**Question 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) 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 with Weiran Liu and Katherine Li. This homework was not released until Sunday afternoon and will be due on Friday. I can see how my entire week will be screwed by the homework.

(b) Please copy the following statement and sign next to it. We just want to make it *extra* clear so that no one inadvertently cheats.

*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 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.

Signature: 

**Question 2.** SVM with custom margins

In the lecture, we covered the soft margin SVM. The objective to be optimized over the training set $\{(\vec{x}_1, y_1), (\vec{x}_2, y_2), \ldots, (\vec{x}_n, y_n)\}$ is

$$\min_{\vec{w},b,\xi_i} \frac{1}{2}||\vec{w}||^2 + C\sum_{i=1}^{n} \xi_i \tag{1}$$

$$s.t. \quad y_i(\vec{w}^\top \vec{x}_i - b) \geq 1 - \xi_i \quad \forall i \tag{2}$$

$$\xi_i \geq 0 \quad \forall i \tag{3}$$

In this problem, we are interested in a modified version of the soft margin SVM where we have a custom margin for each of the $n$ data points. In the standard soft margin SVM, we pay a penalty of $\xi_i$ for each of the data point. In practice, we might not want to treat each training point equally, since with prior knowledge, we might know that some data points are more important than the others. There is some connection to weighted least squares. We formally define the following optimization problem:

$$\min_{\vec{w},b,\xi_i} \frac{1}{2}||\vec{w}||^2 + C\sum_{i=1}^{n} \phi_i\xi_i \tag{4}$$

$$s.t. \quad y_i(\vec{w}^\top \vec{x}_i - b) \geq 1 - \xi_i \quad \forall i \tag{5}$$

$$\xi_i \geq 0 \quad \forall i \tag{6}$$

Note that the only difference is that we have a weighting factor $\phi_i > 0$ for each of the slack variables $\xi_i$ in the objective function. $\phi_i$ are some constants given by the prior knowledge, thus they can be treated as known constants in the optimization problem. Intuitively, this formulation weights each of the violations ($\xi_i$) differently according to the prior knowledge ($\phi_i$).

(a) For the standard soft margin SVM, we have shown that the constrained optimization problem is equal to the following unconstrained optimization problem, i.e. regularized empirical risk minimization problem with hinge loss:

$$\min_{\vec{w},b} \frac{1}{2}||\vec{w}||^2 + C\sum_{i=1}^{n} \max(1 - y_i(\vec{w}^\top \vec{x}_i - b), 0) \tag{7}$$

**What's the corresponding unconstrained optimization problem for the SVM with custom margins?**

$$\min_{\vec{w},b} \frac{1}{2}||\vec{w}||^2 + C\sum_{i=1}^{n} \left[\phi_i \max(1 - y_i(\vec{w}^\top \vec{x}_i - b), 0)\right]$$

We have a weighting factor $\phi_i > 0$ for each of the slack variables $\xi_i$ representing the prior knowledge about the data.

(b) The dual of the standard soft margin SVM is:

$$\max_{\alpha} \alpha^\top \mathbf{1} - \frac{1}{2}\alpha^\top \mathbf{Q}\alpha \tag{8}$$

$$s.t. \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{9}$$

$$0 \le \alpha_i \le C \quad i = 1, \cdots, n \tag{10}$$

where $\mathbf{Q} = (\operatorname{diag} \vec{y})\mathbf{X}\mathbf{X}^T(\operatorname{diag} \vec{y})$

**What's the dual form of the SVM with custom margin? Show the derivation steps in detail.**

Copy the optimization problem from above:

$$\min_{\vec{w},b,\xi_i} \frac{1}{2}||\vec{w}||^2 + C\sum_{i=1}^{n} \phi_i \xi_i$$
$$s.t. \quad y_i(\vec{w}^\top \vec{x}_i - b) \ge 1 - \xi_i \quad \forall i$$
$$\xi_i \ge 0 \quad \forall i$$

Now we can use Lagrangian multiplier to solve this constraint optimization problem. The function of the dual optimization problem can be written as:

$$\max_{\alpha_i,\beta_i} \min_{\vec{w},b,\xi_i} \frac{1}{2}||\vec{w}||^2 + C\sum_{i=1}^{n} \phi_i \xi_i + \sum_{i=1}^{n} \left[\alpha_i * (1 - \xi_i - y_i(\vec{w}^\top \vec{x}_i - b))\right] - \sum_{i=1}^{n} [\beta_i \xi_i]$$
$$s.t. \quad \alpha_i \ge 0, \quad \beta_i \ge 0, \quad \forall i$$

The above optimization problem can be simplified as:

$$\max_{\alpha_i,\beta_i} \min_{\vec{w},b,\xi_i} \frac{1}{2}||\vec{w}||^2 + \sum_{i=1}^{n} [(C\phi_i - \alpha_i - \beta_i)\xi_i] - \sum_{i=1}^{n} \left[\alpha_i * y_i(\vec{w}^\top \vec{x}_i - b)\right] + \sum_{i=1}^{n} \alpha_i$$
$$s.t. \quad \alpha_i \ge 0, \quad \beta_i \ge 0, \quad \forall i$$

Write it in vector form:

$$\max_{\vec{\alpha},\vec{\beta}} \min_{\vec{w},b,\vec{\xi}} \frac{1}{2}||\vec{w}||^2 + (C\vec{\phi} - \vec{\alpha} - \vec{\beta})^\top \vec{\xi} - \vec{\alpha}^\top (\operatorname{diag} y)(\mathbf{X}\vec{w} - b\vec{1}) + \vec{\alpha}^\top \vec{1}$$
$$s.t. \quad \alpha_i \ge 0, \quad \beta_i \ge 0, \quad \forall i$$

Now we take the derivative with respect to all variables and write them out. For simplicity, we denote the above function as $\mathcal{L}$:

---

$$\frac{\partial \mathcal{L}}{\partial \vec{w}} = 0$$

$$\vec{w}^\top - \sum_{i=1}^{n} \alpha_i y_i \vec{x_i}^\top = 0$$

$$(\vec{w^*})^\top = \sum_{i=1}^{n} \alpha_i y_i \vec{x_i}^\top$$

$$(\vec{w}^*)^\top = \vec{\alpha}^\top (\operatorname{diag} y)\mathbf{X}$$

---

$$\frac{\partial \mathcal{L}}{\partial b} = 0$$
$$\sum_{i=1}^{n} \alpha_i^* y_i = 0$$
$$(\vec{\alpha}^*)^\top \vec{y} = 0$$

---

$$\frac{\partial \mathcal{L}}{\vec{\xi}} = 0$$
$$C\vec{\phi} - \vec{\alpha}^* - \vec{\beta}^* = 0$$

---

Now we substitute these equalities to replace $\vec{w}$ with $\vec{\alpha}$:

$$\begin{aligned}
&\mathcal{L} \\
=&\frac{1}{2}||\vec{w}||^2 + (C\vec{\phi} - \vec{\alpha} - \vec{\beta})^\top \vec{\xi} - \vec{\alpha}^\top (\operatorname{diag} y)(\mathbf{X}\vec{w} - b\vec{1}) + \vec{\alpha}^\top \vec{1} \\
=&\frac{1}{2}\vec{\alpha}^\top (\operatorname{diag} y)\mathbf{X}\mathbf{X}^\top (\operatorname{diag} y)\vec{\alpha} + 0 - \vec{\alpha}^\top (\operatorname{diag} y)(\mathbf{X}\mathbf{X}^\top (\operatorname{diag} y)\vec{\alpha} - b\vec{1}) + \vec{\alpha}^\top \vec{1} \\
=&\vec{\alpha}^\top \vec{1} - \frac{1}{2}\vec{\alpha}^\top (\operatorname{diag} y)\mathbf{X}\mathbf{X}^\top (\operatorname{diag} y)\vec{\alpha} \\
=&\vec{\alpha}^\top \vec{1} - \frac{1}{2}\vec{\alpha}^\top \mathbf{Q}\vec{\alpha}
\end{aligned}$$

Therefore, the dual optimization problem is:

$$\begin{aligned}
&\max_{\alpha} \alpha^\top \mathbf{1} - \frac{1}{2}\alpha^\top \mathbf{Q}\alpha \\
&s.t. \sum_{i=1}^{n} \alpha_i y_i = 0 \\
&\quad 0 \le \alpha_i \le C\phi_i \quad i = 1, \cdots, n
\end{aligned}$$

where $\mathbf{Q} = (\operatorname{diag} \vec{y})\mathbf{X}\mathbf{X}^T(\operatorname{diag} \vec{y})$

(c) **From the dual formulation above, how would you kernelize the SVM with custom margins? What role does the $\phi_i$ play in the kernelized version?**

I would kernelize the SVM with custom margins by replacing $XX^\top$ with the gram matrix $\mathbf{K} = \mathbf{\Phi}^\top \mathbf{\Phi}$ where the ij-th entry of $\mathbf{K}$ is $(\mathbf{K})_{ij} = (\mathbf{\Phi}^\top \mathbf{\Phi})_{ij} = k(\vec{x}_i, \vec{x}_j)$.
$\phi_i$ can make the constraints at different directions uneven. We might apply tighter bounds on some $\alpha_i$, where we believe those data points must be correctly classified and apply looser bounds on other $\alpha_i$ for points we don't trust as much. It serves as something like Tikhonov matrix where we regularize different points independently.

**Question 3.** Nearest Neighbors, from A to Z

For this problem, we will use data from the UN to have some fun with the nearest neighbors approach to learning. A lot of the code you will need has been provided for you.

The data we are using is called the "World Values Survey." It consists of survey data collection over several years from almost all countries. The survey asked "Which of these are most important for you and your family?" There were 16 possible responses, including needs like "Freedom from Discrimination and Persecution" and "Better Transport and Roads." The data reported is the fraction of responses in each country that chose each option.

We would like to use these 16 features of each country (the citizen's responses to the survey) to predict that country's HDI (Human Development Index). In reality, the HDI is a complex measure which takes into account lots of data about a country, including factors like life expectancy, education, per capita income, etc. Intuitively though, you might expect citizens of countries with different HDI to have different priorities. For that reason, predicting the HDI from survey data might be a reasonable endeavor.

Note that throughout the problem we will be using RMSE, which stands for Root Mean Squared Error.

(a) (Bonus): **Fill out the "Berkeley's S2018 Values Survey."** The purpose of this is so that you have a sense of how the data was generated, a useful first step in any ML problem. Just for fun, at the end of this problem we will attempt to predict what the HDI of Berkeley would be if it were its own country.

| Please see the attached PDF page: |
|---|

Which of these are most important to you and your family? (Please choose exactly 6 responses). *

- [ ] Action taken on climate change
- [ ] Better transport and roads
- [ ] Support for people who can't work
- [ ] Access to clean water and sanitation
- [x] Better healthcare
- [x] A good education
- [x] A responsive government we can trust
- [ ] Phone and internet access
- [ ] Reliable energy at home
- [x] Affordable and nutritious food
- [ ] Protecting forests rivers and oceans
- [x] Protection against crime and violence
- [ ] Political freedoms
- [ ] Freedom from discrimination and persecution
- [ ] Equality between men and women
- [x] Better job opportunities

SUBMIT

(b) First, we should do some basic data exploration. **Compute the correlation of each feature with HDI. Which feature is the most positively correlated with HDI? Which feature is the most negatively correlated with HDI? Which feature is the least correlated with HDI (closest to 0)?**

Predicting HDI from World Values Survey

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

Action taken on climate change 0.47331289154299033
Better transport and roads -0.43963363862245863
Support for people who can't work -0.33621323672149067
Access to clean water and sanitation -0.018169084455954734
Better healthcare -0.4220123599593434
A good education -0.30397888977158355
A responsive government we can trust 0.32944531498417795
Phone and internet access -0.3516047121577416
Reliable energy at home -0.2854235638359744
Affordable and nutritious food 0.19519330078603303
Protecting forests rivers and oceans 0.6134587562712408
Protection against crime and violence 0.14331869917957565
Political freedoms 0.23809900682145022
Freedom from discrimination and persecution 0.4329323754445626
Equality between men and women 0.27649604349752843
Better job opportunities -0.3973445267396408
Correlation matrix:

$$\begin{bmatrix} 0.4733 & -0.4396 & -0.3362 & -0.0182 & -0.422 & -0.304 & 0.3294 & -0.3516 & -0.2854 \\ 0.1952 & 0.6135 & 0.1433 & 0.2381 & 0.4329 & 0.2765 & -0.3973 & & \end{bmatrix}$$

The feature that is most positively corrected with HDI: Protecting forests rivers and oceans
The feature that is most negatively corrected with HDI: Better transport and roads
The feature that is least corrected with HDI: Access to clean water and sanitation

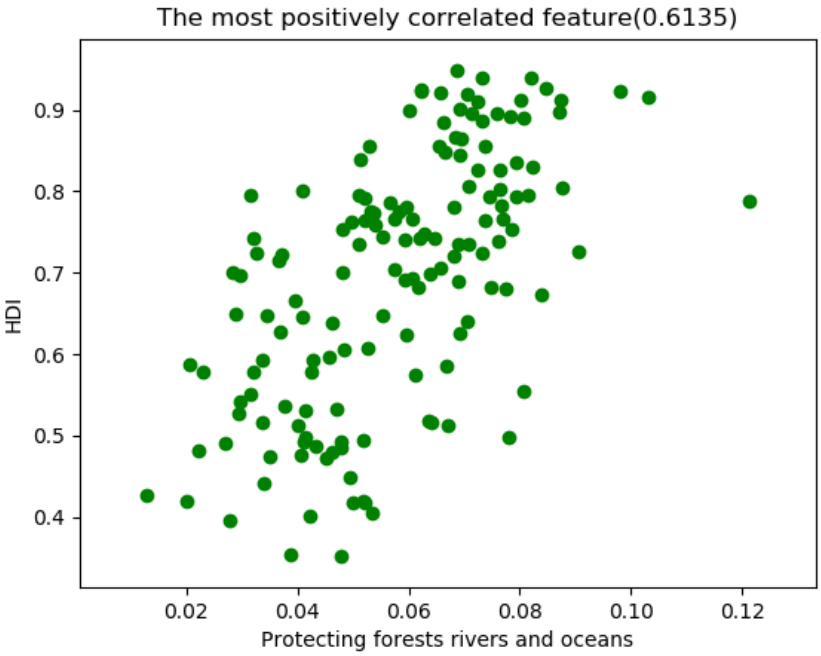
---

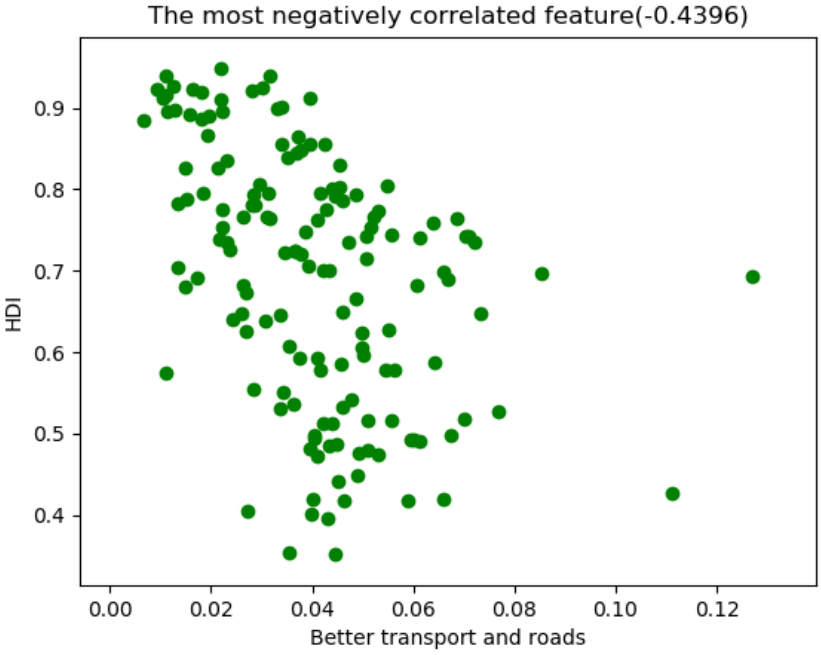
```
def plot_hdi_vs_feature(training_features, training_labels, feature, color, title):
"""
Input:
training_features: world_values responses on the training set
training_labels: HDI (human development index) on the training set
feature: name of one selected feature from training_features
color: color to plot selected feature
title: title of plot to display

Output:
Displays plot of HDI vs one selected feature.
```

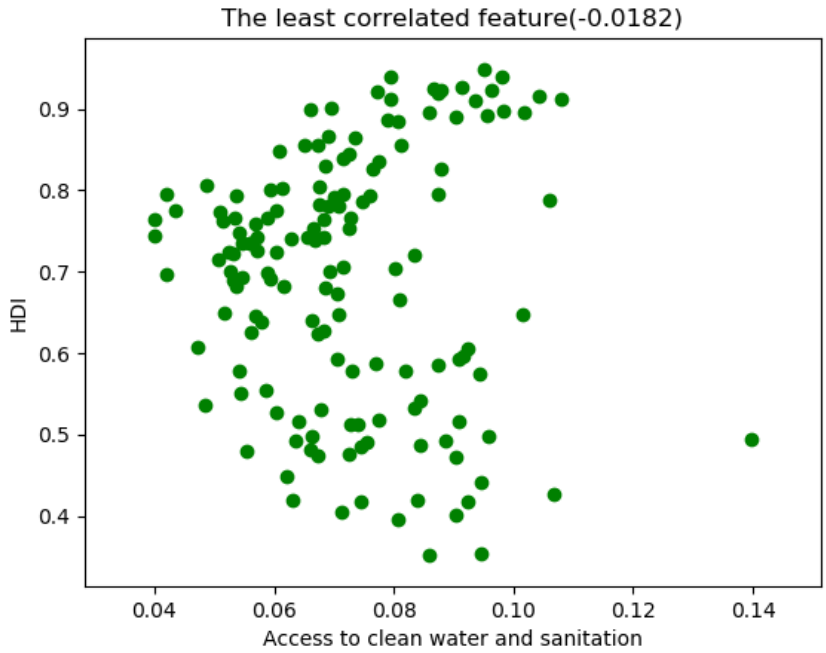
```
"""
plt.scatter(training_features[feature],
training_labels['2015'],
c=color)
plt.title(title)
plt.xlabel(feature)
plt.ylabel('HDI')
#plt.show()
plt.savefig('Figure_3c-'+ title.split('(')[0] + '.png')
plt.close()


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

Output:
Prints correlations between HDI and each feature, separately.
Displays plot of HDI vs one selected feature.
"""
# Calculate correlations between HDI and each feature
correlations = []
for column in training_features.columns:
print(column, training_features[column].corr(training_labels['2015']))
correlations.append(round(training_features[column].corr(training_labels['2015']), 4))
print('Correlation matrix:')
print(correlations)
print()

# Identify three features
idxCorr = np.argmax(correlations)
print('The feature that is most positively corrected with HDI: '+training_features.columns[idxCorr])
plot_hdi_vs_feature(training_features, training_labels, training_features.columns[idxCorr],
'green', 'The most positively correlated feature('+str(correlations[idxCorr])+')')

idxCorr = np.argmin(correlations)
print('The feature that is most negatively corrected with HDI: '+training_features.columns[idxCorr])
plot_hdi_vs_feature(training_features, training_labels, training_features.columns[idxCorr],
'green', 'The most negatively correlated feature('+str(correlations[idxCorr])+')')

idxCorr = np.argmin(np.abs(correlations))
print('The feature that is least corrected with HDI: '+training_features.columns[idxCorr])
plot_hdi_vs_feature(training_features, training_labels, training_features.columns[idxCorr],
'green', 'The least correlated feature('+str(correlations[idxCorr])+')')

#plot_hdi_vs_feature(training_features, training_labels, 'Action taken on climate change',
#                    'green', 'HDI versus ActionTakenClimateChange')
```

(c) **For each of these three features identified in  (most positively correlated, most negatively correlated, least correlated), plot "HDI versus [Feature]."** You will create three plots in total. **What do you observe?**

I observe a positive trend on the first graph; a negative trend on the second graph; and no trend on the last graph.
The first and the second features are better predictors of HDI than the third feature which tells you nothing about HDI.



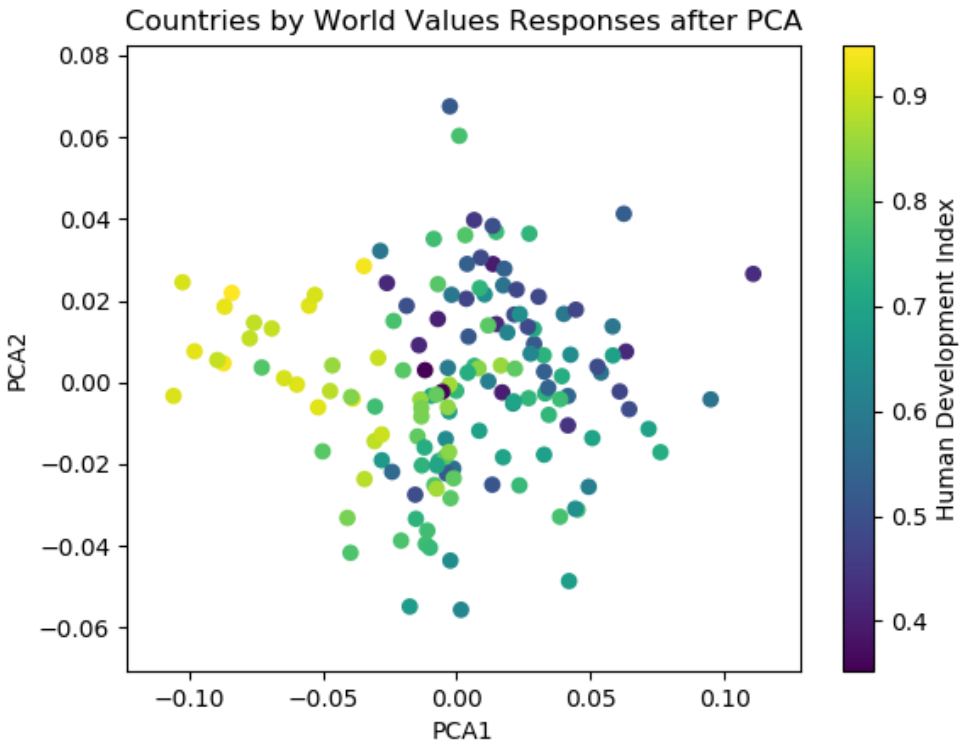






```
def plot_hdi_vs_feature(training_features, training_labels, feature, color, title):
"""
Input:
training_features: world_values responses on the training set
training_labels: HDI (human development index) on the training set
feature: name of one selected feature from training_features
color: color 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.xlabel(feature)
plt.ylabel('HDI')
#plt.show()
plt.savefig('Figure_3c-'+ title.split('(')[0] + '.png')
plt.close()


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

Output:
Prints correlations between HDI and each feature, separately.
Displays plot of HDI vs one selected feature.
"""
# Calculate correlations between HDI and each feature
correlations = []
for column in training_features.columns:
```

```
print(column, training_features[column].corr(training_labels['2015']))
correlations.append(round(training_features[column].corr(training_labels['2015']), 4))
print('Correlation matrix:')
print(correlations)
print()

# Identify three features
idxCorr = np.argmax(correlations)
print('The feature that is most positively corrected with HDI: '+training_features.columns[idxCorr])
plot_hdi_vs_feature(training_features, training_labels, training_features.columns[idxCorr],
'green', 'The most positively correlated feature('+str(correlations[idxCorr])+')')

idxCorr = np.argmin(correlations)
print('The feature that is most negatively corrected with HDI: '+training_features.columns[idxCorr])
plot_hdi_vs_feature(training_features, training_labels, training_features.columns[idxCorr],
'green', 'The most negatively correlated feature('+str(correlations[idxCorr])+')')

idxCorr = np.argmin(np.abs(correlations))
print('The feature that is least corrected with HDI: '+training_features.columns[idxCorr])
plot_hdi_vs_feature(training_features, training_labels, training_features.columns[idxCorr],
'green', 'The least correlated feature('+str(correlations[idxCorr])+')')

#plot_hdi_vs_feature(training_features, training_labels, 'Action taken on climate change',
#                    'green', 'HDI versus ActionTakenClimateChange')
```

(d) Let's visualize the data a bit more. **Plot the data in its first two PCA dimensions, colored by HDI.** The code to do this has been provided for you.




```
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)

print(training_labels)

# 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()
plt.xlabel('PCA1')
```

```
plt.ylabel('PCA2')
plt.savefig('Figure_3d-PCA_heatmap.png')
plt.close()

# 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()
plt.xlabel('PCA1')
plt.ylabel('PCA2')
plt.savefig('Figure_3d-PCA_multicolor.png')
plt.close()
```

(e) Now, let's use our first ML technique. **Use the code provided to train and cross-validate ridge regression to predict a country's HDI from its citizens' world values survey responses. What is the best RMSE?**

Ridge Regression
RMSE: 0.12303337350607803
Pipeline(memory=None, steps=[('ridge', Ridge(alpha=0.02, copy_X=True, fit_intercept=True, max_iter=None, normalize=False, random_state=None, solver='auto', tol=0.001))])
Coefficients

$$\begin{bmatrix} 0.80823467 & -0.74985758 & -0.17800015 & -1.28408103 & -0.66293176 & -0.82203172 \\ 0.73733884 & -0.92891581 & -0.82049672 & 0.39614952 & 2.0708291 & -0.06718981 \\ 0.48310656 & 0.72671425 & 0.42921192 & -0.13808023 & & \end{bmatrix}$$

---

```
def _rmse_grid_search(training_features, training_labels, pipeline, parameters, technique,
    title='Figure.png'):
"""
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("Coefficients")
print(grid.best_estimator_.named_steps[technique].coef_)
else:
idx = np.argmin(-grid.cv_results_['mean_test_score'])
print('The best k is '+ str(grid.cv_results_['param_knn__n_neighbors'][idx]))
# 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()
plt.savefig(title)
plt.close()
```

```
print()
return grid


def regression_grid_searches(training_features, training_labels, testing_features=None,
    title='Figure.png'):
"""
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("Ridge Regression")
_rmse_grid_search(training_features, training_labels,
ridge_regression_pipeline, regression_ridge_parameters, 'ridge', title)

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

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

if testing_features is not None:
print(grid.predict(testing_features))
```

(f) Let's try another ML technique. **Use the code provided to train and cross-validate LASSO regression to predict a country's HDI from its citizens' world values survey responses. What is the best RMSE?**

Lasso Regression
RMSE: 0.12602242808947525
Pipeline(memory=None, steps=[('lasso', Lasso(alpha=0.0002, 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))])
Coefficients

$$\begin{bmatrix} 0.1590192 & -0.72844929 & -0. & -0.85945074 & -0.66274144 & -0.02556703 \\ 0.33904781 & -0.29897158 & -0. & 0. & 3.48536375 & 0. \\ 0. & 0.87057995 & 0.32897045 & -0. & & \end{bmatrix}$$

---

```
def _rmse_grid_search(training_features, training_labels, pipeline, parameters, technique,
    title='Figure.png'):
"""
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("Coefficients")
print(grid.best_estimator_.named_steps[technique].coef_)
else:
idx = np.argmin(-grid.cv_results_['mean_test_score'])
print('The best k is '+ str(grid.cv_results_['param_knn__n_neighbors'][idx]))
# 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()
plt.savefig(title)
plt.close()
```

```
print()
return grid


def regression_grid_searches(training_features, training_labels, testing_features=None,
    title='Figure.png'):
"""
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("Ridge Regression")
_rmse_grid_search(training_features, training_labels,
ridge_regression_pipeline, regression_ridge_parameters, 'ridge', title)

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

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

if testing_features is not None:
print(grid.predict(testing_features))
```

(g) **Examine the model returned by LASSO regression (that is, the 16 feature weights). Does LASSO regression indeed give more 0 weights?**

By examining the model returned by LASSO regression, we can see that LASSO regression indeed give more 0 weights (6/16) than Ridge regression (0/16).

(h) In lecture, we covered $k$-Nearest Neighbors for classification problems. We decided that the class of a test point would be the plurality of the classes of the $k$ nearest training points. That algorithm makes sense when the outputs are discrete, so we can vote. Here, the outputs are continuous. **How would you adapt the $k$ Nearest Neighbors algorithm for a regression problem?**

I found the following lines of code in the provided code:

```
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())
]
)
```

Now we can go into KNeightborsRegressor() and read its source code/comments:

```
'''
Examples
--------
>>> X = [[0], [1], [2], [3]]
>>> y = [0, 0, 1, 1]
>>> from sklearn.neighbors import KNeighborsRegressor
>>> neigh = KNeighborsRegressor(n_neighbors=2)
>>> neigh.fit(X, y) # doctest: +ELLIPSIS
KNeighborsRegressor(...)
>>> print(neigh.predict([[1.5]]))
[ 0.5]
'''
```

Basically, for regression problems, we could take the average value of k nearest neighbors as the predicted value for a test point.

(i) **Which countries are the 7 nearest neighbors of the USA (in order)?**

The nearest countries to the USA:
0: United States(idx=[45]) at a distance of [0.]
1: Ireland(idx=[90]) at a distance of [0.01765082]
2: United Kingdom(idx=[61]) at a distance of [0.02157067]
3: Belgium(idx=[37]) at a distance of [0.03010195]
4: Finland(idx=[108]) at a distance of [0.0302092]
5: Malta(idx=[69]) at a distance of [0.03349605]
6: Austria(idx=[132]) at a distance of [0.03583628]
7: France(idx=[110]) at a distance of [0.03897199]

---

```
# Part i, m, : 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]])
countries_train = pd.read_csv('world-values-train2.csv')
countries_train = countries_train['Country']
print('The nearest countries to the USA: ')
for i, (dist, idx) in enumerate(zip(distances.T, indices.T)):
print(str(i) + ': ' + str(countries_train[idx].to_string(index=False)) + '(idx=' + str(idx) + ') at a
    distance of ' + str(dist))
print()
```

---

Below was my interpretation the first time I read the question. Here is a table from www.distance-between-countries.com:

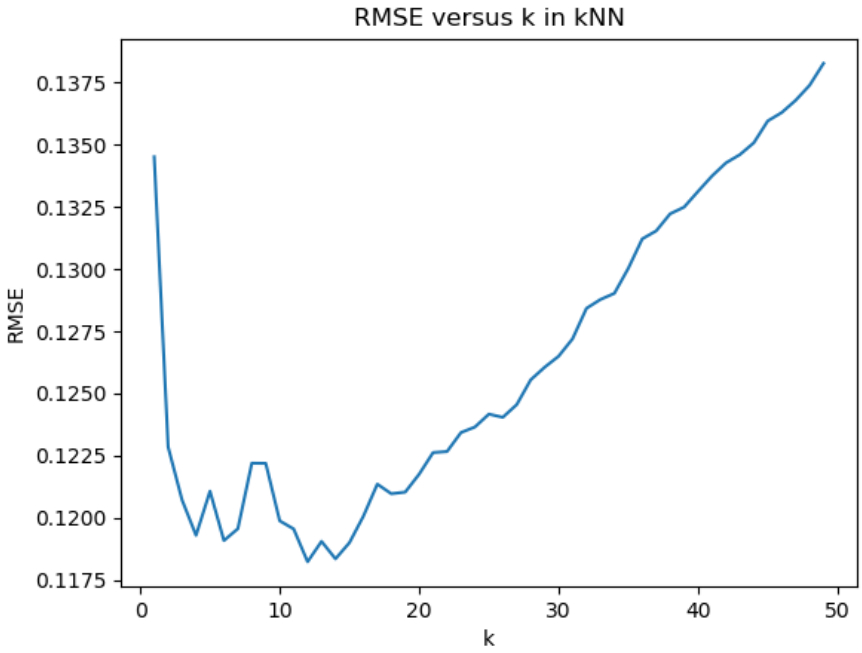
## Countries near USA

Countries distances from USA - ordered by A - B | Countries near USA

| | | |
|---|---|---|
| Up to 1,000 kilometers | | |
| Distance from USA to **Canada** is: 742 kilometer | Distance from Washington to **Ottawa** | Mid |
| Up to 1,500 kilometers | | |
| Distance from USA to **Bermuda** is: 1,318 kilometer | Distance from Washington to **Hamilton** | Mid |
| Up to 2,000 kilometers | | |
| Distance from USA to **Bahamas** is: 1,524 kilometer | Distance from Washington to **Nassau** | Mid |
| Distance from USA to **Cuba** is: 1,813 kilometer | Distance from Washington to **Havana** | Mid |
| Up to 2,500 kilometers | | |
| Distance from USA to **Cayman Islands** is: 2,206 kilometer | Distance from Washington to **George Town** | Mid |
| Distance from USA to **Haiti** is: 2,292 kilometer | Distance from Washington to **Port-au-Prince** | Mid |
| Distance from USA to **Jamaica** is: 2,309 kilometer | Distance from Washington to **Kingston** | Mid |
| Distance from USA to **Dominican Republic** is: 2,357 kilometer | Distance from Washington to **Santo Domingo** | Mid |
| Distance from USA to **Puerto Rico** is: 2,488 kilometer | Distance from Washington to **San Juan** | Mid |

(j) The most important meta-parameter of $k$ nearest neighbors is $k$ itself. **Plot the RMSE of kNN regression versus $k$, where $k$ is the number of neighbors. What is the best value of $k$? What is the RMSE?**

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'))])
The best k is 12




```
def _rmse_grid_search(training_features, training_labels, pipeline, parameters, technique,
    title='Figure.png'):
"""
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)
```

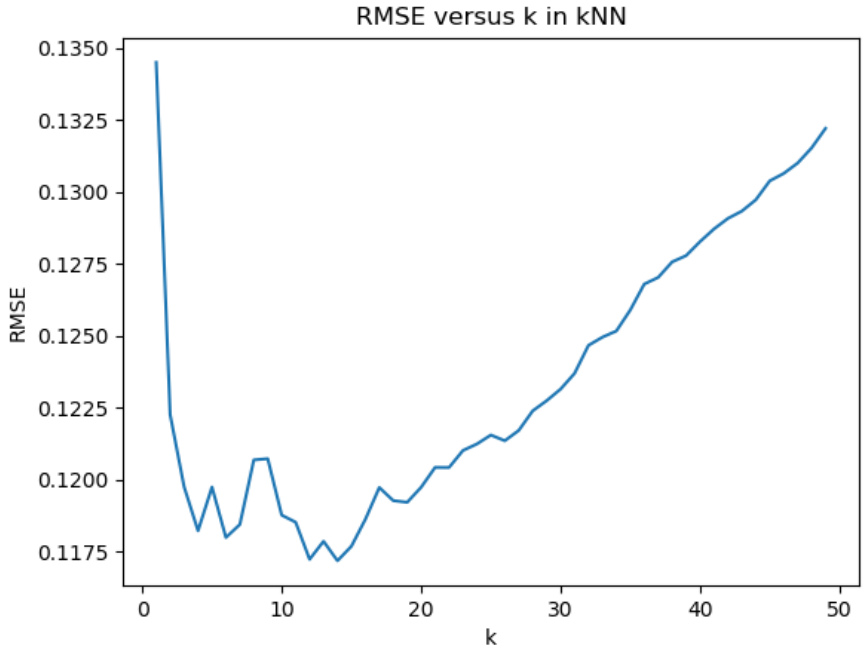
```
print("RMSE:", sqrt(-grid.best_score_))
print(grid.best_estimator_)

# Check Ridge or Lasso Regression
if hasattr(grid.best_estimator_.named_steps[technique], 'coef_'):
print("Coefficients")
print(grid.best_estimator_.named_steps[technique].coef_)
else:
idx = np.argmin(-grid.cv_results_['mean_test_score'])
print('The best k is '+ str(grid.cv_results_['param_knn__n_neighbors'][idx]))
# 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()
plt.savefig(title)
plt.close()

print()
return grid


def regression_grid_searches(training_features, training_labels, testing_features=None,
    title='Figure.png'):
"""
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("Ridge Regression")
_rmse_grid_search(training_features, training_labels,
ridge_regression_pipeline, regression_ridge_parameters, 'ridge', title)

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

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

if testing_features is not None:
print(grid.predict(testing_features))
```

(k) **Explain your plot in () in terms of bias and variance.** This is tricky, so take some time to think about it. Think about the spirit of bias and variance more than their precise definitions.

When we increase k, we average over more neighbors and more data points, so the variance will decrease.
However, when there only exists finite number of data points, increasing k will also result in increasing bias. This is because as we average over more data points, we start to include more points that are further away from our test points whose votes might not represent our test points.
For the extreme case, where we have $k = 1$. The variance of our model is very high and the bias is pretty low. On the other hand, where we have $k = 10000$, the bias is high because we just predict similar values for a bunch of different test points by averaging over a lot of training points. Therefore, there is a bias-variance tradeoff.

(l) We do not need to give every neighbor an equal weight. Maybe closer neighbors are more relevant. For the sake of this problem, let's weight each neighbor by the inverse of its distance to the test point. **Plot the RMSE of kNN regression with distance weighting versus $k$, where $k$ is the number of features. What is the best value of $k$? What is the RMSE?**

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'))])
The best k is 14




```
print("k Nearest Neighbors Regression")
_rmse_grid_search(training_features=values_train, training_labels=hdi_train,
pipeline=k_nearest_neighbors_regression_pipeline,
parameters=regression_knn_parameters_weighted, technique='knn', title='Figure_3l-RMSEvsk.png')
```

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

'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']
}
```

(m) One of the challenges of $k$ Nearest Neighbors is that it is very sensitive to the scale of the features. For example, if one feature takes on values 0 or 0.1 and another takes on values 0 or 10, then the nearest neighbors approach will almost certainly pick nearest neighbors according to the second feature. **Which countries are the 7 nearest neighbors of the USA after scaling (in order)? Compare your result to .**

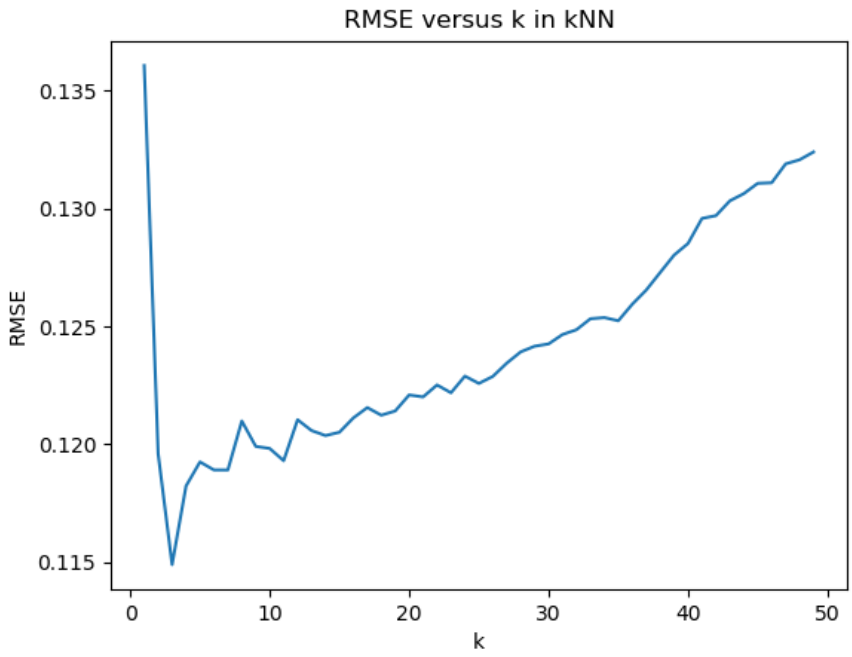
The nearest countries to the USA (after scaling):
0: United States(idx=[45]) at a normalized distance of [0.]
1: Ireland(idx=[90]) at a normalized distance of [1.1895638]
2: United Kingdom(idx=[61]) at a normalized distance of [1.37459492]
3: Finland(idx=[108]) at a normalized distance of [1.81818296]
4: Belgium(idx=[37]) at a normalized distance of [1.94193658]
5: Malta(idx=[69]) at a normalized distance of [2.28988047]
6: France(idx=[110]) at a normalized distance of [2.29063682]
7: Austria(idx=[132]) at a normalized distance of [2.39411947]

```
stdscaler = StandardScaler()
stdscaler.fit(values_train)
values_train = stdscaler.transform(values_train)
nbrs = NearestNeighbors(n_neighbors=8).fit(values_train)
distances, indices = nbrs.kneighbors([values_train[45, :]])
countries_train = pd.read_csv('world-values-train2.csv')
countries_train = countries_train['Country']
print('The nearest countries to the USA (after scaling): ')
for i, (dist, idx) in enumerate(zip(distances.T, indices.T)):
print(str(i) + ': ' + str(countries_train[idx].to_string(index=False))
+ '(idx=' + str(idx) + ') at a normalized distance of ' + str(dist))
print()
```

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

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

('knn', KNeighborsRegressor())
]
)

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

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

('knn', KNeighborsRegressor())
]
)
```

(n) **Add scaling to your $k$ nearest neighbors pipeline (continue to use distance weighting). Plot RMSE versus $k$. What is the best value for $k$? What is the RMSE?**

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'))])
The best k is 3

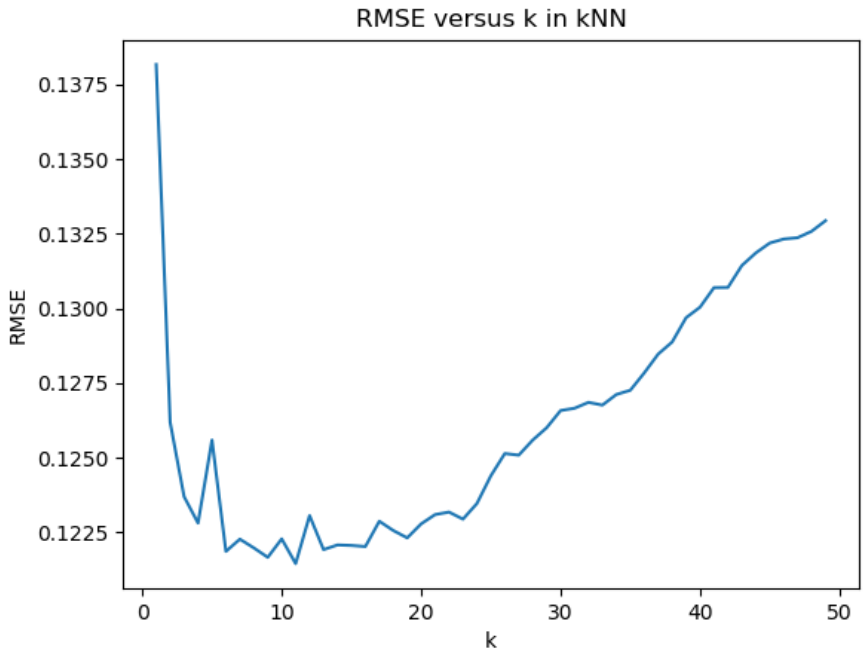



```
# Part n

print("k Nearest Neighbors Regression")
_rmse_grid_search(training_features=values_train, training_labels=hdi_train,
pipeline=k_nearest_neighbors_regression_pipeline_scaled,
parameters=regression_knn_parameters_weighted, technique='knn',
title='Figure_3n-RMSEvsk.png')
```

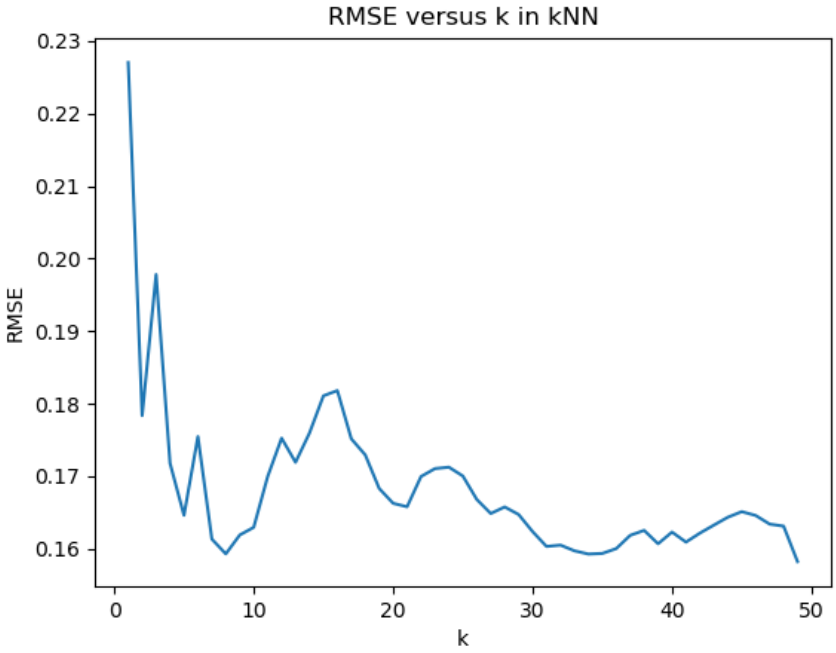
```
k_nearest_neighbors_regression_pipeline_scaled = Pipeline(
[
# Apply PCA to k Nearest Neighbors Regression
# ('pca', PCA()),

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

('knn', KNeighborsRegressor())
]
)
```

(o) (Bonus): **Rather than scaling each feature to have unit variance, explore ways of scaling the features non-uniformly. How much does this help, if at all?**

I tried MinMaxScaler and Binarizer. Neither helps. The detail about these two Scalers can be found here.
Basically, MinMaxScaler scales all data to be between zero and one. Binarizer uses the threshold 0 to make the dataset binary.
k Nearest Neighbors Regression (MinMaxScaler)
RMSE: 0.12144941346487255
Pipeline(memory=None, steps=[('scale', MinMaxScaler(copy=True, feature_range=(0, 1))), ('knn', KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=11, p=2, weights='distance'))])
The best k is 11




k Nearest Neighbors Regression (Binarizer)
RMSE: 0.15824533088817846
Pipeline(memory=None, steps=[('scale', Binarizer(copy=True, threshold=0.0)), ('knn', KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=49, p=2, weights='distance'))])
The best k is 49




```
# Part o
print("k Nearest Neighbors Regression (MinMaxScaler)")
_rmse_grid_search(training_features=values_train, training_labels=hdi_train,
pipeline=k_nearest_neighbors_regression_pipeline_minmax,
parameters=regression_knn_parameters_weighted, technique='knn',
title='Figure_3o-RMSEvsk_minmax.png')
print("k Nearest Neighbors Regression (Binarizer)")
_rmse_grid_search(training_features=values_train, training_labels=hdi_train,
pipeline=k_nearest_neighbors_regression_pipeline_binary,
parameters=regression_knn_parameters_weighted, technique='knn',
title='Figure_3o-RMSEvsk_binary.png')
```

(p) You have been given a set of test features: countries where the responses to the world values survey are given but the HDI is not known. **Using the best model developed so far, predict the HDI values of the countries in the test set. Submit your predictions on Gradescope.**

The current best model is:
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'))])
The best k is 3
These results have been saved to "world-values-hdi-test-predictions.csv"

| | Country | 2015 |
|---|---|---|
| 0 | Egypt | 0.5109996999832642 |
| 1 | Myanmar | 0.6157075259513639 |
| 2 | Namibia | 0.6833111345495279 |
| 3 | Bosnia and Herzegovina | 0.6785222906085072 |
| 4 | Panama | 0.7718752979696814 |
| 5 | Guinea | 0.6041902393336662 |
| 6 | Bahrain | 0.6515989725667951 |
| 7 | Congo (Democratic Republic of the) | 0.6794557870322746 |
| 8 | Costa Rica | 0.7310534590579016 |
| 9 | Cyprus | 0.8053591964492586 |
| 10 | China | 0.6793817419943524 |
| 11 | Micronesia (Federated States of) | 0.6652088081324105 |
| 12 | Malaysia | 0.7793256953694826 |
| 13 | Saint Vincent and the Grenadines | 0.7058476200522537 |
| 14 | Botswana | 0.6656819643795583 |
| 15 | Solomon Islands | 0.5669110097521087 |
| 16 | Peru | 0.6925223471096617 |
| 17 | Saudi Arabia | 0.7270626681280055 |
| 18 | Czech Republic | 0.7049873632311213 |
| 19 | Andorra | 0.7751388762664372 |
| 20 | Nepal | 0.6338426792619273 |
| 21 | Colombia | 0.6508953846760894 |
| 22 | Guinea-Bissau | 0.39709827083113874 |
| 23 | Chile | 0.7344202608929177 |
| 24 | Estonia | 0.7583716005765707 |
| 25 | Georgia | 0.7115726597331336 |
| 26 | Gambia | 0.44080564121620724 |
| 27 | Canada | 0.9046036616176822 |
| 28 | Kyrgyzstan | 0.6281853745417723 |
| 29 | Lebanon | 0.6520822837572857 |
| 30 | Jamaica | 0.6630599026730262 |
| 31 | Portugal | 0.8653921449448525 |
| 32 | Cambodia | 0.7287015223580222 |
| 33 | Luxembourg | 0.9098930186579914 |
| 34 | Denmark | 0.9243620026615066 |
| 35 | Japan | 0.8432591009863225 |
| 36 | Lesotho | 0.49853809244565095 |
| 37 | Belarus | 0.8081057667371241 |

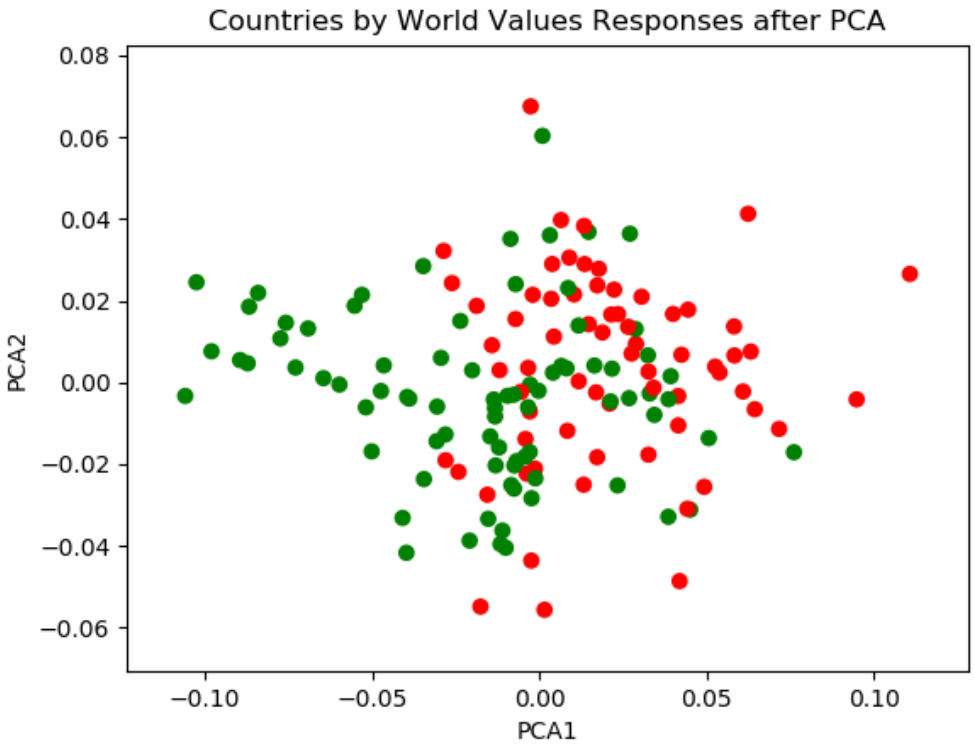
```
print('The current best model is: ')
grid = _rmse_grid_search(training_features=values_train, training_labels=hdi_train,
pipeline=k_nearest_neighbors_regression_pipeline_scaled,
parameters=regression_knn_parameters_weighted, technique='knn',
```

```
title='Figure_3p-RMSEvsk.png')
pred_values = grid.predict(values_test)
countries_test = pd.read_csv('world-values-test.csv')
countries_test = countries_test[['Country']]
pred_values = pred_values.flatten()
countries_test['2015'] = pred_values
countries_test.to_csv('world-values-hdi-test-predictions.csv')
print(countries_test)
```

(q) So far we have dealt with the regression problem. Let's take a brief look at classification. A naive classifier is a classifier which disregards the features and just classifies everything as belonging to a single class. **In any classification problem with $k$ classes, at least what accuracy are we guaranteed to get with the best naive classifier?** (Hint: there are $k$ possible naive classifiers. Use the pigeonhole principle).

First of all, what's the pigeonhole principle?
After a little Google search, it's not hard to find it out on Wikipedia: Pigeonhole principle. In Chinese, we call it "Drawer Principle", which is one of the most important concept we learned in elementary school.
Let's say there are $n$ data points. The best naive classifier is to classify all points to the class of majority. To qualify to be majority, that class must have at least $\lceil \frac{n}{k} \rceil$. It should be noted that on Wikipedia, the formula is written as $\lfloor \frac{n-1}{k} \rfloor + 1$, and these two expressions are equivalent.
**Therefore, we can guarantee an accuracy of $\frac{1}{n} \lceil \frac{n}{k} \rceil \approx \frac{1}{k}$ for the best naive classifier.**

(r) We will split countries into two groups: high HDI (more than 0.7) and low HDI (less than 0.7). **Plot the countries by their first two PCA dimensions again, but now color them by class.**




```
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)

print(training_labels)

# 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()
plt.xlabel('PCA1')
```

```
plt.ylabel('PCA2')
plt.savefig('Figure_3d-PCA_heatmap.png')
plt.close()

# 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()
plt.xlabel('PCA1')
plt.ylabel('PCA2')
plt.savefig('Figure_3d-PCA_multicolor.png')
plt.close()
```

(s) Examine the graph generated in (). **How well do you think a linear SVM would do in classification?**

By examining the graph, I think a linear SVM would do terrible in classification without performing any nonlinear transformation on the data. This is simply because the original data is not linearly separable and two classes overlap a lot in terms of the first two principle components. We will need a very soft margin to account for the overlap.

(t) We will use an SVM classifier to predict whether a country's HDI is "high" or "low" based on the responses of their citizens to the World Values Survey. **Use the code provided to train and cross-validate an SVM classifier using a linear kernel. What is the accuracy of the classifier?**

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))])

```
print("SVM Classification")
_accuracy_grid_search(values_train, hdi_class_train,
svm_classification_pipeline,
classification_svm_parameters)
```

```
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)
}
```

(u) We are going to modify the classifier from (). **Add a PCA step and Scaling step to the SVM pipeline. Your hyper-parameter search should now be over all possible dimensions for the PCA reduction. Does the accuracy improve?**

Yes, it does improve to over 0.8.

SVM Classification (scaled PCA)
Accuracy:

$$[0.8108108108108109]$$

Pipeline(memory=None, steps=[('pca', PCA(copy=True, iterated_power='auto', n_components=7, random_state=None, svd_solver='auto', tol=0.0, whiten=False)), ('scale', StandardScaler(copy=True, with_mean=True, with_std=True)), ('svm', SVC(C=0.02, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False))])

```
print("SVM Classification (scaled PCA)")
_accuracy_grid_search(values_train, hdi_class_train,
svm_classification_pipeline_pca_scaled,
classification_svm_parameters_pca_scaled)
```

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

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

('svm', SVC())
]
)
```

```
classification_svm_parameters_pca_scaled = {
# 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)
```

```
}
```

(v) Change the kernel in from linear to "radial basis function" (rbf). For this part, do not use PCA or Scaling. **What is the accuracy?**

SVM Classification (RBF kernel)
Accuracy:

$$[0.7432432432432432]$$

Pipeline(memory=None, steps=[('svm', SVC(C=73.0, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape='ovr', degree=3, gamma=0.097000000000000003, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False))])

```
print("SVM Classification (RBF kernel)")
_accuracy_grid_search(values_train, hdi_class_train,
svm_classification_pipeline,
classification_svm_parameters_rbf)
```

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

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

('svm', SVC())
]
)
```

```
classification_svm_parameters_rbf = {
# 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)
}
```

(w) Now we are going to use $k$ Nearest Neighbors for the same task. That is, we would like to predict whether a country's HDI is "high" or "low" based on the responses of their citizens to the World Values Survey. **Train and cross-validate a $k$ Nearest Neighbors classifier using distance weighting. What is its accuracy? Does scaling help?**

The scaling helps a little but not much.

k Nearest Neighbors Classification
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'))])

Here is the result after scaling:
k Nearest Neighbors Classification (scaled)
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'))])

```
print("k Nearest Neighbors Classification")
_accuracy_grid_search(values_train, hdi_class_train,
k_nearest_neighbors_classification_pipeline,
classification_knn_parameters)
print('Here is the result after scaling: ')
print("k Nearest Neighbors Classification (scaled)")
_accuracy_grid_search(values_train, hdi_class_train,
k_nearest_neighbors_classification_pipeline_scaled,
classification_knn_parameters)
```

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

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

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

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

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

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

(x) (Bonus): Towards the end of the week, we will post the "Berkeley's S2018 Values Survey." **If this course were an independent country, what do you predict its HDI would be?**

Here are my predictions: The current best model is:
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'))])
The best k is 3

| | Country | 2015 |
|---|---|---|
| 0 | B1 | 0.4619426173007891 |
| 1 | B2 | 0.46217360347834746 |
| 2 | B3 | 0.46152188884173234 |

```
values_test = pd.read_csv('world-values-test-berkeley.csv')
values_test = values_test.drop(['Country'], axis=1)
print('The current best model is: ')
grid = _rmse_grid_search(training_features=values_train, training_labels=hdi_train,
pipeline=k_nearest_neighbors_regression_pipeline_scaled,
parameters=regression_knn_parameters_weighted, technique='knn',
title='Figure_3x-RMSEvsk.png')
pred_values = grid.predict(values_test)
countries_test = pd.read_csv('world-values-test-berkeley.csv')
countries_test = countries_test[['Country']]
pred_values = pred_values.flatten()
countries_test['2015'] = pred_values
countries_test.to_csv('world-values-test-berkeley-predictions.csv')
print(countries_test)
```

(y) (Bonus): **Describe how you would use kNN to revisit the sensor location problem from previous homework. How well do you think it will work?**

Let's say we have $n$ sensors with some object points as our training data and some object points as our test data.
I would kNN to solve this regression problem. First, I would find the k nearest neighbors of the test data points in the $n$ dimensional space. Second, I would use linear interpolation in the 2D space to get an average coordinates of the k nearest neighbors of the test data points. This will be the predicted coordinates of the test data points.

(z) (Bonus): **What did you learn from this problem? Do you have any useful feedback for the problem author?**

I learned that making a question with 26 part is possible.
My suggestion to the author is that a question with 26 parts is long.

---

I learned that kNN can be coupled with scaling, which gives a relatively good performance compared to a more complicated algorithm like kernel SVM.
My suggestion is that graphing always gives better intuition to understand a certain concept than printing out some numbers. More parts could be changed to draw a graph for hyperparameter search.

**Question 4.** Stability: A Unified Approach to Generalization for Classification and Regression

In this problem, we will study how well machine learning algorithms can generalize from the dataset they have been trained on (the training set) to an unseen dataset (the test set). Assume all data is generated from some distribution $\mathcal{D}$ and we sample a training set $S \sim \mathcal{D}^n$ where $\mathcal{D}^n$ is the distribution of $n$ datapoints drawn i.i.d. from $\mathcal{D}$. In this problem we consider estimators of the form

$$\vec{w}(S) = \arg\min_{\vec{w}} L_S(\vec{w}) + \lambda \|\vec{w}\|^2 \tag{11}$$

where $L_S(\vec{w})$ is the empirical loss function

$$L_S(\vec{w}) = \frac{1}{n}\sum_{i=1}^{n} l(\vec{w}, z_i) \qquad \text{where } z_i = (\vec{x}_i, y_i)$$

evaluated on the training data set $S = (z_1, z_2, \ldots, z_n)$. We are interested in the loss

$$L_{\mathcal{D}}(\vec{w}) = \mathbb{E}_{z\sim\mathcal{D}}[l(\vec{w}, z)] \tag{12}$$

on a yet unseen test dataset drawn from $\mathcal{D}$.

We call an estimator $\epsilon_n$**-stable** if

$$\mathbb{E}_{S\sim\mathcal{D}^n, z'\sim\mathcal{D}, i\sim U(n)}[l(\vec{w}(S^{(i)}(z')), z_i) - l(\vec{w}(S), z_i)] \leq \epsilon_n$$

where $S = (z_1, z_2, \ldots, z_n)$ and $S^{(i)}(z') = (z_1, z_2, \ldots, z_{i-1}, z', z_{i+1}, \ldots, z_n)$ and $U(n)$ is the uniform distribution on $\{1, \ldots, n\}$.

We will show that if an estimator is $\epsilon_n$**-stable**, then the test error is close to the training error in expectation, namely

$$\mathbb{E}_{S\sim\mathcal{D}^n}[L_{\mathcal{D}}(\vec{w}(S))] \leq \mathbb{E}_{S\sim\mathcal{D}^n}[L_S(\vec{w}(S))] + \epsilon_n \tag{13}$$

In the first part of the problem, we will establish this result and in the second part, we will apply it to show the generalization properties of ridge regression and soft margin SVMs.

(a) We first link stability to generalization via the fundamental property

$$\mathbb{E}_{S\sim\mathcal{D}^n}[L_{\mathcal{D}}(\vec{w}(S)) - L_S(\vec{w}(S))] = \mathbb{E}_{S\sim\mathcal{D}^n, i\sim U(n), z'\in\mathcal{D}}[l(\vec{w}(S^{(i)}(z'), z_i) - l(\vec{w}(S), z_i)] \tag{14}$$

where $S = (z_1, \ldots, z_n)$ be an iid sequence of samples and $z'$ be another iid sample and $U(n)$ be the uniform distribution over $\{1, 2, \ldots, n\}$. **Show equation** (14) **and conclude that** (13) **holds if** $A$ **is** $\epsilon$**-stable**.

Conceptual hint: When computing the expectation over both training points and a test point, switching any one training point and the test point gives the same result.

Technical hint: Use that $\frac{1}{n}\sum_{i=1}^{n} f(z_i) = \mathbb{E}_{i\sim U(n)} f(z_i)$.

$$\begin{aligned}
&\mathbb{E}_{S\sim\mathcal{D}^n}\left[L_{\mathcal{D}}(\vec{w}(S)) - L_S(\vec{w}(S))\right] \\
=&\mathbb{E}_{S\sim\mathcal{D}^n}\left[\mathbb{E}_{z\sim\mathcal{D}}[l(\vec{w}(S), z)] - \frac{1}{n}\sum_{i=1}^{n} l(\vec{w}(S), z_i)\right] \\
=&\mathbb{E}_{S\sim\mathcal{D}^n}\left[\mathbb{E}_{z\sim\mathcal{D}}[l(\vec{w}(S), z)] - \mathbb{E}_{i\sim U(n)} l(\vec{w}(S), z_i)\right] \\
=&\mathbb{E}_{S\sim\mathcal{D}^n}\left[\mathbb{E}_{z'\sim\mathcal{D}}[l(\vec{w}(S), z')] - \mathbb{E}_{i\sim U(n)} l(\vec{w}(S), z_i)\right]
\end{aligned}$$

Now we use the conceptual hint

$$=\mathbb{E}_{S\sim\mathcal{D}^n}\left[\mathbb{E}_{z'\sim\mathcal{D},i\sim U(n)}[l(\vec{w}(S^{(i)}(z')),z_i)]-\mathbb{E}_{i\sim U(n)}l(\vec{w}(S),z_i)\right]$$

We notice that the second term doesn't depend on $z'$

$$=\mathbb{E}_{S\sim\mathcal{D}^n}\left[\mathbb{E}_{z'\sim\mathcal{D},i\sim U(n)}[l(\vec{w}(S^{(i)}(z')),z_i)]-\mathbb{E}_{z'\sim\mathcal{D},i\sim U(n)}l(\vec{w}(S),z_i)\right]$$

$$=\mathbb{E}_{S\sim\mathcal{D}^n,i\sim U(n),z'\in\mathcal{D}}[l(\vec{w}(S^{(i)}(z'),z_i)-l(\vec{w}(S),z_i)]$$

---

Having proved equation (14), now we can use it to show (13) holds if $\vec{w}(S)$ is $\epsilon_n$-stable.

$$\begin{aligned}\mathbb{E}_{S\sim\mathcal{D}^n,z'\sim\mathcal{D},i\sim U(n)}[l(\vec{w}(S^{(i)}(z')),z_i)-l(\vec{w}(S),z_i)]&\leq\epsilon_n\\ \mathbb{E}_{S\sim\mathcal{D}^n}\left[L_{\mathcal{D}}(\vec{w}(S))-L_S(\vec{w}(S))\right]&\leq\epsilon_n\\ \mathbb{E}_{S\sim\mathcal{D}^n}L_{\mathcal{D}}(\vec{w}(S))-\mathbb{E}_{S\sim\mathcal{D}^n}L_S(\vec{w}(S))&\leq\epsilon_n\\ \mathbb{E}_{S\sim\mathcal{D}^n}L_{\mathcal{D}}(\vec{w}(S))&\leq\mathbb{E}_{S\sim\mathcal{D}^n}L_S(\vec{w}(S))+\epsilon_n\end{aligned}$$

(b) As a simple example, let's walk through the steps of the proof in the simple example where we estimate the mean

$$\mu(S) = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{15}$$

of a dataset $S = (x_1, x_2, \ldots, x_n)$ with scalars $x_1, \ldots, x_n \in \mathbb{R}$. The loss function in this case is

$$l(\mu, x_i) = \frac{1}{2}(\mu - x_i)^2$$

and the regularization parameter is $\lambda = 0$. **First, show that**

$$l(\mu(S^{(i)}(z')), x_i) - l(\mu(S), x_i) \leq \frac{C \cdot B \cdot \max_i |x_i|}{n}$$

where $|\mu(S)|, |\mu(S^{(i)}(z'))| \leq B$ and $C$ is some numerical constant. **Conclude that the generalization error is bounded by**

$$\mathbb{E}_{S\sim\mathcal{D}^n}[L_{\mathcal{D}}(\mu(S)) - L_S(\mu(S))] \leq \frac{C \cdot B \cdot \max_i |x_i|}{n}$$

$$\begin{aligned}
& l(\mu(S^{(i)}(x')), x_i) - l(\mu(S), x_i) \\
=& \frac{1}{2}(\mu(S^{(i)}(x')) - x_i)^2 - \frac{1}{2}(\mu(S) - x_i)^2 \\
=& \frac{1}{2}\mu(S^{(i)}(x'))^2 - \mu(S^{(i)}(x'))x_i + \frac{1}{2}x_i^2 - \frac{1}{2}\mu(S)^2 + \mu(S)x_i - \frac{1}{2}x_i^2 \\
=& \frac{1}{2}\mu(S^{(i)}(x'))^2 - \frac{1}{2}\mu(S)^2 + \mu(S)x_i - \mu(S^{(i)}(x'))x_i \\
=& \frac{1}{2}(\mu(S^{(i)}(x')) - \mu(S))(\mu(S^{(i)}(x')) + \mu(S)) - (\mu(S^{(i)}(x')) - \mu(S))x_i \\
=& (\mu(S^{(i)}(x')) - \mu(S))(\frac{1}{2}\mu(S^{(i)}(x')) + \frac{1}{2}\mu(S) - x_i) \\
\leq& |\mu(S^{(i)}(x')) - \mu(S)||\frac{1}{2}\mu(S^{(i)}(x')) + \frac{1}{2}\mu(S) - x_i| \\
=& |\frac{1}{n}(\sum_{j=1, j\neq i}^{n} x_j + x') - \frac{1}{n}\sum_{j=1}^{n} x_j| \cdot |\frac{1}{2}\mu(S^{(i)}(x')) + \frac{1}{2}\mu(S) - x_i| \\
=& |\frac{x' - x_i}{n}| \cdot |\frac{1}{2}\mu(S^{(i)}(x')) + \frac{1}{2}\mu(S) - x_i| \\
\leq& \frac{|x'| + |x_i|}{n} \cdot (|\frac{1}{2}\mu(S^{(i)}(x'))| + |\frac{1}{2}\mu(S)| + |x_i|) \\
& \text{We know that } \mu(S^{(i)}(x')), \mu(S) \leq \max(|x_i|, |x'|) \leq R \\
\leq& \frac{2R}{n} \cdot (\frac{1}{2}R + \frac{1}{2}R + R) \\
=& \frac{4R^2}{n} \\
=& \frac{CR^2}{n}
\end{aligned}$$

$$
\begin{aligned}
&\mathbb{E}_{S\sim\mathcal{D}^n}[L_{\mathcal{D}}(\mu(S)) - L_S(\mu(S))] \\
=&\mathbb{E}_{S\sim\mathcal{D}^n, i\sim U(n), x'\in\mathcal{D}}[l(\vec{w}(S^{(i)}(x'), z_i) - l(\vec{w}(S), z_i)] \\
\leq&\mathbb{E}_{S\sim\mathcal{D}^n, i\sim U(n), x'\in\mathcal{D}}\frac{CR^2}{n} \\
&\text{All values here do not vary for different } i \\
=&\frac{CR^2}{n}
\end{aligned}
$$

(c) We now consider **ridge regression**. The loss function is (11) with $l(\vec{w}, z) = \frac{1}{2}(\vec{w}^\top \vec{x}_i - y_i)^2$. **First, show that $l(\vec{w}, z)$ is $\beta$-smooth, i. e.**

$$\|\nabla l(\vec{v}, z) - \nabla l(\vec{w}, z)\| \leq \beta \|\vec{v} - \vec{w}\|$$

with $\beta = \max_i \|x_i\|$.

$$\begin{aligned}
&\nabla_{\vec{w}} l(\vec{w}, z) \\
=&\frac{1}{2} 2(\vec{w}^\top \vec{x}_i - y_i)\vec{x}_i \\
=&(\vec{w}^\top \vec{x}_i - y_i)\vec{x}_i \\
=&\vec{x}_i \vec{x}_i^\top \vec{w} - \vec{x}_i y_i
\end{aligned}$$

$$\begin{aligned}
&\|\nabla l(\vec{v}, z) - \nabla l(\vec{w}, z)\| \\
=&\|\vec{x}_i \vec{x}_i^\top \vec{v} - \vec{x}_i y_i - \vec{x}_i \vec{x}_i^\top \vec{w} + \vec{x}_i y_i\| \\
=&\|\vec{x}_i \vec{x}_i^\top (\vec{v} - \vec{w})\| \\
&\text{Here we use Rayleigh quotient inequality} \\
\leq&\lambda_{max}(\vec{x}_i \vec{x}_i^\top)\|\vec{v} - \vec{w}\| \\
&\text{Recall HW0 where we had } \lambda_{max}(\vec{x}_i \vec{x}_i^\top) = \vec{x}_i^\top \vec{x}_i \\
=&\vec{x}_i^\top \vec{x}_i \|\vec{v} - \vec{w}\| \\
=&\|\vec{x}_i\|^2 \|\vec{v} - \vec{w}\|
\end{aligned}$$

The above formula is true $\forall i$. Therefore, we only need to make the upper bound to be the maximum $\|\vec{x}_i\|$. That is:

$$\begin{aligned}
\|\nabla l(\vec{v}, z) - \nabla l(\vec{w}, z)\| &\leq \max_i \|x_i\|^2 \|\vec{v} - \vec{w}\| \\
\|\nabla l(\vec{v}, z) - \nabla l(\vec{w}, z)\| &\leq \beta \|\vec{v} - \vec{w}\|
\end{aligned}$$

(d) **Derive a generalization bound** (13) **for ridge regression.**
You may use *without proof* that for a $\beta$-smooth loss function, we have

$$\mathbb{E}[l(\vec{w}(S^{(i)}(z')), z_i) - l(\vec{w}(S), z_i)] \leq \frac{C\beta}{\lambda n} \tag{16}$$

for some constant $C$.

Because the loss function for ridge regression is a $\beta$-smooth loss function, we have:

$$\begin{aligned}
&\mathbb{E}_{S\sim\mathcal{D}^n}\left[L_{\mathcal{D}}(\vec{w}(S)) - L_S(\vec{w}(S))\right] \\
=&\mathbb{E}_{S\sim\mathcal{D}^n, z'\sim\mathcal{D}, i\sim U(n)}[l(\vec{w}(S^{(i)}(z')), z_i) - l(\vec{w}(S), z_i)] \\
=&\mathbb{E}[l(\vec{w}(S^{(i)}(z')), z_i) - l(\vec{w}(S), z_i)] \\
\leq&\frac{C\beta}{\lambda n} \\
=&\frac{C}{\lambda n}\max_i \|x_i\|^2
\end{aligned}$$

(e) Show that if $l$ is $\rho$-Lipschitz in the first argument, i.e.

$$l(\vec{w}(S^{(i)}(z')), z_i) - l(\vec{w}(S), z_i) \leq \rho \|\vec{w}(S^{(i)}(z')) - \vec{w}(S)\| \tag{17}$$

then the learning algorithm (11) is $\epsilon$-stable with

$$l(\vec{w}(S^{(i)}(z')), z_i) - l(\vec{w}(S), z_i) \leq \frac{2\rho^2}{\lambda n}.$$

Hint: First show

$$\lambda \|\vec{w}(S^{(i)}(z')) - \vec{w}(S)\|^2 \leq \frac{l(\vec{w}(S^{(i)}), z_i) - l(\vec{w}(S), z_i)}{n} + \frac{l(\vec{w}(S), z') - l(\vec{w}(S^{(i)}), z')}{n}. \tag{18}$$

First we define $f(\vec{w}) = L(\vec{w}) + \lambda \|\vec{w}\|^2$
We look at the difference between training errors using two different $\vec{w}$s:

$$\begin{aligned}
& f_S(\vec{w}(S^{(i)}(z'))) - f_S(\vec{w}(S)) \\
=& L_S(\vec{w}(S^{(i)}(z'))) + \lambda \|\vec{w}(S^{(i)}(z'))\|^2 - L_S(\vec{w}(S)) - \lambda \|\vec{w}(S)\|^2 \\
=& \frac{1}{n} \sum_{j=1}^{n} l(\vec{w}(S^{(i)}(z')), z_j) - \frac{1}{n} \sum_{j=1}^{n} l(\vec{w}(S), z_j) + \lambda \left( \|\vec{w}(S^{(i)}(z'))\|^2 - \|\vec{w}(S)\|^2 \right) \\
=& \frac{1}{n} \sum_{j=1, j\neq i}^{n} l(\vec{w}(S^{(i)}(z')), z_j) + \frac{1}{n} l(\vec{w}(S^{(i)}(z')), z_i) - \frac{1}{n} \sum_{j=1, j\neq i}^{n} l(\vec{w}(S), z_i) - \frac{1}{n} l(\vec{w}(S), z_i) \\
& + \lambda \left( \|\vec{w}(S^{(i)}(z'))\|^2 - \|\vec{w}(S)\|^2 \right) \\
=& \left( \frac{1}{n} \sum_{j=1, j\neq i}^{n} l(\vec{w}(S^{(i)}(z')), z_j) + \frac{1}{n} l(\vec{w}(S^{(i)}(z')), z') \right) + \left( \frac{1}{n} l(\vec{w}(S^{(i)}(z')), z_i) - \frac{1}{n} l(\vec{w}(S^{(i)}(z')), z') \right) \\
& - \left( \frac{1}{n} \sum_{j=1, j\neq i}^{n} l(\vec{w}(S), z_i) + \frac{1}{n} l(\vec{w}(S), z') \right) - \left( \frac{1}{n} l(\vec{w}(S), z_i) - \frac{1}{n} l(\vec{w}(S), z') \right) \\
& + \lambda \left( \|\vec{w}(S^{(i)}(z'))\|^2 - \|\vec{w}(S)\|^2 \right) \\
=& L_{S^{(i)}(z')}(\vec{w}(S^{(i)}(z'))) - L_{S^{(i)}(z')}(\vec{w}(S)) + \lambda \left( \|\vec{w}(S^{(i)}(z'))\|^2 - \|\vec{w}(S)\|^2 \right) \\
& + \frac{l(\vec{w}(S^{(i)}(z')), z_i) - l(\vec{w}(S), z_i)}{n} + \frac{l(\vec{w}(S), z') - l(\vec{w}(S^{(i)}(z')), z')}{n}
\end{aligned}$$

We know that the optimal point of data set $S^{(i)}(z')$ is achieved at $\vec{w}(S^{(i)}(z'))$ so we have:

$$L_{S^{(i)}(z')}(\vec{w}(S^{(i)}(z'))) + \lambda \|\vec{w}(S^{(i)}(z'))\|^2 \leq L_{S^{(i)}(z')}(\vec{w}(S)) + \lambda \|\vec{w}(S)\|^2$$

From the taylor expansion of the loss function $f(x)$, we have:

$$\begin{aligned}
& f_S(\vec{w}(S^{(i)}(z'))) - f_S(\vec{w}(S)) \\
\geq & < \nabla_{\vec{w}} f_S(\vec{w}(S)), w(S^{(i)}(z')) - \vec{w}(S) > + \nabla^2_{\vec{w}} f_S(\vec{w}(S)) \|x - y\|^2
\end{aligned}$$

In reality, the loss function is actually $2\lambda$-strong convexity this is because we have:

$$\lambda \|x\| - \lambda \|y\|$$

$$\geq < \lambda x - \lambda y, x - y > + \frac{2\lambda}{2} \|x - y\|$$

This means $\|x\|$ is $2\lambda$-strongly convexed. One property of $\lambda$-strong convexity is that we have a estimate of its second-order gradient:

$$\nabla^2_{\vec{w}} f_S(\vec{w}(S)) \succeq 2\lambda \mathbf{I}$$

Substitute this condition into the taylor expansion will give us:

$$f_S(\vec{w}(S^{(i)}(z'))) - f_S(\vec{w}(S)) \geq \lambda \|w(S^{(i)}(z')) - \vec{w}(S)\|^2$$

This is true becaue we know $\vec{w}(S)$ is the global minimum so that its first-order derivative is always positive. Having proved all these conditions, we can substitute them back and get the following conclusion in the hint:

$$\lambda \|\vec{w}(S^{(i)}(z')) - \vec{w}(S)\|^2 \leq f_S(\vec{w}(S^{(i)}(z'))) - f_S(\vec{w}(S)) \leq \frac{l(\vec{w}(S^{(i)}), z_i) - l(\vec{w}(S), z_i)}{n} + \frac{l(\vec{w}(S), z') - l(\vec{w}(S^{(i)}), z')}{n}$$

---

Now we use this conclusion and substitute it into the formula in the previous part.

$$\begin{aligned}
\lambda \|\vec{w}(S^{(i)}(z')) - \vec{w}(S)\|^2 &\leq \frac{l(\vec{w}(S^{(i)}(z')), z_i) - l(\vec{w}(S), z_i)}{n} + \frac{l(\vec{w}(S), z') - l(\vec{w}(S^{(i)}(z')), z')}{n} \\
\lambda \|\vec{w}(S^{(i)}(z')) - \vec{w}(S)\|^2 &\leq \frac{1}{n}\rho \|\vec{w}(S^{(i)}(z')) - \vec{w}(S)\| + \frac{1}{n}\rho \|\vec{w}(S^{(i)}(z')) - \vec{w}(S)\| \\
\lambda \|\vec{w}(S^{(i)}(z')) - \vec{w}(S)\|^2 &\leq \frac{2\rho}{n} \|\vec{w}(S^{(i)}(z')) - \vec{w}(S)\| \\
\lambda \|\vec{w}(S^{(i)}(z')) - \vec{w}(S)\| &\leq \frac{2\rho}{n} \\
\|\vec{w}(S^{(i)}(z')) - \vec{w}(S)\| &\leq \frac{2\rho}{\lambda n}
\end{aligned}$$

Therefore we have:

$$l(\vec{w}(S^{(i)}(z')), z_i) - l(\vec{w}(S), z_i) \leq \rho \|\vec{w}(S^{(i)}(z')) - \vec{w}(S)\| \leq \frac{2\rho^2}{\lambda n}$$

---

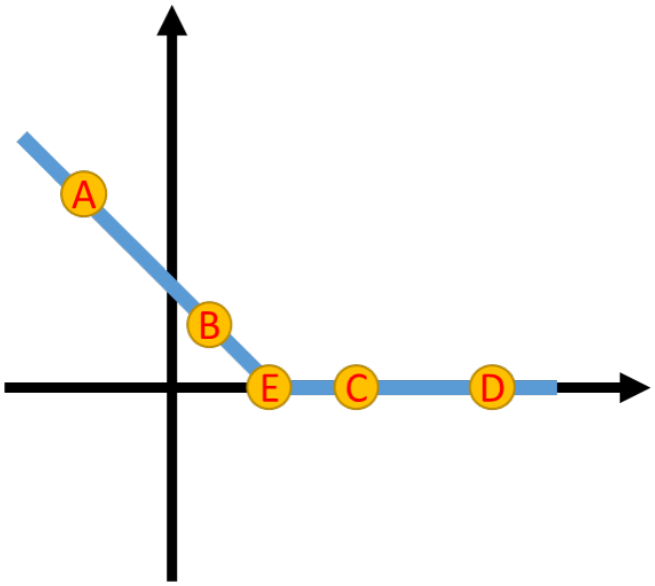
References:

1. Statistical learning theory
2. convex analysis

(f) We now consider the **soft margin SVMs**. The associated loss function is the hinge loss $l(\vec{w}, z_i) = \max(0, 1 - y_i \vec{w}^\top \vec{x}_i)$. **First show that $l(\vec{w}, z_i)$ is $\rho$-Lipschitz with $\rho = \max_i \|x_i\|$.**

$$\begin{aligned} & l(\vec{v}, z_i) - l(\vec{w}, z_i) \\ =& \max(0, 1 - y_i \vec{v}^\top \vec{x}_i) - \max(0, 1 - y_i \vec{w}^\top \vec{x}_i) \end{aligned}$$

Now we plot the hinge loss function and try to simplify this problem.




On the graph above, there are three possible combinations for $y_i \vec{v}^\top \vec{x}_i$ and $y_i \vec{w}^\top \vec{x}_i$. Without losing generality, we only consider the following possibilities assuming $y_i \vec{v}^\top \vec{x}_i \geq y_i \vec{w}^\top \vec{x}_i$.

1. C and D
   This is the simplest scenario:
   $$l(\vec{v}, z_i) - l(\vec{w}, z_i) = 0 \leq \|\vec{v} - \vec{w}\|$$
   Here $\rho$ can be any real number.
2. A and B
   This is not hard either:
   $$\begin{aligned} & l(\vec{v}, z_i) - l(\vec{w}, z_i) \\ =& 1 - y_i \vec{v}^\top \vec{x}_i - (1 - y_i \vec{w}^\top \vec{x}_i) \\ =& y_i \vec{x}_i^\top (\vec{w} - \vec{v}) \\ \leq& \|y_i \vec{x}_i\| \|\vec{v} - \vec{w}\| \\ & \text{Because } y_i \text{ is either 1 or } -1, \text{ we have} \\ \leq& \|\vec{x}_i\| \|\vec{v} - \vec{w}\| \end{aligned}$$

The above formula is true $\forall i$. Therefore, the upper bound is given by the maximum value of $\|\vec{x}_i\|$ and we have:

$$l(\vec{v}, z_i) - l(\vec{w}, z_i) \leq \max_i \|\vec{x}_i\| \cdot \|\vec{v} - \vec{w}\|$$
$$\text{where } \rho = \max_i \|\vec{x}_i\|$$

3. A and C
   This is a bit more complicated because we are dealing with a nonlinear function. Notice that the point E, which is the kink of the hinge loss function has a corresponding $\vec{w}^*$ that $1 - y_i(\vec{w}^*)^\top \vec{x}_i = 0$

$$\begin{aligned} & l(\vec{v}, z_i) - l(\vec{w}, z_i) \\ =& 1 - y_i \vec{v}^\top \vec{x}_i - 0 \\ =& 1 - y_i \vec{v}^\top \vec{x}_i - (1 - y_i(\vec{w}^*)^\top \vec{x}_i) \\ =& y_i \vec{x}_i^\top (\vec{w}^* - \vec{v}) \end{aligned}$$

   We also know the following is true:

$$\begin{aligned} 1 - y_i(\vec{w}^*)^\top \vec{x}_i &\geq 1 - y_i \vec{w}^\top \vec{x}_i \\ y_i \vec{x}_i^\top \vec{w}^* &\leq y_i \vec{x}_i^\top \vec{w} \\ y_i \vec{x}_i^\top (\vec{w} - \vec{w}^*) &\geq 0 \end{aligned}$$

   Therefore, we have

$$\begin{aligned} & l(\vec{v}, z_i) - l(\vec{w}, z_i) \\ =& y_i \vec{x}_i^\top (\vec{w}^* - \vec{v}) \\ \leq& y_i \vec{x}_i^\top (\vec{w}^* - \vec{v}) + y_i \vec{x}_i^\top (\vec{w} - \vec{w}^*) \\ =& y_i \vec{x}_i^\top (\vec{w} - \vec{v}) \\ \leq& \|\vec{x}_i\| \|\vec{v} - \vec{w}\| \end{aligned}$$

   For the same reason in scenario 2, we only need to set the upper bound to be the maximum value of $\|x_i\|$.

**As a summary, we have prove that the hinge loss function is $l(\vec{w}, z_i)$ is $\rho$-Lipschitz with $\rho = \max_i \|x_i\|$.**

(g) **Derive a generalization bound** (13) **for soft margin SVMs.**

Using the conclusion from part(e) and part(f), we have:

$$l(\vec{w}(S^{(i)}(z')), z_i) - l(\vec{w}(S), z_i) \leq \frac{2\rho^2}{\lambda n} \leq \frac{2}{\lambda n}(\max_i \|x_i\|^2).$$

$$\begin{aligned}
&\mathbb{E}_{S\sim\mathcal{D}^n}\left[L_{\mathcal{D}}(\vec{w}(S)) - L_S(\vec{w}(S))\right] \\
=&\mathbb{E}_{S\sim\mathcal{D}^n, z'\sim\mathcal{D}, i\sim U(n)}[l(\vec{w}(S^{(i)}(z')), z_i) - l(\vec{w}(S), z_i)] \\
=&\mathbb{E}[l(\vec{w}(S^{(i)}(z')), z_i) - l(\vec{w}(S), z_i)] \\
\leq&\frac{2}{\lambda n}(\max_i \|x_i\|^2)
\end{aligned}$$

**Question 5.** Your Own Question

**Write your own question, and provide a thorough solution.**

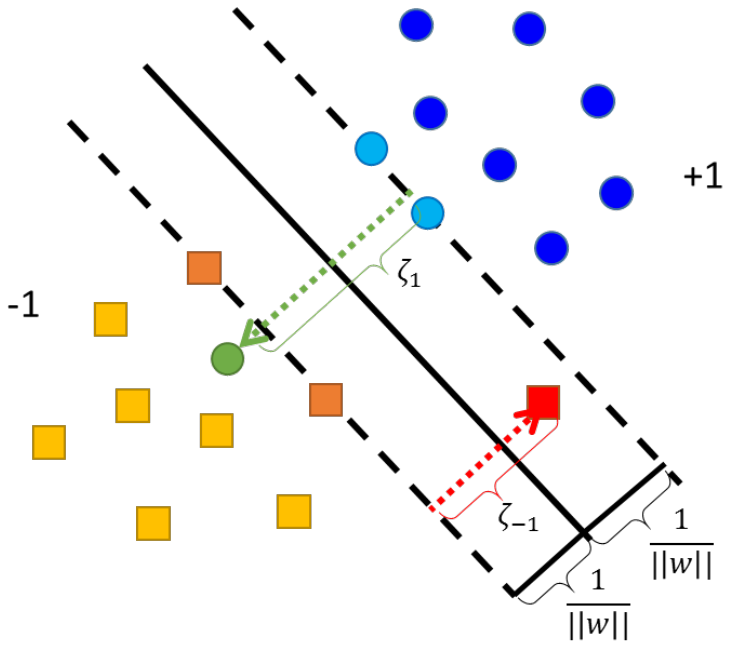
Writing your own problems is a very important way to really learn the material. The famous "Bloom's Taxonomy" that lists the levels of learning is: Remember, Understand, Apply, Analyze, Evaluate, and Create. Using what you know to create is the top-level. We rarely ask you any HW questions about the lowest level of straight-up remembering, expecting you to be able to do that yourself. (e.g. make yourself flashcards) But we don't want the same to be true about the highest level.

As a practical matter, having some practice at trying to create problems helps you study for exams much better than simply counting on solving existing practice problems. This is because thinking about how to create an interesting problem forces you to really look at the material from the perspective of those who are going to create the exams.

Besides, this is fun. If you want to make a boring problem, go ahead. That is your prerogative. But it is more fun to really engage with the material, discover something interesting, and then come up with a problem that walks others down a journey that lets them share your discovery. You don't have to achieve this every week. But unless you try every week, it probably won't happen ever.

Redraw a graph to have a better understanding of SVM and its dual problem. One thing I have to point out here is that:

1. If a point lies on the margin, it doesn't mean $\alpha_i \neq 0$ or $\alpha_i = 0$.
2. That is, some points on the margin are support vectors and some are not.




| +1 | α | β | $y_i f(x_i)$ | ζ | -1 |
|---|---|---|---|---|---|
| ● | 0 | C | >1 | 0 | ■ |
| ● | (0, C) | (0, C) | 1 | 0 | ■ |
| ● | C | 0 | <1 | >0 | ■ |