hw06

July 25, 2022

[1]:
```
# Initialize Otter
import otter
grader = otter.Notebook("hw06.ipynb")
```

1 Homework 6: Modeling and Analyzing COVID-19 Cases

1.1 Probability and Estimators

1.2 Due Date: Thursday, July 28, 11:59 PM PDT

Content Warning

This assignment includes an analysis of daily COVID-19 cases by U.S. county through 2021. If you feel uncomfortable with this topic, **please contact your GSI or the instructors.**

Collaboration Policy

Data science is a collaborative activity. While you may talk with others about the homework, we ask that you **write your solutions individually**. If you do discuss the assignments with others please **include their names** below.

Collaborators: *list collaborators here*

1.3 Introduction

In this homework, we will investigate a dataset that contains information about COVID-19 cases in the United States, vaccination rates, and various other metadata that can assist in modeling various aspects of COVID-19.

Through this homework assignment, you will demonstrate your experience with: * Bootstrap sampling * Bias-variance tradeoff and decomposition * Biased and unbiased estimators * Multicollinearity in features

[2]:
```
# Run this cell to set up your notebook
import numpy as np
import pandas as pd
import sklearn.linear_model as lm
from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt
import seaborn as sns
```

```
from IPython.display import Markdown

import scipy.stats

import warnings
warnings.filterwarnings("ignore")
```

1.4 Question 1: Random Variables

Question 1 is a written problem and should be submitted as a separate PDF to the Written portal of Gradescope. All other questions in this assignment are submitted as part of this notebook.

Question 1 PDF File: https://ds100.org/su22/hw/hw06/hw06_student.pdf Question 1 Overleaf Template: https://ds100.org/su22/hw/hw06/hw06_template.zip

1.5 Question 2: Exploratory Data Analysis

Let's perform some initial exploratory data analysis to examine and visualize potential trends in a COVID-19 dataset.

[3]:
```
# just run this cell
covid_data = pd.read_csv('data/covid_data.csv')
covid_data.head(5)
```

[3]:

	UID	iso2	iso3	code3	FIPS	Admin2	Province_State	Country_Region	\
0	84001001	US	USA	840	1001	Autauga	Alabama	US	
1	84001003	US	USA	840	1003	Baldwin	Alabama	US	
2	84001005	US	USA	840	1005	Barbour	Alabama	US	
3	84001007	US	USA	840	1007	Bibb	Alabama	US	
4	84001009	US	USA	840	1009	Blount	Alabama	US	

	Lat	Long_	…	POPESTIMATE2018	POPESTIMATE2019	\
0	32.539527	-86.644082	…	55533	55769	
1	30.727750	-87.722071	…	218071	223565	
2	31.868263	-85.387129	…	24887	24657	
3	32.996421	-87.125115	…	22300	22313	
4	33.982109	-86.567906	…	57770	57840	

	POPESTIMATE042020	POPESTIMATE2020	COUNTYFP	NEVER	RARELY	SOMETIMES	\
0	56130	56145	1001	0.053	0.074	0.134	
1	227989	229287	1003	0.083	0.059	0.098	
2	24652	24589	1005	0.067	0.121	0.120	
3	22199	22136	1007	0.020	0.034	0.096	
4	57932	57879	1009	0.053	0.114	0.180	

	FREQUENTLY	ALWAYS
0	0.295	0.444
1	0.323	0.436

```
2       0.201   0.491
3       0.278   0.572
4       0.194   0.459

[5 rows x 638 columns]
```

The data are at county granularity; each row corresponds to COVID-19 data from a U.S. county. Here are some highlights and data sources:

- The first few columns encode county and state data; for example, check out the FIPS numeric encoding for U.S. counties.
- The next 600 columns record daily COVID-19 cases in the county for the date range 1/22/2020 to 9/12/2021. COVID-19 case data are from CSSE at Johns Hopkins University GitHub.
- The next few columns include county populations from U.S. census data, the latest of which is 2020.
- The last 5 columns record mask usage survey data on a 5-point scale from `NEVER` to `ALWAYS`. Data are collected in July 2020 from the New York Times GitHub.

We can use `covid_data.describe()` to see various statistics about the numerical features of the provided COVID-19 data. Do any particular statistics stand out to you? Which might be useful when modeling?

Note: This isn't a question (i.e. it's worth no points); this is just food for thought as you start to explore the dataset.

[4]:
```
# just run this cell
covid_data.describe()
```

[4]:
```
                UID   code3          FIPS          Lat        Long_  \
count  3.141000e+03  3141.0   3141.000000  3141.000000  3141.000000
mean   8.403039e+07   840.0  30392.602674    38.448156   -92.272006
std    1.515661e+04     0.0  15156.613190     5.292540    12.909318
min    8.400100e+07   840.0   1001.000000    19.601212  -174.159600
25%    8.401818e+07   840.0  18179.000000    34.693167   -98.218207
50%    8.402918e+07   840.0  29177.000000    38.373019   -90.396561
75%    8.404508e+07   840.0  45081.000000    41.802830   -83.436796
max    8.405604e+07   840.0  56045.000000    69.314792   -67.628135

            1/22/20      1/23/20      1/24/20      1/25/20      1/26/20  …  \
count  3141.000000  3141.000000  3141.000000  3141.000000  3141.000000  …
mean      0.000318     0.000318     0.000637     0.000637     0.001592  …
std       0.017843     0.017843     0.025230     0.025230     0.039873  …
min       0.000000     0.000000     0.000000     0.000000     0.000000  …
25%       0.000000     0.000000     0.000000     0.000000     0.000000  …
50%       0.000000     0.000000     0.000000     0.000000     0.000000  …
75%       0.000000     0.000000     0.000000     0.000000     0.000000  …
max       1.000000     1.000000     1.000000     1.000000     1.000000  …

       POPESTIMATE2018  POPESTIMATE2019  POPESTIMATE042020  POPESTIMATE2020  \
```

```
count     3.141000e+03     3.141000e+03       3.141000e+03     3.141000e+03
mean      1.040525e+05     1.045274e+05       1.048677e+05     1.048949e+05
std       3.326200e+05     3.332156e+05       3.335848e+05     3.333719e+05
min       8.700000e+01     8.700000e+01       9.000000e+01     8.700000e+01
25%       1.096300e+04     1.093600e+04       1.091900e+04     1.092100e+04
50%       2.583700e+04     2.573500e+04       2.570100e+04     2.565800e+04
75%       6.813600e+04     6.828200e+04       6.822000e+04     6.824100e+04
max       1.006153e+07     1.001160e+07       9.968969e+06     9.943046e+06

           COUNTYFP        NEVER       RARELY    SOMETIMES   FREQUENTLY  \
count   3141.000000  3141.000000  3141.000000  3141.000000  3141.000000
mean   30392.602674     0.079952     0.082929     0.121340     0.207728
std    15156.613190     0.058543     0.055469     0.058007     0.063581
min     1001.000000     0.000000     0.000000     0.001000     0.029000
25%    18179.000000     0.034000     0.040000     0.079000     0.164000
50%    29177.000000     0.068000     0.073000     0.115000     0.204000
75%    45081.000000     0.113000     0.115000     0.156000     0.247000
max    56045.000000     0.432000     0.384000     0.422000     0.549000

            ALWAYS
count  3141.000000
mean      0.508044
std       0.152190
min       0.115000
25%       0.393000
50%       0.497000
75%       0.613000
max       0.889000

[8 rows x 630 columns]
```

1.5.1 Question 2a

In this homework, we will use linear regression to predict the number of COVID-19 cases on September 12th, 2021 using linear regression. **per capita** (i.e. the number of COVID-19 cases in a county divided the population of the county). Define a column `'9/12/2021_cpc'` in `covid_data` corresponding to the number of cases per capita on September 12th, 2021.

Note that we will **always** use the `'POPESTIMATE2020'` as the population of each county.

Hint: The number of cases per capita should be the total number of cases in a county divided by the population of the county.

[5]:
```
covid_data['9/12/2021_cpc'] = covid_data['9/12/21'] /␣
↪covid_data['POPESTIMATE2020'] # SOLUTION
```

[6]:
```
grader.check("q2a")
```

[6]:
```
q2a results: All test cases passed!
```

1.5.2 Question 2b

Assign `mask_data` that has six columns from the original `covid_data` table: the five mask columns and the `9/12/2021_cpc` column.

Note: You should make a **copy** of these columns using `df.copy()` (link).

[7]:
```
mask_data = covid_data[['NEVER', 'RARELY', 'SOMETIMES', 'FREQUENTLY', 'ALWAYS',␣
↪'9/12/2021_cpc']].copy() # SOLUTION
mask_data
```

[7]:

	NEVER	RARELY	SOMETIMES	FREQUENTLY	ALWAYS	9/12/2021_cpc
0	0.053	0.074	0.134	0.295	0.444	0.165411
1	0.083	0.059	0.098	0.323	0.436	0.152429
2	0.067	0.121	0.120	0.201	0.491	0.134003
3	0.020	0.034	0.096	0.278	0.572	0.171440
4	0.053	0.114	0.180	0.194	0.459	0.158538
…	…	…	…	…	…	…
3136	0.061	0.295	0.230	0.146	0.268	0.143205
3137	0.095	0.157	0.160	0.247	0.340	0.196238
3138	0.098	0.278	0.154	0.207	0.264	0.158496
3139	0.204	0.155	0.069	0.285	0.287	0.144330
3140	0.142	0.129	0.148	0.207	0.374	0.122942

[3141 rows x 6 columns]

[8]:
```
grader.check("q2b")
```

[8]:
```
q2b results: All test cases passed!
```

1.5.3 Question 2c

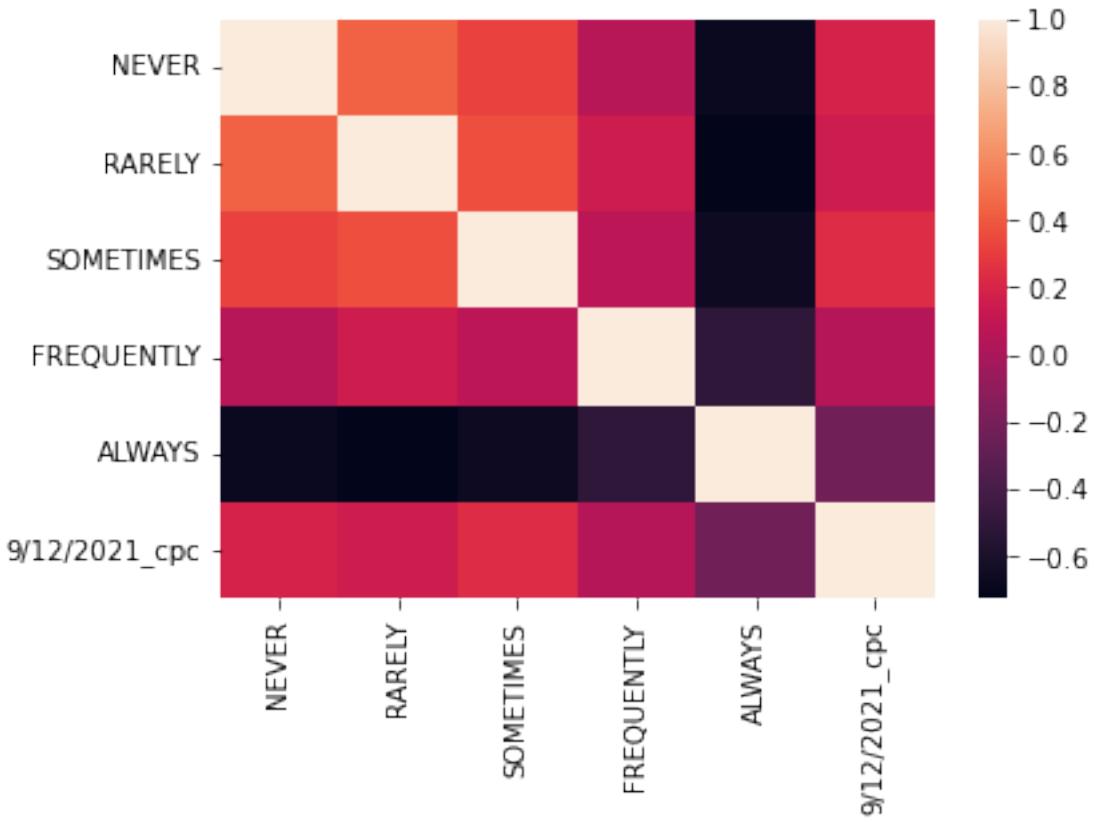
In our first model, we will use county-wise mask usage data to predict the number of COVID-19 cases on September 12th, 2021 (i.e., the column `9/12/2021_cpc`). Create a visualization that shows the pairwise correlation between each combination of columns in `mask_data`. For 2-D visualizations, consider Seaborn's heatmap.

Hint: You should be plotting 36 values corresponding to the pairwise correlations of the six columns in `mask_data`.

[9]:
```
# BEGIN SOLUTION

sns.heatmap(mask_data[['NEVER', 'RARELY', 'SOMETIMES', 'FREQUENTLY', 'ALWAYS',␣
↪'9/12/2021_cpc']].corr())
# END SOLUTION
```

[9]: <AxesSubplot:>



1.5.4 Question 2d

(1) Describe the trends and takeaways visible in the visualization of pairwise correlations you plotted in Question 2c.

(2) Consider the following linear regression model

$$\hat{y} = \theta^T x,$$

where $\hat{y}$ is the predicted number of COVID-19 cases per capita on 9/12/2021 and x is the five mask usage features. Comment on the quality of predictions and interpretability of features if we fit this linear model to the data.

Type your answer here, replacing this text.

SOLUTION:

The correlations are weak to moderate and mostly positive (except for `ALWAYS` which is weakly and negatively correlated) with the response variable, which would imply that our regression model will likely not be very good. Moreover, there is a lot more correlation between features than there is between the features and response variable.

1.6 Question 3: Creating a Preliminary COVID-19 Model

This question will guide you through creating a supervised learning framework that will predict the number of COVID-19 cases per capita given various COVID-19 safety protocols that have been implemented. Then, we will investigate the bias, variance, and observational noise of this framework in the next question.

Note that any answer responses without the appropriate work (i.e. code or explanation) will be subject to additional review and will not receive any credit.

1.6.1 Question 3a

Train a linear regression model using Scikit-learn, with an intercept term to predict the number of COVID-19 cases per capita for September 12, 2021 using county-wise mask usage data from `mask_data`. Use `train_test_split` to evaluate your model's RMSE on a held-out test set with 33% of the COVID-19 data; call the resulting splits `X_train`, `X_test`, `y_train`, and `y_test`.

To pass the autograder, make sure to set the parameter `random_state` to 42 in your call to `train_test_split` to generate a reproducible data split (documentation).

[10]:
```
# Create train/test sets
X = ...
y = ...
X_train, X_test, y_train, y_test = ..., ..., ..., ...

# fit the linear model and make predictions
...

# compute RMSE on train and test sets
train_rmse_cpc = ...
test_rmse_cpc = ...

# BEGIN SOLUTION
X = mask_data[['NEVER', 'RARELY', 'SOMETIMES', 'FREQUENTLY', 'ALWAYS']]
y = mask_data['9/12/2021_cpc']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,␣
↪random_state=42)

model = lm.LinearRegression()
model.fit(X_train, y_train)

train_rmse_cpc = np.sqrt(np.mean((model.predict(X_train) - y_train) ** 2))
test_rmse_cpc = np.sqrt(np.mean((model.predict(X_test) - y_test) ** 2))
# END SOLUTION

train_rmse_cpc, test_rmse_cpc
```

[10]:
```
(0.03552339478226883, 0.037954861477287)
```

[11]:
```
grader.check("q3a")
```

[11]:
```
q3a results: All test cases passed!
```

1.6.2 Question 3b

Visualize the model performance from part (a) by plotting two visualizations: (1) the predictions vs observations on the test set and (2) the residuals for the test set.

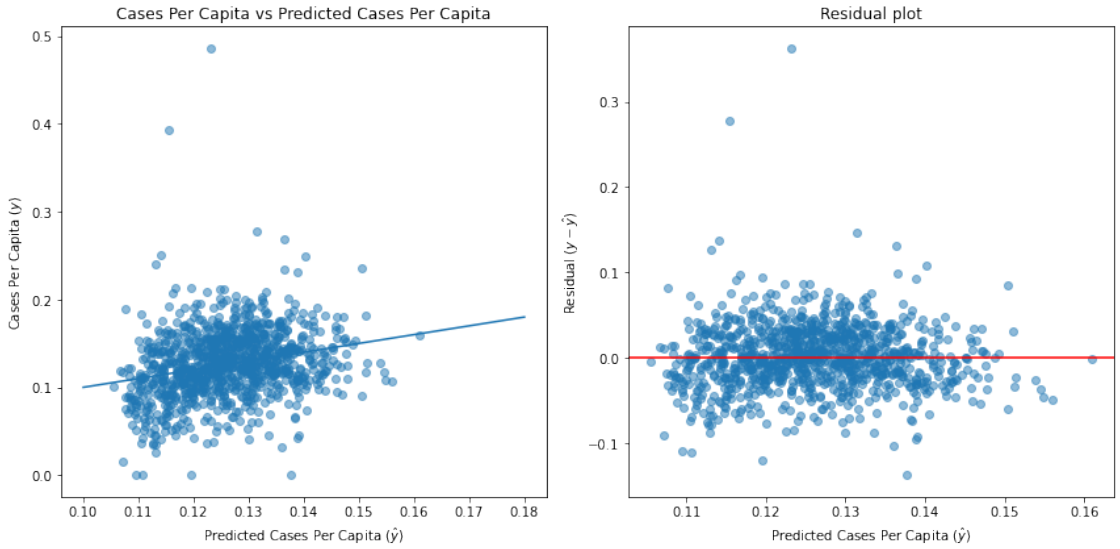
Some notes: * We've used `plt.subplot` (documentation) so that you can view both visualizations side-by-side. For example, `plt.subplot(121)` sets the plottable area to the first column of a 1x2 plot grid; you can then call Matplotlib and Seaborn functions to plot that area, before the next `plt.subplot(122)` area is set. * Remember to add a guiding line to both plot where $\hat{y} = y$, i.e., where the residual is 0. * Remember to label your axes.

[12]:
```
plt.figure(figsize=(12,6))      # do not change this line

plt.subplot(121)                # do not change this line
# (1) predictions vs observations
# BEGIN SOLUTION
y_pred = model.predict(X_test)
plt.scatter(y_pred, y_test, alpha=0.5)
plt.plot(np.linspace(0.1, 0.18, 10), np.linspace(0.1, 0.18, 10))
plt.ylabel("Cases Per Capita $(y)$")
plt.xlabel("Predicted Cases Per Capita $(\hat{y})$")
plt.title("Cases Per Capita vs Predicted Cases Per Capita");
# END SOLUTION

plt.subplot(122)               # do not change this line
# (2) residual plot
# BEGIN SOLUTION
plt.scatter(y_pred, y_test - y_pred, alpha=0.5)
plt.ylabel("Residual $(y - \hat{y})$")
plt.xlabel("Predicted Cases Per Capita $(\hat{y})$")
plt.title("Residual plot")
plt.axhline(y = 0, color='r');
# END SOLUTION

plt.tight_layout()             # do not change this line
```



1.6.3 Question 3c

Describe what the plots in part (b) indicates about this linear model. Justify your answer.

Type your answer here, replacing this text.

SOLUTION:

This indicates that our linear model is doing a poor job because the predictions are not at all on the $y = x$ line that would correspond with perfect predictions.

1.7 Question 4: Performing Multicollinearity Analysis

This question will guide you through performing an analysis that can reveal potential multicollinearity in our features, which is unideal.

Note that any answer responses without the appropriate work (i.e. code or math) will be subject to additional review and will not receive any credit.

1.7.1 Question 4a

Fill in the blanks below to implement the `bootstrap_sample` function, that returns k randomly drawn samples from a dataset $\mathcal{D}$ of size n with replacement, each of size n (i.e. same size as the dataset). In other words, the returned object should be a Python list `samples` containing k Pandas DataFrames, each of which have n rows.

Hint: Take a look at the documentation for `df.sample`!

[13]:
```
def bootstrap_sample(data, k):
    """
    Performs bootstrap sampling on data to obtain k samples of size n.
```

```
    Arguments:
        data - Dataset contained as a Pandas DataFrame
        k - Number of randomly drawn samples

    Returns:
        samples - List containing k Pandas DataFrames of size n each
                  corresponding to each sample
    """
    samples = []
    return ...

# BEGIN SOLUTION
def bootstrap_sample(data, k):
    """
    Performs bootstrap sampling on data to obtain k samples of size n.

    Arguments:
        data - Dataset contained as a Pandas DataFrame
        k - Number of randomly drawn samples

    Returns:
        samples - List containing k Pandas DataFrames of size n each
                  corresponding to each sample
    """
    return [data.sample(len(data), replace = True) for _ in range(k)]
# END SOLUTION
bootstrap_sample(mask_data, 1)[0]
```

[13]:

	NEVER	RARELY	SOMETIMES	FREQUENTLY	ALWAYS	9/12/2021_cpc
1919	0.068	0.044	0.143	0.126	0.620	0.146495
469	0.049	0.087	0.226	0.209	0.430	0.106090
1987	0.073	0.027	0.080	0.199	0.621	0.137887
2555	0.034	0.145	0.170	0.209	0.441	0.125105
1628	0.091	0.200	0.128	0.209	0.372	0.095166
…	…	…	…	…	…	…
87	0.042	0.095	0.148	0.349	0.365	0.066444
2256	0.017	0.013	0.076	0.101	0.792	0.113873
1220	0.023	0.010	0.047	0.131	0.788	0.132315
1904	0.010	0.067	0.167	0.197	0.560	0.099198
540	0.185	0.068	0.203	0.142	0.403	0.199457

[3141 rows x 6 columns]

[14]:
```
grader.check("q4a")
```

[14]:
```
q4a results: All test cases passed!
```

1.7.2 Question 4b

Using the function from the previous part, generate 1000 bootstrapped samples from the original `mask_data` dataframe. Use Scikit-learn to fit a linear regression model of mask features (with an intercept term) to predict the `9/12/2021_cpc` response. You should fit your model to **each** of the 1000 datasets such that we have 1000 trained models. Make sure to store each of the 1000 trained models in the Python list `models`.

Note: You *should not* create any validation or testing sets in this subpart; you should fit your model to the entire resampled dataframe.

[15]:
```
np.random.seed(42)

datasets = ...
models = []
...

# BEGIN SOLUTION
datasets = bootstrap_sample(mask_data, 1000)
for df in datasets:
    X_b = df[['NEVER', 'RARELY', 'SOMETIMES', 'FREQUENTLY', 'ALWAYS']]
    y_b = df['9/12/2021_cpc']
    models.append(lm.LinearRegression(fit_intercept = True))
    models[-1].fit(X_b, y_b)
# END SOLUTION

# These take up a lot of memory, so we should remove them!
del datasets
```

[16]:
```
grader.check("q4b")
```

[16]:
```
q4b results: All test cases passed!
```

1.7.3 Question 4c

Fill in the blanks below in the `confidence_interval` function to generate a 95% confidence interval for each of our parameters θ_i, including an intercept term if applicable. All of the helper code to extract coefficients from our trained models has been implemented for you already.

[17]:
```
def extract_coefs(models, include_intercept = True):
    """
    NOTE: This function has already been implemented. You do not need to modify␣
 ↪this!

    Extracts coefficients of all the linear regression models in models, and␣
 ↪returns
    it as a NumPy array with one model's coefficients as each row.

    Arguments:
```

```
        models - Contains k sklearn LinearRegression models, each with p + 1␣
↪coefficients
        include_intercept - Whether to include intercept in returned␣
↪coefficients

    Returns:
        coef_array - Coefficients of all k models, each with p + 1 coefficients␣
↪(if intercept
                     enabled, otherwise p). Returned object is k x (p + 1)␣
↪NumPy array.
    """
    coef_array = np.zeros(shape = (len(models), len(models[0].coef_) + 1))
    for i, m in enumerate(models):
        coef_array[i, 0] = m.intercept_
        coef_array[i, 1:] = m.coef_
    if include_intercept:
        return coef_array
    return coef_array[:, 1:]

def confidence_interval(coefs):
    """
    Calculates confidence intervals for each theta_i based on coefficients of
    bootstrapped models. Returns output as a list of confidence intervals.

    Arguments:
        coefs - Output of extract_coefs, a k x (p + 1) or k x p NumPy array␣
↪containing
                coefficients of bootstrapped models

    Returns:
        cis - Confidence intervals of each parameter theta_i in the form of a
              list like this: [(0.5, 0.75), (0.2, 0.4), ...]
    """
    cis = []

    # FILL IN CODE BELOW
    for i in range(...):
        theta_i_values = ...
        theta_i_lower_ci, theta_i_upper_ci = np.percentile(...), np.percentile(.
↪..)
        cis.append((theta_i_lower_ci, theta_i_upper_ci))

    return cis

# BEGIN SOLUTION
def confidence_interval(coefs):
```

```
    """
    Calculates confidence intervals for each theta_i based on coefficients of
    bootstrapped models. Returns output as a list of confidence intervals.

    Arguments:
        coefs - Output of extract_coefs, a k x (p + 1) or k x p NumPy array␣
 ↪containing
                coefficients of bootstrapped models

    Returns:
        cis - Confidence intervals of each parameter theta_i in the form of a
              list like this: [(0.5, 0.75), (0.2, 0.4), ...]
    """
    cis = []
    for i in range(coefs.shape[1]):
        theta_i_values = coefs[:, i]
        theta_i_lower_ci, theta_i_upper_ci = np.percentile(theta_i_values, 2.
 ↪5), np.percentile(theta_i_values, 97.5)
        cis.append((theta_i_lower_ci, theta_i_upper_ci))
    return cis
# END SOLUTION

# compute confidence intervals
model_coefs = extract_coefs(models)
cis = confidence_interval(model_coefs)

# pretty print in a table
display(Markdown('#### Confidence Intervals:'))
md_list = ["|parameter|lower|upper|",
           "----|----|----|"]
md_list += [fr"|$\theta_{i}$|{lci}|{uci}|" for i, (lci, uci) in enumerate(cis)]
display(Markdown('\n'.join(md_list)))
```

parameter	lower	upper
θ_0	-2.6200726729046937	1.2507871842262086
θ_1	-1.0896828874135185	2.798332585238362
θ_2	-1.1461331540818134	2.744519025435908
θ_3	-1.018294369222972	2.8436420433762395
θ_4	-1.1582158941591825	2.7329392573407363
θ_5	-1.1477854499077667	2.7195109052029274

Confidence Intervals:

[18]:
```
grader.check("q4c")
```

[18]:
```
q4c results: All test cases passed!
```

1.7.4 Question 4d

Interpret the confidence intervals above for each of the θ_i, where θ_0 is the intercept term and the remaining θ_i for $i > 0$ are parameters corresponding to mask usage features. What does this indicate about our data and our model?

Describe a mathematical reason why this could be happening.

Hint: Take a look at the design matrix!

Type your answer here, replacing this text.

SOLUTION:

From Question 2, we know that there is collinearity between the terms. In fact, there is linear dependence; the sum of all mask usage features subtracted from the bias vector yields $\vec{0}$. Therefore while Scikit-Learn can fit models to bootstrap 95% confidence intervals (many of which contain zero), the parameters are neither statistically significant nor interpretable, since at any point one of them is redundant.

1.8 Question 5: Performing Bias-Variance Analysis

This question will guide you through performing an analysis that can estimate the bias and variance of our models, which can be helpful in modeling.

Note that any answer responses without the appropriate work (i.e. code or explanation) will be subject to additional review and will not receive any credit.

1.8.1 Question 5a

We will use the same bootstrapped models contained in the Python list `models` to estimate our **model variance**. To do this, recall that the model variance on a data point is simply the variance of our predictions on that sample point. From the bias-variance decomposition in lecture, for a parametric model $\hat{Y}(x) = f_{\hat{\theta}}(x)$:

$$\text{model variance} = \text{Var}(f_{\hat{\theta}}(x))$$

To investigate the variance in our test predictions, we sample a particular data point (x_i, y_i). Define the **model risk** for this point as the mean square error over all possible fitted models:

$$\mathbb{E}\left[\left(y_i - f_{\hat{\theta}}(x_i)\right)^2\right]$$

Note that in contrast to lecture, you are considering a particular observation of the random response variable $Y = y_i$. Therefore model risk is an expectation over the estimate $\widehat{\theta}$, which is a function of the random sample you used to fit your model.

Using the bootstrapped estimates `models`, approximate the ratio of model variance to model risk for the datapoint $i = 100$, i.e., (x_{100}, y_{100}). You can interpret this ratio as the the proportion of the expected square error on the data point "captured" by the model variance. Since you bootstrapped 1000 models, you can generate 1000 predictions for the given x_i. Recall that `X` is the design matrix of mask features, and `y` is the `9/12/2021_cpc` response.

Assign `prop_var` to the computed, approximate ratio:

$$\frac{\text{Var}\left(f_{\hat{\theta}}(x_{100})\right)}{\mathbb{E}\left[\left(y_{100} - f_{\hat{\theta}}(x_{100})\right)^2\right]}$$

[19]:
```
prop_var = ...

# BEGIN SOLUTION
X_test_pt, y_test_pt = X.iloc[100], y.iloc[100]
preds = [model.predict([X_test_pt])[0] for model in models]
preds = pd.DataFrame({'preds': preds})
preds['error'] = (preds['preds'] - y_test_pt) ** 2
prop_var = preds['preds'].var() / preds['error'].mean()

# END SOLUTION
prop_var
```

[19]:
```
0.001318235885796681
```

[20]:
```
grader.check("q5a")
```

[20]:
```
q5a results: All test cases passed!
```

1.8.2 Question 5b

Comment on the ratio `prop_var`, which is the proportion of the expected square error on the data point captured by the model variance. Is the model variance the dominant term in the bias-variance decomposition? If not, what term(s) dominate the bias-variance decomposition?

Justify your answer.

Type your answer here, replacing this text.

SOLUTION:

The model variance only accounts for <0.5% of the total expected error, which is essentially nothing. This indicates that almost all the error is likely linked to the model bias and observational variance.

1.8.3 Question 5c

Using the bias-variance decomposition above, calculate the average variance and average mean square error across 250 randomly sampled (x_i, y_i) points. In other words, estimate the following quantities across all x_i and y_i in `X_sample` and `y_sample`:

$$\frac{1}{250}\sum_{i=1}^{250}\text{Var}\left(f_{\hat{\theta}}(x_i)\right)$$

and

$$\frac{1}{250}\sum_{i=1}^{250}\mathbb{E}\left[\left(y_i - f_{\hat{\theta}}(x_i)\right)^2\right]$$

[21]:
```
np.random.seed(42)

X_sample = X.sample(250)          # generate 250 x_i
y_sample = y.loc[X_sample.index] # ...and select the corresponding y_i

avg_var, avg_mse = ..., ...
# BEGIN SOLUTION
avg_var, avg_mse = 0, 0
for i in range(len(X_sample)):
    X_pt, y_pt = X_sample.iloc[i], y_sample.iloc[i]
    preds = [model.predict([X_pt])[0] for model in models]
    preds = pd.DataFrame({'preds': preds})
    preds['error'] = (preds['preds'] - y_pt) ** 2
    avg_var += preds['preds'].var()
    avg_mse += preds['error'].mean()
avg_var /= len(X_sample)
avg_mse /= len(X_sample)
# END SOLUTION
avg_var, avg_mse
```

[21]:
```
(2.5371356178727863e-06, 0.0016097766310807606)
```

[22]:
```
grader.check("q5c")
```

[22]:
```
q5c results: All test cases passed!
```

1.8.4 Question 5d

Propose a solution to reducing the mean square error using the insights gained from the bias-variance decomposition above. Please show all quantities and work that informs your analysis.

Assume that the standard bias-variance decomposition used in lecture can be applied here.

Type your answer here, replacing this text.

SOLUTION:

We can reduce the MSE by reducing the bias since there is clearly not enough complexity in our model. This is based on how little the variance contributes to the expected error for both a singular point and across a sample of the dataset (students should provide numbers here, but since they are variable across runs of the notebook, we do not provide numbers here). Since the dominant term is the noise and bias squared terms, and the noise is uncontrollable, we should try to reduce the bias by increasing the complexity of the model.

1.9 Question 6: Improving our Model

1.9.1 Question 6a

Suppose we decide to add a feature to our model corresponding to the number of cases per capita the week before (i.e. September 5, 2021). Calculate the cases per capita on September 5, 2021 from the original `covid_data` table, and store it in `mask_data` as a column named `'9/5/2021_cpc'`.

Hint: This should be similar to Question 2a!

[23]:
```
mask_data['9/5/2021_cpc'] = covid_data['9/5/21'] /␣
↪covid_data['POPESTIMATE2020'] # SOLUTION
mask_data
```

[23]:
```
      NEVER  RARELY  SOMETIMES  FREQUENTLY  ALWAYS  9/12/2021_cpc  \
0     0.053   0.074      0.134       0.295   0.444       0.165411
1     0.083   0.059      0.098       0.323   0.436       0.152429
2     0.067   0.121      0.120       0.201   0.491       0.134003
3     0.020   0.034      0.096       0.278   0.572       0.171440
4     0.053   0.114      0.180       0.194   0.459       0.158538
…     …       …          …           …       …           …
3136  0.061   0.295      0.230       0.146   0.268       0.143205
3137  0.095   0.157      0.160       0.247   0.340       0.196238
3138  0.098   0.278      0.154       0.207   0.264       0.158496
3139  0.204   0.155      0.069       0.285   0.287       0.144330
3140  0.142   0.129      0.148       0.207   0.374       0.122942

      9/5/2021_cpc
0         0.160513
1         0.148561
2         0.129489
3         0.163896
4         0.153113
…              …
3136      0.135706
3137      0.191599
3138      0.151026
3139      0.139175
3140      0.118938

[3141 rows x 7 columns]
```

[24]:
```
grader.check("q6a")
```

[24]:
```
q6a results: All test cases passed!
```

1.9.2 Question 6b

Add the feature that we generated in the previous subpart into our design matrix, and train a Scikit-Learn linear regression model **without an intercept term**. Use `train_test_split` to evaluate your model's RMSE on a held-out validation set with 33% of the county-wise data `mask_data`.

To pass the autograder, make sure to set the parameter `random_state` to 42 in your call to `train_test_split`.

Hint: This should be similar to Question 3a!

[25]:
```
# Create train/test sets
X_improved = ...
y_improved = ...
X_improved_train, X_improved_test, y_improved_train, y_improved_test = ..., ...
 ↪, ..., ...

# fit the linear model and make predictions
...

# compute RMSE on train and test sets
train_rmse_improved_cpc = ...
test_rmse_improved_cpc = ...

# BEGIN SOLUTION NO PROMPT
X_improved = mask_data[['NEVER', 'RARELY', 'SOMETIMES', 'FREQUENTLY', 'ALWAYS',␣
 ↪'9/5/2021_cpc']]
y_improved = mask_data['9/12/2021_cpc']

X_improved_train, X_improved_test, y_improved_train, y_improved_test =␣
 ↪train_test_split(X_improved, y_improved, test_size=0.33, random_state=42)

model = lm.LinearRegression(fit_intercept = False)
model.fit(X_improved_train, y_improved_train)

train_rmse_improved_cpc = np.sqrt(np.mean((model.predict(X_improved_train) -␣
 ↪y_improved_train) ** 2))
test_rmse_improved_cpc = np.sqrt(np.mean((model.predict(X_improved_test) -␣
 ↪y_improved_test) ** 2))
# END SOLUTION

train_rmse_improved_cpc, test_rmse_improved_cpc
```

[25]:
```
(0.002412718984782543, 0.002426958021684323)
```

[26]:
```
grader.check("q6b")
```

[26]:
```
q6b results: All test cases passed!
```

1.9.3 Question 6c

Compare the RMSE of our improved model with an extra feature with the intercept term removed with the RMSE obtained in the model from Question 3a.

Comment on what you would *expect* to happen if you repeated the multicollinearity and bias-variance analyses on this new model using bootstrapping. Specifically, what would you expect to happen with this new model bias?

Hint: If you wish, you may want to carry out this analysis by adding a cell below this. Please delete it afterwards and note that you *may* run into memory issues if you run it too many times!

Type your answer here, replacing this text.

SOLUTION:

There is no clear linear dependence anymore without a bias term, so we're likely to see *less* multicollinearity (i.e. more features are statistically significant) but it'll likely still be present since our mask features are correlated per Question 1b. We would expect to have lesser model bias and lesser expected error.

Note that we need not see lower model variance since the error has also decreased!

Closing note: The model you built in Question 6 is called an *autoregressive model*. To understand more about autoregressive models and collinearity, check out this paper pre-print.

1.10 Congratulations!

Congrats! You are finished with this homework assignment.

1.11 Detailed Submission Instructions

There are two parts to this assignment. 1. Question 1 is a written problem and should be submitted as a separate PDF to the Written portal of Gradescope. Please see the top of this notebook for the question writeup.

1. All other questions are submitted as part of this notebook. Please see the following cells to generate the zip file for the Coding portal of Gradescope.

To double-check your work, the cell below will rerun all of the autograder tests.

[27]:
```
grader.check_all()
```

[27]:
```
q2a results: All test cases passed!

q2b results: All test cases passed!

q3a results: All test cases passed!

q4a results: All test cases passed!

q4b results: All test cases passed!
```

```
q4c results: All test cases passed!

q5a results: All test cases passed!

q5c results: All test cases passed!

q6a results: All test cases passed!

q6b results: All test cases passed!
```

1.12 Submission

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. The cell below will generate a zip file for you to submit. **Please save before exporting!**

[ ]:
```
# Save your notebook first, then run this cell to export your submission.
grader.export()
```