```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun May 22 20:19:40 2022

@author: hillarywolff
"""
import numpy as np
import pandas as pd
from numpy import ravel

from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import cross_val_score, GridSearchCV,
KFold
from sklearn.metrics import mean_squared_error
from sklearn.cross_decomposition import PLSRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
import seaborn as sns
from matplotlib import pyplot as plt
from itertools import combinations
from sklearn import model_selection
from sklearn.preprocessing import StandardScaler

def heat_map(cm):
    fig, ax = plt.subplots()
    ax = sns.heatmap(cm, annot=True,
                cmap='Blues')

    ax.set_title('Confusion Matrix with labels\n');
    ax.set_xlabel('\nPredicted Values')
    ax.set_ylabel('Actual Values ');
    ax.xaxis.set_ticklabels(['False','True'])
    ax.yaxis.set_ticklabels(['False','True'])

    return fig


PATH = r"/Users/hillarywolff/Documents/GitHub/machine_learning/PS4/"
# ch. 6, #9
# a. split the data set into a training set and a test set
college = pd.read_csv(PATH+'Data-College.csv')
college = college.drop('Unnamed: 0', axis=1)
college['Private'] = np.where(college['Private'].str.contains('Yes'),
```

```
1, 0)
X = college.drop('Apps', axis=1)
y = college[['Apps']]

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.7, random_state=37)

# b. fit a linear model using least squares on the training set and
report the
# test error
model = LinearRegression().fit(X_train, y_train)
y_pred_ols = model.predict(X_test)
mean_squared_error(y_test, y_pred_ols)
# 1382455

# e. fit a PCR model on the training set, with M chosen by cross-
validation.
# report the test error obtained, along with the value of M selected
by X-val
n_components = np.arange(2, 18)
key = []
values = []
for i in n_components:
    pcr = make_pipeline(StandardScaler(), PCA(n_components=i),
LinearRegression())
    pcs = pcr.fit(X_train, y_train)
    y_predict = pcr.predict(X_test)
    key.append(i)
    values.append(mean_squared_error(y_test, y_predict))
pcr_mse = dict(zip(key, values))

min(pcr_mse, key=pcr_mse.get)
# M=17, error: 1382455.02


# f. fit a PLS model on the training set, with M chosen by cross-
validation.
# report the test error obtained, along with the value of M selected
by X-val
pls = PLSRegression()
components = np.arange(2, 18)
param_grid = [{'n_components' : components}]

grid_search_pls = GridSearchCV(pls, param_grid, cv=10, scoring =
'neg_mean_squared_error')
grid_predict = grid_search_pls.fit(X_train, y_train)
y_pred = grid_predict.predict(X_test)
print(mean_squared_error(y_test, y_pred))
print(grid_search_pls.best_params_)
```

```
# M=10, MSE=1406884.07



# g. comment on the results obtained. how accurrately can we predict
the number
# of college applications received? is there much difference among the
test
# errors resulting from these five approaches?

# looking at the three tests run, we end up with MSE for OLS at
1382455, PCR at
# 1370097, and PLS at 1407576. This means that our PCR is the best
predictor
# of college applications received since it has the lowest MSE. The
three
# tests are not that far from each other in MSE result, specifically
comparing
# OLS to PCR since they are very similar to each other. Our results
however are
# incredibly inaccurate given how high the MSEs are.


# ch. 8, #4
# a. sketch the tree corresponding to the partition of the predictor
space illustrated
# in the left-handed panel of figure 8.14. the numbers inside the
boxes indicate
# the mean of Y within each region.

#### see attached


# b. Create a diagram similar to the left-handed panel of figure 8.14,
using
# the tree illustrated in the right=hand panel of the same figure. you
should
# divide up the predictor space into the correct regions, and indicate
the mean
# for each region

##### see attached

# modified question 9

# i. Create a training set containing a random sample of 800
observations, and a test
# set containing the remaining observations.

oj = pd.read_csv(PATH+'Data-OJ.csv')
```

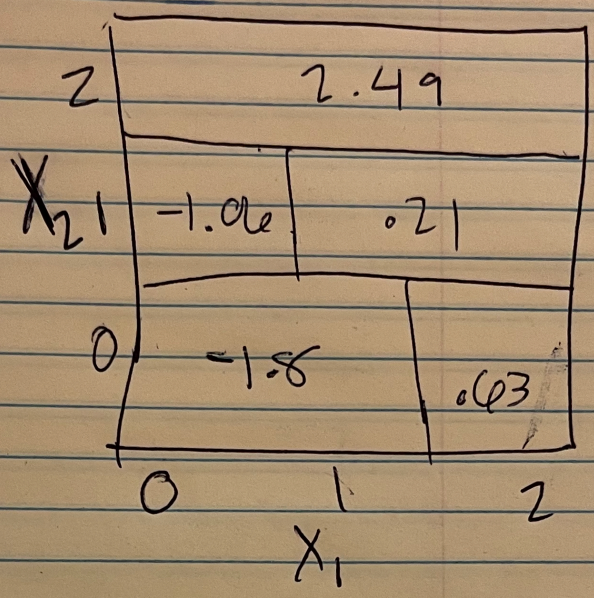
```
oj['Store7'] = np.where(oj['Store7'].str.contains('Yes'), 1, 0)
oj['Purchase'] = np.where(oj['Purchase'].str.contains('CH'), 1, 0)

X = oj.drop(['Purchase'], axis=1)
y = oj[['Purchase']]

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=.252, random_state=37)

# ii. Fit a full, unpruned tree to the training data, with Purchase as
the response and the
# other variables as predictors. What is the training error rate?

model = DecisionTreeClassifier(random_state=37, criterion='gini')
model.fit(X_train, y_train)
y_pred = model.predict(X_train)
mean_squared_error(y_train, y_pred)
# 0.01

# iii. Create a plot of the tree The plot is a mess, isn’t it? For the
purposes of this
# question, fit another tree with the max_depth parameter set to 3 in
order to get an
# interpretable plot. How many terminal nodes does the tree have? Pick
one of the
# terminal nodes, and interpret the information displayed.
#
plt.figure(figsize=(12,12))
tree.plot_tree(model, fontsize=10)
plt.show()

model = DecisionTreeClassifier(max_depth = 3, random_state=1,
criterion='gini')
model.fit(X_train, y_train)

plt.figure(figsize=(12,12))
tree.plot_tree(model, fontsize=10)
plt.show()


# how many terminal nodes: 8
# pick one and interpret the results

# gini = 0.206. this gini coefficient tells us that the points within
this
# node are fairly equally distributed, since the closer to 0 the more
equal
# the distribution
# samples = 17. this branch has 17 data points that fall into it from
our dataset
```

```
# value = [15, 2] 15 coded 1 (purchase), 2 coded 0(not purchase)


# iv. Predict the response on the test data, and produce a confusion
matrix comparing
# the test labels to the predicted test labels. What is the test error
rate?
#
y_predict = model.predict(X_test)
cm = confusion_matrix(y_test, y_predict)
heat_map(cm)

print('\nTest error rate: ', 1 - accuracy_score(y_test, y_predict))
# 0.185

# v. Determine the optimal tree size by tuning the ccp_alpha argument
in scikit-
# learn's DecisionTreeClassifier You can use GridSearchCV for this
pur-
# pose.

alpha_param = (10**np.linspace(start=-2, stop=-.5, num=100))
parameters = {'ccp_alpha': alpha_param}
cv_tree = GridSearchCV(model, parameters)
cv_tree.fit(X_train, y_train)
alpha = cv_tree.best_params_['ccp_alpha']

model = DecisionTreeClassifier(ccp_alpha=alpha)
model.fit(X_train, y_train)
print(model.get_n_leaves())

# optimal tree size is 5

# vi. Produce a plot with tree size on the x-axis and cross-validated
classification error
# rate on the y-axis calculated using the method in the previous
question. Which tree
# size corresponds to the lowest cross-validated classification error
rate?
#
tree_size = []
for i in alpha_param:
    model = DecisionTreeClassifier(ccp_alpha=i)
    model_fit = model.fit(X_train, y_train)
    tree_size.append(model_fit.get_n_leaves())

cv_scores = []
for mean_score in zip(cv_tree.cv_results_["mean_test_score"]):
    cv_scores.append(mean_score[0])
```

```
error = [1-i for i in cv_scores]
plt.figure(figsize=(10,8))
sns.lineplot(x=tree_size, y=error)
plt.xlabel("Tree size", fontsize= 16)

# vii. Produce a pruned tree corresponding to the optimal tree size
obtained using cross-
# validation. If cross-validation does not lead to selection of a
pruned tree, then
# create a pruned tree with five terminal nodes.
#

# pruned tree
model = DecisionTreeClassifier(max_depth = 5, ccp_alpha=alpha)
model.fit(X_train, y_train)
y_pred = model.predict(X_train)
mean_squared_error(y_train, y_pred)
# pruned train error: 0.166

# viii. Compare the training error rates between the pruned and
unpruned trees. Which is
# higher? Briefly explain.

# unpruned tree
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_train)
mean_squared_error(y_train, y_pred)
# unpruned error: 0.01

# the error rate is higher for our pruned tree while our unpruned tree
has a very
# low training error. This has a low training error meaning it has
been overfitted


# ix. Compare the test error rates between the pruned and unpruned
trees. Which is
# higher? Briefly explain.
# pruned tree
model = DecisionTreeClassifier(max_depth = 5, ccp_alpha=alpha)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
mean_squared_error(y_test, y_pred)
# pruned test error: 0.181


# unpruned tree
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test)
mean_squared_error(y_test, y_pred)
# unpruned error: 0.262

# looking at the test error, the unpruned tree has a higher test error
than
# the pruned model. having a high test error again corresponds with
overfitting
# which can happen when we don't use the ideal number of nodes
```

4.B



4.A