PS5

October 17, 2025

0.1 Problem set 5

0.2 Name: [Yawen Tan]

0.3 Link to your PS5 github repo: [https://github.com/IsabellaTan/Brown-DATA1030-HW5.git]

0.3.1 Problem 0

-2 points for every missing green OK sign.

Make sure you are in the DATA1030 environment.

[1]:
```
from __future__ import print_function
from packaging.version import parse as Version
from platform import python_version

OK = '\x1b[42m[ OK ]\x1b[0m'
FAIL = "\x1b[41m[FAIL]\x1b[0m"

try:
    import importlib
except ImportError:
    print(FAIL, "Python version 3.12.10 is required,"
                " but %s is installed." % sys.version)

def import_version(pkg, min_ver, fail_msg=""):
    mod = None
    try:
        mod = importlib.import_module(pkg)
        if pkg in {'PIL'}:
            ver = mod.VERSION
        else:
            ver = mod.__version__
        if Version(ver) == Version(min_ver):
            print(OK, "%s version %s is installed."
                  % (lib, min_ver))
        else:
            print(FAIL, "%s version %s is required, but %s installed."
                  % (lib, min_ver, ver))
```

```python
    except ImportError:
        print(FAIL, '%s not installed. %s' % (pkg, fail_msg))
    return mod


# first check the python version
pyversion = Version(python_version())

if pyversion >= Version("3.12.10"):
    print(OK, "Python version is %s" % pyversion)
elif pyversion < Version("3.12.10"):
    print(FAIL, "Python version 3.12.10 is required,"
                " but %s is installed." % pyversion)
else:
    print(FAIL, "Unknown Python version: %s" % pyversion)


print()
requirements = {'numpy': "2.2.5", 'matplotlib': "3.10.1",'sklearn': "1.6.1",
                'pandas': "2.2.3",'xgboost': "3.0.0", 'shap': "0.47.2",
                'polars': "1.27.1", 'seaborn': "0.13.2"}

# now the dependencies
for lib, required_version in list(requirements.items()):
    import_version(lib, required_version)
```

```
[ OK ] Python version is 3.12.10

[ OK ] numpy version 2.2.5 is installed.
[ OK ] matplotlib version 3.10.1 is installed.
[ OK ] sklearn version 1.6.1 is installed.
[ OK ] pandas version 2.2.3 is installed.
[ OK ] xgboost version 3.0.0 is installed.
[ OK ] shap version 0.47.2 is installed.
[ OK ] polars version 1.27.1 is installed.
[ OK ] seaborn version 0.13.2 is installed.
```

0.3.2 Problem 1

We will investigate the importance of preprocessing your dataset in this exercise. You will work with the unprocessed diabetes dataset in problem 1a, and you will preprocess it in 1b. You'll train a regression model in both problems and you'll compare the feature coefficients. This exercise is a preparation for one of the last topics we will cover in class: interpretability.

The coefficients of a linear model (the w's) can be used to measure the importance of features if and only if all features have the same mean and standard deviation (even the one-hot encoded features!). Here is why:

The feature coefficients are determined by two things: - the importance of the feautre - more important features have larger w's - the order of magnitude of the feature relative to the target variable - if a feature tends to be orders of magnitudes larger than the target variable, the feature's weight will be small in order to bring the feature to the same order of magnitude as the target variable - similarly, if a feature tends to be orders of magnitudes smaller than the target variable, the feature's weight will be large in order to bring the feature to the same order of magnitude as the target variable

This is an ambiguity since a feature could have a large w because either it is important and/or because the feature values tend to be much smaller than the target variable.

The only way to remove this ambiguity is to standardize all features. If all features have the same mean and the same standard deviation (most commonly 0 mean and 1 standard deviation when you use the StandardScaler), all features have the same order of magnitudes. Then the feature coefficient is purely determined by the importance of the feature and it can be used to rank features based on how important they are.

0.3.3 Problem 1a (10 points)

First, read in the dataset into a pandas dataframe using the tab delimited file linked at this page. Loop through 10 different random states. Split the dataset into training and test sets (70-30 ratios). We won't tune the regularization parameter, so we don't need a validation set for this exercise. Do not preprocess the sets. Train a linear regression model with Ridge regularization with alpha = 1. Use RMSE as your evaluation metric and print out the mean and standard deviation of the training and test scores. For each random state, save the test score and the model's coefficients for each feature.

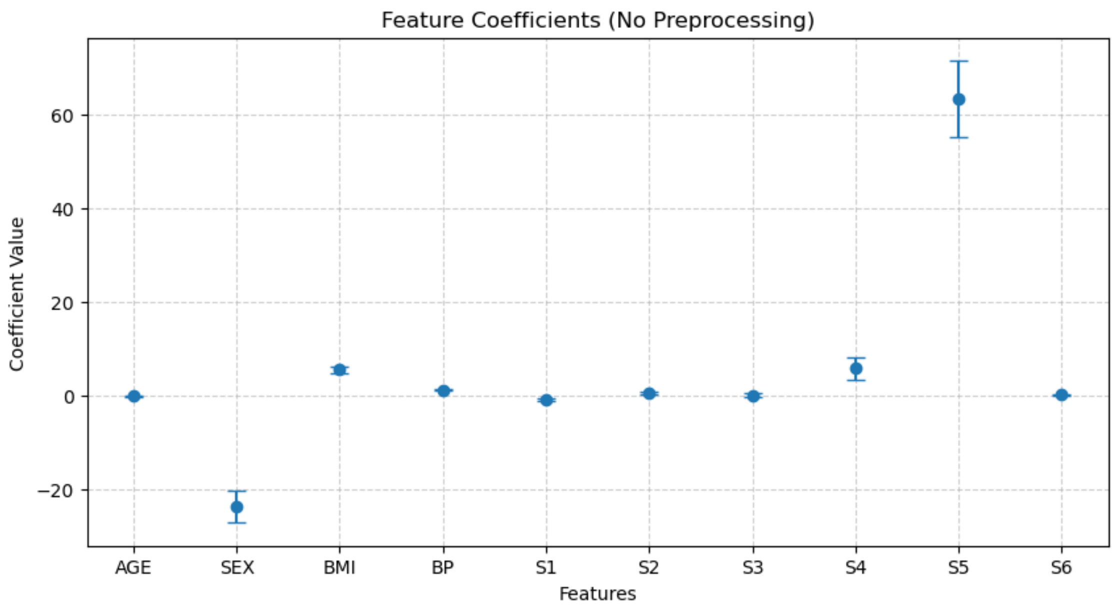
Once you loop through the 10 random states, prepare one plot which shows the mean and standard deviation of each feature's coefficient. Make sure to have proper x and y ticks and labels, as well as a title.

Hint: use plt.errorbar() or something similar for the figure.

[13]:
```python
# your code here
# Load libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.metrics import root_mean_squared_error
from sklearn.preprocessing import StandardScaler
from scipy.stats import ttest_rel
# Load dataset
url = "https://www4.stat.ncsu.edu/~boos/var.select/diabetes.tab.txt"
df = pd.read_csv(url, sep="\t")
# Separate features and target variable
X = df.drop(columns=['Y'])
y = df['Y']
# Create lists to store results
train_rmse_list = []
```

```python
test_rmse_list = []
coef_list = []
# Loop through the 10 random states
for state in range(10):
    # Split the dataset into training and test sets (70-30 ratios)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=state)
    # Fit Ridge regression model with alpha=1 using training set
    model = Ridge(alpha=1)
    model.fit(X_train, y_train)
    # Find predictions for training and test sets
    train_pred = model.predict(X_train)
    test_pred = model.predict(X_test)
    # Calculate RMSE for training and test sets
    train_rmse = root_mean_squared_error(y_train, train_pred)
    test_rmse = root_mean_squared_error(y_test, test_pred)
    # Append results to lists
    train_rmse_list.append(train_rmse)
    test_rmse_list.append(test_rmse)
    coef_list.append(model.coef_)

# Store the RMSE lists for question 1b
test_rmse_1a = test_rmse_list.copy()
# Calculate mean and standard deviation of RMSEs for training and test sets
train_mean = np.mean(train_rmse_list)
train_std = np.std(train_rmse_list)
test_mean = np.mean(test_rmse_list)
test_std = np.std(test_rmse_list)
# Print results
print(f"Training RMSE: mean = {train_mean:.3f}, std = {train_std:.3f}")
print(f"Testing RMSE:  mean = {test_mean:.3f}, std = {test_std:.3f}")

# Covert coef_list to a NumPy array for easier calculations
coef_array = np.array(coef_list)
# Calculate mean and standard deviation of coefficients across the 10 runs
coef_mean = np.mean(coef_array, axis=0) # axis=0 means we calculate based on␣
 ↪columns
coef_std = np.std(coef_array, axis=0)
# Plot the coefficients with error bars
plt.figure(figsize=(10, 5))
# fmt='o' means we want to use circle to represent the mean coefficients
# capsize=5 means the horizontal lines at the end of the error bars will have a␣
 ↪length of 5
plt.errorbar(X.columns, coef_mean, yerr=coef_std, fmt='o', capsize=5)
plt.title("Feature Coefficients (No Preprocessing)")
plt.xlabel("Features")
plt.ylabel("Coefficient Value")
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```

```
Training RMSE: mean = 53.563, std = 0.962
Testing RMSE:  mean = 53.892, std = 2.443
```



0.3.4 Problem 1b (10 points)

Now let's repeat what you did in 1a but this time preprocess the sets with the standard scaler.

Notice how the coefficients change as a result of preprocessing.

Answer the following questions:

Q1: Do you see a statistically significant change in the test score?

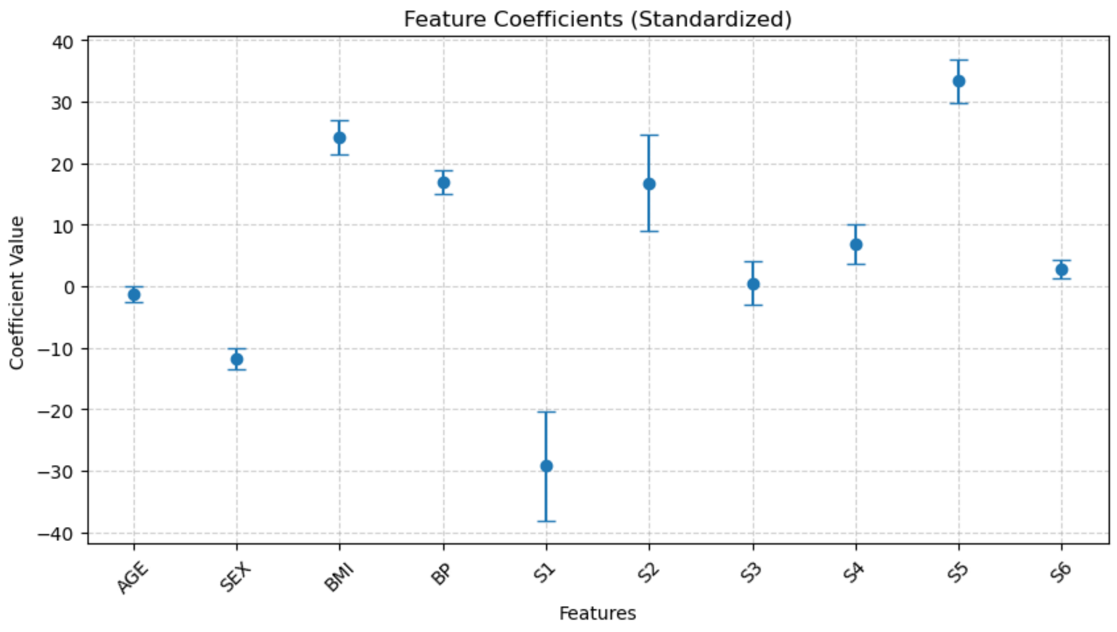
Q2: How would you order the features based on the coefficients for feature importance? Explain your answer in a few sentences.

Q3: Print out names of the top 3 most important features!

[14]:
```
# your code here

# Load dataset
url = "https://www4.stat.ncsu.edu/~boos/var.select/diabetes.tab.txt"
df = pd.read_csv(url, sep="\t")
# Separate features and target variable
X = df.drop(columns=['Y'])
y = df['Y']
# Create lists to store results
```

```
train_rmse_list = []
test_rmse_list = []
coef_list = []
# Loop through the 10 random states
for state in range(10):
    # Split the dataset into training and test sets (70-30 ratios)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=state)
    # Standardize the features
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)
    # Fit Ridge regression model with alpha=1 using training set
    model = Ridge(alpha=1)
    model.fit(X_train, y_train)
    # Find predictions for training and test sets
    train_pred = model.predict(X_train)
    test_pred = model.predict(X_test)
    # Calculate RMSE for training and test sets
    train_rmse = root_mean_squared_error(y_train, train_pred)
    test_rmse = root_mean_squared_error(y_test, test_pred)
    # Append results to lists
    train_rmse_list.append(train_rmse)
    test_rmse_list.append(test_rmse)
    coef_list.append(model.coef_)

# Store the RMSE lists for question 1b
test_rmse_1b = test_rmse_list.copy()
# Calculate mean and standard deviation of RMSEs for training and test sets
train_mean = np.mean(train_rmse_list)
train_std = np.std(train_rmse_list)
test_mean = np.mean(test_rmse_list)
test_std = np.std(test_rmse_list)
# Print results
print(f"Training RMSE: mean = {train_mean:.3f}, std = {train_std:.3f}")
print(f"Testing RMSE:  mean = {test_mean:.3f}, std = {test_std:.3f}")

# Covert coef_list to a NumPy array for easier calculations
coef_array = np.array(coef_list)
# Calculate mean and standard deviation of coefficients across the 10 runs
coef_mean = np.mean(coef_array, axis=0) # axis=0 means we calculate based on␣
 ↪columns
coef_std = np.std(coef_array, axis=0)
# Plot the coefficients with error bars
plt.figure(figsize=(10, 5))
# fmt='o' means we want to use circle to represent the mean coefficients
# capsize=5 means the horizontal lines at the end of the error bars will have a␣
↪length of 5
plt.errorbar(X.columns, coef_mean, yerr=coef_std, fmt='o', capsize=5)
# Rotate x-axis labels for better readability
plt.xticks(rotation=45)
plt.title("Feature Coefficients (Standardized)")
plt.xlabel("Features")
plt.ylabel("Coefficient Value")
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()



# Question 1b Q1
# Perform a t-test on the two sets of test RMSEs
t_stat, p_value = ttest_rel(test_rmse_1a, test_rmse_1b)
# Print the t-statistic and p-value
print("t-statistic:", t_stat)
print("p-value:", p_value)
# Determine if the change is statistically significant at the 0.05 level
if p_value < 0.05:
    print("Test RMSE change is statistically significant (p < 0.05).")
else:
    print("Test RMSE change is NOT statistically significant (p >= 0.05).")

# Question 1b Q3
# Sort features by the absolute value of their mean coefficients in descending␣
↪order
sorted_idx = np.argsort(np.abs(coef_mean))[::-1]
# Get the indices of the top 3 features
top_3_idx = sorted_idx[:3]
# Get the names of the top 3 features
top_3_features = X.columns[top_3_idx]
# Print the top 3 features
print("Top 3 features based on absolute mean coefficients:")
for feature in top_3_features:
    print(feature)
```

```
Training RMSE: mean = 53.568, std = 0.965
Testing RMSE:  mean = 53.870, std = 2.442
```



```
t-statistic: 1.301952913446945
p-value: 0.22526606010058245
Test RMSE change is NOT statistically significant (p >= 0.05).
Top 3 features based on absolute mean coefficients:
S5
S1
BMI
```

your answers here

Q1: Do you see a statistically significant change in the test score?

No, there is no statistically significant change in the test score after standardization. The paired t-test gives a t-statistic of 1.302 and a p-value of 0.225, which is greater than 0.05. For the unprocessing data, the test RME has mean 53.892 and standard deviation 2.443; For the standardized data, the test RME has mean 53.870 and standard deviation 2.442. Because Ridge Regression comes with regularization, it ensures that the model is not prone to overfitting and can automatically adjust the magnitude of the coefficients, so RMSE is not sensitive to standardization.

Q2: How would you order the features based on the coefficients for feature importance? Explain your answer in a few sentences.

We can sort the coefficients by abs(coef_mean) , where larger values indicate more important features. The sign of coef_mean indicates the 'positive' or 'negative' impact on the target variable, and the absolute value indicates its importance to the target variable.

Q3: Print out names of the top 3 most important features!

Top 3 features based on absolute mean coefficients: S5, S1, BMI

0.3.5 Problem 2

In class we have seen both l1 (Lasso) and l2 (Ridge) regression. These are two basic ways to perform regularization. In the following problem, we will explore the **elastic net**, a third regularization technique that combines both l1 and l2 penalties. We will use this in a classification context.

The basic idea of the elastic net is that the cost function in regression becomes

$L(\theta) = \frac{1}{n}\sum_{i=1}^{n}[(\theta_0 + \sum_{j=1}^{m}\theta_j x_{ij} - y_i)^2] + \alpha\rho\sum_{j=0}^{m}|\theta_j| + \alpha(1-\rho)\sqrt{\sum_{j=0}^{m}\theta_j^2}$,

and the cost function in classification is

$L(\theta) = -\frac{1}{N}\sum_{i=1}^{n}[y_i \ln(\frac{1}{1+e^{-\theta_0+\sum_{j=1}^{m}\theta_j x_{ij}}}) + (1-y_i)\ln(1-\frac{1}{1+e^{-\theta_0+\sum_{j=1}^{m}\theta_j x_{ij}}}))] + \alpha\rho\sum_{j=0}^{m}|\theta_j| + \alpha(1-\rho)\sqrt{\sum_{j=0}^{m}\theta_j^2}$,

where α is the regularization parameter and ρ is the l1 ratio (how much weight we assign to the l1 term over the l2 term in the cost function). Basically, an elastic net uses the weighted sum of the l1 and l2 regularization terms. The weight of the l1 term is ρ and the weight of the l2 term is $(1-\rho)$ where ρ is between 0 and 1.

You can read more about the elastic net here.

Note that there are a large number of regularization techniques available in sklearn, the complete list of linear models are described here.

0.3.6 Problem 2a (3 points)

Load the training and validation sets from train.csv and val.csv from the `data` folder.

Run a logistic regression model without regularization on the data and print the accuracy score of the validation set. Use the 'saga' solver.

[ ]:
```
# your code here
# Load libraries
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
# Load the datasets
train_df = pd.read_csv("data/train.csv")
val_df = pd.read_csv("data/val.csv")
# Separate features and target variable for training set and validation set
X_train = train_df.drop(columns=['y'])
y_train = train_df['y']
X_val = val_df.drop(columns=['y'])
y_val = val_df['y']
# Initialize and fit the logistic regression model
model = LogisticRegression(penalty=None, solver='saga',␣
  ↪max_iter=5000,random_state=321) # We set penalty to 'none' to disable␣
  ↪regularization and let solver to 'saga' and let max_iter to 5000 to ensure␣
  ↪convergence
model.fit(X_train,y_train)
# Make predictions on the validation set
```

```
y_val_pred = model.predict(X_val)
# Calculate and print the accuracy on the validation set
val_acc = accuracy_score(y_val, y_val_pred)
# Print results
print(f"Validation Accuracy (no regularization): {val_acc:.3f}")
```

```
Validation Accuracy (no regularization): 0.708
```

0.3.7 Problem 2b (7 points)

Perform l1 regularization on the data.

The value of the alpha should contain 21 uniformly spaced values in log from 1e-2 to 1e2.

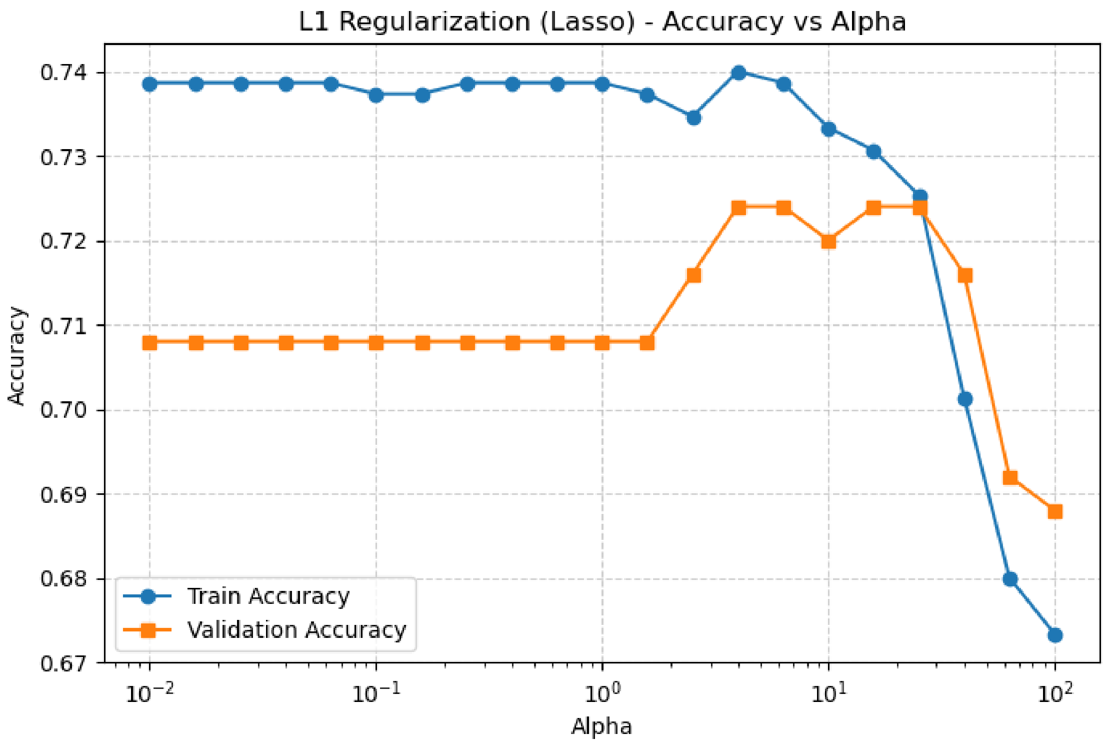
Again, use the 'saga' solver and if you see a converge warning, fix it without ignoring the warning.

Plot the train and validation accuracy scores.

Print the best validation accuracy score and the corresponding alpha values. If multiple alpha values give you euqlly good validation scores, print out all the alpha values, not just one.

```
[ ]: # your code here
# Let alpha vary from 1e-2 to 1e2 (21 values in total)
alphas = np.logspace(-2, 2, 21)
# Initialize lists to store training and validation accuracies
train_acc_list = []
val_acc_list = []

# Loop through each alpha value
for alpha in alphas:
    model = LogisticRegression(
        penalty='l1', # set penalty to 'l1' for Lasso regularization
        solver='saga',
        C=1/alpha,
        max_iter=5000,
        random_state=321)
    model.fit(X_train, y_train)

    # make predictions on training set and calculate training accuracy
    y_train_pred = model.predict(X_train)
    train_acc = accuracy_score(y_train, y_train_pred)
    # save to the lsit
    train_acc_list.append(train_acc)

    # make predictions on validation set and calculate validation accuracy
    y_val_pred = model.predict(X_val)
    val_acc = accuracy_score(y_val, y_val_pred)
    # save to the list
    val_acc_list.append(val_acc)
```

```
plt.figure(figsize=(8,5))
plt.plot(alphas, train_acc_list, marker='o', label='Train Accuracy') #␣
  ↪marker='o' let the data points be represented by circles
plt.plot(alphas, val_acc_list, marker='s', label='Validation Accuracy') #␣
  ↪marker='s' let the data points be represented by squares
plt.xscale('log') # set x-axis to logarithmic scale in order to better␣
  ↪visualize the results
plt.xlabel('Alpha')
plt.ylabel('Accuracy')
plt.title('L1 Regularization (Lasso) - Accuracy vs Alpha')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()

# Find the alpha that gives the highest validation accuracy
val_acc_array = np.array(val_acc_list) # Convert to NumPy array for easier␣
  ↪calculations
best_val_acc = val_acc_array.max() # find the maximum validation accuracy
best_alpha_idx = np.where(val_acc_array == best_val_acc)[0] # find the index/
  ↪indices of the maximum validation accuracy
best_alphas = alphas[best_alpha_idx] # find the corresponding alpha value(s)
# Print the results
print(f"Best Validation Accuracy: {best_val_acc:.3f}")
print("Alpha with best validation accuracy:", best_alphas)
```



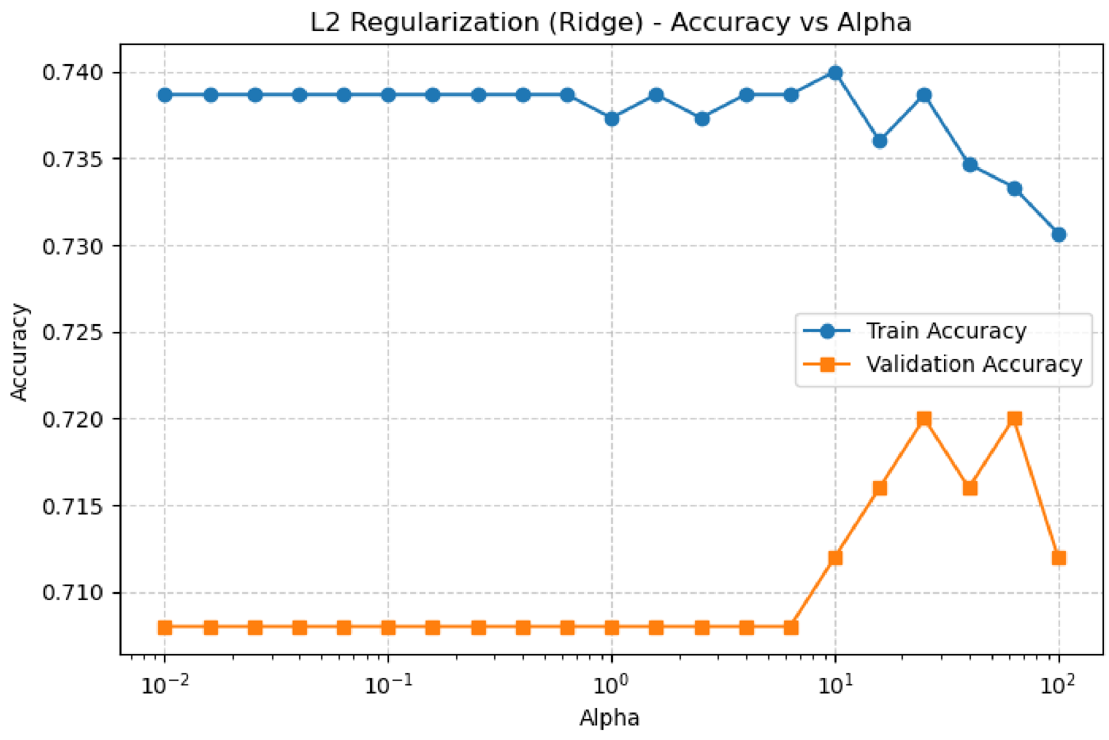
```
Best Validation Accuracy: 0.724
Alpha with best validation accuracy: [ 3.98107171  6.30957344 15.84893192
25.11886432]
```

0.3.8 Problem 2c (1 point)

Perform l2 regularization on the data. The alpha values and all the other instructions are the same as in 2b.

```
[ ]: # your code here

# Let alpha vary from 1e-2 to 1e2 (21 values in total)
alphas = np.logspace(-2, 2, 21)
# Initialize lists to store training and validation accuracies
train_acc_list = []
val_acc_list = []

# Loop through each alpha value
for alpha in alphas:
    model = LogisticRegression(
        penalty='l2', # set penalty to 'l2' for ridge regularization
        solver='saga',
        C=1/alpha,
        max_iter=5000,
        random_state=321)
    model.fit(X_train, y_train)

    # make predictions on training set and calculate training accuracy
    y_train_pred = model.predict(X_train)
    train_acc = accuracy_score(y_train, y_train_pred)
    # save to the lsit
    train_acc_list.append(train_acc)

    # make predictions on validation set and calculate validation accuracy
    y_val_pred = model.predict(X_val)
    val_acc = accuracy_score(y_val, y_val_pred)
    # save to the list
    val_acc_list.append(val_acc)

plt.figure(figsize=(8,5))
plt.plot(alphas, train_acc_list, marker='o', label='Train Accuracy') #␣
 ↪marker='o' let the data points be represented by circles
plt.plot(alphas, val_acc_list, marker='s', label='Validation Accuracy') #␣
 ↪marker='s' let the data points be represented by squares
```

```
plt.xscale('log') # set x-axis to logarithmic scale in order to better␣
 ↪visualize the results
plt.xlabel('Alpha')
plt.ylabel('Accuracy')
plt.title('L2 Regularization (Ridge) - Accuracy vs Alpha')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()

# Find the alpha that gives the highest validation accuracy
val_acc_array = np.array(val_acc_list) # Convert to NumPy array for easier␣
 ↪calculations
best_val_acc = val_acc_array.max() # find the maximum validation accuracy
best_alpha_idx = np.where(val_acc_array == best_val_acc)[0] # find the index/
 ↪indices of the maximum validation accuracy
best_alphas = alphas[best_alpha_idx] # find the corresponding alpha value(s)
# Print the results
print(f"Best Validation Accuracy: {best_val_acc:.3f}")
print("Alpha with best validation accuracy:", best_alphas)
```



```
Best Validation Accuracy: 0.720
Alpha with best validation accuracy: [25.11886432 63.09573445]
```

0.3.9 Problem 2d (10 points)

Let's train an elastic net now! The elastic net has two parameters: alpha and rho (the l1 ratio). The l1 ratio should be linearly spaced between 0 and 1 with 21 values in between, while alpha should be spaced as outlined above. (1 point) Use the 'saga' solver. The reason we use the saga solver everywhere in Problem 2 is that it is the only solver in LogisticRegression that works with an elastic net.

Calculate the train and validation accuracy scores for all combinations of alpha and rho.

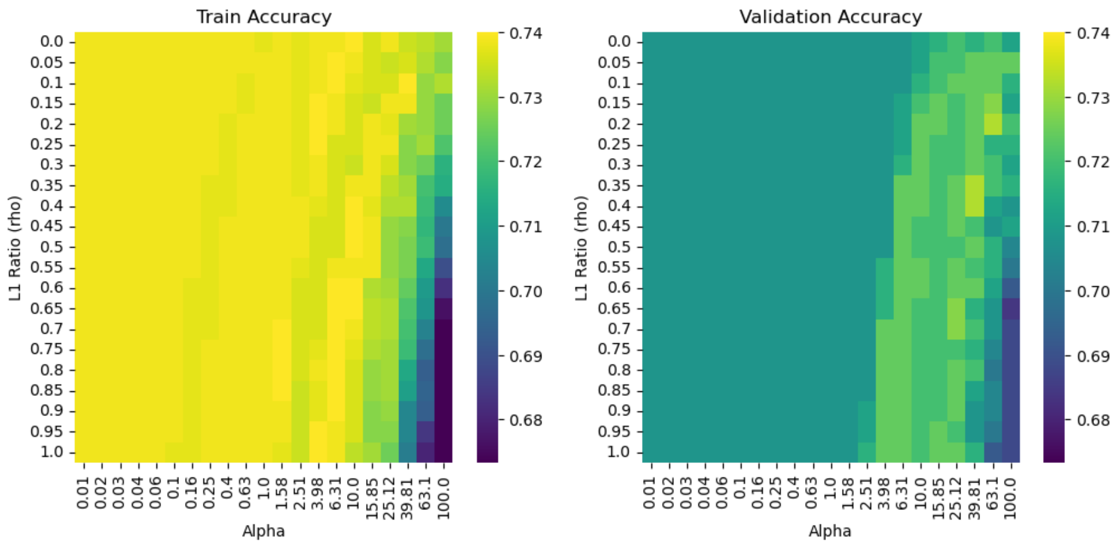
Print the best validation score and the corresponding alpha and rho values.

Prepare heatmaps to show the train and validation scores. Make sure that the data range covered by the two heatmaps are the same so you can easily compare the two heatmaps and you can identify the high bias and high variance regions. Label the plot and add a colorbar. Make the x and y ticks look pretty.

Which of the four approaches gave you the best validation score?

[29]:
```python
# your code here
# Load libraries
import seaborn as sns
# Let alpha vary from 1e-2 to 1e2 (21 values in total)
alphas = np.logspace(-2, 2, 21)
# Let rho vary from 0 to 1 (21 values in total)
rhos = np.linspace(0, 1, 21)
# Initialize metrix to store training and validation accuracies
# Because we have alpha and rho, so we need a 2D metrix to store the results
train_scores = np.zeros((len(rhos), len(alphas)))
val_scores = np.zeros((len(rhos), len(alphas)))

# Loop through each combination of alpha and rho
for i, rho in enumerate(rhos): # i is the index, rho is the value
    for j, alpha in enumerate(alphas): # j is the index, alpha is the value
        # Initialize and fit the logistic regression model with elastic net␣
 ↪regularization
        model = LogisticRegression(
            penalty='elasticnet', # set penalty to 'elasticnet' for elastic net␣
 ↪regularization
            solver='saga', # 'saga' solver supports elastic net regularization
            l1_ratio=rho, # set l1_ratio to rho
            C=1/alpha, # set C to 1/alpha
            max_iter=5000, # set max_iter to 5000 to ensure convergence
            random_state=321
        )
        model.fit(X_train, y_train) # fit the model

        # Find predictions and calculate accuracies of training set
        y_train_pred = model.predict(X_train)
        train_scores[i, j] = accuracy_score(y_train, y_train_pred)
        # Find predictions and calculate accuracies of validation set
        y_val_pred = model.predict(X_val)
        val_scores[i, j] = accuracy_score(y_val, y_val_pred)

# Find the combination of alpha and rho that gives the highest validation␣
 ↪accuracy
best_val_acc = val_scores.max()
best_indices = np.argwhere(val_scores == best_val_acc)# find the index/indices␣
 ↪of the maximum validation accuracy
# Print the best validation accuracy
print(f"Best Validation Accuracy: {best_val_acc:.3f}")
# Loop through best_indices to print all combinations of best alpha and best rho
for row, col in best_indices:
    # Extract the best alpha and best rho
    best_alpha = alphas[col]
    best_rho = rhos[row]
    # Print combination of best alpha and best rho
    print(f"Best alpha: {best_alpha}")
    print(f"Best l1_ratio (rho): {best_rho}")


plt.figure(figsize=(12,5))
# Determine the common color scale limits
vmin = min(train_scores.min(), val_scores.min()) # find the minimum value␣
 ↪between training and validation scores
vmax = max(train_scores.max(), val_scores.max()) # find the maximum value␣
 ↪between training and validation scores

# Plot training set heatmap
plt.subplot(1, 2, 1) # 1 row, 2 columns, 1st subplot
sns.heatmap(train_scores, xticklabels=np.round(alphas,2), yticklabels=np.
 ↪round(rhos,2),
            vmin=vmin, vmax=vmax, cmap='viridis')
# Set title and labels
plt.title('Train Accuracy')
plt.xlabel('Alpha')
plt.ylabel('L1 Ratio (rho)')

# Plot validation set heatmap
plt.subplot(1, 2, 2) # 1 row, 2 columns, 2nd subplot
sns.heatmap(val_scores, xticklabels=np.round(alphas,2), yticklabels=np.
 ↪round(rhos,2),
            vmin=vmin, vmax=vmax, cmap='viridis')
# Set title and labels
plt.title('Validation Accuracy')
plt.xlabel('Alpha')
plt.ylabel('L1 Ratio (rho)')

plt.show()

# Answer the question
print('Question: Which of the four approaches gave you the best validation␣
 ↪score?')
print('Answer: Among the four approaches, Elastic Net achieved the highest␣
 ↪validation accuracy of 0.732, outperforming the other methods: no␣
 ↪regularization (0.708), L1 regularization (0.724), and L2 regularization (0.
 ↪720). Compared to L1 and L2, Elastic Net combines both penalties, which␣
 ↪likely helped it balance sparsity and coefficient shrinkage, leading to␣
 ↪better generalization on the validation set.')
```

```
Best Validation Accuracy: 0.732
Best alpha: 63.095734448019364
Best l1_ratio (rho): 0.2
Best alpha: 39.810717055349734
Best l1_ratio (rho): 0.35000000000000003
Best alpha: 39.810717055349734
Best l1_ratio (rho): 0.4
```



```
Question: Which of the four approaches gave you the best validation score?
Answer: Among the four approaches, Elastic Net achieved the highest validation
accuracy of 0.732, outperforming the other methods: no regularization (0.708),
L1 regularization (0.724), and L2 regularization (0.720). Compared to L1 and L2,
Elastic Net combines both penalties, which likely helped it balance sparsity and
coefficient shrinkage, leading to better generalization on the validation set.
```