CS4049: Introduction to Machine Learning and Data Mining

Assessment 1

1. Loading the data

```
# importing libraries

import os
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
from sklearn.model_selection import KFold
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split
```
✓ 0.7s

Fig.1

We import all libraries we are going to use at the start of the notebook.

```
# importingng the body_fat_data.csv dataset

data_path = os.path.join(os.getcwd(),'body_fat_data.csv')
data = pd.read_csv(data_path, index_col=0) # we load the data into a pandas dataframe, skipping the first column
```
✓ 0.1s

Fig.2

In order to load the body fat data csv file into a dataframe, we get the path to it (from the same directory as the notebook) and save the dataframe to the variable `data` omitting the first column(*Fig.2)*. We do this as the first column of the csv file contains indices from 0 through 251 that we are not going to use as the `pandas.DataFrame` construct provides indices automatically(*Fig.3*).

```
data.head()
```
✓ 0.4s

	Percent body fat using Siri equation 495/Density	Weight (lbs)	Height (inches)	Adiposity index = Weight/Height^2 (kg/m^2)	Neck circumference (cm)	Chest circumference (cm)	...
0	12.3	154.25	67.75	23.7	36.2	93.1	
1	6.1	173.25	72.25	23.4	38.5	93.6	
2	25.3	154.00	66.25	24.7	34.0	95.8	
3	10.4	184.75	72.25	24.9	37.4	101.8	
4	28.7	184.25	71.25	25.6	34.4	97.3	

Fig.3

The data provided contains 17 columns of body measurements/metrics recorded for 252 men(rows).

2. Preprocessing

2.1. Removing outliers/wrong data

In order to remove any outliers that might skew the model's prediction we create a plot of the distribution of each column in the dataframe `data`*(Fig. 4)*.

```
# finding outliers

for col in data.columns:    # plotting each column's values' distribution

    plt.hist(data[col], rwidth=0.8)
    plt.grid()
    plt.title(label=col)
    plt.show()
✓ 4.1s
```

Fig.4

From all 17 graphs and by looking at the csv file, we identify 3 outliers in the data that need to be removed*(Fig.5, Fig.6)*.

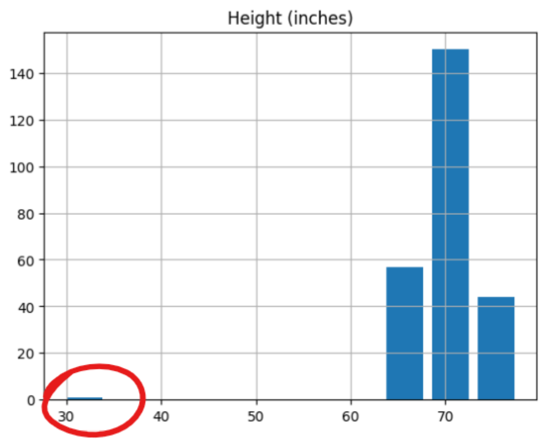

Fig.5

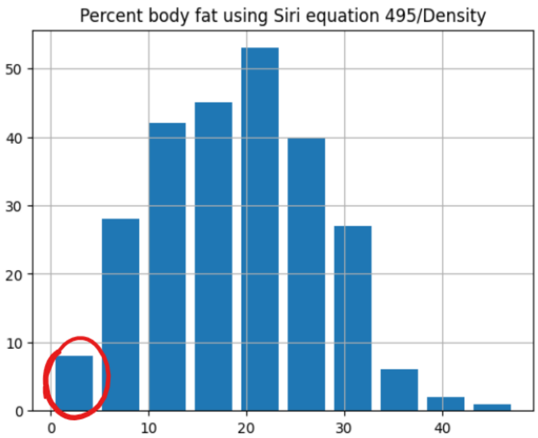

Fig.6

If we look ath the csv file we can find that the `Height (inches)` outlier comes from entry 41, where height is set to the unrealistic value of `29.5` *(Fig.7)*.

	Percent body fat	Density gm/cm^3	Age	Weight (lbs)	Height (inches)	...
41	32.9	1.03	44	205	29.5	

Fig.7

The other 2 outliers we need to remove are entries 181 and 171. Both of them contain unrealistic values of `0` and `0.7` for the `Percent body fat…` column*(Fig. 8)*.

	Percent body fat
181	0
171	0.7

Fig.8

We `drop` (remove) the aforementioned entries(41, 171, 181) from the dataframe and confirm that they have been removed by checking the shape of `data` with `data.shape`.

```
# removing outliers

print(data.shape)
data = data.drop([41, 171, 181]) # 41 - height too low(29.5 in),   171 and 181 - body fat too low(0 and 0.7)
print(data.shape)
✓ 0.1s

(252, 17)
(249, 17)
```

Fig.9

*We can note that entry 215 displays considerably higher entries for `Percent body fat` and the rest of the columns but they all fit each other and are therefore not considered as outliers.

2.2. Feature Selection

Now that we have removed all outliers we drop the columns containing data that cannot be measured or calculated only using a scale and measuring tape, as specified*(Fig.10)*

```
# feature selection
# removing columns of data that cannot be measured or calculated only using a scale and measuring tape

data.drop(["Density gm/cm^3", "Fat Free Weight (1 - fraction of body fat) * Weight, using Brozek formula (lbs)"], axis=1, inplace=True)
✓ 0.4s
```

Fig.10

The next step of feature selection is choosing which features can be eliminated based on their correlation with the target. In order to display this we create a heatmap*(Fig.11, Fig.12)*.

```
# displaying heatmap to help us eliminate features based on correlation

correlation = data.corr()
plt.figure(figsize = (15,12))
sns.heatmap(correlation, annot = True)
✓ 0.8s
```

Fig.11

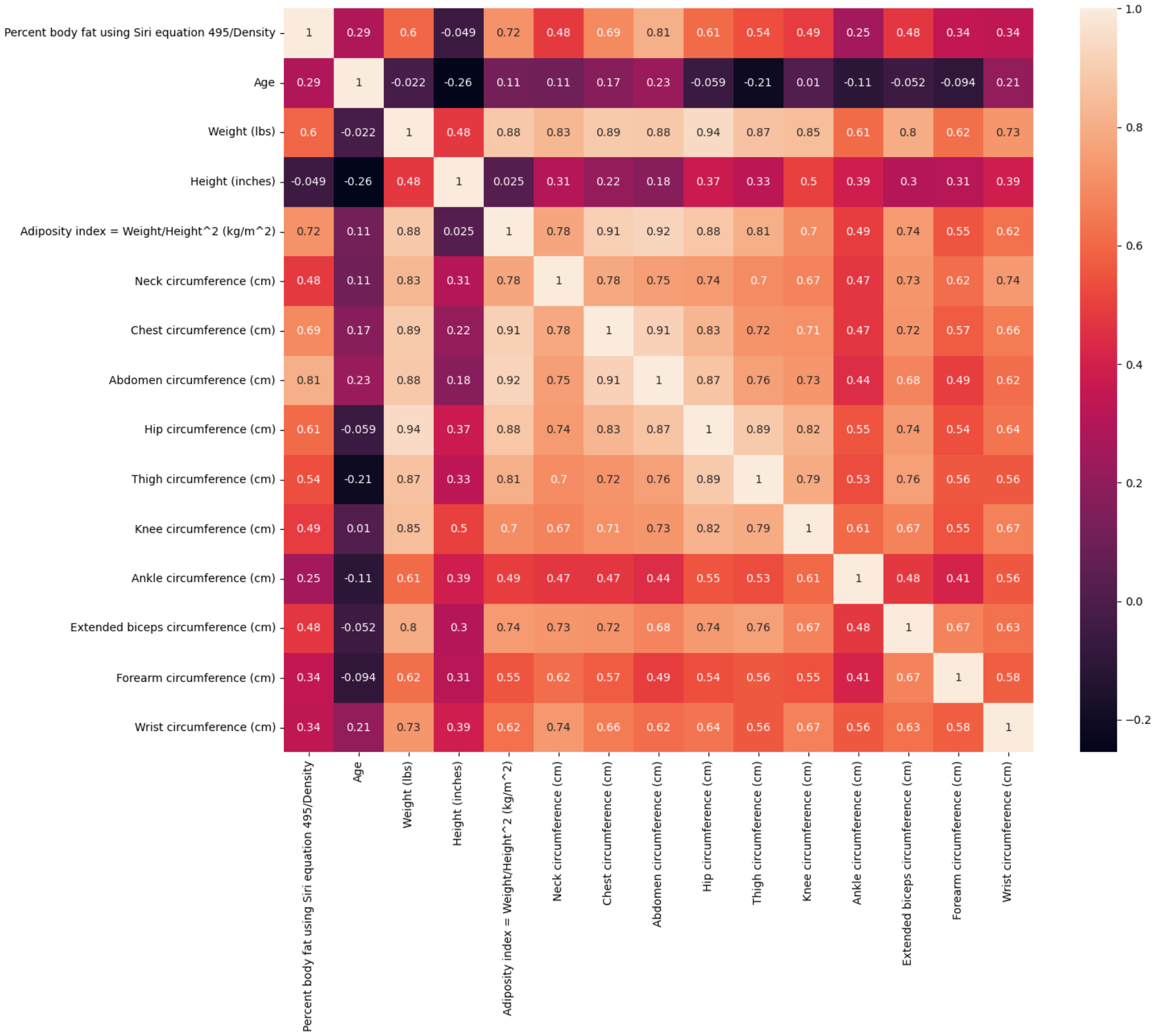

Fig.12

The heatmap contains the correlation between all the columns of the dataframe represented as numbers between -1 and 1. From this we decide to eliminated the columns of data with the least correlation with `Percent body fat`, namely `Age`, `Ankle circumference (cm)`, and `Forearm circumference (cm)`.*(Fig.13)*

```
# removing features with least correlation to target value

data.drop(["Age", "Ankle circumference (cm)", "Forearm circumference (cm)"], axis=1, inplace=True)
✓ 0.4s
```

Fig.13

2.3. Standardization

Feature scaling is an important part of preprocessing. The data provided in the csv file uses a wide range of measurements in different scales such as inches, cm, lbs, etc. Our data also follows the normal distribution. Because of this we choose to standardize the datain each column*(Fig.14)*.

```
#standardizing the data

standardized = (data-data.mean())/data.std()
standardized
```
✓ 0.1s

Fig.14

	Percent body fat using Siri equation 495/Density	Age	Weight (lbs)	Height (inches)	Neck circumference (cm)	Chest circumference (cm)	Abdomen circumference (cm)	Hip circumference (cm)	Thigh circumference (cm)	Knee circumference (cm)	Ankle circumference (cm)	Extended biceps circumference (cm)	Forearm circumference (cm)	V circumfer
0	-0.847019	-1.733852	-0.860825	-0.994878	-0.757709	-0.938968	-0.700746	-0.773101	-0.087100	-0.550688	-0.718206	-0.105961	-0.642821	-1.23
1	-1.603004	-1.812850	-0.207275	0.734103	0.194331	-0.879009	-0.907061	-0.176767	-0.145454	-0.550688	0.167692	-0.608197	0.103971	-0.04
2	0.738111	-1.812850	-0.869424	-1.571205	-1.668356	-0.615191	-0.447542	-0.105775	0.029606	0.121681	0.522051	-1.177399	-1.738116	-1.77
3	-1.078692	-1.496855	0.188294	0.734103	-0.260992	0.104314	-0.588211	0.178193	0.126862	-0.550688	-0.186667	0.027969	0.352902	-0.04
4	1.152684	-1.654853	0.171096	0.349885	-1.502784	-0.435315	0.687188	0.277582	0.729846	1.508442	0.522051	-0.038996	-0.493463	-0.58
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
247	-1.005532	1.979097	-1.548772	-1.283041	-1.295818	-1.406646	-0.850793	-1.582410	-1.915503	-1.601264	-0.954445	-2.248838	-1.489186	0.27
248	1.750156	2.137095	0.747251	-0.226442	1.187764	0.907761	1.156084	0.646741	0.029606	0.920119	0.049572	0.965478	-0.045387	1.99
249	1.225844	2.137095	0.257089	-1.667259	0.359903	1.219546	1.765650	0.249185	0.165764	-0.550688	-0.954445	-0.340338	-0.742393	-0.26
250	0.823465	2.137095	0.394678	0.061722	0.359903	0.883777	0.809101	-0.304553	-0.670633	1.256304	-0.245727	-0.608197	0.352902	1.67
251	1.542870	2.295093	0.970834	-0.130387	1.146371	1.375439	1.484312	1.015899	-0.028747	1.508442	0.876410	0.463241	0.651619	2.86

249 rows × 14 columns

Fig.15

We check that the standardization has been applied in *Fig.15*.

3. Performing cross validation and ridge regression

We start by splitting the dataframe into target(`Percent body fat`) and features(`the rest of the body measurements`)*(Fig.16)*.

```
# splitting the data into features and target

X = standardized.iloc[:, 1:].values
y = standardized.iloc [:, 0].values
```
✓ 0.4s

Fig.16

Then we prepare a list of possible alphas(λ) values in range `0` to `50` to try in our ridge regression model and split the data in 5 with `KFold`. We also set `shuffle=True` in order to get a better estimate from varying the data in the folds*(Fin.17)*.

```
alphas = np.arange(0, 50.5, 0.5) # creating a range of alpha(lambda) values to try

fold = KFold(n_splits=5, shuffle=True) # initiate KFold with shuffling of the data for a better estimation
```
✓ 0.6s

Fig.17

*Note: because of the shuffle, best alpha(λ) varies with the varying training/testing set but primarily stays in the range `1` to `5`.

Next, we perform cross validation. For each alpha we train a Ridge regression model with that alpha, and then test it. The 5 (from 5 folds) tests are then scored and their mean saved alongside the corresponding alpha(λ) we used. Finally, we find the highest mean score and its corresponding alpha and display them*(Fig.17)*. That is how we determine the best complexity parameter.

```
# finding the best value for alpha(lambda)

mean_scores_list = []

for i in alphas:
    ridge_model = Ridge(alpha = i) # inintiating a Ridge model
    li = []

    # below split code templated from example (Bronshtein, 2022) cited in report
    for train_index, test_index in fold.split(X): # splitting the data for training and testing
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

        ridge_model.fit(X_train, y_train) # fitting the model to the training data
        score = ridge_model.score(X_test, y_test) # scoring each test

        li.append(score) # save the list of scores to li

    mean_scores_list.append([np.mean(li), i]) # find the mean of scores for each alpha value and save to list


best_score_and_alpha = max(mean_scores_list) # storing the highest score and corresponding alpha from all runs

print("[Best mean score, Best alpha]")
print(best_score_and_alpha)
✓ 0.4s
[Best mean score, Best alpha]
[0.693353365261926, 1.5]
```

Fig.17 Code template from (Bronshtein, 2022)

We can also show the optimal alpha(λ) by graphing the scores and their corresponding alpha(λ)*(Fig.18)*. We can confirm that by looking at the graph on *Fig.19*. λ = 1.5 has the highest score of 0.6933...

```
# plotting each score to its corresponding alpha(lambda) value

plt.plot([x[1] for x in mean_scores_list],[np.mean(x[0]) for x in mean_scores_list])
plt.ylabel("Score")
plt.xlabel("Alpha(Lambda)")
✓ 0.2s
```

Fig.18

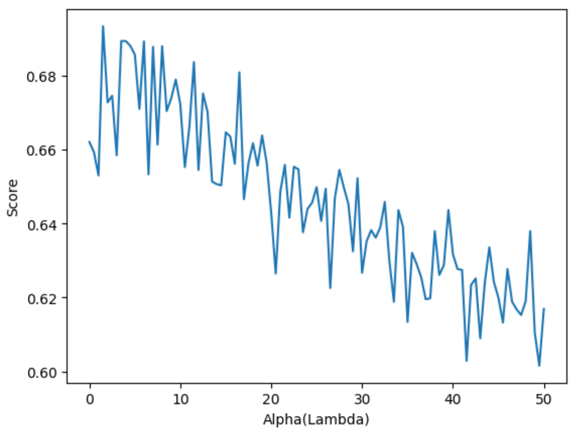

Fig.19

Now that we have chosen the best hyper-parameter λ, we apply the model for regression purposes. In order to confirm that the model we have trained can predict `Percent body fat`, we test that on the non-standardized data*(Fig.20)* and then plot the predictions against the test values*(Fig.21)*.

```
# splitting the non-standardized data for the purposes of testing the model
Xtesting = data.iloc[:, 1:].values
ytesting = data.iloc[:, 0].values

# making a train/test split
X_train, X_test, y_train, y_test = train_test_split(Xtesting, ytesting, test_size=0.2)

# using the selected best alpha(lambda) and fitting the model to the data
ridge = Ridge(alpha=best_score_and_alpha[1]).fit(X_train, y_train)

# plotting predictions against test values
prediction = ridge.predict(X_test)
print(ridge.score(X_test, y_test))
plt.scatter(prediction, y_test)
plt.ylabel("Test value")
plt.xlabel("Prediction")
```
✓ 0.1s

```
0.7686264063990945
```

Fig.20

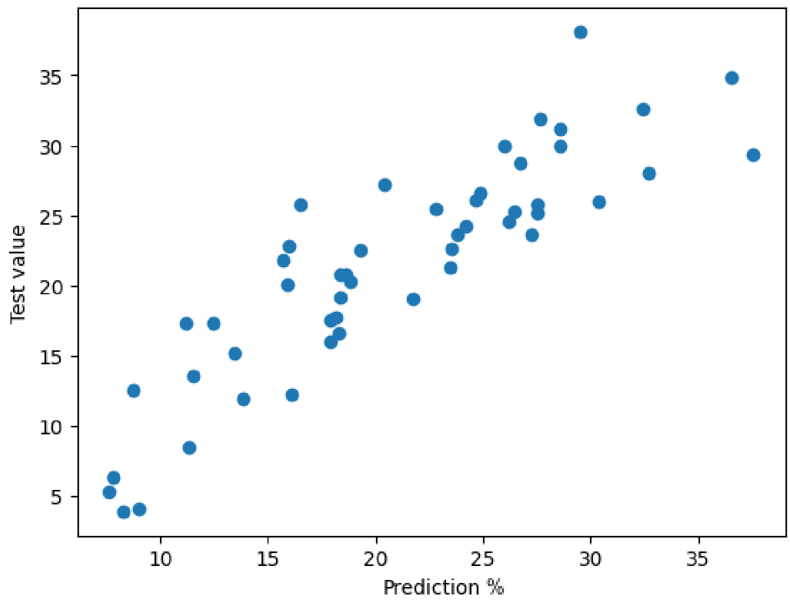

Fig.21

In conclusion, using Ridge regression with cross validation is a viable method of predicting body fat percentage in men and can be determined using only a scale and tape measure.

References

Bronshtein, A. (2022) Train/test split and cross validation in Python, Towards Data Science. Available at: https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6 (Accessed: November 10, 2022).

Sklearn (2022) Sklearn.model_selection.Kfold, scikit. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html (Accessed: November 10, 2022).

Sklearn (2022) Sklearn.linear_model.Ridge, scikit. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html#sklearn.linear_model.Ridge (Accessed: November 10, 2022).

Sklearn (2022) Sklearn.model_selection.train_test_split, scikit. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html#sklearn.model_selection.train_test_split (Accessed: November 10, 2022).

Sklearn (2022) Sklearn.utils.shuffle, scikit. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.utils.shuffle.html (Accessed: November 10, 2022).