```
import sys

import numpy as np
import sklearn
import pandas as pd
import matplotlib.pyplot as plt
from tabulate import tabulate
from scipy.stats import pearsonr

from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn import linear_model


from google.colab import files
data_file = files.upload()
```

Choose Files SAheart.data.csv

- **SAheart.data.csv**(text/csv) - 25106 bytes, last modified: 10/3/2022 - 100% done

```
Saving SAheart.data.csv to SAheart.data (1).csv
```

```
import io
data = pd.read_csv(io.BytesIO(data_file['SAheart.data.csv']), index_col=0)
```

```
data.head()
```

row.names	sbp	tobacco	ldl	adiposity	famhist	typea	obesity	alcohol	age	chd
1	160	12.00	5.73	23.11	Present	49	25.30	97.20	52	1
2	144	0.01	4.41	28.61	Absent	55	28.87	2.06	63	1
3	118	0.08	3.48	32.28	Present	52	29.14	3.81	46	0
4	170	7.50	6.41	38.03	Present	51	31.99	24.26	58	1
5	134	13.60	3.50	27.78	Present	60	25.99	57.34	49	1

```
data["famhist"] = data["famhist"] == "Present"
data["famhist"]=data["famhist"].astype(int)
```

```
data.corr()
```

	sbp	tobacco	ldl	adiposity	famhist	typea	obesity	alcoh
sbp	1.000000	0.212247	0.158296	0.356500	0.085645	-0.057454	0.238067	0.1400
tobacco	0.212247	1.000000	0.158905	0.286640	0.088601	-0.014608	0.124529	0.2008
ldl	0.158296	0.158905	1.000000	0.440432	0.161353	0.044048	0.330506	-0.0334
adiposity	0.356500	0.286640	0.440432	1.000000	0.181721	-0.043144	0.716556	0.1003
famhist	0.085645	0.088601	0.161353	0.181721	1.000000	0.044809	0.115595	0.0805
typea	-0.057454	-0.014608	0.044048	-0.043144	0.044809	1.000000	0.074006	0.0394
obesity	0.238067	0.124529	0.330506	0.716556	0.115595	0.074006	1.000000	0.0516
alcohol	0.140096	0.200813	-0.033403	0.100330	0.080520	0.039498	0.051620	1.0000

```
#features listed in order of correlation with label (highest to lowest)
features_corr = data[['age',·'tobacco',·'ldl',·'adiposity',·'sbp',·'typea',·'obesity',·'alcoh
features_corr.head()
```

row.names	age	tobacco	ldl	adiposity	sbp	typea	obesity	alcohol
1	52	12.00	5.73	23.11	160	49	25.30	97.20
2	63	0.01	4.41	28.61	144	55	28.87	2.06
3	46	0.08	3.48	32.28	118	52	29.14	3.81
4	58	7.50	6.41	38.03	170	51	31.99	24.26
5	49	13.60	3.50	27.78	134	60	25.99	57.34

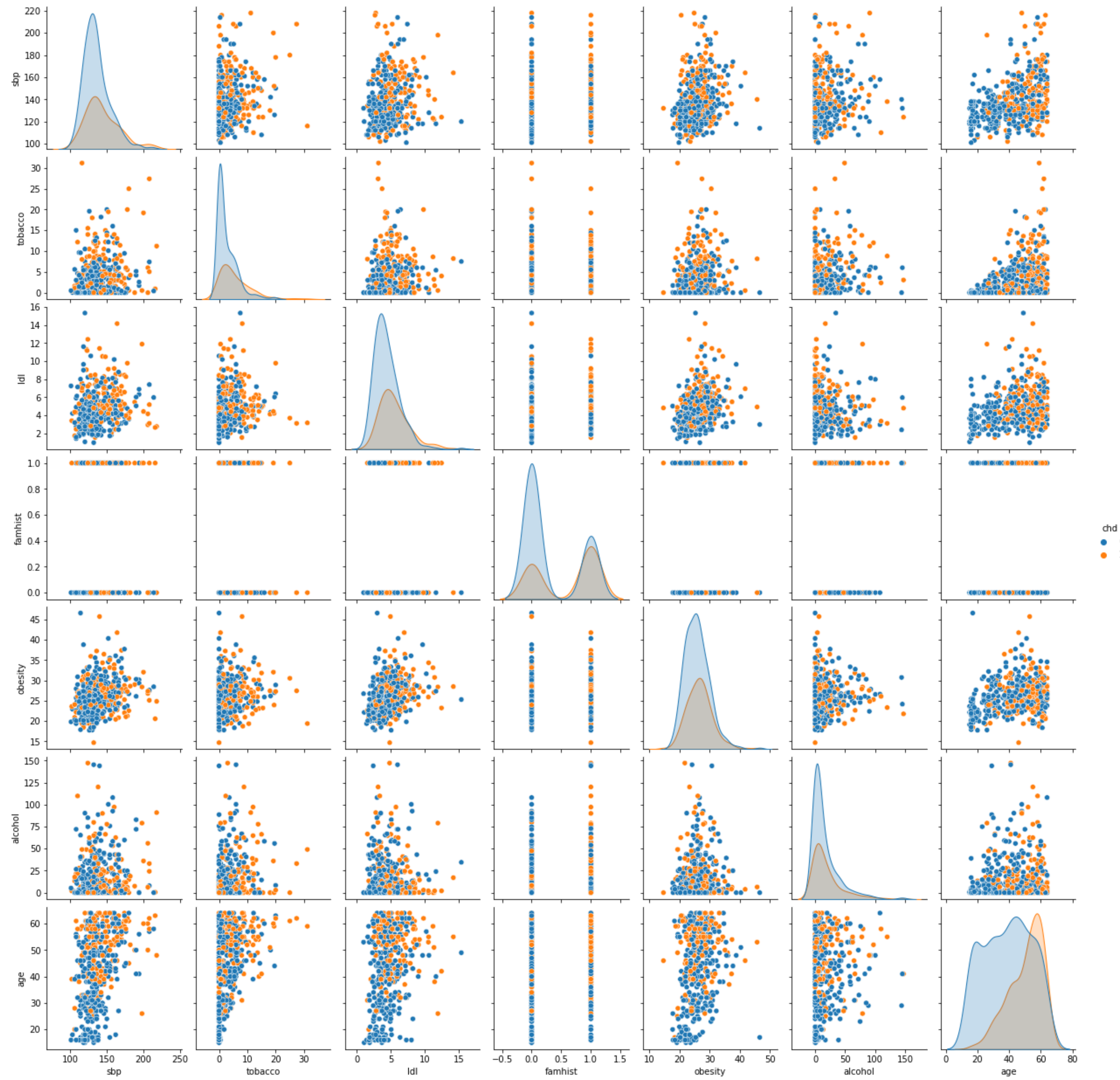
```
# Import data, convert to numpy arrays, and preprocess string ground truth to ints
feature_names = ['Intercept'] + [d for d in data.columns if d != 'chd' and d != 'famhist']

train_data = np.concatenate((np.ones((data.shape[0],1)), data[list(col for col in data.column
test_data = data['chd'].to_numpy().reshape((len(data),1))


# Split data
x_train, x_test, y_train, y_test= train_test_split(train_data, test_data, test_size=0.2, rand
x_val, x_test, y_val, y_test = train_test_split(x_test, y_test, test_size = 0.5, random_state


# Normalize train data
def normalize(x, mean, std):
  for i in range(1, x.shape[1]):
    #x[:,i] = (x[:,i] - np.mean(x_train[:,i])) / (np.std(x_train[:,i]) + 1e-5)
    x[:,i] = (x[:,i] - mean[i]) / (std[i] + 1e-5)
  return x
```

```
def count_correct_predictions(y_hat, y_test):
  return sum(y_hat.T == y_test)


def sigmoid(z):
    return 1 / (1 + np.exp(-z))


def mse_error(y, y_hat):
  return mean_squared_error(y_hat, y.T)



def get_minibatch(x, y, batchsize):
  num_batches = x.shape[0] // batchsize

  for i in range(num_batches):
    # draw random numbers from 0 to the number of data
    indx = np.random.randint(0, x.shape[0], batchsize)
    yield (x[indx,:].reshape(batchsize, -1), y[indx,:].reshape(batchsize, 1))


# Normalize train, validation, and test features
x_train_mean = np.zeros(x_train.shape[1])
x_train_std = np.zeros(x_train.shape[1])
for i in range(1, x_train.shape[1]):
    x_train_mean[i] = np.mean(x_train[:,i])
    x_train_std[i] = np.std(x_train[:,i])

x_train = normalize(x_train, x_train_mean, x_train_std)
x_val = normalize(x_val, x_train_mean, x_train_std)
x_test = normalize(x_test, x_train_mean, x_train_std)
```

Figure 4.12

```
import seaborn as sns

sns.pairplot(data[['sbp', 'tobacco', 'ldl', 'famhist', 'obesity', 'alcohol', 'age', 'chd']],
```

<seaborn.axisgrid.PairGrid at 0x7f2848e77150>



1. Unregularized Model

```
def initialize_weights(size):
  return np.zeros((1, size)), 0


def model_optimize(w, X, Y):
    # number of training data
    m = X.shape[0]

    #Prediction
    final_result = sigmoid(np.matmul(w, X.T))

    Y_T = Y.T
    #eqn 4.20
    log_likelihood = np.sum(Y_T*np.log(final_result) + (1-Y_T)*(np.log(1-final_result)))
    #

    #Gradient calculation
    dw = (Y - final_result) * X

    grads = {"dw": dw}

    return grads, log_likelihood


def model_predict(w, x_train, y_train, learning_rate, no_iterations, batchsize):
    log_likelihoods = []
    for i in range(no_iterations):
        # SGD
        for x_batch, y_batch in get_minibatch(x_train, y_train, batchsize):
          #
          grads, log_likelihood = model_optimize(w, x_batch, y_batch)
          #
          dw = grads["dw"]

          #weight update
          w = w + learning_rate * (dw)
          #

        log_likelihoods.append(log_likelihood)

    #final parameters
    coeff = {"w": w}
    gradient = {"dw": dw}
```

```
def predict(final_pred, m):
    y_pred = np.zeros((1,m))
    for i in range(final_pred.shape[1]):
        if final_pred[0][i] > 0.5:
            y_pred[0][i] = 1
        else:
            y_pred[0][i] = 0
    return y_pred


no_iterations = 100
batchsize = 1
learning_rate = 0.00001


w, b = initialize_weights(x_train.shape[1])


# Fit training data
coeff, gradient, log_likelihoods = model_predict(w, x_train, y_train, learning_rate, no_itera
y_hat = predict(sigmoid(np.dot(coeff['w'], x_test.T)), x_test.shape[0])


# mse = mse_error(y_hat, y_test)
num_correct_plain = count_correct_predictions(y_hat, y_test)[0]
print(f'Number of correct predictions: {num_correct_plain}/{y_test.shape[0]}')
print(f'Percent correct: {num_correct_plain / y_test.shape[0]}')
```

```
Number of correct predictions: 38/47
Percent correct: 0.8085106382978723
```

2. Stepwise Model

```
#stepwise forward selection function to select best features for prediction
def forward_selection(x_train, y_train, x_val, y_val, learning_rate, no_iterations):
    max_num_correct = 0
    feature_dict = {'alcohol': 7, 'obesity': 6, 'typea': 5, 'sbp': 1, 'adiposity': 4, 'ldl':

    selected_features = []
    selected_features_indices = []
    # coeff_dict = {"Feature": [], "Coefficients": []}

    for item in features_corr.columns:
        w = np.zeros((1, len(selected_features)+2))
        print(selected_features)

        if item in selected_features:
            continue
```

```
        print("Trying feature", item)
        key = feature_dict[item]

        print("Current feautrues: ", selected_features_indices + [key])
        features = [0] + selected_features_indices + [key]
        print("1: ", w.shape)
        print("2: ", len(features))
        coeff, gradient, log_likelihood = model_predict(w, x_train[:, features], y_train, lea
        w_opt = coeff['w']

        # Compute number of correct predictions on validation set
        y_hat = predict(sigmoid(np.dot(w_opt, x_val[:, features].T)), x_val.shape[0])
        num_correct = count_correct_predictions(y_hat, y_val)[0]
        print(num_correct)

        # Store optimal weights
        if num_correct >= max_num_correct:
          max_num_correct = num_correct
          selected_features.append(item)
          selected_features_indices.append(key)
          W = w_opt

          print(w_opt)
    return W, selected_features

W, selected_features = forward_selection(x_train, y_train, x_val, y_val, .0001, 100)
```

```
[]
Trying feature age
Current feautrues:  [8]
1:  (1, 2)
2:  2
29
[[-0.38662991  0.44173369]]
['age']
Trying feature tobacco
Current feautrues:  [8, 2]
1:  (1, 3)
2:  3
31
[[-0.36750811  0.39378581  0.32344194]]
['age', 'tobacco']
Trying feature ldl
Current feautrues:  [8, 2, 3]
1:  (1, 4)
2:  4
30
['age', 'tobacco']
Trying feature adiposity
Current feautrues:  [8, 2, 4]
1:  (1, 4)
2:  4
30
```

```
['age', 'tobacco']
Trying feature sbp
Current feautrues:  [8, 2, 1]
1:  (1, 4)
2:  4
30
['age', 'tobacco']
Trying feature typea
Current feautrues:  [8, 2, 5]
1:  (1, 4)
2:  4
28
['age', 'tobacco']
Trying feature obesity
Current feautrues:  [8, 2, 6]
1:  (1, 4)
2:  4
30
['age', 'tobacco']
Trying feature alcohol
Current feautrues:  [8, 2, 7]
1:  (1, 4)
2:  4
31
[[-0.37777937  0.37974465  0.31780951  0.02413945]]
```

```
feature_dict = {'alcohol': 7, 'obesity': 6, 'typea': 5, 'sbp': 1, 'adiposity': 4, 'ldl': 3, '
indices = [0]
for i in selected_features:
  value = feature_dict[i]
  indices.append(value)
x_best = x_test[:, indices] #extract best feature columns
y_pred = predict(sigmoid(np.dot(W, x_best.T)), x_best.shape[0])


num_correct_stepwise = count_correct_predictions(y_pred, y_test)[0]
print(f"Number of correct predictions: {num_correct_stepwise}/{y_test.shape[0]}")
print(f'Percent correct: {num_correct_stepwise / y_test.shape[0]}')
```

```
Number of correct predictions: 33/47
Percent correct: 0.7021276595744681
```

3. L2 Regularized

```
def model_optimize_l2norm(w, X, Y, lamb):
    # number of training data
    m = X.shape[0]

    #Prediction
    final_result = sigmoid(np.matmul(w, X.T))
    Y_T = Y.T
    # cost = (-1/m)*(np.sum((Y_T*np.log(final_result)) + ((1-Y_T)*(np.log(1-final_result))))
```

```
    log_likelihood = np.sum(Y_T*np.log(final_result) + (1-Y_T)*(np.log(1-final_result))) + (l
    #

    #Gradient calculation
    dw = (Y - final_result) * X

    grads = {"dw": dw}

    return grads, log_likelihood


def model_predict_l2norm(w, x_train, y_train, x_val, y_val, learning_rate, no_iterations, lam
    max_percent_correct = 0
    W = 0
    gradient = {"dw": 0}
    opt_lambda = 0

    # Loop through lambda to find optimal lambda for l2 penalty
    for lamb in lambdas:
      for i in range(no_iterations):
          #
          # SGD
          for x_batch, y_batch in get_minibatch(x_train, y_train, batchsize):
            grads, log_likelihood = model_optimize_l2norm(w, x_batch, y_batch, lamb)
            #
            dw = grads["dw"]

            # gradient ascent
            w = w + (learning_rate * (dw)) - lamb*w
            #

          # select lambda that gives the highest percent correct
          y_hat = predict(sigmoid(np.dot(w, x_val.T)), x_val.shape[0])
          num_correct = count_correct_predictions(y_hat, y_val)[0]

          if num_correct > max_percent_correct:
            #final parameters
            W = w
            gradient = {"dw": dw}
            opt_lambda = lamb
            max_percent_correct = num_correct

    return W, gradient, opt_lambda


def predict(final_pred, m):
    y_pred = np.zeros((1,m))
    for i in range(final_pred.shape[1]):
        if final_pred[0][i] > 0.5:
            y_pred[0][i] = 1
        else:
```

```
            y_pred[0][i] = 0
    return y_pred

# Set hyperparameters for logistic regression with L2Norm
no_iterations = 100
learning_rate = 0.0001
batchsize = 1
lambdas = [1e-5, 1e-4, 0.001, 0.01, 0.1, 1]
# lambdas = [1e-4, 1e-3]


# Initialize weight and bias to zeros.
# Initialize using other distribution might help.
w, b = initialize_weights(x_train.shape[1])


# Fit training data
W, gradient, min_lambda = model_predict_l2norm(w, x_train, y_train, x_val, y_val, learning_ra


y_hat = predict(sigmoid(np.dot(W, x_test.T)), x_test.shape[0])


# Report result
num_correct_l2 = count_correct_predictions(y_hat, y_test)[0]
print(f'Number of correct predictions: {num_correct_l2}/{y_test.shape[0]}')
print(f'Percent correct: {num_correct_l2 / y_test.shape[0]}')
```

```
Number of correct predictions: 34/47
Percent correct: 0.723404255319149
```

```
# Report results in table
from tabulate import tabulate
print("Result of classifying SA Heart data")
table = [['Model', '% Correct (%)'],
         ['Plain', num_correct_plain / y_test.shape[0]],
         ['L2 Reg', num_correct_l2 / y_test.shape[0]],
         ['Stepwise', num_correct_stepwise / y_test.shape[0]]]

print(tabulate(table, headers='firstrow'))
```

```
Result of classifying SA Heart data
Model       % Correct (%)
--------  ---------------
Plain            0.808511
L2 Reg           0.723404
Stepwise         0.702128
```

Using Wisconsin Breast Cancer Data

```
data2 = files.upload()
```

Choose Files data.csv

- **data.csv**(text/csv) - 125204 bytes, last modified: 10/10/2022 - 100% done

```
Saving data.csv to data.csv
```

```
data2_pd = pd.read_csv(io.BytesIO(data2['data.csv']))
```

```
data2_pd = pd.read_csv('/content/data.csv', header = 0)
```

```
data2_pd.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothnes
0	842302	M	17.99	10.38	122.80	1001.0	(
1	842517	M	20.57	17.77	132.90	1326.0	(
2	84300903	M	19.69	21.25	130.00	1203.0	(
3	84348301	M	11.42	20.38	77.58	386.1	(
4	84358402	M	20.29	14.34	135.10	1297.0	(

5 rows × 33 columns

```
#convert diagnosis to binary
data2_pd.diagnosis[data2_pd.diagnosis == 'M'] = 0
data2_pd.diagnosis[data2_pd.diagnosis == 'B'] = 1
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  This is separate from the ipykernel package so we can avoid doing imports until
```

```
data2_pd = data2_pd.drop(['Unnamed: 32', 'id'], axis=1)
data2_pd.head()
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	co
0	0	17.99	10.38	122.80	1001.0	0.11840	
1	0	20.57	17.77	132.90	1326.0	0.08474	
2	0	19.69	21.25	130.00	1203.0	0.10960	
3	0	11.42	20.38	77.58	386.1	0.14250	
4	0	20.29	14.34	135.10	1297.0	0.10030	

5 rows × 31 columns

```
#change diagnosis column to int64 datatype
data2_pd['diagnosis'] = data2_pd['diagnosis'].apply(pd.to_numeric)
# print(data2_pd.dtypes)


#get correlation between features and target
corr_dict = {}
keys = data2_pd.columns

for k in keys:
  if k == 'diagnosis':
    continue
  corr = data2_pd[k].corr(data2_pd['diagnosis'])
  corr_dict[k] = corr
print(corr_dict)
```

```
{'radius_mean': -0.7300285113754558, 'texture_mean': -0.4151852998452037, 'perimeter_mea
```

```
#order from highest to lowest correlation
corr_dict = dict(sorted(corr_dict.items(), key=lambda item: item[1], reverse = True))
# print(corr_dict)


data2_train = data2_pd[list(col for col in data2_pd.columns if col != "Unnamed: 32" and col !
data2_train.dropna()
data2_train = data2_train.to_numpy()

data2_test = data2_pd['diagnosis'].to_numpy()
data2_test = data2_test.reshape(len(data2_test), 1)


# Split data
x_train2, x_test2, y_train2, y_test2 = train_test_split(data2_train, data2_test, test_size=0.
x_val2, x_test2, y_val2, y_test2 = train_test_split(x_test2, y_test2, test_size = 0.5, random
```

```
# Normalize train, validation, and test features
x_train2_mean = np.zeros(x_train2.shape[1])
x_train2_std = np.zeros(x_train2.shape[1])
for i in range(1, x_train2.shape[1]):
    x_train2_mean[i] = np.mean(x_train2[:,i])
    x_train2_std[i] = np.std(x_train2[:,i])

x_train2 = normalize(x_train2, x_train2_mean, x_train2_std)
x_val2 = normalize(x_val2, x_train2_mean, x_train2_std)
x_test2 = normalize(x_test2, x_train2_mean, x_train2_std)
```

1. Unregularized

```
no_iterations = 100
batchsize = 1
learning_rate = 0.00001

w, b = initialize_weights(x_train2.shape[1])

# Fit training data
coeff, gradient, log_likelihoods = model_predict(w, x_train2, y_train2, learning_rate, no_ite
y_hat = predict(sigmoid(np.dot(coeff['w'], x_test2.T)), x_test2.shape[0])

# mse = mse_error(y_hat, y_test)
num_correct_plain2 = count_correct_predictions(y_hat, y_test2)[0]
print(f'Number of correct predictions: {num_correct_plain2}/{y_test2.shape[0]}')
print(f'Percent correct: {num_correct_plain2 / y_test2.shape[0]}')
```

```
Number of correct predictions: 54/57
Percent correct: 0.9473684210526315
```

2. Stepwise Model

```
indices = {c: i for i, c in enumerate(data2_pd.columns)}
del indices["diagnosis"]
# print(indices)


for i in indices.keys():
  value = indices[i] - 1
  indices[i] = value

# print(indices)


def forward_selection2(x_train, y_train, x_val, y_val, learning_rate, no_iterations):
    n_features = x_train2.shape[1]
```

```
    max_num_correct = 0

    selected_features = []
    selected_features_indices = []

    for item in corr_dict.keys():
        w = np.zeros((1, len(selected_features)+2))
        if item in selected_features:
            continue
        print("Trying feature", item)
        value = indices[item]
        features = [0] + selected_features_indices + [value]

        coeff, gradient, cost = model_predict(w, x_train[:, features], y_train, learning_rate
        w_opt = coeff['w']

        # Compute number of correct predictions on validation set
        y_hat = predict(sigmoid(np.dot(w_opt, x_val[:, features].T)), x_val.shape[0])
        num_correct = count_correct_predictions(y_hat, y_val)[0]

        if num_correct > max_num_correct:
          max_num_correct = num_correct
          selected_features.append(item)
          selected_features_indices.append(value)
          W = w_opt

    return W, selected_features, selected_features_indices

W, selected_features, selected_features_indices = forward_selection2(x_train2, y_train2, x_va
```

```
Trying feature smoothness_se
Trying feature fractal_dimension_mean
Trying feature texture_se
Trying feature symmetry_se
Trying feature fractal_dimension_se
Trying feature concavity_se
Trying feature compactness_se
Trying feature fractal_dimension_worst
Trying feature symmetry_mean
Trying feature smoothness_mean
Trying feature concave points_se
Trying feature texture_mean
Trying feature symmetry_worst
Trying feature smoothness_worst
Trying feature texture_worst
Trying feature area_se
Trying feature perimeter_se
Trying feature radius_se
Trying feature compactness_worst
Trying feature compactness_mean
Trying feature concavity_worst
Trying feature concavity_mean
Trying feature area_mean
```

```
Trying feature radius_mean
Trying feature area_worst
Trying feature perimeter_mean
Trying feature radius_worst
Trying feature concave points_mean
Trying feature perimeter_worst
Trying feature concave points_worst
```

```
indices_best = [0] + selected_features_indices

x_best = x_test2[:, indices_best] #extract best feature columns
y_pred = predict(sigmoid(np.dot(W, x_best.T)), x_best.shape[0])


# Report result
num_correct_stepwise2 = count_correct_predictions(y_pred, y_test2)[0]
print(f'Number of correct predictions:",  {count_correct_predictions(y_pred, y_test2)[0]}/{y_
print(f'Percent correct predictions:,  {count_correct_predictions(y_pred, y_test2)[0]/y_test2
```

```
Number of correct predictions:",  53/57
Percent correct predictions:,  0.9298245614035088
```

3. L2 Regularized

```
# Set hyperparameters for logistic regression with L2Norm
no_iterations = 100
learning_rate = 0.001
lamb = 0.001
batchsize = 1
lambdas = [1e-5, 1e-4, 0.001, 0.01, 0.1, 1]


# Initialize weight and bias to zeros.
# Initialize using other distribution might help.
w, b = initialize_weights(x_train2.shape[1])


# Fit training data
W, gradient, min_lambda = model_predict_l2norm(w, x_train2, y_train2, x_val2, y_val2, learnin



y_hat2 = predict(sigmoid(np.dot(W, x_test2.T)), x_test2.shape[0])


num_correct_l22 = count_correct_predictions(y_hat2, y_test2)[0]
print(f'Number of correct predictions: {num_correct_l22}/{y_test2.shape[0]}')
print(f'Number of correct predictions: {num_correct_l22 / y_test2.shape[0]}')
```

```
Number of correct predictions: 54/57
Number of correct predictions: 0.9473684210526315
```

```
# Report results in table
from tabulate import tabulate
print("Resuls of classifying breast cancer data")
table = [['Model', '% Correct (%)'],
         ['Plain', num_correct_plain2 / y_test2.shape[0]],
         ['L2 Reg', num_correct_l22 / y_test2.shape[0]],
         ['Stepwise', num_correct_stepwise2 / y_test2.shape[0]]]

print(tabulate(table, headers='firstrow'))
```

```
Resuls of classifying breast cancer data
Model       % Correct (%)
--------  ---------------
Plain            0.947368
L2 Reg           0.947368
Stepwise         0.929825
```

Stretch Goal 1: Implement L1 Regularization

```
indices = np.random.permutation(len(x_train))


'''
Reference :
Stochastic Gradient Descent Training for L1-regularized Log-linear Models with Cumulative Pen
Yoshimasa Tsuruoka, Junichi Tsujii, Sophia Ananiadou†
'''

def train_l1_reg(x, y, x_val, y_val, iterations, lambdas, learning_rate):
  min_lambda = -1
  max_num_correct = 0
  num_features = x.shape[1]
  num_data = x.shape[0]
  W = np.zeros((1, x.shape[1]))

  # Initialize q (L1 penalty actually received)
  u = 0
  # q = np.zeros((1, num_features))[0]
  q = 0
  for lamb in lambdas:
    w = np.array([1e-5] * x.shape[1])
    for i in range(iterations):
      # total L1 penalty could have received until interation i
      u = u + learning_rate * lamb/x.shape[0]

      x_single = x[indices[i], :]
      y_single = y[indices[i], :]
      w, q = update_weights(x_single, y_single, w, u, q)
```

```
    # Compute number of correct predictions on validation set
    y_hat = predict(sigmoid(np.dot(w.reshape(1, len(w)), x_val.T)), x_val.shape[0])
    num_correct = count_correct_predictions(y_hat, y_val)[0]
    #print("num correct: ", num_correct)

    if num_correct >= max_num_correct:
      W = w
      max_num_correct = num_correct
      min_lambda = lamb

  return W, min_lambda



def update_weights(x, y, w, u, q):
  for i in range(len(w)):
    if w[i] == 0:
      continue

    else:
      # Update weight using gradient descent
      dw = (y - np.dot(w, x.T)) * x[i]
      w[i] = w[i] + learning_rate * dw

      temp = w[i]

      # Implement SGD-L1 (Cumulative)
      w[i] = max(0, w[i] - (u + q)) + min(0, w[i] + (u - q))

      q = q + (w[i] - temp)

  return w, q


# Initialize hyperparameters
iterations = 200
lambdas = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1]
# lambdas = [0.001]
learning_rate = 0.001


W, min_lambda = train_l1_reg(x_train, y_train, x_val, y_val, iterations, lambdas, learning_ra


print(feature_names)
feature_coeffs = {}
for i in range(len(W)):
  feature_coeffs[feature_names[i]] = W[i]

print(dict(sorted(feature_coeffs.items(), key=lambda item: item[1], reverse=True)))
```

```
['Intercept', 'sbp', 'tobacco', 'ldl', 'adiposity', 'typea', 'obesity', 'alcohol', 'age
{'tobacco': 0.03174083252850607, 'ldl': 0.02644000966519652, 'typea': 0.012898784532879
```

```
y_hat_bin = predict(sigmoid(np.dot(W.reshape(1,len(W)), x_test.T)), x_test.shape[0])
print(f'Number of correct predictions: {count_correct_predictions(y_hat_bin, y_test)[0]}/{y_t
print(f'Percent correct predictions: {count_correct_predictions(y_hat_bin, y_test)[0] / y_tes
```

```
Number of correct predictions: 34/47
Percent correct predictions: 0.723404255319149
```

The top three significant features collected from L1 SGD were sbp, tobacco, and ldl. In stepwise, tobacco and age showed to be the most significant. The two models both showed that tobacco is an important factor in determining heart disease. L1 didn't select ldl to be important but this can be due to small data and the fact that we are using correlation to order significance in stepwise. However, the two models resulted in reasonable performance and their results generally agree.

Stretch Goal 2: Multinomial Regression

```
from google.colab import files
data_file = files.upload()
```

Choose Files iris.data

- **iris.data**(n/a) - 4551 bytes, last modified: 10/8/2022 - 100% done

```
Saving iris.data to iris.data
```

```
header=['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class']
data = pd.read_csv(io.BytesIO(data_file['iris.data']), names=header)
```

```
data.head()
```

	sepal_length	sepal_width	petal_length	petal_width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
data.describe()
```

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000

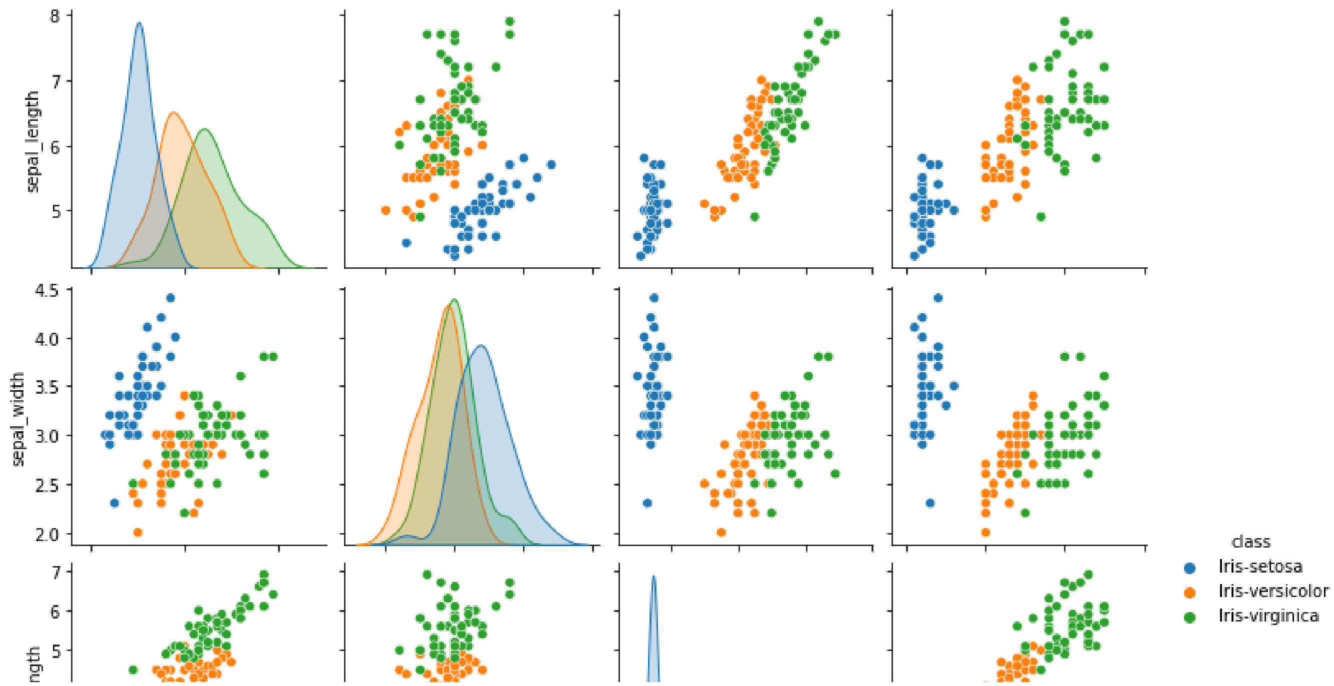
```
data.corr()
```

	sepal_length	sepal_width	petal_length	petal_width
sepal_length	1.000000	-0.109369	0.871754	0.817954
sepal_width	-0.109369	1.000000	-0.420516	-0.356544
petal_length	0.871754	-0.420516	1.000000	0.962757
petal_width	0.817954	-0.356544	0.962757	1.000000

Figure 4.12

```
import seaborn as sns

sns.pairplot(data[['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class']], h
```

```
<seaborn.axisgrid.PairGrid at 0x7f28aa811410>
```




```
from tensorflow.keras.utils import to_categorical

data = data.replace(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], [0, 1, 2])

input = np.concatenate((np.ones((data.shape[0],1)), data[list(col for col in data.columns if
output =  data['class'].to_numpy().reshape((len(data),1))


def one_hot(output):
  one_hot_output = to_categorical(output, num_classes=3)
  return one_hot_output


X_train, X_test, y_train, y_test = train_test_split(input, output, test_size=0.2, random_stat


# Normalize train data
def normalize(x, mean, std):
  for i in range(1, x.shape[1]):
    x[:,i] = (x[:,i] - mean[i]) / (std[i] + 1e-5)
  return x


# Normalize train, validation, and test features
X_train_mean = np.zeros(X_train.shape[1])
X_train_std = np.zeros(X_train.shape[1])
for i in range(1, X_train.shape[1]):
    X_train_mean[i] = np.mean(X_train[:,i])
    X_train_std[i] = np.std(X_train[:,i])
```

```
X_train = normalize(X_train, X_train_mean, X_train_std)
#X_val = normalize(X_val, X_train_mean, X_train_std)
X_test = normalize(X_test, X_train_mean, X_train_std)


actual_output = y_test
y_train = one_hot(y_train)
y_test = one_hot(y_test)


no_iterations = 1000000
learning_rate = 0.00001


w = np.zeros((3, X_train.shape[1]))


def multi_model_optimize(w, X, Y):
    # number of training data
    m = X.shape[0]

    #Prediction
    final_result = sigmoid(np.matmul(w, X.T))
    """
    print("final result shape: ", np.shape(final_result))
    print("Y.T shape: ", np.shape(Y.T))
    print("X shape: ", np.shape(X))
    """
    Y_T = Y.T
    #eqn 4.20
    log_likelihood = np.sum(Y_T*np.log(final_result) + (1-Y_T)*(np.log(1-final_result)))
    #

    #Gradient calculation
    dw = (Y.T - final_result) @ X

    grads = {"dw": dw}

    return grads, log_likelihood


def multi_model_predict(w, x_train, y_train, learning_rate, no_iterations):
    log_likelihoods = []
    for i in range(no_iterations):
        # SGD

        grads, log_likelihood = multi_model_optimize(w, x_train, y_train)
        #
        dw = grads["dw"]

        #weight update
        w = w + learning_rate * (dw)
```

```
        #

        log_likelihoods.append(log_likelihood)

    #final parameters
    coeff = {"w": w}
    gradient = {"dw": dw}

    return coeff, gradient, log_likelihoods

def multi_predict(final_pred, m):
    y_pred = np.zeros(np.shape(final_pred))
    for i in range(final_pred.shape[1]):
      y_pred[np.where(final_pred==np.max(final_pred[:,i]))] = 1
    return y_pred.T


# Fit training data
coeff, gradient, log_likelihoods = multi_model_predict(w, X_train, y_train, learning_rate, no
y_hat = multi_predict(sigmoid(np.dot(coeff['w'], X_test.T)), X_test.shape[0])


def multi_accuracy(actual, predicted):
  header = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
  print(f'Number of total correct predictions of: {sum(actual == predicted)}/{actual.shape[0]
  print(f'Percent correct: {sum(actual == predicted) / actual.shape[0]}%')
  for i in [0,1,2]:
    index = np.where(actual == i)
    num_correct = sum(actual[index] == predicted[index])
    total = actual[index].shape[0]
    print(f'Number of correct predictions of {header[i]}: {num_correct}/{total}')
    print(f'Percent correct: {num_correct / total}')



multi_accuracy(actual_output.reshape((np.shape(np.argmax(y_hat, axis=1)))), np.argmax(y_hat,
```

```
     Number of total correct predictions of: 29/30
     Percent correct: 0.9666666666666667%
     Number of correct predictions of Iris-setosa: 11/11
     Percent correct: 1.0
     Number of correct predictions of Iris-versicolor: 13/13
     Percent correct: 1.0
     Number of correct predictions of Iris-virginica: 5/6
     Percent correct: 0.8333333333333334
```