Mini-Project 1: Logistic Regression

Jack Wei
McGill University
yi.wei4@mail.mcgill.ca

Grey Yuan
McGill University
linwei.yuan@mail.mcgill.ca

Yu Wu
McGill University
yu.wu4@mail.mcgill.ca

Abstract

Logistic regression is a discriminative classifier that models the conditional probability distributions of classes with a logistic function and categorizes samples by a linear decision boundary. In this project, the logistic regression model was implemented to classify the Pima Indian Diabetes (PID) data set and Red Wine Quality (RWQ) data set. The decision boundary of the classifier was determined by the gradient descent algorithm. Second orders and square roots of features, as well as multiples of two features, were constructed to create models with higher complexity. A special case of wrapper feature selection methodology, Recursive Feature Elimination (RFE) was then implemented to select the optimal subset of features. The proposed models are selected based on the average 10-fold cross-validation accuracy. From experimentation, we have analyzed the effect of learning rates on model convergence. Moreover, by experimenting with RFE, we have discovered that simpler logistic regression models, ones with a smaller number of features, produce slightly better results and are significantly more efficient than higher complexity models regarding running time.

1 Introduction

Logistic regression is a machine learning model that classifies new samples based on a modeled probability. It is commonly used in many fields including sociology, biostatistics, clinical, quantitative psychology, econometrics and marketing[1], [2], [3]. In this project, the binary logistic regression, which is also called binomial logistic regression, has been used to predict whether patients have diabetes, and whether the quality of red wines are high. The binary logistic regression, as the name suggests, includes two possible values for the class label, which is normally labeled as 1 and 0. In this report, label 1 suggests the patient has diabetes and 0 without diabetes in the PID data set; this corresponds to high- and low-quality wine in the RWQ data set. Besides that, there are features which quantify the medical conditions of patients and the physicochemical features of red wines.

In this project, we developed a logistic regression model that minimizes the cross-entropy loss by gradient descent. The initial weight vector is arbitrarily set to zero. This, however, would not affect the model's performance as the cross-entropy loss function given below is convex, and therefore gradient descent is deterministic and the loss of the model converges to the global minimum[4].

$$\text{cross-entropy}(\mathbf{D}) = -\sum_{i=1}^{n} y_i \ln(\sigma(\mathbf{w^T x_i}) + (1 - y_i) \ln(1 - \sigma(\mathbf{w^T x_i}))$$

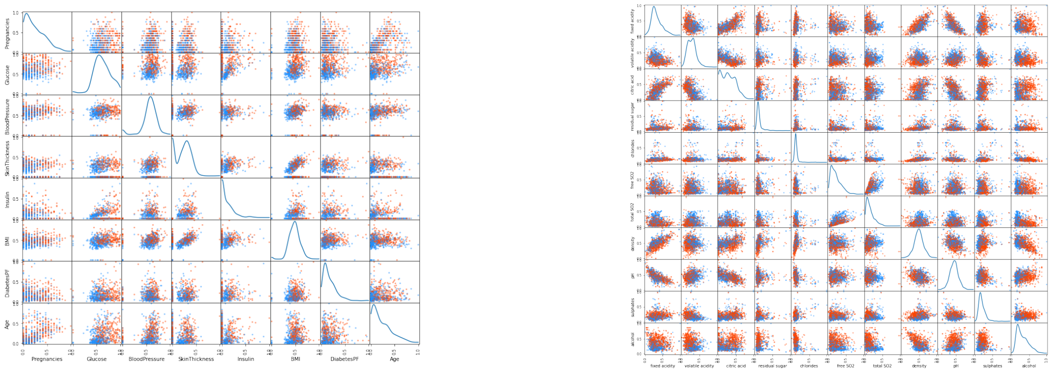
To increase our model complexity, we have expanded the basis of feature space by including second-order terms and square roots of features as well as interaction terms as multiples of two features. This

Preprint. Under review.

makes evaluation of potential models difficult since all possible subsets of features correspond to a new model. This problem motivated the development of various feature selection techniques[5]. As part of our project, we have implemented a Recursive Feature Elimination (RFE) algorithm to facilitate the selection of features[6]. RFE is a greedy algorithm that recursively eliminates features that were assigned the lowest absolute value in the weight vector. This immensely reduces the search space since instead of testing models from all elements of the power set of features, we just need to test m models, where m is the number of total features. To determine the optimal number of features to select, we cross-validated all m models by 10-fold cross-validation and choose the one with the best scores. In our case we use average validation accuracy and running time as our scoring metrics.

As part of our search for the optimal model, we investigated the impact of learning rate on the convergence of gradient descent. By the analysis in section 3.1, we have concluded that the optimal learning rate is $\frac{1}{n}$ where n is number of samples. Lower learning rate of $\frac{1}{100n}$ results in slower convergence whereas higher learning rate of 0.01 results converging to a higher loss. With the optimal learning rate and best set of features obtained from cross-validated RFE, we thus propose our models for the two data sets. Our final models include 7 out of 24 features and 6 out of 20 features respectively. This suggests that only a small subset of features is necessary to support the predictive capabilities of our models.

2 Datasets Analysis



(a) Scatter matrix of diabetes dataset

(b) Scatter matrix of red wine dataset

Figure 1: Data analysis of both datasets.

2.1 PID Data Set

2.1.1 Statistical Analysis

The PID data set consists of eight medical indicators as its attributes gathering from 600 female patients in an Indian hospital. According to those attributes, patients are divided into two categories: patients with diabetes (268 patients, class 1) and normal patients (332 patients, class 0). All the data in this data set has been normalized. To show a clearer illustration of the data distribution, a scatter matrix plot of the eight medical predictors is plotted in Figure 1(a), where orange corresponds to class 1 and blue corresponds to class 0. Along the diagonal, it is noticeable that not all the distributions within a single class follow a Gaussian distribution. The most obvious observation we can make that is: Insulin data are skewed to smaller values, making its distribution close to a logarithmic distribution. Similarly, another class, Skin Thickness, also exhibits a non-Gaussian distribution. It varies more than other features and shows a bi-modal distribution.

2.1.2 Feature Construction

As a measure to increase model complexity, we constructed a number of features. To avoid high dimensionality, we chose the features that make the most significant contribution to our prediction weights. Therefore, preliminary experiments were made to select the features that give a large absolute value of weights of the logistics regression model. As a result, Glucose, Age, BMI, Blood Pressure, DPF are considered as important indicators for the logistics regression model. Then, various non-linear combinations are applied to construct additional features for the model selection algorithm. For example, the operations chosen are: square, square root, and multiplication of two features. With operation like these, we constructed 16 different expanded features for the model selection algorithm. The features added are given in table 1.

Table 1: Features inserted in both red wine and diabetes datasets

red wine	$volatileacidity^2$	$freeSO_2^2$	pH^2	$sulphates^2$
	$pH * volatileacidity$	$pH * citricacid$	$sulphate * freeSO_2$	$fixed * density$
	$sqrt(volatileacidity)$			
diabetes	$glucose^2$	age^2	BMI^2	$pressure^2$
	$glucose * BMI$	$glucose * DPF$	$glucose * insulin$	$glucose * age$
	$sqrt(age)$	$sqrt(BMI)$	$sqrt(pressure)$	$BMI * pressure$
	$pressure * age$	$sqrt(glucose)$	$BMI * DPF$	$BMI * age$

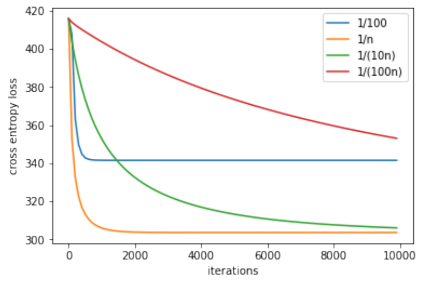
2.2 RWQ Data Set

The RWQ data Set consists of eleven physicochemical features as its attributes gathering from 1599 observations from red wines. According to those attributes, red wines are divided into two categories: high quality (855 red wines, class 1) and low quality (744 red wines, class 0). All the data in this data set has already been normalized. Similar to the PID data set, a scatter matrix plot of the eleven physicochemical features is plotted in Figure 1(b), where orange corresponds to class 1 and blue corresponds to class 0. For feature selection, we have carried out a similar experiment and expanded the feature set. The features added is shown in table 1.

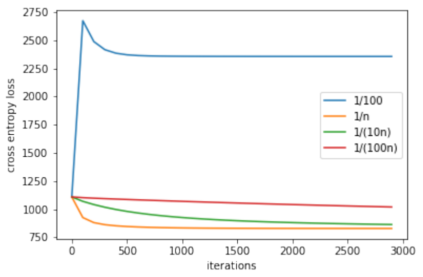
3 Result

3.1 Learning Rate Investigation

Learning rate is a hyper-parameter in gradient descent optimization determining the magnitude of steps toward a local minimum of the loss function. Due to the convexity of cross-entropy loss function, logistic regression always converges to a global minimum. We investigated the effect of learning rates, α, on the models' convergence for the two data sets. A set of values were investigated starting from 0.01 to $\frac{1}{100n}$ where n is the sample size. With the number of iterations fixed at 10000, the results obtained are illustrated in Fig.2.



(a) PID data set

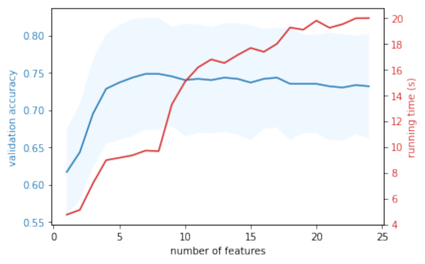


(b) Red Wine data set

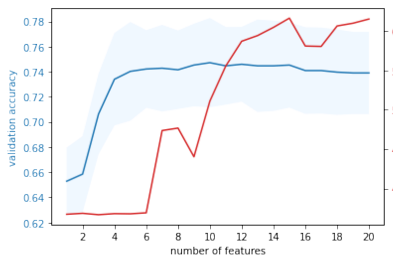
Figure 2: Impact of learning rates on convergence

For the higher learning rate of 0.01, the trained weights converge slowly to higher loss which indicates overshooting. In this case, the weight vector oscillates between its optimal value and fails to reach a lower value. In comparison, choosing learning rates of $0.1n$ and $0.01n$ results in slower convergence compared to $\frac{1}{n}$.

The stopping criteria for this experimentation is the number of iterations for the purpose of comparison. As a result, the running time is independent of the learning rate. However, changing the stopping criteria to $\|\mathbf{w_{k+1}} - \mathbf{w_k}\|^2 < \epsilon$, where $\|\mathbf{w_{k+1}} - \mathbf{w_k}\|$ is the norm of the weight vector's variation between iterations, the running time for smaller learning rates would be longer as more iterations are required for it decrease below a threshold, ϵ. Therefore, we propose a learning rate of $\frac{1}{n}$ for all our models.



(a) Diabetes RFE result



(b) Red Wine data set

Figure 3: Recursive Feature Elimination Validation Accuracy and Run Time Results[†]

[†]The shaded area is the error margin of the validated accuracy

3.2 Pima Indian Diabetes

The PID data set with the constructed features has a total of 24 features. After running through the cross-validated RFE algorithm with a learning rate of $\frac{1}{n}$ and a maximum learning rate of 10000. We have obtained the validation accuracy of the subsets of the feature space with the n best features. The result is summarized in fig.3(a).

The maximum mean cross-validation accuracy is achieved with 7 and 8 features selected. This corresponds to a validation accuracy of 74.83%. As shown, further increasing the number of feature selected, in other words, increasing the model's feature dimension only slightly reduces accuracy. This pattern can be explained by the logistic regression model's ability to give higher weights to the most relevant features. Therefore, while irrelevant features are not removed as in the case of models with smaller dimensions of feature space, they have less impact on the model's prediction.

Another crucial metric is running time, training models with higher number of features is time consuming as shown in fig.3(a). While training model with 7 features has a running time of 9.71 s, training that with 24 features takes 19.96 s. Considering both running time and validation accuracy, the optimal number of features is 7.

3.2.1 Proposed Model for PID data set

Based on the analysis outlined in the previous sections, the proposed model for the PID dataset is a model with 7 features, learning rate $\alpha = \frac{1}{n}$, epsilon $\epsilon = 1e - 6$. The 7 selected features are tabulated in table 2.

Table 2: Diabetes features selected by model selection algorithm

$Glucose$	$Glucose^2$	Age^2	BMI^2	$sqrt(Glucose)$	$sqrt(Age)$	$BMI * DPF$

The proposed model produces an average validation accuracy of $74.33 \pm 7.5\%$ and an averaged training accuracy of $75.54 \pm 1.0\%$. The small difference between validation and training accuracy indicates the model is not over-fitting the data set.

3.3 Red Wine

The expanded red wine data set has 20 features in total. Following the cross-validated RFE algorithm with the same hyper-parameters as for the PID data set in 3.2, the result is given in fig.3(b).

As in the previous section, we aim to balance the validation accuracy and running time. Hence, we pick the feature set with 6 features. This set produces an average validation accuracy of 74.2% and running time of $36.9s$ during RFE,

3.3.1 Proposed Model for Red Wine data set

The proposed model for the Red Wine data set is a model with 6 features, learning rate $\alpha = \frac{1}{n}$, epsilon $\epsilon = 1e - 6$. The 6 selected features are given in Table 3.

Table 3: Red wine features selected by model selection algorithm

$FreeSO_2$	$TotalSO_2$	$Sulphates$	$Alcohol$	$Sulphate^2$	$sqrt(VolatileAcidity)$

This model gives an average validation accuracy of $74.28 \pm 3.1\%$ and an average training accuracy of $74.71 \pm 0.2\%$, which, again, shows that our model is not over-fitting the data.

4 Discussion and Conclusion

4.1 Feature Selection Pattern

In conclusion, we can clearly observe a pattern from the algorithm's feature selection on the PID data set. As in Table 2, the important features are Glucose, Age, BMI, and DPF which match the assumption that we have made from observing the weights on each feature. Glucose is the most predictive feature as all relating features (original, square, square root) remains after running through RFE. This conforms the medical domain knowledge as glucose in blood directly connected to diabetes. Then, Age is also a necessary indicator whose square and square root are both selected. The major reason that Age's original form is not an optimum choice is because its data are not very far from each other, which means most of tested people are in similar age range. So, applying non-linear would signify these differences making it a more optimized indicator. BMI and DPF are both direct medical indicators of diabetes. This signifies that the RFE algorithm perform well on selecting the most relevant features. For red wine quality data set, we have similar feature selection pattern (as in Table 3) where the selected indicators match the scientific result and domain knowledge.

It is also noticeable that both data sets have less features in the optimized feature selection than their original feature set. Therefore, we can conclude that models having more features does not lead to higher accuracy, and features that have large absolute value of weights in our model are more valuable for prediction.

4.2 Further Improvements

The disadvantage of the DFE algorithm its high running time. Despite it being a greedy algorithm, the algorithm trains and evaluates them in a brute force manner, resulting in high running time. Other feature selection techniques such as filter based feature selection could be implemented and compared with our approach in order to find the optimal technique for feature selection[5].

If time is not a constraint, we would have a better chance of getting more better subsets of features by feeding the algorithm other constructed features. Therefore, we could expand our model to order-3 polynomials or even high orders and explore other interactions terms between features.

5 Statement of Contribution

Jack Wei: Implementation of RFE algorithm, optimize model, feature processing, report
Grey Yuan: draft programming, feature selection, data analysis, graphs, tables, report
Yu Wu: draft programming, preliminary data process, tables, report and report organization
This project is completed as a team of 3, and each member worked actively and made great contributions to the project.

References

[1] Bertsimas, Dimitris, & Angela King. (2017) “Logistic Regression: From Art to Science.” *Statistical Science*, vol. 32, no. 3, Institute of Mathematical Statistics, pp. 367–84

[2] Morgan, S. Philip, & Jay D. Teachman. (1988) “Logistic Regression: Description, Examples, and Comparisons.” *Journal of Marriage and Family* 50, no. 4 : 929–36.

[3] Bender, R, & U Grouven. (1997) “Ordinal logistic regression in medical research.” *Journal of the Royal College of Physicians of London* vol. 31,5 : 546-51.

[4] Jason D. M. Rennie (2005) Regularized Logistic Regression is Strictly Convex. http://people.csail.mit.edu/jrennie/writing/convexLR.pdf

[5] Isabelle Guyon & André Elisseeff. (2003) An introduction to variable and feature selection. *J. Mach. Learn.* Res. 3, null (3/1/2003), 1157–1182.

[6] Scikit−yb (2016) Recursive Feature Elimination. https://www.scikit-yb.org/en/latest/api/modelselection/rfecv.html

6 Appendix

```
# -*- coding: utf-8 -*-
"""ECSE551_Assignment1.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1hzJmwYJJV5LuJwFPlJhce1cxcYd-lHtC

# ECSE 551 Assignment 1
"""

from google.colab import drive

import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from numpy import linalg as LA
from time import time
from google.colab import files

drive.mount('/content/gdrive')

path = "./gdrive/MyDrive/ECSE_551_Assignment_1/Data/"

"""## Loading data"""

df_d = pd.read_csv(path + '/diabetes.csv', header=None)
df_d = df_d.sample(frac=1, random_state=0).reset_index(drop=True) #shuffle df_d

df_rw = pd.read_csv(path + 'red_wine.csv', header=None)
df_rw = df_rw.sample(frac=1, random_state=0).reset_index(drop=True)  #shuffle df_d

df_d.head()

df_rw.head()

df_d.columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPF', 'Age', 'Outcome']
df_d.head()

df_rw.columns = ['fixed_acidity', 'volatile_acidity', 'citric_acid', 'residual_sugar', 'chlorides', 'free_SO2', 'total_SO2', 'density', 'pH'
df_rw.head()

"""##Data Analysis"""

print("diabetes␣sample␣size␣=␣" + str(len(df_d.index)))
print("red␣wine␣sample␣size␣=␣" + str(len(df_rw.index)))

df_d['Outcome'].value_counts()

"""Diabetes correlation heat matrix"""

df_rw['Output'].value_counts()

plt.matshow(df_d.corr())
plt.show()

"""Red Wine correlation heat matrix"""

plt.matshow(df_rw.corr())
plt.show()
```

```
color = ['dodgerblue', 'orangered']
colors = df_d['Outcome'].map(lambda x: color[x])
pd.plotting.scatter_matrix(df_d.iloc[:,:-1], figsize= (14, 12), diagonal='kde', color=colors);

color = ['dodgerblue', 'orangered']
colors = df_rw['Output'].map(lambda x: color[x])
pd.plotting.scatter_matrix(df_rw.iloc[:,:-1], figsize= (18, 16), diagonal='kde', color=colors);

"""## Preprocessing"""

# convert to numpy
np_d = df_d.to_numpy()
np_rw = df_rw.to_numpy()

# Get feature set and label set
X_train_d = np_d[:,:-1]
Y_d = np_d[:,-1]

X_train_rw = np_rw[:,:-1]
Y_rw = np_rw[:,-1]

"""## Model"""

class Logistic_Regression_Model:
  def __init__(self):
    # weight vector
    self.w = None # Type: np.array

  def calculate_loss(self, w, X, y):
    n = X.shape[0]
    loss = 0
    for i in range(n):
      y_pred = self.sigmoid(np.matmul(w, X[i]))
      loss -= y[i] * np.log(y_pred) + (1 - y[i]) * np.log(1 - y_pred)
    return loss

  def fit(self, X , y, alpha_k, epsilon = 0, max_iter = float("inf"), compute_loss = False, verbose = False):
    if compute_loss: loss_list = []

    # Get the dimensions of X
    n, m = X.shape

    # Generate and append bias column
    bias = np.ones((n,1), dtype=np.double)
    X = np.append(X, bias, axis = 1)

    # Initialize w as all 0s
    w = np.zeros(m+1) # dimension 1*(m+1)

    # Gradient Descent
    k = 0                     # number of iterations
    l2_norm = float("inf")    # initial error is infinity
    if verbose: print("L2-Norm␣of␣w:")
    while(l2_norm > epsilon and k < max_iter):
      delta = np.zeros(m+1)

      for i in range(n):
        y_pred = self.sigmoid(np.matmul(w, X[i])) # 1D matmul is dot product
        delta -= X[i] * (y[i] - y_pred)

      new_w = w - alpha_k * delta

      l2_norm = LA.norm(new_w - w) ** 2

      # Print norm for every 100 iterations
      if k % 100 == 0:
        if verbose:
          print("Iteration␣" + str(k) + ":␣" + str(l2_norm))
        if compute_loss:
          loss = self.calculate_loss(w, X, y);
          loss_list.append((k, loss))

      w = new_w
      k += 1

    self.w = w
    if compute_loss: return np.array(loss_list)

  def predict(self, X):
    predicted = []

    bias = np.ones((X.shape[0],1), dtype=np.double)
    X = np.append(X,bias, axis=1)
    for x in X:
      if(self.sigmoid(np.matmul(self.w, x)) >= 0.5):
        predicted.append(1)
      else:
        predicted.append(0)

    return np.array(predicted)
  def sigmoid(self, x):
    return 1 / (1 + np.exp(-x))

"""## Preliminary Experimention

### Diabetes
"""

model = Logistic_Regression_Model()
model.fit(X_train_d, Y_d, alpha_k=1/X_train_d.shape[0], epsilon=0.0000001, max_iter=15000, verbose=True)

print("Our␣model␣weights:␣")
print(model.w)

"""#### learning rate"""

n = X_train_d.shape[0]
learning_rates = [1/100, 1/n, 1/(10 * n), 1/ (100 * n)]
loss_data_lists = []

for learning_rate in learning_rates:
  model = Logistic_Regression_Model()
  loss_data = model.fit(X_train_d, Y_d, epsilon=0, alpha_k=learning_rate, max_iter=10000, compute_loss=True, verbose=True)
  loss_data_lists.append(loss_data)

for loss_data in loss_data_lists:
  plt.plot(loss_data[:,0], loss_data[:,1])
plt.legend(["1/100", "1/n", "1/(10n)", "1/(100n)"])
plt.xlabel('iterations')
plt.ylabel('cross␣entropy␣loss')
plt.show()

"""#### Epsilon"""

epsilons = [0.01, 0.0001, 0.000001, 0.00000001, 0.0000000001, 0.000000000001]
running_time_list = []

model = Logistic_Regression_Model()

for epsilon in epsilons:
  start = time()
  model.fit(X_train_d, Y_d, epsilon=epsilon, alpha_k=1/X_train_d.shape[0], verbose=True)
  end = time()

  running_time_list.append(end - start)

print(running_time_list)

plt.bar(['1e-2', '1e-4', '1e-6', '1e-8', '1e-10', '1e-12'],running_time_list)
plt.ylabel('running␣time␣(s)')
plt.show()

"""### Red Wine"""

model = Logistic_Regression_Model()
model.fit(X_train_rw, Y_rw, alpha_k=1/X_train_rw.shape[0], epsilon=0.0000001, max_iter=15000, verbose=True)

print("Our␣model␣weights:␣")
print(model.w)

n = X_train_rw.shape[0]
learning_rates = [1/100, 1/n, 1/(10 * n), 1/ (100 * n)]
loss_data_lists = []

for learning_rate in learning_rates:
  model = Logistic_Regression_Model()
  loss_data = model1.fit(X_train_rw, Y_rw, epsilon=0, alpha_k=learning_rate, max_iter=3000, compute_loss=True, verbose=True)
  loss_data_lists.append(loss_data)

for loss_data in loss_data_lists:
  plt.plot(loss_data[:,0], loss_data[:,1])
plt.legend(["1/100", "1/n", "1/(10n)", "1/(100n)"])
plt.xlabel('iterations')
plt.ylabel('cross␣entropy␣loss')
plt.show()

"""## Feature Engineering"""

df_d.insert(8, 'Glucose␣**2', '')
df_d.insert(9, 'Age␣**2', '')
df_d.insert(10, 'BMI␣**2', '')
df_d.insert(11, 'Pressure␣**2', '')
df_d.insert(12, 'sqrt␣Glucose', '')
df_d.insert(13, 'sqrt␣Age', '')
df_d.insert(14, 'sqrt␣BMI', '')
df_d.insert(15, 'sqrt␣Pressure', '')

df_d.insert(16,'Glucose␣*␣BMI','')
df_d.insert(17,'Glucose␣*␣DPF','')
df_d.insert(18, 'Glucose␣*␣insulin','')
df_d.insert(19,'Glucose␣*␣Age','')
df_d.insert(20,'BMI␣*␣DPF','')
df_d.insert(21,'BMI␣*␣Pressure','')
df_d.insert(22,'BMI␣*␣Age','')
df_d.insert(23,'Pressure␣*␣Age','')

df_d['Glucose␣**2'] = df_d.apply((lambda row: row.Glucose ** 2), axis=1)
df_d['Age␣**2'] = df_d.apply((lambda row: row.Age ** 2), axis=1)
df_d['BMI␣**2'] = df_d.apply((lambda row: row.BMI ** 2), axis=1)
df_d['Pressure␣**2'] = df_d.apply((lambda row: row.BloodPressure ** 2), axis=1)
df_d['sqrt␣Glucose'] = df_d.apply((lambda row: np.sqrt(row.Glucose)), axis=1)
df_d['sqrt␣Age'] = df_d.apply((lambda row: np.sqrt(row.Age)), axis=1)
df_d['sqrt␣BMI'] = df_d.apply((lambda row: np.sqrt(row.BMI)), axis=1)
df_d['sqrt␣Pressure'] = df_d.apply((lambda row: np.sqrt(row.BloodPressure)), axis=1)

df_d['Glucose␣*␣BMI'] = df_d.apply((lambda row: row.Glucose * row.BMI), axis=1)
df_d['Glucose␣*␣DPF'] = df_d.apply((lambda row: row.Glucose * row.DiabetesPF), axis=1)
df_d['Glucose␣*␣insulin'] = df_d.apply((lambda row: row.Glucose * row.Insulin), axis=1)
df_d['Glucose␣*␣Age'] = df_d.apply((lambda row: row.Glucose * row.Age), axis=1)
df_d['BMI␣*␣DPF'] = df_d.apply((lambda row: row.BMI * row.DiabetesPF), axis=1)
df_d['BMI␣*␣Pressure'] = df_d.apply((lambda row: row.BloodPressure * row.BMI), axis=1)
df_d['BMI␣*␣Age'] = df_d.apply((lambda row: row.BMI * row.Age), axis=1)
df_d['Pressure␣*␣Age'] = df_d.apply((lambda row: row.BloodPressure * row.Age), axis=1)

df_rw.insert(11, 'volatile␣acidity␣**2', '')
df_rw.insert(12, 'free␣SO2␣**2', '')
df_rw.insert(13, 'pH␣**2', '')
df_rw.insert(14, 'sulphates␣**2', '')
df_rw.insert(15, 'sqrt␣volatile␣acidity', '')

df_rw.insert(16,'pH␣*␣volatile␣acidity','')
df_rw.insert(17,'pH␣*␣citric␣acid','')
df_rw.insert(18, 'sulphate␣*␣free␣SO2','')
df_rw.insert(19,'fixed␣*␣density','')

df_rw['volatile␣acidity␣**2'] = df_rw.apply((lambda row: row.volatile_acidity** 2), axis=1)
df_rw['free␣SO2␣**2'] = df_rw.apply((lambda row: row.free_SO2 ** 2), axis=1)
df_rw['pH␣**2'] = df_rw.apply((lambda row: row.pH ** 2), axis=1)
df_rw['sulphates␣**2'] = df_rw.apply((lambda row: row.sulphates ** 2), axis=1)
df_rw['sqrt␣volatile␣acidity'] = df_rw.apply((lambda row: np.sqrt(row.volatile_acidity)), axis=1)

df_rw['pH␣*␣volatile␣acidity'] = df_rw.apply((lambda row: row.pH * row.volatile_acidity), axis=1)
df_rw['pH␣*␣citric␣acid'] = df_rw.apply((lambda row: row.pH * row.citric_acid), axis=1)
df_rw['sulphate␣*␣free␣SO2'] = df_rw.apply((lambda row: row.sulphates * row.free_SO2), axis=1)
df_rw['fixed␣*␣density'] = df_rw.apply((lambda row: row.fixed_acidity * row.density), axis=1)

# convert to numpy
np_d = df_d.to_numpy()
np_rw = df_rw.to_numpy()

# Get feature set and label set
X_train_d = np_d[:,:-1]
Y_d = np_d[:,-1]

X_train_rw = np_rw[:,:-1]
Y_rw = np_rw[:,-1]

"""## Automated Feature Selection

Recursive Feature Elimination (RFE)
"""

def RFE(model_class, X_train, Y, num_features_selected, original_X_train, i=None):

  n = X_train.shape[1] # number of features left
  print("number␣of␣feature␣left␣is␣" + str(n))
  m = model_class()
  m.fit(X_train, Y, alpha_k=1/X_train.shape[0], epsilon=0.0000001, max_iter=7000, verbose=True)

  print("weight␣w␣shape" + str(m.w.shape))
  if n <= num_features_selected:  # no features left
    print("returned!")
    print(i)
    return m, i

  i = np.argmin(np.abs(m.w[:-1])) # :-1 because we do not consider dummy (bias) term
  X_train_del = np.delete(X_train, i, 1)

  return RFE(model_class, X_train_del, Y, num_features_selected, original_X_train, i)

#k-fold cross, validation set as i th subset, dataset= name of the dataset
def seperate(k, i, dataset):
  #count the number of rows of the dataset
  n,m = dataset.shape
  #devide into k sets to get the unber of rows in each subset
  subset_rows = int(n/k)
  #create and initialize the validation set and training set
  vs = []
  ts = []
  for j in range(n):
    if j < subset_rows*(i+1) and j >= subset_rows*i:
       vs.append(dataset[j])
    else:
       ts.append(dataset[j])
  return np.array(ts), np.array(vs)

def Accu_eval(ground_truth, predicted):
  return (ground_truth == predicted).mean()

def append_index(index, new_element, stack):
  if new_element == None: return

  while len(stack) != 0:
    if(new_element >= stack.pop()):
      new_element += 1

  index.append(new_element)

"""### Diabetes"""

row, col = np_d.shape
 col = col - 1
 final_result = []

 for i in range(10):

   X_train_set, X_val_set = seperate(10, i, np_d)

   X_train = X_train_set[:,:-1]
   Y_train = X_train_set[:,-1]
   X_val = X_val_set[:,:-1]
   Y_val = X_val_set[:,-1]

   model_result = []

   del_i = []
   stack = []

   for num_features in range(col, 0, -1):
      if len(del_i) != 0 : del_X_train = np.delete(X_train, del_i, 1)
      else: del_X_train = X_train

      m, index = RFE(Logistic_Regression_Model, del_X_train, Y_train, num_features, X_train)

      append_index(del_i, index, stack.copy())
      if index != None: stack.append(index)
      X_val_del = np.delete(X_val, del_i, 1)

      if len(del_i) == 0: predicted = m.predict(X_val)
      else: predicted = m.predict(X_val_del)

      print(del_i)

      accuracy = Accu_eval(Y_val, predicted)
      print(accuracy)
      model_result.append((accuracy, del_i.copy()))

   final_result.append(model_result.copy())

pd.DataFrame(final_result).to_csv('Diabetes␣RFE␣result.csv', header=None, index=None)

df_d_rfe = pd.read_csv(path + '/Diabetes␣RFE␣accuracy␣result.csv')
df_d_rfe.head()

"""### Red Wine"""

row, col = np_rw.shape
 col -= - 1
 final_result = []

 for i in range(10):

   X_train_set, X_val_set = seperate(10, i, np_rw)

   X_train = X_train_set[:,:-1]
   Y_train = X_train_set[:,-1]
   X_val = X_val_set[:,:-1]
   Y_val = X_val_set[:,-1]

   model_result = []

   del_i = []
   stack = []

   for num_features in range(col, 0, -1):
      if len(del_i) != 0 : del_X_train = np.delete(X_train, del_i, 1)
      else: del_X_train = X_train

      print("At␣" + str(i) + "␣th␣fold")
      m, index = RFE(Logistic_Regression_Model, del_X_train, Y_train, num_features, X_train)

      append_index(del_i, index, stack.copy())
      if index != None: stack.append(index)
      X_val_del = np.delete(X_val, del_i, 1)

      if len(del_i) == 0: predicted = m.predict(X_val)
```

```
        else: predicted = m.predict(X_val_del)

        print(del_i)

        accuracy = Accu_eval(Y_val, predicted)
        print(accuracy)
        model_result.append((accuracy, del_i.copy()))

    final_result.append(model_result.copy())

pd.DataFrame(final_result).to_csv('Red␣Wine␣RFE␣result.csv', header=None, index=None)

files.download('Red␣Wine␣RFE␣result.csv')

"""## Result Analysis

### Diabetes
"""

df_d_rfe = pd.read_csv(path + 'Diabetes␣RFE␣accuracy␣result.csv')

df_d_rfe.describe().loc["mean"]

deleting_list = [3, 4, 2, 11, 7, 18, 21, 22, 0, 16, 14, 19, 5, 6, 17, 15, 23, 1, 12, 20, 8, 9, 13]

running_time_list = []
row, col = np_d.shape
col = col - 1

X_train_set, X_val_set = seperate(10, 1, np_d)

X_train = X_train_set[:,:-1]
Y_train = X_train_set[:,-1]

model1 = Logistic_Regression_Model()

for i in range(col):
  del_i = deleting_list[0:i]
  X_train_del = np.delete(X_train, del_i, 1)

  start = time()
  model1.fit(X_train_del, Y_train, alpha_k=1/X_train.shape[0], epsilon=0.000001, verbose=True)
  end = time()

  running_time_list.append(end - start)

mean = df_d_rfe.describe().loc["mean"]
x = range(20,0,-1)

fig, ax1 = plt.subplots()

color = 'tab:blue'
ax1.set_xlabel('number␣of␣features')
ax1.set_ylabel('validation␣accuracy', color=color)

ax1.plot(x, df_d_rfe.describe().loc["mean"])
ax1.tick_params(axis='y', labelcolor=color)

ax2 = ax1.twinx()

color = 'tab:red'
ax2.set_ylabel('running␣time␣(s)', color=color)
ax2.plot(x, running_time_list, color=color)
ax2.tick_params(axis='y', labelcolor=color)

ax1.fill_between(x, df_d_rfe.describe().loc["mean"] - df_d_rfe.describe().loc["std"], df_d_rfe.describe().loc["mean"] + df_d_rfe.describe().l

plt.show()

"""Running time

Our proposed model for diabetes data is when alpha = 1/n, epsilon = 0.000001, X_train = X_t - X_indices of [3, 4, 22, 11, 2, 7, 21, 0, 18, 1
"""

model = Logistic_Regression_Model()
scores = []

del_i = [3, 4, 22, 11, 2, 7, 21, 0, 18, 16, 14, 5, 6, 17, 19, 15, 23]

for i in range(10):

    X_train_set, X_val_set = seperate(10, i, np_d)

    X_train = X_train_set[:,:-1]

    Y_train = X_train_set[:,-1]
    X_val = X_val_set[:,:-1]
    Y_val = X_val_set[:,-1]

    X_train = np.delete(X_train, del_i, 1)
    X_val = np.delete(X_val, del_i, 1)
    model.fit(X_train, Y_train, alpha_k=1/X_train.shape[0], epsilon=0.000001, verbose=True)

    train_predicted = model.predict(X_train)
    val_predicted = model.predict(X_val)

    training_accuracy = Accu_eval(Y_train, train_predicted)
    validation_accuracy = Accu_eval(Y_val, val_predicted)
    scores.append((training_accuracy, validation_accuracy))

unzipped_scores = np.array(list(zip(*scores)))

print("10␣fold-cross␣training␣accuracy:␣" + str(unzipped_scores[0].mean()))
print("10␣fold-cross␣validation␣accuracy:␣" + str(unzipped_scores[1].mean()))

print("10␣fold-cross␣training␣std:␣" + str(unzipped_scores[0].std()))
print("10␣fold-cross␣validation␣std:␣" + str(unzipped_scores[1].std()))

"""### Red Wine"""

df_rw_rfe = pd.read_csv(path + 'Red␣Wine␣RFE␣accuracy␣result.csv', header=None)

df_rw_rfe.describe().loc["mean"]

df_rw_rfe.head()

df_rw_rfe.describe()

deleting_list =  [19, 16, 3, 7, 0, 17, 11, 8, 2, 18, 13, 4, 1, 12]

running_time_list = []
row, col = np_rw.shape
col = col - 1

X_train_set, X_val_set = seperate(10, 1, np_rw)

X_train = X_train_set[:,:-1]
Y_train = X_train_set[:,-1]

model1 = Logistic_Regression_Model()

for i in range(col):
  del_i = deleting_list[0:i]
  X_train_del = np.delete(X_train, del_i, 1)

  start = time()
  model1.fit(X_train_del, Y_train, alpha_k=1/X_train.shape[0], epsilon=0.000001, verbose=True)
  end = time()

  running_time_list.append(end - start)

mean = df_rw_rfe.describe().loc["mean"]
x = range(20,0,-1)

fig, ax1 = plt.subplots()

color = 'tab:blue'
ax1.set_xlabel('number␣of␣features')
ax1.set_ylabel('validation␣accuracy', color=color)

ax1.plot(x, df_rw_rfe.describe().loc["mean"])
ax1.tick_params(axis='y', labelcolor=color)

ax2 = ax1.twinx()

plt.xticks(range(20,0,-2))

color = 'tab:red'
ax2.set_ylabel('running␣time␣(s)', color=color)
ax2.plot(x, running_time_list, color=color)
ax2.tick_params(axis='y', labelcolor=color)


ax1.fill_between(x, df_rw_rfe.describe().loc["mean"] - df_rw_rfe.describe().loc["std"], df_rw_rfe.describe().loc["mean"] + df_rw_rfe.describe

plt.show()

"""Our proposed model for red wine data is when alpha = 1/n, epsilon = 0.000001, X_train = X_t - X_indices of
[19, 16, 3, 7, 0, 17, 11, 8, 2, 18, 13, 4, 1, 12]"""

model = Logistic_Regression_Model()
scores = []

del_i = [19, 16, 3, 7, 0, 17, 11, 8, 2, 18, 13, 4, 1, 12]

for i in range(10):

    X_train_set, X_val_set = seperate(10, i, np_rw)

    X_train = X_train_set[:,:-1]

    Y_train = X_train_set[:,-1]
    X_val = X_val_set[:,:-1]
    Y_val = X_val_set[:,-1]

    X_train = np.delete(X_train, del_i, 1)
    X_val = np.delete(X_val, del_i, 1)

    model.fit(X_train, Y_train, alpha_k=1/X_train.shape[0], epsilon=0.000001, verbose=True)

    train_predicted = model.predict(X_train)
    val_predicted = model.predict(X_val)

    training_accuracy = Accu_eval(Y_train, train_predicted)
    validation_accuracy = Accu_eval(Y_val, val_predicted)
    scores.append((training_accuracy, validation_accuracy))

unzipped_scores = np.array(list(zip(*scores)))

print("10␣fold-cross␣training␣accuracy:␣" + str(unzipped_scores[0].mean()))
print("10␣fold-cross␣validation␣accuracy:␣" + str(unzipped_scores[1].mean()))

print("10␣fold-cross␣training␣std:␣" + str(unzipped_scores[0].std()))
print("10␣fold-cross␣validation␣std:␣" + str(unzipped_scores[1].std()))
```