

## ▼ Mini Assignment 4. Linear Classification

Please add one code cell after each question and create a program to answer the question. Make sure your code runs without error. After you are finished, click on File -> Print -> Save as PDF to create a PDF output and upload it on Brightspace.

NOTE: Make sure your PDF does not have your name or any identifying information in the name or content of the file. Anonymity is essential for the peer-review process.

### ▼ The Case: Diabetes Prediction

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

The datasets consists of several medical predictor variables and one target variable, Outcome. Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

You may [download the dataset here](#). The dataset includes 9 features and over 700 records. It contains the following variables:

- **Pregnancies:** Number of times pregnant
- **Glucose:** Plasma glucose concentration a 2 hours in an oral glucose tolerance test
- **BloodPressure:** Diastolic blood pressure (mm Hg)
- **SkinThickness:** Triceps skin fold thickness (mm)
- **Insulin:** 2-Hour serum insulin (mu U/ml)
- **BMI:** Body mass index (weight in kg/(height in m)<sup>2</sup>)
- **DiabetesPedigreeFunction:** Diabetes pedigree function
- **Age:** Age (years)
- **Outcome:** Class variable (0 or 1) 268 of 768 are 1, the others are 0

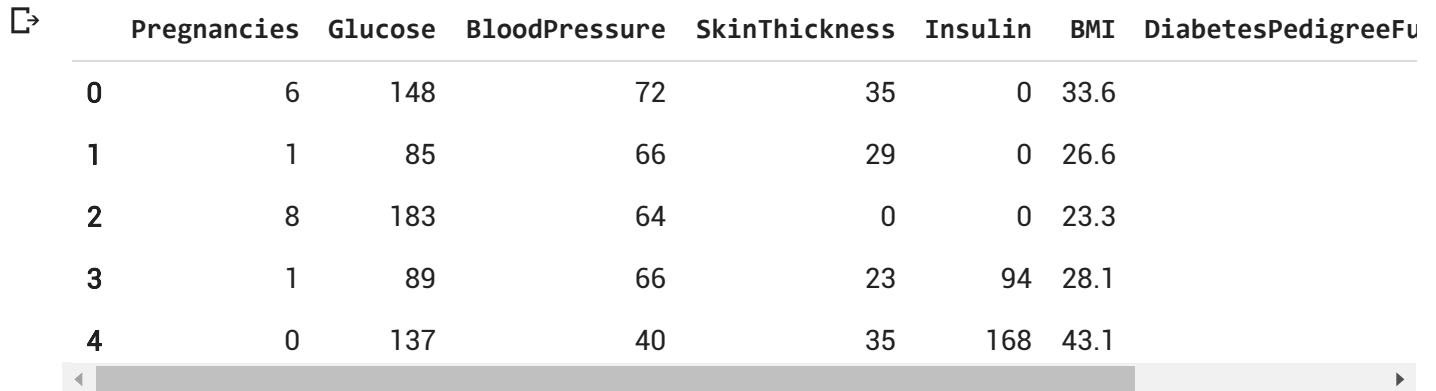
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import pandas as pd
```

1. Read the dataset into a dataframe, and check out the first few rows and column data types.

```
df = pd.DataFrame(pd.read_csv('/content/drive/MyDrive/colabData/diabetes.csv'))
df.head()
```



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	

2. Inspect the dataset for missing values and treat them if there is any.

```
df.isna().sum()
```

```
Pregnancies      0
Glucose           0
BloodPressure     0
SkinThickness     0
Insulin           0
BMI               0
DiabetesPedigreeFunction  0
Age               0
Outcome           0
dtype: int64
```

3. We would like to create linear classifiers to predict the outcome (if the patient has diabetes). Create predictor (X) and target (y) datasets. Then normalize/standardize the predictor dataset, and split X and y into train and test datasets. make sure to use stratification.

```
X = df.drop("Outcome", axis = 1)
y = df.Outcome
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

```
ssc = StandardScaler()
X_scaled = ssc.fit_transform(X)
```

```
SEED = 1234
```

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size = 0.2, random_stat
```

4. Use simple linear regression to predict the outcome class (0 or 1). Print the accuracy rate for the train and the test sets. Tune the model with C hyperparameter (inverse of regularization) to get the best fit.

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score
from matplotlib import pyplot as plt
import numpy as np
```

```
lr = LinearRegression()
lr.fit(X_train, y_train)
```

```
scores = []
xlist = []
```

```
y_predictions = lr.predict(X_test)
```

```
for x in np.linspace(0, 1, 20):
    test_predictions = y_predictions > x
    xlist.append(x)
    score = accuracy_score(y_test, test_predictions)
    scores.append(score)
```

```
winnerscore = max(scores)
winnerindex = scores.index(winnerscore)
winner = xlist[winnerindex]
```

```
y_pred1 = y_predictions > winner
```

```
print(accuracy_score(y_test, y_pred1))
```

```
0.8181818181818182
```

5. Use logistic linear regression to predict the outcome. Print the accuracy rate for the train and the test sets. Tune the model with C hyperparameter (inverse of regularization) to get the best fit.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

```

lr2 = LogisticRegression()

params = {
    "C": [0.05, 0.1, 0.15, 0.2, 0.25, 0.5, 1]
}

search = GridSearchCV(lr2, param_grid = params)
search.fit(X_train, y_train)

best_lr2 = search.best_estimator_
print(search.best_params_)

y_pred2 = best_lr2.predict(X_test)
print(accuracy_score(y_test, y_pred2))

{'C': 0.2}
0.8181818181818182

```

6. Use linear support vector machine to predict the outcome. Print the accuracy rate for the train and the test sets. Tune the model with C hyperparameter (inverse of regularization) to get the best fit.

```

from sklearn.svm import LinearSVC
from sklearn.model_selection import GridSearchCV

lsvc = LinearSVC(max_iter=100000)

params = {
    "C": [0.01, 0.05, .1, .15, .2, .5, 1]}

search = GridSearchCV(lsvc, param_grid = params)
search.fit(X_train, y_train)
print(search.best_params_)

{'C': 0.1}

params2 = {"C": [0.09, 0.095, 0.1, 0.105, 0.11]}
search2 = GridSearchCV(lsvc, param_grid=params2)
search2.fit(X_train, y_train)
print(search2.best_params_)

{'C': 0.1}

best_lsvc = search2.best_estimator_
print(best_lsvc.score(X_test, y_test))
print(best_lsvc.score(X_test, y_test))

```

```
y_pred3 = best_lsvc.predict(X_test)
```

```
0.8116883116883117
```

```
0.8116883116883117
```

7. Use SVM with non-linear kernel trick to predict the outcome. Print the accuracy rate for the train and the test sets. Tune the model with C and gamma hyperparameters to get the best fit.

```
from sklearn.svm import SVC
```

```
svc = SVC()
```

```
params = {
```

```
    "C": [0.05, 0.1, 0.15, 0.2, 0.5, 1, 2, 5],
```

```
    "gamma": [0.001, 0.005, 0.01, 0.05, 0.1, 0.15, 0.2, 0.5, 5]}
```

```
search = GridSearchCV(svc, param_grid = params)
```

```
search.fit(X_train, y_train)
```

```
print(search.best_params_)
```

```
    {'C': 5, 'gamma': 0.01}
```

```
best_svc = search.best_estimator_
```

```
print(best_svc.score(X_test, y_test))
```

```
print(best_svc.score(X_train, y_train))
```

```
y_pred4 = best_svc.predict(X_test)
```

```
0.7987012987012987
```

```
0.7899022801302932
```

8. Print the test set classification report for the top four models. Which one offers a better f1-score?

```
from sklearn.metrics import classification_report
```

```
predictions = [y_pred1, y_pred2, y_pred3, y_pred4]
```

```
for pred in predictions:
```

```
    print(classification_report(y_test, pred))
```

	precision	recall	f1-score	support
0	0.83	0.90	0.87	100
1	0.78	0.67	0.72	54
accuracy			0.82	154
macro avg	0.81	0.78	0.79	154
weighted avg	0.82	0.82	0.81	154

	precision	recall	f1-score	support
0	0.83	0.91	0.87	100
1	0.80	0.65	0.71	54
accuracy			0.82	154
macro avg	0.81	0.78	0.79	154
weighted avg	0.82	0.82	0.81	154

	precision	recall	f1-score	support
0	0.83	0.90	0.86	100
1	0.78	0.65	0.71	54
accuracy			0.81	154
macro avg	0.80	0.77	0.78	154
weighted avg	0.81	0.81	0.81	154

	precision	recall	f1-score	support
0	0.81	0.91	0.85	100
1	0.78	0.59	0.67	54
accuracy			0.80	154
macro avg	0.79	0.75	0.76	154
weighted avg	0.80	0.80	0.79	154

ANSWER: Somehow, the standard linear regression model ended up giving me the most accurate results.

---

✓ 0s completed at 6:40 PM

