به نام خدا

گزارش سوال ۵ از تمرین اول
مهدی کافی ۹۹۲۱۰۷۵۳

۱.۱.۵
در این بخش با استفاده از دستورات کتابخانه pandas و به صورت زیر داده‌ها را خوانده‌ایم و سپس برای آشنایی با داده‌های خوانده‌شده از دو دستور head و describe استفاده کردیم.

```python
# Packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
# Load Data
train_data = pd.read_csv("Diabetes/Train.csv")
test_data = pd.read_csv("Diabetes/Test.csv")
```

```python
train_data.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	1	164	82	43	67	32.8	0.341	50	False
1	4	90	0	0	0	28.0	0.610	31	False
2	1	138	82	0	0	40.1	0.236	28	False
3	4	110	92	0	0	37.6	0.191	30	False
4	2	93	64	32	160	38.0	0.674	23	True

```python
train_data.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
count	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000
mean	3.801303	120.698697	69.097720	20.856678	78.412052	31.853746	0.464230	33.086319
std	3.401314	31.805415	19.301258	15.892553	113.145367	7.800839	0.311473	11.925163
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.245250	24.000000
50%	3.000000	117.000000	72.000000	23.000000	34.000000	32.000000	0.367000	29.000000
75%	6.000000	139.000000	80.000000	32.000000	125.750000	36.400000	0.613750	40.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.288000	81.000000

برای کنترل کیفیت داده‌ها دستورات زیر را استفاده کرده‌ایم تا از وجود مقادیر null و یا nan آگاه شویم و متوجه می‌شویم که داده‌ها این مقادیر را ندارند.

```
# Quality Control
train_data.isnull().sum()
```

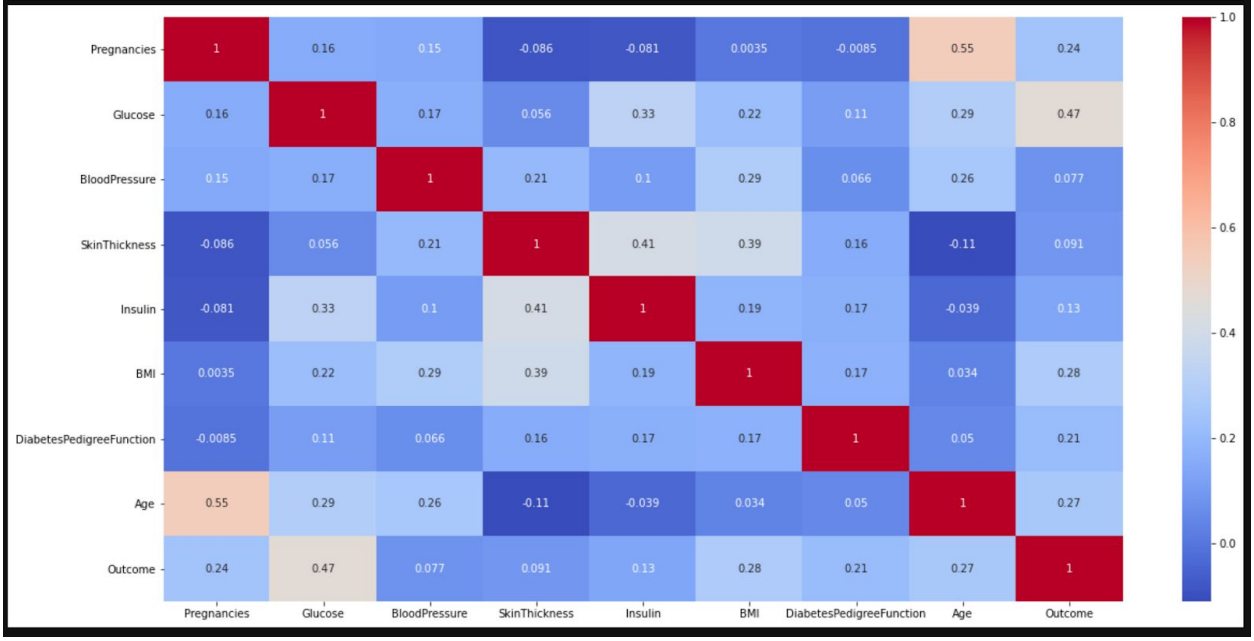
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

```
train_data.isna().sum()
```

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

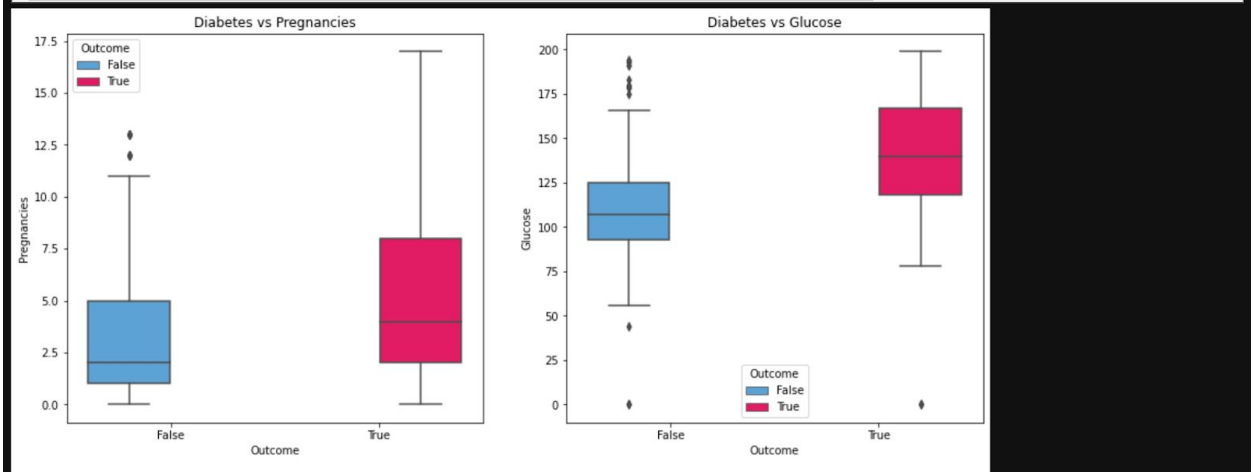
سپس با استفاده از دستور corr از کتابخانه پانداس مقادیر همبستگی بین دو به دوی پارامترها را به دست می‌آوریم با آن‌ها را رسم می‌کنیم.

```
corr = train_data.corr()
plt.figure(figsize=(20,10))
sns.heatmap(corr, cmap='coolwarm', annot = True)
plt.show()
```



سپس boxplot برای هر ویژگی را نسبت به ستون آخر که بیمار بودن یا نبودن نمونه را مشخص می‌کند رسم می‌کنیم؛ تمامی نمودارها در جوپیتر نوت بوک این سوال آمده‌است و در ادامه نمودار دو ویژگی آمده‌است.

```
attrs_data = header_data[:-1]
fig, axes = plt.subplots(4, 2, figsize=(15, 30))

for idx in range(0, len(attrs_data), 2):
    sns.boxplot(ax=axes[int(idx/2), 0], x=train_data["Outcome"], y=train_data[attrs_data[idx]], hue=train_data
    axes[int(idx/2), 0].set_title("Diabetes vs "+ attrs_data[idx], fontsize=12)
    sns.boxplot(ax=axes[int(idx/2), 1], x=train_data["Outcome"], y=train_data[attrs_data[idx+1]], hue=train_da
    axes[int(idx/2), 1].set_title("Diabetes vs "+ attrs_data[idx+1], fontsize=12)
```



۲.۱.۵

سپس با دستور mean میانگین هر ستون را به دست می‌آوریم و چاپ می‌کنیم و سپس تابعی به نام make_categorical ایجاد می‌کنیم، این تابع به این صورت عمل می‌کند که با گرفتن دیتا فریم و اسامی ستون‌ها یا همان ویژگی‌های آن، مقادیر مین و میانگین و ماکس هر ستون را به دست آورده با استفاده از تابع cut در کتابخانه پانداس داده‌های آن ستون را به دو بخش کمتر از میانگین و بیشتر از میانگین تبدیل کرده و به جای آن‌ها مقادیر low و high می‌گذارد. با استفاده از این تابع داده‌های آموزش و تست را به صورت categorical در می‌آوریم.

5.2

```
col_means = train_data.mean(axis=0)
col_means
```

```
Pregnancies                 3.801303
Glucose                   120.698697
BloodPressure              69.097720
SkinThickness              20.856678
Insulin                    78.412052
BMI                        31.853746
DiabetesPedigreeFunction    0.464230
Age                        33.086319
Outcome                     0.348534
dtype: float64
```

```
def make_categorical(data, header_data):
    for header in header_data[:-1]:
        minimum = data[header].min()
        mean = data[header].mean()
        maximum = data[header].max()
        bins = [minimum, mean, maximum]
        labels = ["low", "high"]
        data[header] = pd.cut(data[header], bins=bins, labels=labels, include_lowest=True)
    return data
```

```
make_categorical(train_data, header_data)
make_categorical(test_data, header_data)
train_data
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	low	high	high	high	low	high	low	high	False
1	high	low	low	low	low	low	high	low	False
2	low	high	high	low	low	high	low	low	False
3	high	low	high	low	low	high	low	low	False
4	low	low	low	high	high	high	high	low	True

برای مشخص کردن ویژگی‌ای که باید برای هر گره درخت انتخاب کنیم، نیاز به محاسبه آنتروپی داریم، برای این منظور تابع entropy را ایجاد می‌کنیم.

```
import math
def entropy(p=0, n=0):
    """This function gives the numbers of positive and negative samples
    and returns the entropy of the dataset."""
    s = p + n
    return 0 if p == 0 or n == 0 else -p/s*math.log2(p/s) - n/s*math.log2(n/s)
```

این تابع برای دسته‌بندی‌هایی با دو کلاس عمل می‌کند و با گرفتن تعداد اعضای مثبت و منفی مقدار آنتروپی را محاسبه می‌کند.
سپس تابعی به نام pseduo_info_gain ایجاد می‌کنیم که این تابع مقدار دقیق information gain را محاسبه نمی‌کند بلکه به صورت زیر عمل می‌کند که مقدار $H_s(Y|X_i)$ را محاسبه می‌کند و منطق کد برای انتخاب ویژگی، ویژگی‌ای است که مقدار خروجی تابع برای آن کمتر باشد.

$$j = \underset{i \in \text{remaining atts.}}{\text{argmax}} \; Gain(S, X_i)$$

$$= \underset{i \in \text{remaining atts.}}{\text{argmax}} \; H_S(Y) - H_S(Y|X_i)$$

$$= \underset{i \in \text{remaining atts.}}{\text{argmin}} \; H_S(Y|X_i)$$

```python
def pseudo_info_gain(attribute, outcome):
    """
    This function gives the attribute and outcome columns and returns a measure.
    Returned measure determines the quality of the data division based on the given attribute.
    The measure is interpreted by the lower the measure, the better the quality.
    """
    lvl_1 = outcome[attribute == "low"]
    lvl_1_cls_count = lvl_1.value_counts()
    ent_1 = entropy(*lvl_1_cls_count)
    lvl_2 = outcome[attribute == "high"]
    lvl_2_cls_count = lvl_2.value_counts()
    ent_2 = entropy(*lvl_2_cls_count)
    lvl_1_count = lvl_1_cls_count.sum()
    lvl_2_count = lvl_2_cls_count.sum()
    s = lvl_1_count + lvl_2_count
    pseudo_info_gain = lvl_1_count/s*ent_1 + lvl_2_count/s*ent_2
#     print(attribute.name, pseudo_info_gain)
    return pseudo_info_gain
```

سپس کلاس درخت تصمیم را ایجاد می‌کنیم؛ در سازنده این کلاس مقدار را ماکزیمم عمق را باید وارد کنیم. این کلاس تابعی دارد که بهترین ویژگی را انتخاب کند برای گره که به صورت زیر تعریف شده‌است.

```python
class DecisionTree():
    def __init__(self, max_depth):
        self.depth = 0
        self.max_depth = max_depth

    def best_split_attr(self, x, outcome):
        """
        This function give the feature matrix and outcome vector
        and returns the best attribute for splitting with maximum information gain.
        """
        min_gain = 1
        best_attr = None
        attributes = x.columns.to_list()
        for attr in attributes:
            gain = pseudo_info_gain(x[attr], outcome)
            if gain <= min_gain:
                min_gain = gain
                best_attr = attr
        return best_attr, min_gain

    def all_same(self, y):
        return y.eq(y.iloc[0]).all()
```

تابع all_same به این منظور تعریف شده‌است که میسنجد آیا تمام داده‌های مانده در گره، در یک کلاس قرار میگیرند یا خیر.
سپس برای ساخت درخت با داده‌ها نیاز به تابع fit داریم، این تابع با استفاده از توابع تعریف شده در بالا، ویژگی هر گره را انتخاب کرده و سپس فرزندان هر گره را نیز به صورت بازگشتی میسازد تا به شرایط پایانی برسد که یا خالی شدن نمونه‌ها در گره یا خالی شدن ویژگی‌ها یا یکسان شدن خروجی تمام نمونه‌ها در گره و یا رسیدن به حد اکثر عمق است.

```
def fit(self, x, y, param_node={}, depth=0):
    """
    This function gives the feature matrix, outcome vector, parameter node, and depth
    and returns the decision tree fitted on the given data.
    inputs:
    x: feature matrix
    y: outcome vector
    par_node: parameter node dictionary
    depth: depth integer
    """
    if y.size == 0:
        return None
    if x.size == 0:
        return {"value": y.mode()[0]}
    if self.all_same(y):
        return {"value": y.iloc[0]}
    if depth >= self.max_depth:
        return None
    else:
        attrs = x.columns.to_list()
        split_col = self.best_split_attr(x, y)[0]
        y_left = y[x[split_col] == "low"]
        y_right = y[x[split_col] == "high"]
        child_attrs = attrs
        child_attrs.remove(split_col)
        x_left = x[x[split_col] == "low"]
        x_right = x[x[split_col] == "high"]
        param_node = {"splitting column": split_col, "value": y.mode()[0]}
        param_node["left"] = self.fit(x_left[child_attrs], y_left, {}, depth+1)
        param_node["right"] = self.fit(x_right[child_attrs], y_right, {}, depth+1)
        self.depth += 1
        self.dec_tree = param_node
        return param_node
```

در ادامه تابع query تعریف شده‌است که این تابع خروجی یک نمونه را با پیمایش درخت تصمیم، پیش‌بینی می‌کند.

```
def query(self, sample):
    """
    This function predicts the outcome of a single sample.
    """
    pred_tree = self.dec_tree
    for itr in range(self.max_depth+1):
        left = pred_tree.get('left', None)
        right = pred_tree.get('right', None)
        if left == None and right == None:
            return pd.Series(pred_tree['value'])

        else:
            level = sample[pred_tree['splitting column']]
            if level == "low":
                if left == None:
                    return pd.Series(pred_tree['value'])
                else:
                    pred_tree = pred_tree['left']
            else:
                if right == None:
                    return pd.Series(pred_tree['value'])
                else:
                    pred_tree = pred_tree['right']
```

تابع predict که در ادامه آمده‌است، به این صورت عمل می‌کند که نمونه‌ها را می‌گیرد و بردار خروجی را پیش‌بینی می‌کند.

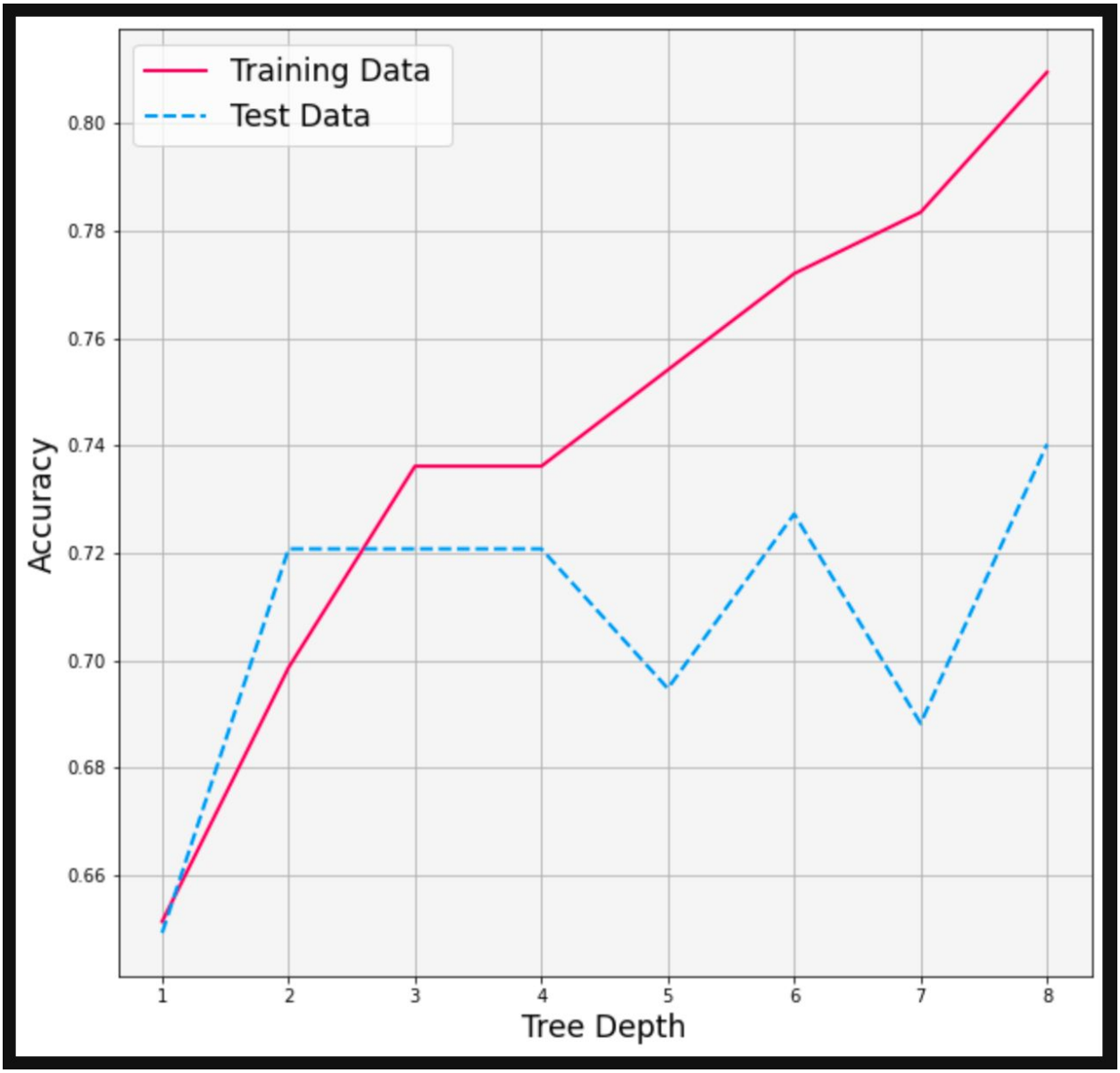
```
    def predict(self, x):
        """
        This function gives the feature matrix
        and returns the predicted outcome for every sample in the matrix.
        """
        predictions = pd.Series(dtype=bool)
        for sample in x.iterrows():
            predictions = predictions.append(self.query(sample[1]), ignore_index=True)
        return predictions


outcome = train_data["Outcome"]
max_depth = 5
clf = DecisionTree(max_depth)
m = clf.fit(train_data[header_data[:-1]], outcome)
from pprint import pprint
pprint(m)
```

۳.۱.۵

تابع دقت را تعریف می‌کنیم که مقادیر درست پیش‌بینی شده را بر تعداد کل داده تقسیم می‌کند، سپس مقدار دقت را روی داده‌های آموزش و تست به ازای عمق‌های بین ۱ تا ۸ به دست می‌آوریم و آن‌ها را رسم می‌کنیم.

5.3

```
def accuracy(predict, outcome):
    correct = predict == outcome
    return correct.sum()/outcome.size

train_data = pd.read_csv("Diabetes/Train.csv")
test_data = pd.read_csv("Diabetes/Test.csv")
header_data = train_data.columns.to_list()
make_categorical(train_data, header_data)
make_categorical(test_data, header_data)
train_acc_depth = {}
outcome_train = train_data["Outcome"]
train_data = train_data[header_data[:-1]]
test_acc_depth = {}
outcome_test = test_data["Outcome"]
test_data = test_data[header_data[:-1]]
for max_depth in range(1, 9):
    clf = DecisionTree(max_depth)
    clf.fit(train_data, outcome_train)
    train_acc_depth[str(max_depth)] = accuracy(clf.predict(train_data), outcome_train)
    test_acc_depth[str(max_depth)] = accuracy(clf.predict(test_data), outcome_test)
```



دیده می‌شود که با اضافه شدن عمق درخت، پیچیدگی مدل بیشتر شده و مدل برای داده آموزش بهتر عمل می‌کند ولی در مواقعی دیده می‌شود که دقت مدل برای داده تست کم می‌شود که احتمالا به دلیل اورفیت شدن مدل بر روی داده آموزش است ولی در نهایت دقت برای هر دو داده آموزش و تست بیشتر می‌شود که احتمالا نشان از آن است که از توزیع‌های نزدیک به هم داده‌های آموزش و تست آمده‌اند.

۴.۱.۵

در این بخش برای cross_validation ابتدا داده‌ها را shuffle می‌کنیم و سپس بخش‌های متفاوت داده را به عنوان بخش validation انتخاب کرده، درخت را بر روی سایر بخش‌ها می‌سازیم و سپس دقت آن‌را برای بخش validation محاسبه می‌کنیم و دقت محاسبه‌شده به ازای هر بخش validation را با میانگین میگیریم و این کار را برای تمام عمق‌ها تکرار می‌کنیم و سپس عمقی که بالاترین دقت دارد را انتخاب می‌کنیم. بهترین عمق ۶ به دست آمده‌است.

5.4

```
train_data = pd.read_csv("Diabetes/Train.csv")
test_data = pd.read_csv("Diabetes/Test.csv")
header_data = train_data.columns.to_list()
make_categorical(train_data, header_data)
make_categorical(test_data, header_data)
#shuffle the training data
train_data = train_data.sample(frac = 1).reset_index(drop = True)
n_samples = train_data.shape[0]
import numpy as np
boundaries = np.linspace(0, n_samples, 6, dtype=np.int64)
c_val_acc_depth = {}
for max_depth in range(1, 9):
    clf = DecisionTree(max_depth)
    valid_accs = []
    for idx in range(len(boundaries)-1):
        c_valid = train_data.iloc[boundaries[idx]:boundaries[idx+1]]
        c_train_data = train_data.drop(np.r_[boundaries[idx]:boundaries[idx+1]], axis= 0)
        c_valid_out = c_valid["Outcome"].reset_index(drop=True)
        c_valid_features = c_valid[header_data[:-1]].reset_index(drop=True)
        c_train_out = c_train_data["Outcome"].reset_index(drop=True)
        c_train_features = c_train_data[header_data[:-1]].reset_index(drop=True)
        clf.fit(c_train_features, c_train_out)
        valid_accs.append(accuracy(clf.predict(c_valid_features), c_valid_out))
    c_val_acc_depth[str(max_depth)] = np.array(valid_accs).mean()
print(c_val_acc_depth)
best_depth = int(max(c_val_acc_depth, key=c_val_acc_depth.get))
print("Best Depth:", best_depth)
```

```
{'1': 0.651499400239904, '2': 0.6987205117952818, '3': 0.7361988537918166, '4': 0.7117686258829802, '5': 0.7232040517126482, '6': 0.7036518725842995, '7': 0.6922431027588964, '8': 0.6841263494602159}
Best Depth: 3
```

سپس درخت را روی کل داده آموزش با عمق ۶ به دست آمده می‌سازیم و سپس مقادیر مختلف ارزیابی را برای این درخت محاسبه می‌کنیم.

```
#Precision = TP/(TP+FP)
correct = test_outcome == predicts
tp = predicts[correct].sum()
tp_fp = predicts.sum()
precision = tp/tp_fp
print("Precision:", precision)
```

```
Precision: 0.627906976744186
```

```
#Recall = TP/(TP+FN)
wrong = test_outcome != predicts
fn = predicts[wrong].size - predicts[wrong].sum()
tp_fn = tp + fn
recall = tp/tp_fn
print("Recall:", recall)
```

```
Recall: 0.5
```

```
f_score = 2*precision*recall/(precision+recall)
print("f_score:", f_score)
```

```
f_score: 0.5567010309278351
```

```
#Sensitivity equals to Recall
sensitivity = recall
print("Sensitivity:", sensitivity)
```

```
Sensitivity: 0.5
```

```
#Specificity = TN/(TN+FP)
tn = predicts[correct].size - predicts[correct].sum()
fp = predicts[wrong].sum()
tn_fp = tn+fp
specificity = tn/(tn_fp)
print("Specificity:", specificity)
```

```
Specificity: 0.84
```

معیار دقت به این صورت است که مقادیری که درست پیش‌بینی می‌شوند به کل داده محاسبه می‌شود ولی اگر فرض کنیم که داده‌های بیمارانی را بررسی می‌کنیم که مشکوک به سرطان هستند. ۱۰۰ داده داریم که ۳۰ تای آن‌ها مبتلا به سرطان و ۷۰ تای آن‌ها سالم هستند حال اگر مدل ما ۲۵ نمونه سرطانی و ۶۵ مورد سالم تشخیص دهد این مدل دقتی برابر ۹۰ درصد دارد. ولی با توجه به اینکه ۵ مورد سرطانی را تشخیص نداده‌است مرتکب خطای بسیار شده‌است. در چنین مواقعی از معیار f-score استفاده می‌کنیم.
معیار دقت در مواقعی کارآمد است که هر دو کلاس از اهمیت یکسانی برخوردار باشند و معیار f-score زمانی مناسب است که مقادیر false negative و false positive اهمیت بیشتری دارند.

معیار sensitivity نسبت داده‌های درست تشخیص داده شده مثبت را به کل داده‌های مثبت می‌سنجد و مدلی با معیار sensitivity بالا، افراد با وضعیت مثبت مانند بیماران را به خوبی تشخیص می‌دهد و تعداد کمی false negative تولید می‌کند.

معیار specificity نسبت داده‌های درست تشخیص داده شده منفی را به کل داده‌های منفی می‌سنجد و مدلی با معیار specificity بالا، افراد با وضعیت منفی مانند افراد سالم را به خوبی تشخیص می‌دهد و تعداد کمی false positive تولید می‌کند.

۲.۵
۱.۲.۵

در این بخش stochastic gradient descent را پیادسازی می‌کنیم. در ابتدا داده‌ها را خوانده و سپس ردیف‌هایی که مقادیر هر یک از Glocuse ،BMI و Blood Pressure برای آن صفر باشد را از داده حذف می‌کنیم و سپس ستون خروجی را به صورت ۱و ۱- در می‌آوریم و سپس داده‌ها را نرمال می‌کنیم.

Perceptron

1. Stochastic Gradient Descent

```
# Packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Data Preperation
train_data = pd.read_csv("Diabetes/Train.csv")
test_data = pd.read_csv("Diabetes/Test.csv")
train_data = train_data[train_data["BMI"] != 0]
train_data = train_data[train_data["Glucose"] != 0]
train_data = train_data[train_data["BloodPressure"] != 0]
train_data["Outcome"].replace([True, False], [1, -1], inplace=True)
train_data = train_data.reset_index(drop=True)
# Min-Max Scaler (Normalization)
headers = train_data.columns.to_list()
attrs = headers[:-1]
for attr in attrs:
    col_min = train_data[attr].min()
    col_max = train_data[attr].max()
    train_data[attr] = (train_data[attr] - col_min) / (col_max - col_min)
train_data.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	579.000000	579.000000	579.000000	579.000000	579.000000	579.000000	579.000000	579.000000	579.000000
mean	0.224017	0.500630	0.493691	0.219850	0.098288	0.288580	0.176751	0.202763	-0.316062
std	0.199791	0.196790	0.125150	0.157958	0.135714	0.138541	0.142697	0.200166	0.949559

داده‌های آموزش و validation را جدا می‌کنیم و سپس به هر دو داده ستونی برابر با ۱ اضافه می‌کنیم برای مقدار bias.

```
# Train Validation Split
valid_data = train_data.sample(frac=0.15, random_state=42)
valid_indices = valid_data.index.to_list()
train_data.drop(index=valid_indices, inplace=True)
train_data.reset_index(drop=True, inplace=True)
valid_data.reset_index(drop=True, inplace=True)

train_expect = train_data["Outcome"]
train_feature = train_data.drop(labels="Outcome", axis=1)
train_feature.insert(0, "Ones", 1)
valid_expect = valid_data["Outcome"]
valid_feature = valid_data.drop(labels="Outcome", axis=1)
valid_feature.insert(0, "Ones", 1)

def accuracy(predict, outcome):
    correct = predict == outcome
    return correct.sum()/outcome.size


def predict(x, w):
    return np.sign(x@w)[0]
```

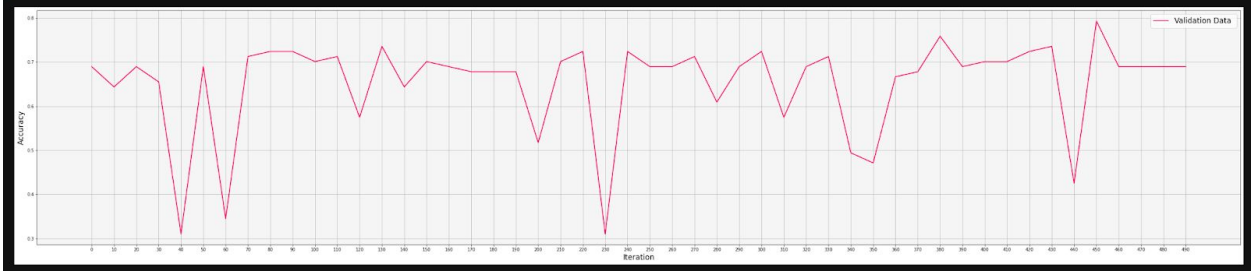
در ادامه تابع fit را تعریف می‌کنیم که با نشان دادن هر نمونه به مدل وزن‌ها را آپدیت می‌کند و وزن‌ها را در یک دیکشنری میریزد و سپس تابع evaluate با داشتن این دیکشنری، بر روی داده validation با هر یک از وزن‌های دیکشنری ورودی مقدار دقت را محاسبه می‌کند و بهترین وزن و دقت را محاسبه می‌کند.

```
def fit(train_feature, train_expect, n_iteration=100, learning_rate=0.01):
    iter_weights = {}
    d = train_feature.iloc[0].shape[0]
    w = np.zeros((d, 1))
    n = train_feature.shape[0]
    for t in range(n_iteration):
        i = t % n
        x = train_feature.iloc[i]
        y = train_expect.iloc[i]
        y_hat = predict(x, w)
        if y_hat != y:
            w = w + learning_rate*x.values.reshape((x.size, 1))*y
        iter_weights[str(t)] = w
    return iter_weights


iter_weights = fit(train_feature, train_expect, n_iteration=500)

def evaluate(valid_feature, valid_expect, iter_weights):
    iter_acc = {}
    max_accr, best_weight = 0, None
    for iteration, weight in iter_weights.items():
        accr = accuracy(predict(valid_feature, weight), valid_expect)
        if int(iteration) % 10 == 0:
            iter_acc[iteration] = accr
        if accr > max_accr:
            max_accr = accr
            best_weight = weight
    return iter_acc, best_weight, max_accr
```

سپس دقت را به ازای هر بردار وزن ایجاد شده با دیدن یک نمونه، روی داده validation رسم می‌کنیم.

```
iter_acc, weight, max_accr = evaluate(valid_feature, valid_expect, iter_weights)

fig, ax = plt.subplots(figsize=(50, 10))
ax.plot(iter_acc.keys(), iter_acc.values(),
        label="Validation Data ", color="#FF005E", linewidth=2)
ax.grid(True)
ax.legend(fontsize="xx-large")
ax.set_facecolor('#f5f5f5')
ax.set_ylabel("Accuracy", fontsize="xx-large")
ax.set_xlabel("Iteration", fontsize="xx-large")
```

```
Text(0.5, 0, 'Iteration')
```



در انتها داده تست را آماده کرده و با بهترین وزن به دست آمده از مرحله قبل، داده‌های تست را پیش بینی می‌کنیم و معیارهای متفاوت ارزیابی را برای مدل روی داده تست محاسبه می‌کنیم.

```
tp_fp = predicts == 1
tp_fp = tp_fp.sum()
precision = tp/tp_fp
print("Precision:", precision)
#Recall = TP/(TP+FN)
wrong = predicts != test_expect
fn = predicts[wrong]
fn = fn == -1
fn = fn.sum()
tp_fn = tp+fn
recall = tp/tp_fn
print("Recall:", recall)
#F-Score
f_score = 2*precision*recall/(precision+recall)
print("f_score:", f_score)
# Sensitivity equals to Recall
sensitivity = recall
print("Sensitivity:", sensitivity)
#Specificity = TN/(TN+FP)
neg = predicts == -1
neg = neg.sum()
tn = neg - fn
pos = predicts == 1
pos = pos.sum()
fp = pos - tp
tn_fp = tn+fp
specificity = tn/(tn_fp)
print("Specificity:", specificity)
```

```
Precision: 0.7021276595744681
Recall: 0.6470588235294118
f_score: 0.673469387755102
Sensitivity: 0.6470588235294118
Specificity: 0.851063829787234
```

۲.۲.۵

در این بخش باید الگوریتم perceptron را به صورت Full Batch پیادهسازی کنیم که بسیار شبیه به بخش قبل است و تنها تفاوت، این است که با دیدن تمام دادهها بردار وزن را آپدیت می‌کنیم.

```python
#Full Batch
def fit(feature, expect, n_epoch=10, learning_rate=0.01):
    epoch_weight = {}
    d = feature.shape[1]
    w = np.random.random((d, 1))
    for epoch in range(1, n_epoch+1):
        x = feature
        y = expect
        y_hat = predict(x, w)
        wrong = y_hat != y
        x = x[wrong]
        y = y[wrong]
        w = w + (learning_rate*x.T@y).values.reshape((d, 1))
        epoch_weight[str(epoch)] = w
    return epoch_weight

epoch_weights = fit(train_feature, train_expect, n_epoch=20)
train_feature
```

	Ones	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	1	0.058824	0.774194	0.591837	0.434343	0.079196	0.298569	0.119005	0.483333
1	1	0.058824	0.606452	0.591837	0.000000	0.000000	0.447853	0.071493	0.116667
2	1	0.117647	0.316129	0.408163	0.323232	0.189125	0.404908	0.269683	0.033333
3	1	0.058824	0.367742	0.265306	0.151515	0.042553	0.122699	0.202715	0.083333
4	1	0.117647	0.438710	0.520408	0.323232	0.000000	0.357873	0.031674	0.000000

در انتها مانند بخش قبل معیارهای متفاوت ارزیابی را روی داده تست با وزن به دست آمده محاسبه می‌کنیم.

```
Precision: 0.5581395348837209
```

```
#Recall = TP/(TP+FN)
wrong = predicts != test_expect
fn = predicts[wrong]
fn = fn == -1
fn = fn.sum()
tp_fn = tp+fn
recall = tp/tp_fn
print("Recall:", recall)
```

```
Recall: 0.47058823529411764
```

```
f_score = 2*precision*recall/(precision+recall)
print("f_score:", f_score)
```

```
f_score: 0.5106382978723404
```

```
#Sensitivity equals to Recall
sensitivity = recall
print("Sensitivity:", sensitivity)
```

```
Sensitivity: 0.47058823529411764
```

```
#Specificity = TN/(TN+FP)
neg = predicts == -1
neg = neg.sum()
tn = neg - fn
pos = predicts == 1
pos = pos.sum()
fp = pos - tp
tn_fp = tn+fp
specificity = tn/(tn_fp)
print("Specificity:", specificity)
```

```
Specificity: 0.7978723404255319
```

۳.۲.۵

در این بخش نیز به جای اینکه با دیدن کل داده و یا یک نمونه، مقادیر وزن‌ها را آپدیت کنیم، هر بار با دیدن بخشی از داده (Mini Batch) وزن‌ها را آپدیت می‌کنیم.

```python
#Mini Batch
def fit(feature, expect, n_epoch=10, learning_rate=0.01, n_batch=5):
    batch = 1
    batch_weight = {}
    d = feature.shape[1]
    w = np.random.random((d, 1))
    n = train_feature.shape[0]
    indices = np.linspace(0, n, num= n_batch+1, dtype=np.int32)
    for epoch in range(1, n_epoch+1):
        random_state = np.random.randint(51)
        feature = feature.sample(frac=1, random_state=random_state)
        expect = expect.sample(frac=1, random_state=random_state)
        for idx in range(len(indices)-1):
            x = feature.iloc[indices[idx]:indices[idx+1]]
            y = expect.iloc[indices[idx]:indices[idx+1]]
        y_hat = predict(x, w)
        wrong = y_hat != y
        x = x[wrong]
        y = y[wrong]
        w = w + (learning_rate*x.T@y).values.reshape((d, 1))
        batch_weight[str(batch)] = w
        batch += 1
    return batch_weight

batch_weights = fit(train_feature, train_expect, n_epoch=20)
```

در انتها مانند بخش قبل معیارهای متفاوت ارزیابی را روی داده تست با وزن به دست آمده محاسبه می‌کنیم.

```
Precision: 0.5581395348837209
```

```
#Recall = TP/(TP+FN)
wrong = predicts != test_expect
fn = predicts[wrong]
fn = fn == -1
fn = fn.sum()
tp_fn = tp+fn
recall = tp/tp_fn
print("Recall:", recall)
```

```
Recall: 0.47058823529411764
```

```
f_score = 2*precision*recall/(precision+recall)
print("f_score:", f_score)
```

```
f_score: 0.5106382978723404
```

```
#Sensitivity equals to Recall
sensitivity = recall
print("Sensitivity:", sensitivity)
```

```
Sensitivity: 0.47058823529411764
```

```
#Specificity = TN/(TN+FP)
neg = predicts == -1
neg = neg.sum()
tn = neg - fn
pos = predicts == 1
pos = pos.sum()
fp = pos - tp
tn_fp = tn+fp
specificity = tn/(tn_fp)
print("Specificity:", specificity)
```

```
Specificity: 0.7978723404255319
```