به نام خدا

ارائه پروژه های مربوط به svm و decision tree

علیرضا اسلامی خواه

99521064

استاد عبدی

ترم 1401-1402

در ابتدا:

پروژه مربوط به درخت تصمیم

اول : درخت مربوط به داده های رستوران

در ابتدا ما یک فایل اکسل برای ذخیره داده ها برای استفاده از آنها درست کردیم که مانند جزوه میبود.

	A	B	C	D	E	F	G	H	I	J	K
1	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	Wait
2	Yes	No	No	Yes	Some	3	No	Yes	French	0-10	Yes
3	Yes	No	No	Yes	Full	1	No	No	Thai	30-60	No
4	No	Yes	No	No	Some	1	No	No	Burger	0-10	Yes
5	Yes	No	Yes	Yes	Full	1	Yes	No	Thai	30-10	Yes
6	Yes	No	Yes	No	Full	3	No	Yes	French	>60	No
7	No	Yes	No	Yes	Some	2	Yes	Yes	Italian	0-10	Yes
8	No	Yes	No	No	None	1	Yes	No	Burger	0-10	No
9	No	No	No	Yes	Some	2	Yes	Yes	Thai	0-10	Yes
10	No	Yes	Yes	No	Full	1	Yes	No	Burger	>60	No
11	Yes	Yes	Yes	Yes	Full	3	No	Yes	Italian	30-10	No
12	No	No	No	No	None	1	No	No	Thai	0-10	No
13	Yes	Yes	Yes	Yes	Full	1	No	No	Burger	30-60	Yes
14											

سپس شروع به درست کردن درخت تصمیم گرفتیم.

اصلی ترین تابع درخت تصمیم همانطور که در شکل زیر هم نشان داده میشود تابع مربوط به درخت تصمیم است.

```
function DECISION-TREE-LEARNING(examples, attributes, parent_examples) returns a tree

  if examples is empty then return PLURALITY-VALUE(parent_examples)
  else if all examples have the same classification then return the classification
  else if attributes is empty then return PLURALITY-VALUE(examples)
  else
      A ← argmax_{a ∈ attributes} IMPORTANCE(a, examples)
      tree ← a new decision tree with root test A
      for each value v_k of A do
          exs ← {e : e ∈ examples and e.A = v_k}
          subtree ← DECISION-TREE-LEARNING(exs, attributes − A, examples)
          add a branch to tree with label (A = v_k) and subtree subtree
      return tree
```

که ما این تابع را به همین اسم پیاده سازی کردیم.

!! توجه جزئیات تمام کدها به صورت کامنت در کد نوشته شده است و اینجا موضوعات کلی گفته میشود.!!

تابع بعدی که بسیار مهم بود تابعی به اسم entropy بود که مانند شکل زیر پیاده سازی شد.

$$\text{Entropy:} \quad H(V) = \sum_k P(v_k) \log_2 \frac{1}{P(v_k)} = -\sum_k P(v_k) \log_2 P(v_k)$$

که بعدا هم در بدست آوردن gain یا همان ارزش هر ویژگی مورد استفاده قرار گرفت.

$$Gain(A) = B(\frac{p}{p+n}) - Remainder(A)$$

کد آن مانند شکل زیر پیاده سازی شده است.

```
48  def Entropy(Example , attribute):
49      c = Example[attribute]              # مقادیر ویژگی
50      c , remainder = pd.factorize(c)     # تبدیل مقادیر به عدد
51      Vk = np.bincount(c)                 # تعداد مقادیر
52      Entropy = 0                         # انتروپی
53      pb = Vk / len(c)                    # احتمال هر مقدار
54      for p in pb:                        # برای هر مقدار
55          if p > 0 :
56              Entropy = Entropy + p*np.log2(p)  # انتروپی را محاسبه میکنیم
57      return -Entropy                     # انتروپی را برمیگردانیم
58
```

سپس به پیاده سازی gain یا همان معیار تشخیص ویژگی بعدی پرداختیم.

```
57      return -Entropy                     # انتروپی را برمیگردانیم
58
59  def ig(table , attribute , y):
60      parent_entropy = Entropy(table , y)   #parent entropy
61      d = table[attribute].unique()         # مقادیر ویژگی
62      Vk = list()                           # جدول های مقادیر ویژگی
63      for Value in d:                       # برای هر مقدار ویژگی
64          Vk.append(table[table[attribute] == Value]) # جدول های مقادیر ویژگی را میسازیم
65      Remainder = 0                         # مقدار باقی مانده
66      for i in range(len(Vk)):              # برای هر جدول
67          prob = Vk[i].shape[0]/table.shape[0]        # احتمال مقدار ویژگی
68          Remainder = Remainder + prob*Entropy(Example=Vk[i] , attribute=y)   # مقدار باقی مانده را محاسبه میکنیم
69      return parent_entropy - Remainder                                       # اینفورمیشن گین را برمیگردانیم
70
```

در کنار همه این توابع ما نیاز داشتیم گره ها را با استفاده از کلاسی تعریف کنیم که یک ساختار کلی داشته باشد.

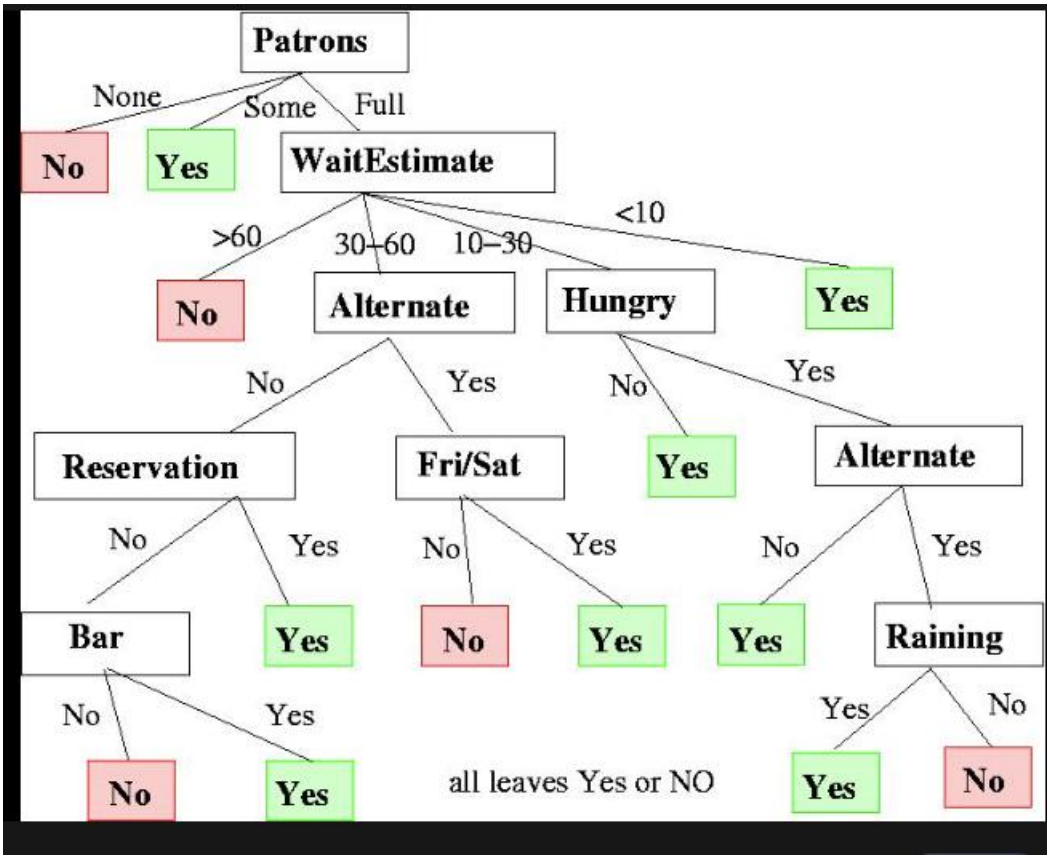
این ساختار شامل بچه ها ، ویژگی ، ارزش ها و .. میباشد.

```python
class node:
    def __init__(self , attribute, table = None, Value = None, children = None, ig = None):
        self.attribute = attribute      # نام ویژگی
        self.table = None               # جدول داده ها
        self.Value = []                 # مقادیر ویژگی
        self.children = []              # فرزندان
        self.ig = None          # اطلاعات گرفته شده

```

توابع classification و plurarity هم مانند آنچه در جزوه گفته شد پیاده سازی شدند.

```python
def PluralityValue(examples , y):
    number = examples.shape[0]                  # تعداد مثال ها
    results = list(examples[y])                 # مقادیر متغیر هدف
    attr1 = results[0]                          # مقدار اول
    attr2 = results[1]                          # مقدار دوم
    count1 = 0                                  # تعداد مقدار اول
    count2 = 0                                  # تعداد مقدار دوم
    for i in range(number):                     # برای هر مثال
        if results[i] == attr1:                 # اگر مقدار اول بود
            count1 += 1                         # تعداد اول را افزایش میدهیم
        elif results[i] == attr2:               # اگر مقدار دوم بود
            attr2 = results[i]                  # مقدار دوم را تغییر میدهیم
            count2 += 1
    if count1 > count2:
        return node(attr1)                      # اگر تعداد اول بیشتر بود یک گره با مقدار اول برمیگرداند
    else :
        return node(attr2)                      # اگر تعداد دوم بیشتر بود یک گره با مقدار دوم برمیگرداند


def clasification(Examples , y):
    if Examples.empty:                          # اگر جدول خالی بود
        return
    length = Examples.shape[0]                          # تعداد مثال ها
    res = list(Examples[y])                             # مقادیر متغیر هدف
    Value = res[0]                                      # مقدار اول
    for i in range(length):                             # برای هر مثال
        if res[i] != Value:                             # اگر مقدار متغیر هدف مثال متفاوت از مقدار اول بود
            return -1                                   # برمیگرداند -1
    return res[0]                                       # مقدار اول را برمیگرداند
```

در انتها هم به ران کردن کد و رفع اشکال های آن پرداختیم که همچین خروجی را به ما داد.



که به صورت دقیق تر ترمینال خروجی هر یک از گره ها چاپ میشود.

```
C:\Users\lenovo\Desktop\DT>python -u "c:\Users\lenovo\Desktop\DT\Desicion_Tree.py"
The root is Pat and the IG is 0.5408520829727552 and the Entropy is 1.0 and the number of examples is 12

The child is Yes and the IG is None and the Entropy is -0.0 and the number of examples is 4

The child is Hun and the IG is 0.2516291673878229 and the Entropy is 0.9182958340544896 and the number of examples is 6

The child is No and the IG is None and the Entropy is -0.0 and the number of examples is 2

The child is Type and the IG is 0.5 and the Entropy is 1.0 and the number of examples is 4

The child is No and the IG is None and the Entropy is -0.0 and the number of examples is 2

The child is Yes and the IG is None and the Entropy is 0 and the number of examples is 0

The child is Fri and the IG is 1.0 and the Entropy is 1.0 and the number of examples is 2

The child is Yes and the IG is None and the Entropy is -0.0 and the number of examples is 1

The child is No and the IG is None and the Entropy is -0.0 and the number of examples is 1

The child is No and the IG is None and the Entropy is -0.0 and the number of examples is 1

The child is Yes and the IG is None and the Entropy is -0.0 and the number of examples is 1
```

که اطلاعاتی را شامل خود ویژگی گره و gain آن و آنتروپی آن به ما میدهد.
در انتها هم تعداد نمونه ها یا همان examples را میدهد.

درخت تصمیم برای نمونه دیابت:

تمامی بخش ها شبیه بخش قبل بود با چند تفاوت:

اول اینکه ما نیاز داشتیم داده ها را گسسته سازی کنیم. برای این کار روش زیر را انجام دادیم :

```
در این تابع ابتدا بعد از گرفتن ستون دلخواه ابتدا کوچکترین و بزرگترین را
    در آن مشخص کرده و سپس با استفاده از این دو متغیر مقدار گام را محاسبه میکنیم
    دانستن مقدار گام به ما کمک میکند که بفهمیم چه مقادیری در چه بازه ای قرار دارند
    برای گذاشتن مقادیر در بازه های مختلف از حلقه های دوتایی استفاده کردیم
    و همانگونه که در کد پایین هم قبل مشاهده است ما مقادیر را در بازه های مناسب قرار میدهیم
```

منظور از گسسته سازی اینست که ما مثلا برای داده ها یک بازه تعیین کردیم و با استفاده از → گفتیم که عدد مشخص شده در سطر در این بازه قرار دارد.

گسسته سازی باعث شد ما بتوانیم به نوعی مقادیر را صفر و یکی کنیم.

نمونه خروجی کد برای دیابت:

```
The child is 0 and the IG is None and the Entropy is 0 and the number of examples is 0

The child is 0 and the IG is None and the Entropy is 0 and the number of examples is 0

The child is 0 and the IG is None and the Entropy is 1.0 and the number of examples is 4

The child is 0 and the IG is None and the Entropy is 0 and the number of examples is 0

The child is 1 and the IG is None and the Entropy is 0.9709505944546686 and the number of examples is 5

The child is 1 and the IG is None and the Entropy is 0 and the number of examples is 0

The child is 1 and the IG is None and the Entropy is 0 and the number of examples is 0

The child is 0 and the IG is None and the Entropy is 0.7219280948873623 and the number of examples is 5

The child is 0 and the IG is None and the Entropy is 0 and the number of examples is 0

The child is 0 and the IG is None and the Entropy is 0 and the number of examples is 0

The number of nodes is 354
deghat dar train: 0.7785016286644951
 deghat dar test: 0.6948051948051948
```

که در دو خط آخر با استفاده از تابع fit که مشخص کردیم میزان دقت درخت های ساخته شده را سنجیدیم.

توجه شود که در تابع main تعریف شده ما 80 درصد را برای آموزش و 20 درصد را برای آزمایش تعریف کردیم.

میزان دقت درخت ها هم با استفاده از همین فرمول بدست آوردیم.

Accuracy= (TP + TN) / (Total number of observation)

پروژه مربوط به svm

هدف اصلی SVM به حداکثر رساندن حاشیه بین دو کلاس مختلف است. این بدان معناست که شما می‌خواهید مطمئن شوید که به تعداد نقاط یک کلاس در یک طرف مرز تصمیم‌گیری و به همان تعداد نقطه در کلاس دیگر در طرف دیگر قرار دارند.

هنگامی که این اتفاق می افتد، تمام نقاط با درجه جدایی بالاتر به درستی طبقه بندی می شوند در حالی که تمام نقاط با درجه جدایی پایین تر به اشتباه طبقه بندی می شوند.
ابتدا یک کلاس SVM ایجاد کردیم و مقداری را مقداردهی اولیه کردیم.
عبارت C به عنوان عبارت خطا شناخته می شود که برای بهینه سازی تابع باید اضافه کنیم

```python
# svm.py

import numpy as np


class SVM:

    def __init__(self, C = 1.0):
        # C = error term
        self.C = C
        self.w = 0
        self.b = 0
```

سپس به بررسی تابع fit میپردازیم:

آنچه در این روش برازش اتفاق می افتد واقعاً ساده است، ما در تلاش هستیم تا تلفات را در تکرارهای متوالی کاهش دهیم و بهترین پارامترهای w و b را پیدا کنیم. توجه داشته باشید که در اینجا از Batch Gradient Descent استفاده می کنیم. وزن‌ها (w) و سوگیری (b) در هر تکرار با استفاده از گرادیان‌ها و نرخ یادگیری به‌روزرسانی می‌شوند که منجر به به حداقل رساندن ضرر می‌شود. هنگامی که پارامترهای بهینه یافت می شوند، روش به سادگی آن را همراه با تلفات برمی گرداند.

سپس با استفاده از کتابخانه mtplotlib شروع به به تصویر کشیدن داده ها و محدوده آن ها کردیم.

نمونه خروجی نشان داده شده:

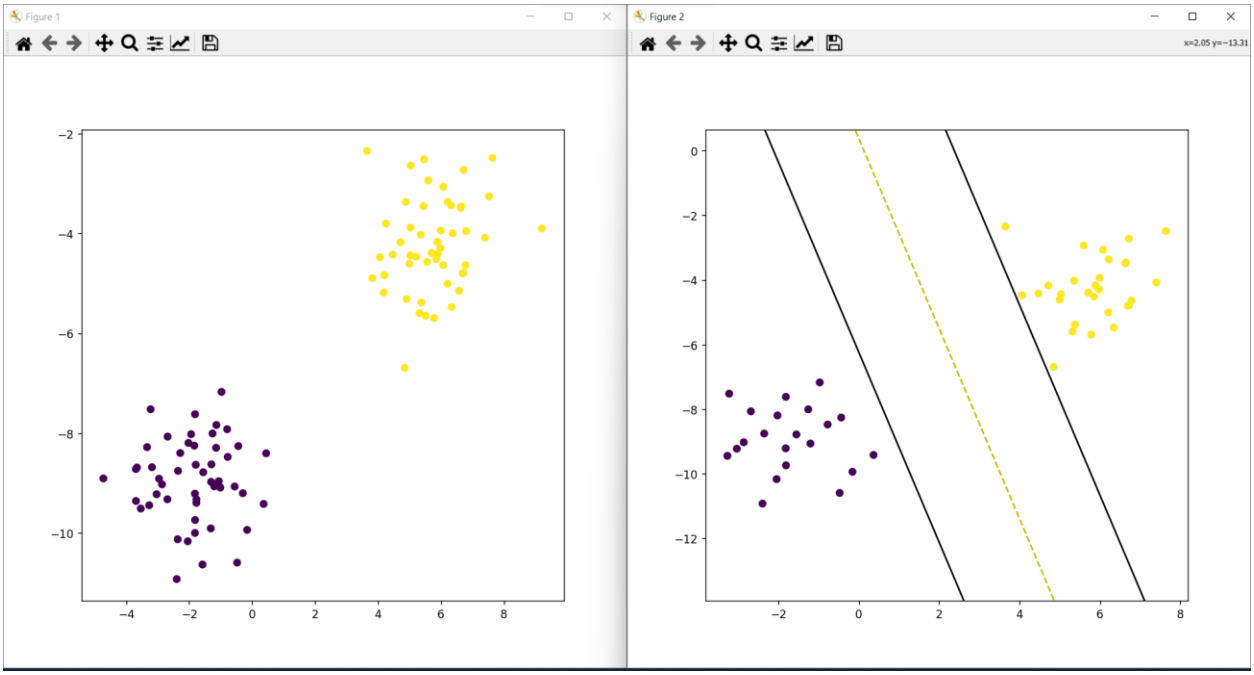


Figure 1
Figure 2
x=2.05 y=−13.31