پروژه دوم

پیاده سازی درخت تصمیم

درس هوش مصنوعی

نگین حقیقی - 99521226

استاد درس: دکتر آرش عبدی

نیم سال اول 1402-1401

سخن آغازین:

در این پروژه قصد داریم درخت تصمیم را پیاده سازی کنیم. و برای بررسی صحت کد، ابتدا داده های 12 گانه مثال رستوران را آزمایش کرده و سپس در مرحله دوم، داده های مربوط به تشخیص دیابت مورد استفاده قرار خواهد گرفت.

در مرحله اول داده ها را به دو مجموعه آموزشی و آزمایشی تقسیم میکنیم. (در این کد، به نسبت ٪80 آموزشی و ٪20 آزمایشی تقسیم کردم)

ورودی به فرمت (x,y) است که x برداری از مقادیر ویژگی های ورودی و y یک مقدار دودوئی خروجی است.

خروجی برنامه، درخت پیدا شده در مرحله اول و دوم است که به ترتیب از root چاپ شده و فرزندانش را چاپ میکند. در هر گره، مشخص شده برگ است یا نه و اگر نیست، مشخص شده کدام ویژگی در آن تست میشود و مقدار دست آورد اطلاعات و آنتروپی در زیرشاخه هایش ذکر شده و همچنین دقت خروجی نمایش داده میشود.

پیاده سازی و توابع:

ابتدا کلاس های Treenode و DesitionTree را پیاده سازی میکنیم. در کلاس TreeNode یکسری ویژگی های مربوط به گره ها را تعریف میکنیم که برای مثال، فرزاندانش، value آن در صورت برگ بودن، feature و مقدار تقسیم آن و مقدار آنتروپی و دست آورد اطلاعات ذخیره شده است. در کلاس درخت تصمیم نیز، root درخت، ماکزیمم عمق درخت و تعداد فیچرها ذخیره شده است. و حالا به تعریف یکسری توابع داخل این کلاس، میپردازیم. و در زیر شرح مختصری از هر تابع را خواهیم گفت.

تابع ExtendTree:

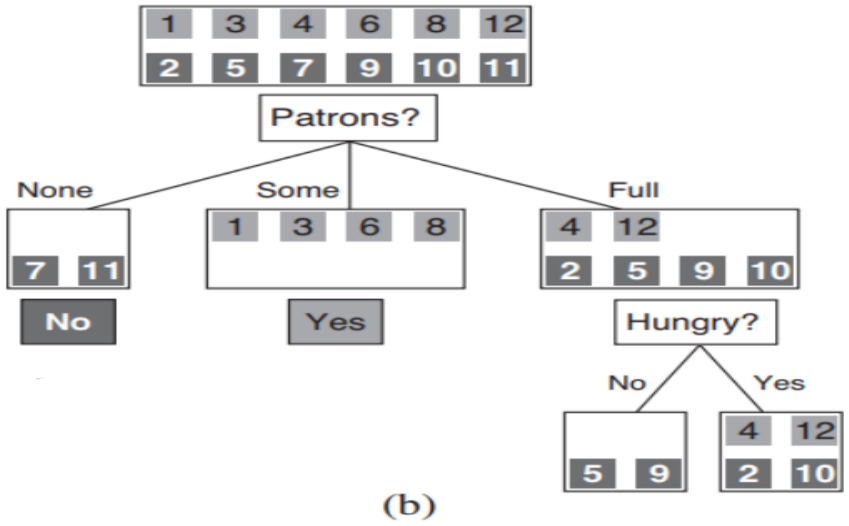
```
34
35      def ExpandTree(self, X, y, depth=0):
36          sampleCount = X.shape[0]
37          featureCount = X.shape[1]
38          LableCount = len(np.unique(y))
39
40          if (depth>=self.MaxDepth or LableCount==1 or sampleCount<2):
41              c = Counter(y)
42              leaf_Val = c.most_common(2)[0][0]
43              return TreeNode(leafVal=leaf_Val)
44
45          best_feature, best_thresh, Bestgain, best_entp = self.SplitWithBest(X, y, featureCount)
46          # create childs
47          leftIdxs, rightIdxs = self.Splitnode(X[:, best_feature], best_thresh)
48          Left = self.ExpandTree(X[leftIdxs, :], y[leftIdxs], depth+1)
49          Right = self.ExpandTree(X[rightIdxs, :], y[rightIdxs], depth+1)
50          return TreeNode(best_feature, best_thresh,Bestgain, best_entp, Left, Right)
51
```

در ابتدا میدانیم x.shape مقدار i وj یک آرایه دو بعدی را به ما میدهد (در قالب یک تاپل). پس تعداد سمپل های ما میشود x.shape[0] و تعداد ستون های جدول یا همان تعداد فیچرها (ویژگی ها) میشود عدد دوم یعنی x.shape[1]

در ادامه قرار است این تابع را به طور بازگشتی صدا بزنیم، فلذا نیاز است شرط اتمام بازگشتی بودن را همینجا مشخص کنیم. سه شرط اصلی وجود دارد برای اینکه یک گره تقسیم نشود و ادامه پیدا نکند و به گره برگ تبدیل شود. یا اینکه به مقدار max depth یا همان ماکزیمم عمق برسیم و دیگر قادر به ادامه دادن نباشیم. و یا اینکه تعداد lableهای آن یک باشد. یعنی همه خروجی ها یا 0 باشند یا 1 (یا yes باشند یا no) و یا اینکه کلا 1 سمپل داشته باشیم(یا کمتر) پس در اینصورت اصلا نمیتوان گره رو ادامه داد و درخت را بیشتر گسترش داد.

در این صورت باید یک لیبل به این گره نسبت داده شود. اگر فقط یک سمپل داشته باشد، خب لیبل گره همان است. (مثلا فقط یکدانه yes داشته باشیم) و یا اگر همه مقادیر یکی باشند مثلا 4 تا no فقط داشته باشیم طبیعی است که لیبل گره، همان no (یا صفر) شود. و یا اینکه اگر به مقدار ماکزیمم عمق رسیده باشیم و یک تعدادی yes و یک تعدادno داشته باشیم، باید بین آنها ماکزیمم گرفته شود و همان به عنوان لیبل انتخاب شود. مثلا اگر 7 تا no داریم و 2 تا yes بدیهی است که لیبل گره no خواهد شد. پس طبق توضیحات فوق، مقدار لیبل گره باید با تابع most_common تعیین شود.

شکل برای درک بهتر:

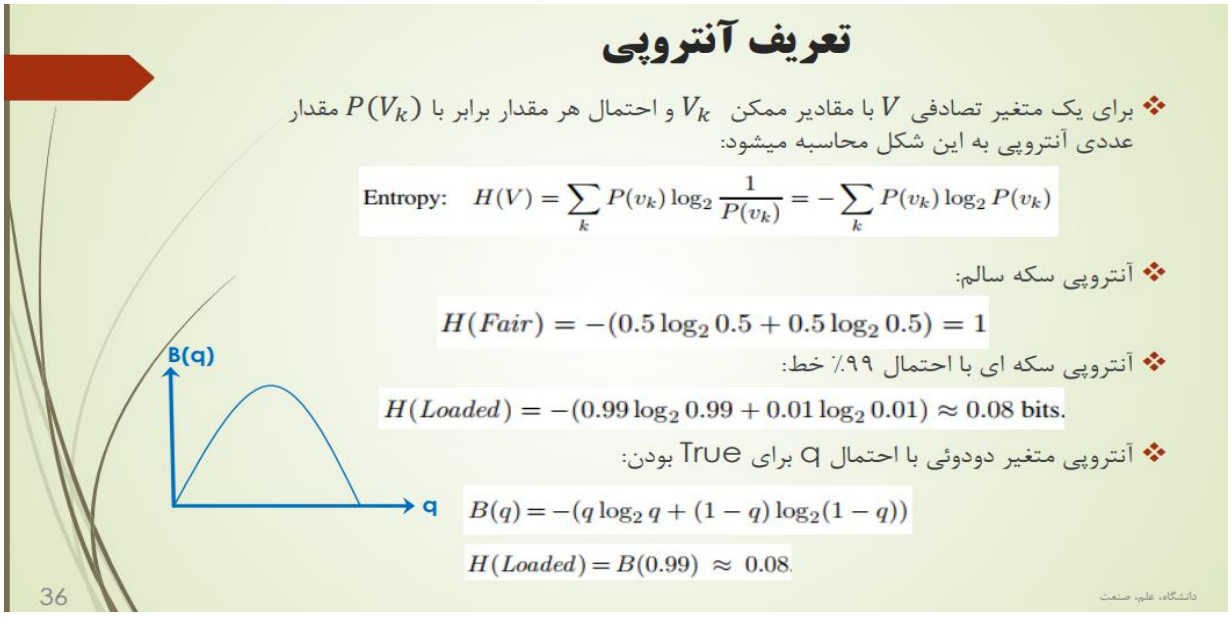


(b)

و سپس تابع را به طور بازگشتی برای فرزندان گره صدا خواهیم زد.

تابع MyEntropy:

همانطور که در صفحه 29 اسلایدها مشخص شده و از قبل نیز میدانیم، مهم است که کدام ویژگی اول انتخاب شود و معمولا ویژگی مهم تر را باید در گره های اولیه بررسی کرد. برای این منظور نیاز به تابع آنتروپی داریم تا برایمان این مورد را تعیین کند.



همانطور که در اسلاید بالا میبینید، نیاز به پیاده سازی دستی فرمول آنتروپی داریم. برای این کار کد زیر را پیاده سازی کرده ام. به این صورت که در مرحله ی اول تعداد yes وno های خروجی را باید تعیین کنیم و سپس فرمول آنتروپی را در خط 58 (و57) محاسبه میکنیم.

```
52      def MyEntropy(self, y):
53          ps = np.bincount(y)/ len(y) #[0's count, 1's count]
54          array = []
55          for p in ps:
56              if p>0:
57                  array.append(p * np.log(p))
58          result_Entp = -1 * np.sum(array)
59          if np.sum(array) == 0:
60              return 0
61          return result_Entp
62
```

تابع محاسبه دست آورد اطلاعات نیز به صورت زیر نوشته شده است:

```
91
92      def InformationGain(self, y, X_column, amountThr):
93          EntropyParent = self.MyEntropy(y)
94          leftIdxs, rightIdxs = self.Splitnode(X_column, amountThr)
95          if len(leftIdxs) == 0 or len(rightIdxs) == 0:
96              return 0
97          # weighted avgerage entropy of childs
98          y_count = len(y)
99          left_count = len(leftIdxs)
100         right_count = len(rightIdxs)
101         left_entp = self.MyEntropy(y[leftIdxs])
102         right_entp = self.MyEntropy(y[rightIdxs])
103         childEntropy = (left_count/y_count)*left_entp + (right_count/y_count)*right_entp
104
105         information_gain = EntropyParent - childEntropy
106         arr = []
107         arr.append(information_gain)
108         arr.append(childEntropy)
109         return arr
110
```

که همانطور که میبینید ابتدا آنتروپی برای parent حساب شده و سپس گره split شده که در ادامه آن را نیز حتما بررسی خواهیم کرد. و سپس آنتروپی برای فرزندان گره محاسبه شده و در آخر یک آنتروپی وزن دار برای تمام فرزاندان بدست آمده است. با کم کردن این دو مقدار از هم، مقدار information gain بدست می آید. و در آخر نیز مقدار آنتروپی و دست آورد اطلاعات ریترن میشود.

توابع split with best و split node :

این دو تابع همانطور که پیداست قرار است بر اساس بهترین ویژگی، گره را split کند.

برای همین تمام ستونهای جدول که همان ویژگی های ما است را بررسی میکنیم و بهترین gain را پیدا میکنیم. و با تابع split node فرزندان گره را بر اساس مقدار threshold تعیین میکنیم. (تابع argwhere ایندکس عناصری را میدهد که شرط داده شده را ارضا کنند) و در نتیجه این ایندکس ها در آخر ریترن خواهد شد.

```
62
63      def SplitWithBest(self, X, y, feat_idxs):
64          Bestgain = -1
65          split_idx = None
66          split_threshold = None
67          for feat_idx in range(feat_idxs):
68              X_column = X[:, feat_idx]
69              amountThrs = np.unique(X_column)
70              for thr in amountThrs:
71                  tmp = self.InformationGain(y, X_column, thr)
72                  if tmp != 0:
73                      gain = tmp[0]
74                      entp = tmp[1]
75                  else:
76                      gain = 0
77                      entp = 0
78                  if gain > Bestgain:
79                      Bestgain = gain
80                      best_entp = entp
81                      split_idx = feat_idx
82                      split_threshold = thr
83          return split_idx, split_threshold, Bestgain, best_entp
84
85      def Splitnode(self, X_column, s_amountThrsh):
86          l_idx = np.argwhere(X_column <= s_amountThrsh)
87          IDXLeft = l_idx.flatten()
88          r_idx = np.argwhere(X_column > s_amountThrsh)
89          IDXRight = r_idx.flatten()
90          return IDXLeft, IDXRight
91
```

بخش اصلی کد(main):

ابتدا از فایل مورد نظر شروع به خواندن میکنیم. و طبق خط 131 و 132 ورودی های X و y را تعیین میکنیم. همانطور که قبلا گفتیم x آرایه ای از مقادیر ویژگی های ورودی و y آرایه ای از مقدار دودوئی خروجی هاست. و سپس در خط 134 مقدار داده های test و train را مشخص میکنیم(داده های آموزشی و آزمایشی) که باز هم طبق سخنان پیشتر گفته شده، مقدار 20درصد و 80 درصد برای داده های آزمایشی و آموزشی در نظر گرفته ام. (کامنت های خط 138 تا 141 برای داده های تماما آموزشی و بدون آزمایشی است)

و سپس در خط های 143 و 144 درخت تصمیم را ساخته ایم.

```
127
128   # dataset = pd.read_csv(".\diabetes.csv")
129   dataset = pd.read_csv(".\DataRestaurant.csv")
130   datelen = len(dataset.columns)-1
131   X = dataset.iloc[:,:-1].values
132   y = dataset.iloc[:, datelen].values
133
134   X_train, X_test, y_train, y_test = train_test_split(
135       X, y, test_size=0.2, random_state=0 #1234
136   )
137
138   # X_train = X
139   # X_test = []
140   # y_train = y
141   # y_test = []
142
143   clf = DecisionTree(MaxDepth=50)
144   clf.fitRoot(X_train, y_train)
145
```

توابع traverse tree وfind tree :

برای تابع اول قصد داریم درخت را با یک ورودی x طی کنیم و از root شروع میکنیم و به طور بازگشتی تابع را روی فرزندان هر گره صدا میزنیم و شرط اتمام بازگشتی نیز برگ بودن گره است. اگر گره leaf باشد، یعنی خروجی دودویی ما است و باید مقدار value آن را ریترن کنیم.

در تابع find_decisitionTree کل درخت را میخواهیم پرینت کنیم تا در کنسول نمایش داده شود. مجدد برای اینکار از root شروع کرده و هر گره را بررسی کرده و اگر برگ نبود و به انتهای درخت نیز نرسیده بودیم، تابع را به طور بازگشتی روی فرزندان گره صدا میزنیم. و برای هر گره

مقدار threshold و ویژگی گرهfeature و دست آورد اطلاعات و آنتروپیش را چاپ میکنیم. و اگر به برگ رسیدیم مقدار value آن را که خروجی ما است چاپ میکنیم. و در آخر نیز برای شرط اتمام بازگشتی صدا زده شدن تابع، اگر به nodeای که none باشد برسیم یعنی درخت تمام شده و باید return کنیم و از تابع خارج شویم.

```
111     def TreeTraverse(self, x, node):
112         if node.IsLeaf():
113             return node.leafVal
114         if x[node.feature] <= node.amountThr:
115             return self.TreeTraverse(x, node.Left)
116         return self.TreeTraverse(x, node.Right)
117 
118 def Find_DecisionTree (node):
119     if node == None:
120         return
121     if node.IsLeaf():
122         print("- leaf\t" , node.leafVal)
123     else:
124         print("- feat\t" ,node.feature , "  - thresh\t" , node.amountThr, "  - information gain\t", node.gain, "  - entropy\t", node.Entropy)
125     Find_DecisionTree(node.Left)
126     Find_DecisionTree(node.Right)
127 
```

بررسی خروجی نهایی برنامه:

در خط 157 درخت تصمیم نهایی را همانطور که بالاتر توضیح دادیم چاپ کرده ام در خط 147 تا 155 به تست کردن ورودی های تست و بررسی صحت آنها پرداختم. باید دقت داده های آزمایشی را بدست بیاوریم فلذا باید تک تک ورودی های داخل آرایه ی X_test را روی درخت تصمیم طی کرده و برای هرکدام تابع Traverse را صدا بزنیم و به خروجی درخت برسیم. تمام این خروجی ها را در آرایه Result نگه میداریم.

حال برای محاسبه دقت داده های آزمایشی، باید ببینیم چه تعداد از این خروجی ها با خروجی های درست منطبق است (y_test) و با تقسیم بر طول آرایه یعنی len(y_test) مقدار درست دقت بدست می آید. و آن را در متغییری به اسم Deghat ریخته و چاپش میکنیم.

```
145
146     #test X_test
147     ResArray = []
148     for xx in X_test:
149         res = clf.TreeTraverse(xx, clf.root)
150         ResArray.append(res)
151     Result = np.array(ResArray)
152
153     #calculate right answers (deghat)
154     deghat = np.sum(y_test == Result) / len(y_test)
155     print("deghat(accuracy): ", deghat)
156
157     Find_DecisionTree(clf.root)
```

نمایش خروجی برای داده های مثال رستوران و داده های تشخیص دیابت:

درخت تصمیم مربوط به مثال 12تایی رستوران مطابق شکل زیر است که همانطور که مشاهده میکنید ویژگی ها و مقادیر خروجی و زیرشاخه هایش پیداست و از بالاترین گره یعنی root شروع میشود و فرزندانش به طور عمقی طی میشود تا به برگ برسد. و همچنین مقدار دقت نیز برای داده های آزمایشی بدست آمده و نشان داده شده است. (مقدار دقت داده های آموزشی 100 و منها این عدد برای داده های آزمایشی، اختلاف این دو را خواهد داد)

```
E:\neg\term5\hoosh masnooie\projects\Project2\Project2_99521226>Project2_99521226.py
deghat(accuracy):  0.6666666666666666
- feat   9   - thresh    20   - information gain        0.22164078547545663   - entropy        0.46532079112186675
- feat   4   - thresh    1   - information gain  0.2120742666998533   - entropy  0.38619532188540395
- leaf   0
- feat   2   - thresh    0   - information gain  0.2195121486796562   - entropy  0.23104906018664842
- leaf   1
- feat   1   - thresh    0   - information gain  0.6931471805599453   - entropy  0.0
- leaf   1
- leaf   0
- leaf   0
```

خروجی مربوط به داده های تشخیص دیابت:

خروجی به دلیل بزرگ بودن و زیاد بودن داده های جدول، بزرگ است و به همین دلیل در عکس زیر تنها بخشی از آن را برای مثال نشان داده ایم. میزان دقت و همچنین خود درخت تصمیم در خروجی پرینت شده است:

```
E:\neg\term5\hoosh masnooie\projects\Project2\Project2_99521226>Project2_99521226.py
deghat(accuracy):  0.7337662337662337
- feat   1   - thresh    123.0   - information gain      0.09117258495016212   - entropy         0.5622081176268778
- feat   5   - thresh    26.4    - information gain      0.05441080933782194   - entropy         0.423974433117583
- feat   6   - thresh    0.673   - information gain      0.017602822069490306   - entropy        0.03981510096057002
- leaf   0
- feat   6   - thresh    0.678   - information gain      0.2145591551764051   - entropy  0.0
- leaf   1
- leaf   0
- feat   7   - thresh    28.0    - information gain      0.04652942788478431   - entropy         0.516880226717072
- feat   5   - thresh    30.9    - information gain      0.027899187083508248   - entropy        0.3524855589846186
- feat   0   - thresh    6.0    - information gain       0.11251593411963928   - entropy         0.0
- leaf   0
- leaf   1
- feat   2   - thresh    52.0    - information gain      0.04276238997726162   - entropy         0.4192745194891954
- feat   6   - thresh    0.496   - information gain      0.6365141682948128   - entropy  0.0
- leaf   1
- leaf   0
- feat   6   - thresh    0.498   - information gain      0.03159797987954621   - entropy         0.37252028504181545
- feat   2   - thresh    72.0    - information gain      0.06078715435149365   - entropy         0.19984994518242863
- leaf   0
- feat   5   - thresh    45.2    - information gain      0.07797052459516374   - entropy         0.36169935480617926
- feat   4   - thresh    36.0    - information gain      0.13807734814588196   - entropy         0.2386928131105548
- feat   1   - thresh    87.0    - information gain      0.17441604792151594   - entropy         0.46209812037329684
- leaf   0
- feat   5   - thresh    33.3    - information gain      0.31825708414740644   - entropy         0.37489009641253884
- leaf   1
- feat   1   - thresh    95.0    - information gain      0.5623351446188083   - entropy  0.0
- leaf   1
- leaf   0
- leaf   0
- leaf   1
- feat   5   - thresh    38.2    - information gain      0.10584209720644394   - entropy         0.4651806137364723
- feat   5   - thresh    32.5    - information gain      0.1843976993692441   - entropy  0.4710840745321486
- leaf   0
- feat   1   - thresh    87.0    - information gain      0.11687963343115981   - entropy         0.5740436758826583
- feat   0   - thresh    2.0    - information gain       0.5004024235381879   - entropy  0.0
- leaf   0
- leaf   1
- feat   2   - thresh    68.0    - information gain      0.3094481730497441   - entropy  0.3014161290051494
```

چالش ها:

به چالش های زیادی در طی انجام پروژه برخورد کردم که هم شامل مشکلات کدی بود هم مشکلات مربوط به خود الگوریتم و درخت تصمیم. برای مثال برای تِرورس کردن درخت و نحوه تست داده ها و یا نحوه محاسبه فرمول آنتروپی و یا برای کد، پیدا کردن توابع مفیدی مثل most_common و flattern وbincount و یا بهبود و افزایش دقت کد از جمله چالش های مواجه در طی انجام پروژه بود. برای افزایش دقت نیز راه های متفاوتی وجود دارم که بنظر میتوان با افزایش داده های آموزشی، درخت تصمیم دقیق تری پیاده سازی کرد. و ..

پایان

نگین حقیقی-پاییز1401

99521226