Implementation of ID3 algorithm classification

1. نحوه پیاده سازی:

1. در ابتدا تابع کل که آنتروپی کل گره میباشد را محاسبه میکنیم.

```
def totalEntropy(train_data, label, class_list):
    total_row = train_data.shape[0] #the total size of the dataset
    total_entr = 0

    for c in class_list: #for each class in the label
        total_class_count = train_data[train_data[label] == c].shape[0] #number of the class
        total_class_entr = - (total_class_count/total_row)*np.log2(total_class_count/total_row) #entropy of the class
        total_entr += total_class_entr #adding the class entropy to the total entropy of the dataset

    return total_entr
```

I. train_data: *a pandas dataframe/dataset.*
II. label: *string, name of the label of the dataframe (<=50K)*
III. class_list: *list, unique classes of the label ([Yes, No]).*

2. در مرحله بعدی آنتروپی هر شاخه از نود را محاسبه میکنیم.

I. feature_value_data: *a pandas dataframe/dataset, which contains rows that has a specific value of a feature*
II. label: *string, name of the label of the dataframe (<=50K)*
III. class_list: *list, unique classes of the label ([Yes, No]).*

```
def informationgain(feature_name, train_data, label, class_list):
    feature_value_list = train_data[feature_name].unique() #unqiue values of the feature
    total_row = train_data.shape[0]
    feature_info = 0.0

    for feature_value in feature_value_list:
        feature_value_data = train_data[train_data[feature_name] == feature_value] #filtering rows with that feature_value
        feature_value_count = feature_value_data.shape[0]
        feature_value_entropy = feature_entropy(feature_value_data, label, class_list) #calculcating entropy for the feature
        feature_value_probability = feature_value_count/total_row
        feature_info += feature_value_probability * feature_value_entropy #calculating information of the feature value

    return totalEntropy(train_data, label, class_list) - feature_info #calculating information gain by subtracting
```

3. در این مرحله information gian را برای ویژگی ها محاسبه میکنیم.

I. feature_name: *string, the name of the feature that we want to find it's information gain(EX.* Male).
II. train_data: *a pandas dataframe/dataset.*
III. label: *string, name of the label of the dataframe (<=50K)*

```
def feature_entropy(feature_value_data, label, class_list):
    class_count = feature_value_data.shape[0]
    entropy = 0

    for c in class_list:
        label_class_count = feature_value_data[feature_value_data[label] == c].shape[0] #row count of class c
        entropy_class = 0
        if label_class_count != 0:
            probability_class = label_class_count/class_count #probability of the class
            entropy_class = - probability_class * np.log2(probability_class)  #entropy
        entropy += entropy_class
    return entropy
```

IV. class_list: *list, unique classes of the label ([Yes, No]).*

4. در این مرحله از بین ویژگی ها آن ویژگی که بیشترین بهره اطلاعاتی را دارد انتخاب میکنیم.

```
def find_most_informative_feature(train_data, label, class_list,size_ID3):
    feature_list = train_data.columns.drop(label) #finding the feature names in the dataset
    max_info_gain = -1
    max_info_feature = None

    for feature in feature_list:  #for each feature in the dataset
        feature_info_gain = informationgain(feature, train_data, label, class_list)
        if max_info_gain < feature_info_gain: #selecting feature name with highest information gain
            max_info_gain = feature_info_gain
            max_info_feature = feature
    size_ID3.append(max_info_feature)
    return max_info_feature,size_ID3
```

I. feature_name: *string, the name of the feature that we want to find it's information gain(EX.* Male).
II. train_data: *a pandas dataframe/dataset.*
III. label: *string, name of the label of the dataframe (<=50K).*
IV. class_list: *list, unique classes of the label ([Yes, No]).*
V. Size_ID3: number of tree nodes per stage.

5. در این مرحله برای شاخه های نود های انتخاب شده labale را پیدا میکنیم.ممکن است به labale value مسیله برسم و یا به نود بعدی.اگر به labale value رسیدیم آن شاخه را از داده ها حذف میکنیم.

I. feature_name: *string, the name of the feature that we want to find it's information gain(EX.* Male).
II. train_data: *a pandas dataframe/dataset.*
III. train_data_column: train data without column of selected feature.
IV. label: *string, name of the label of the dataframe (<=50K).*
V. class_list: *list*, *unique classes of the label ([Yes, No]).*

```
def generate_sub_tree(feature_name, train_data,train_data_column, label, class_list):
    feature_value_count_dict = train_data[feature_name].value_counts(sort=False) #dictionary of the count of unqiue feature val
    tree = {} #sub tree or node
    maxvote=[-1,-2]
    i=0
    fe = train_data_column.columns #finding the feature names in the dataset
    for feature_value, count in feature_value_count_dict.iteritems():
        feature_value_data = train_data[train_data[feature_name] == feature_value] #dataset with only feature_name = feature_va
        assigned_to_node = False #flag for tracking feature_value is pure class or not
        for c in class_list: #for each class
            class_count = feature_value_data[feature_value_data[label] == c].shape[0] #count of class c
            if class_count == count: #count of feature_value = count of class (pure class)
                tree[feature_value] = c #adding node to the tree
                train_data = train_data[train_data[feature_name] != feature_value] #removing rows with feature_value
                assigned_to_node = True
            elif (len(list(train_data_column.columns))==2) & (class_count!=0):

                maxvote[i]=class_count
                i=i+1
            else:
                pass

        if  (maxvote[0]>0) & (maxvote[1]>0):


            assigned_to_node = True
            if(maxvote[0]>maxvote[1]):
                tree[feature_value] = class_list[0]
                train_data = train_data[train_data[feature_name] != feature_value]
            elif maxvote[1]>maxvote[0]:
                tree[feature_value] = class_list[1]
                train_data = train_data[train_data[feature_name] != feature_value]
            else:

                tree[feature_value] = class_list[0]
                train_data = train_data[train_data[feature_name] != feature_value]
        maxvote=[-1,-2]
        i=0
        if not assigned_to_node: #not pure class
            tree[feature_value] = "?" #should extend the node, so the branch is marked with ?

    return tree, train_data
```

6. در این مرحله ساختمان داده درخت را با استفاده از تابع بازگشتی از مرحله 1 تا 5 پیاده سازی میکنیم.

```
def make_tree(root, prev_feature_value, train_data,train_data_column, label, class_list,size_ID3):
    if train_data.shape[0] != 0: #if dataset becomes enpty after updating
        max_info_feature,size_ID3 = find_most_informative_feature(train_data_column,label, class_list,size_ID3) #most infor
        tree, train_data = generate_sub_tree(max_info_feature, train_data,train_data_column, label, class_list) #getting tr
        next_root = None
        if prev_feature_value != None: #add to intermediate node of the tree
            root[prev_feature_value] = dict()
            root[prev_feature_value][max_info_feature] = tree
            next_root = root[prev_feature_value][max_info_feature]
        else: #add to root of the tree
            root[max_info_feature] = tree
            next_root = root[max_info_feature]
        for node, branch in list(next_root.items()): #iterating the tree node
            if branch == "?": #if it is expandable
                feature_value_data = train_data[train_data[max_info_feature] == node] #using the updated dataset
                a=train_data_column.drop(columns=max_info_feature).copy()
                make_tree(next_root, node, feature_value_data,a,label, class_list,size_ID3) #recursive call with updated da
    else:
        return
```

I. root: *dictionary, the current pointed node/feature of the tree. It is contineously being updated.*
II. prev_feature_value: *Any datatype (Int or Float or String etc.) depending on the datatype of the previous feature, the previous value of the pointed node/feature.*
III. train_data: *a pandas dataframe/dataset.*
IV. train_data_column: train data without column of selected feature.
V. label: *string, name of the label of the dataframe (<=50K).*
VI. class_list: *list, unique classes of the label ([Yes, No]).*
VII. Size_ID3: number of tree nodes per stage.

7. در این مرحله تابعی پیاده سازی میکنیم تا بتوانیم تابع بازگشتی ها را صدا بزنیم.

```
def id3(train_data_m, label,size_ID3):
    train_data = train_data_m.copy() #getting a copy of the dataset
    tree = {} #tree which will be updated
    class_list = train_data[label].unique() #getting unqiue classes of the label
    make_tree(tree, None, train_data_m, train_data_m,label, class_list,size_ID3) #start calling recursion
    return tree,size_ID3
```

1. train_data_m: *a pandas dataframe/dataset.*
2. label: *string, name of the label of the dataframe (<=50K).*
3. Size_ID3: number of tree nodes per stage.

8. در این مرحله تابع پیش بینی را پیاده سازی میکنیم.

```
def predict(tree, instance):
    if not isinstance(tree, dict): #if it is leaf node
        return tree #return the value
    else:
        root_node = next(iter(tree)) #getting first key/feature name of the dictionary
        feature_value = instance[root_node] #value of the feature
        if feature_value in tree[root_node]: #checking the feature value in current tree node
            return predict(tree[root_node][feature_value], instance) #goto next feature
        else:
            return None
```

I. tree: *(nested) dictionary, a decision tree.*

II. instance: *series/dataframe row, a row of dataset. The row may not contain label.*

9. در این مرحله تابع ارزیابی را پیاده سازی میکنیم.

```
def evaluate(tree, test_data_m, label):
    correct_preditct = 0
    wrong_preditct = 0
    for index, row in test_data_m.iterrows(): #for each row in the dataset
        result = predict(tree, test_data_m.iloc[index]) #predict the row
        if result == test_data_m[label].iloc[index]: #predicted value and expected value is s
            correct_preditct += 1 #increase correct count
        else:
            wrong_preditct += 1 #increase incorrect count
    accuracy = correct_preditct / (correct_preditct + wrong_preditct) #calculating accuracy
    return accuracy
```

I. tree: *(nested) dictionary, a decision tree.*
II. train_data_m: *a pandas dataframe/dataset.*
III. label: *string, name of the label of the dataframe (<=50K).*

1. بخش اول:

a) همان طور که مشاهده میشود با آموزش 45 داده ها مقدار صحت بر روی داده های تست حدود 72 درصد میباشد و میانگین نیز همان 72 و 73 درصد میباشد. و میانگین نیز حدود 73 درصد میباشد همین طور سایز درخت نیز بین 790 تا 900 میباشد.

```
for i in range(3):
    t,s=id3(adult_train.sample(frac=0.45),'<=50K',[])
    print('size 0.45 is %i:'%(len(s)))
    accuracy_train=evaluate(t,adult_train,'<=50K')
    print('accuracy_train:',accuracy_train)
    accuracy_test=evaluate(t,adult_test,'<=50K')
    print('accuracy_test:',accuracy_test)
    ac_test.append(accuracy_test)
```

```
size 0.45 is 821:
accuracy_train: 0.8024802480248024
accuracy_test: 0.7358735873587359
size 0.45 is 862:
accuracy_train: 0.7977797779777978
accuracy_test: 0.7212721272127213
size 0.45 is 792:
accuracy_train: 0.7984798479847984
accuracy_test: 0.7309730973097309
```

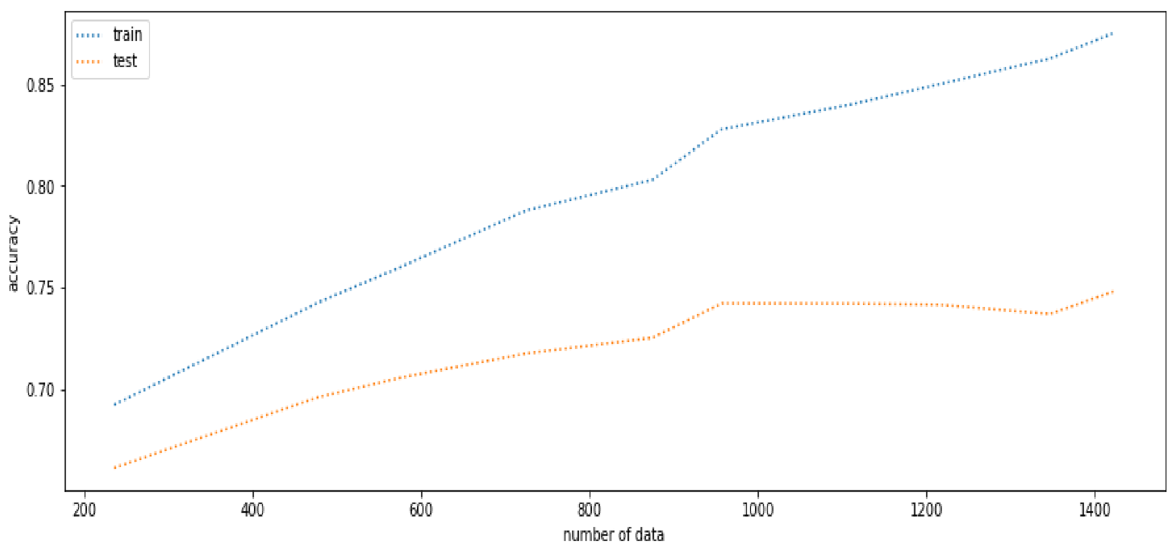
```
mean_accuracy_test_45=sum(ac_test)/3
print(mean_accuracy_test_45)
```

```
0.7293729372937294
```

b) همانطور که مشاهده میشود با افزایش سایز داده های آموزشی دقت بر روی داده های تست بیشتر میشود. و بیشترین دقت نیز مربوط به کل داده های آموزشی است که حاصل میشود. پس با توجه به نمودار میتوان گفت هر چقدر داده ها برای آموزش بیشتر میشود صحت بر روی داده های تست بیشتر میشود و همین طور سایز درخت نیز بیشتر میشود.

	سایز درخت				صحت بر روی داد های آموزشی				صحت بر روی داد های تست			
	دفعات آموزش				دفعات آموزش				دفعات آموزش			
	1	2	3	mean	1	2	3	mean	1	2	3	mean
0.45	821	862	792	825	0.80248	0.7978	0.7985	0.7996	0.7359	0.7213	0.731	0.729373
0.55	945	992	989	975.3333	0.8131	0.8158	0.8206	0.8165	0.7296	0.729	0.7346	0.73104
0.65	1068	1037	1070	1058.333	0.8298	0.8288	0.8329	0.8305	0.7301	0.7405	0.736	0.735507
0.75	1136	1157	1201	1164.667	0.8469	0.846	0.8457	0.8462	0.7392	0.7425	0.7403	0.740641
100	1424	1424	1424	1424	0.8754	0.8754	0.8754	0.8754	0.7483	0.7483	0.7483	0.748275

گزارش سایز و صحت



در نمودار مشخص هست وقتی 60 الی 70 درصد داده ها را آموزش میدهیم بر روی مجموعه تست بهتر جواب میگیریم پس مشخص میشود وقتی تعداد داده ها بیشتر از 70 درصد داده ها را آموزش میدهیم با overfitting روبرو میشویم.