Offline1

April 28, 2018

```
In [1]: import pandas as pd
        import numpy as np
        import math
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.model_selection import train_test_split

In [2]: data = pd.read_csv('bank-additional-full.csv',sep=';')

In [3]: data.head()
```

Out[3]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous	poutcome	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed	y
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191.0	no
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191.0	no
2	37	services	married	high.school	no	yes	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191.0	no
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191.0	no
4	56	services	married	high.school	no	no	yes	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191.0	no

[5 rows x 21 columns]

```
In [4]: myDataY=data[data.y=='yes']

In [5]: myDataN=data[data.y=='no']

In [6]: def shuffle2(df, n=1, axis=0):
            for _ in range(n):
```

```
            df.apply(np.random.shuffle, axis=axis)
            return df
```

```
In [7]: myFinalData=pd.concat([myDataN.sample(n=4444,replace=False,),myDataY])
        myFinalData = myFinalData.sample(frac=1).reset_index(drop=True)

        tmpHolder=myFinalData.dtypes

        #converting to categorical type
        for i in range(len(tmpHolder)):
            if(tmpHolder[i]=='object'):
                myFinalData[tmpHolder.index[i]]=myFinalData[tmpHolder.index[i]].astype('category
        myFinalData=myFinalData.drop(columns=['duration'])
```

```
In [8]: trainSet,testSet=train_test_split(myFinalData,test_size=0.25)
```

0.1 Decision Stump

```
In [9]: def entropy(probs):
            return sum([-prob*np.log2(prob) for prob in probs])
```

```
In [10]: def entropy_list(list_item):
             #print(list_item)
             from collections import Counter
             mp = Counter(x for x in list_item)
             total=1.0*len(list_item)

             probs = [tt/total for tt in mp.values()]
             return entropy(probs)
```

```
In [11]: def information_gain_cat(df, split_attribute_name, target_attribute_name):

             dfTmp=df.copy(deep=True)
             #Split
             df_split = dfTmp.groupby(split_attribute_name)

             dct={}

             nrows=len(dfTmp.index)*1.0

             #Merge
             for name,group in df_split:
             #    print(name,entropy_categorical(np.array(group['y'],dtype=pd.Series)),len(group)
                 dct[name]=[entropy_list(np.array(group[target_attribute_name],dtype=pd.Series))

             dd = pd.DataFrame(dct)
             dd=dd.T
             dd.columns=['Entropy','ObsProp']
```

```
            #print(dd)

            #Gain

            new_entropy = sum( dd['Entropy'] *dd['ObsProp'] )


            old_entropy = entropy_list(dfTmp[target_attribute_name])


            return old_entropy-new_entropy
```

In [12]:
```
information_gain_cat(trainSet,'job','y')
```

Out[12]:
```
0.031088731643523104
```

In [13]:
```
def information_gain_num(df,split_attribute_name,target_attribute_name):
    dfTemp=df.copy(deep=True)

    dfTemp=dfTemp.sort_values(split_attribute_name)

    split_points=dfTemp[split_attribute_name].unique()
    #print(dfTemp[split_attribute_name].unique())

    gains={}

    cols=list(dfTemp.columns)

    #print(cols)


    pos=cols.index(split_attribute_name)
    #print(pos)

    parent_entr=entropy_list(dfTemp[target_attribute_name])


    best_split=-np.inf
    best_gain=-np.inf


    for split_point in split_points:

        subset_df1 = dfTemp[dfTemp[split_attribute_name] <= split_point]
        subset_df2 = dfTemp[dfTemp[split_attribute_name] > split_point]

        s1_entr=entropy_list(subset_df1[target_attribute_name])
        s2_entr=entropy_list(subset_df2[target_attribute_name])
```

```
                child_entr=s1_entr*len(subset_df1.index)+s2_entr*len(subset_df2.index)
                child_entr/=len(dfTemp.index)
                tmpGain=parent_entr-child_entr

                if((best_gain==-np.inf and best_split==-np.inf)or(tmpGain>best_gain)):
                    best_split=split_point
                    best_gain=tmpGain
                #print(split_point,tmpGain)

            return best_gain,best_split
```

```
In [14]: information_gain_num(myFinalData,'emp.var.rate','y')
```

```
Out[14]: (0.14470345494572923, -1.1)
```

```
In [15]: class DecisionNode:
            def __init__(self,attr_name,kind):
                self.attr_name=attr_name
                self.kind=kind

            def setThreshold(self,threshold):
                if(self.kind=='numeric'):
                    self.threshold=threshold
                else:
                    raise ValueError("Threshold can't be set for categorical variable")

            def setLessThanEq(self,cls):
                if(self.kind=='numeric'):
                    self.less_than_eq_class=cls
                else:
                    raise ValueError("Invalid for categorical variable")

            def setGreaterThan(self,cls):
                if(self.kind=='numeric'):
                    self.greater_than_class=cls
                else:
                    raise ValueError("Invalid for categorical variable")

            def setLevelClassMap(self,levelClassMap):
                if(self.kind=='categorical'):
                    self.levelClassMap=levelClassMap
                else:
                    raise ValueError("Invalid for numeric variable")
```

```
In [16]: def isCategorical(df,attr):
            if(df[attr].dtype.name=='category'):
```

```
            return True
        return False
```

In [17]:
```
from operator import itemgetter
def DecisionStump(mydf,all_attr,target_attr,default_class=None):
    df=mydf.copy(deep=True)

    gains=[]
    for attr in all_attr:
        if(isCategorical(df,attr)):
            ig = information_gain_cat(df=df,split_attribute_name=attr,target_attribute_
            splt=-np.inf
        else:
            ig,splt=information_gain_num(df=df,split_attribute_name=attr,target_attribu
        gains.append((ig,splt))

    mx_gain=max(gains,key=itemgetter(0))[0]

    #print(mx_gain)

    mx_gain_idx=-1
    for i in range(len(gains)):
        if(gains[i][0]==mx_gain):
            mx_gain_idx=i
            break
#     print(gains)
#     print(all_attr[mx_gain_idx])

    mx_gain_attr=all_attr[mx_gain_idx]

  #  print(mx_gain_attr)

    #splitting for Numeric Attributes
    if(not(isCategorical(df,mx_gain_attr))):
        thresh=gains[mx_gain_idx][1]
        df1=df[df[mx_gain_attr]<=thresh]
        df2=df[df[mx_gain_attr]>thresh]
        dn=DecisionNode(mx_gain_attr,'numeric')
        dn.setThreshold(thresh)
        ly=df1[df1.y=='yes'].shape[0]
        ln=df1[df1.y=='no'].shape[0]

        if(ly>=ln):
            dn.setLessThanEq('yes')
        else:
            dn.setLessThanEq('no')
        gy=df2[df2.y=='yes'].shape[0]
        gn=df2[df2.y=='no'].shape[0]
```

```
            if(gy>=gn):
                dn.setGreaterThan('yes')
            else:
                dn.setGreaterThan('no')

            return dn


        else:
            dn=DecisionNode(mx_gain_attr,'categorical')
            df_split = df.groupby(mx_gain_attr)

            dct={}
            for name,group in df_split:
                y=group[group.y=='yes'].shape[0]
                n=group[group.y=='no'].shape[0]
                if(y>=n):
                    dct[name]='yes'
                else:
                    dct[name]='no'
                #print(name,group[group.y=='yes'].shape[0],group[group.y=='no'].shape[0])

            print(dct)
                #dct[name]=[entropy_list(np.array(group[target_attribute_name],dtype=pd.Ser
            dn.setLevelClassMap(dct)
            return dn
```

In [18]:
```
def Predict(df,h):
    attr=h.attr_name
    dataPoints=df[attr]
    y_pred=[]
    if(h.kind=='categorical'):
        dct=h.levelClassMap
        y_pred= [dct[x] for x in dataPoints]
    else:
        thresh=h.threshold
        for x in dataPoints:
            if(x<=thresh):
                y_pred.append(h.less_than_eq_class)
            else:
                y_pred.append(h.greater_than_class)
    return y_pred
```

0.2 ADABOOST

In [19]:
```
# {yes,no}==>{+1,-1}

    class AdaBoost:
```

```
    def __init__(self,k,target_attr):
        self.hyp= []
        self.hyp_wgt = []
        self.k=k
        self.target_attr=target_attr

    def train(self,df):

        sampN, _ = df.shape
        samp_wgt = np.ones(sampN) / sampN

        ktmp=self.k


        while ktmp>0:
          #  print(samp_wgt)

            sampled_data=df.sample(n=sampN,replace=True,weights=samp_wgt)
            cols=list(sampled_data.columns)
            cols.remove(self.target_attr)

            h=DecisionStump(mydf=sampled_data,all_attr=cols,target_attr=self.target_att


#             h = DecisionTreeClassifier(max_depth=1)

#             h.fit(X, y, sample_weight=samp_wgt)
#             pred = h.predict(X)

            y_pred=Predict(df=sampled_data,h=h)
            y_actual=np.array(sampled_data[self.target_attr],dtype=pd.Series)


            yy_pred=[]
            yy_actual=[]

            for i in range(len(y_actual)):
                if(y_pred[i]=='yes'):
                    yy_pred.append(1)
                else:
                    yy_pred.append(-1)
                if(y_actual[i]=='yes'):
                    yy_actual.append(1)
                else:
                    yy_actual.append(-1)


            misses=[]
```

```
            for i in range(len(yy_actual)):
                if(yy_actual[i]!=yy_pred[i]):
                    misses.append(np.float64(1))
                else:
                    misses.append(np.float64(0))

        #  print(misses)

            eps = np.dot(samp_wgt,misses)# samp_wgt.dot(yy_pred != yy_actual)

#             print(type(eps))
#             print(eps)

            if(eps>0.5):
                continue

            alpha = (np.log(1 - eps) - np.log(eps)) / 2.0

#             print(type(samp_wgt[0]))
#             print(type(alpha))
#             print(type(yy_actual))
#             print(type(yy_pred))


            yy_actual=np.array(yy_actual,dtype=np.float64)
            yy_pred=np.array(yy_pred,dtype=np.float64)

            samp_wgt = samp_wgt * np.exp(- alpha * yy_actual * yy_pred)
  #          print("after ",samp_wgt)


#             for i in range(len(samp_wgt)):
#               #  print(t, " th samp_wgt ",samp_wgt[i])
#                 if(math.isnan(samp_wgt[i])):
#                     samp_wgt[i]=0
            #print(max(samp_wgt))
            #print("HELLOWORLD ",np.sum(samp_wgt))
            samp_wgt = samp_wgt / samp_wgt.sum()
            samp_wgt=np.array(samp_wgt,dtype=pd.Series)
            #print("HELLOWORLD ",samp_wgt.sum())

#             print(t,"th iteration")
#             print(samp_wgt)
#             print(np.sum(samp_wgt,dtype=np.float128))
#             print()

            self.hyp.append(h)
            self.hyp_wgt.append(alpha)
```

```
                ktmp-=1
            return samp_wgt

        def test(self,X):

            sampN, _ = X.shape
            samp_wgt = np.ones(sampN) / sampN
            y=np.zeros(sampN)


            for (h, alpha) in zip(self.hyp, self.hyp_wgt):
                y_pred=Predict(df=X,h=h)

                yy_pred=[]
                for i in range(len(y_pred)):
                    if(y_pred[i]=='yes'):
                        yy_pred.append(np.float64(1))
                    else:
                        yy_pred.append(np.float64(-1))

#               print(type(y))
#               print(type(alpha))
#               print(type(yy_pred))
                yy_pred=np.array(yy_pred,dtype=np.float64)

                y = y + alpha * yy_pred

#           print(y)

            y = np.sign(y)

            return y
```

In [20]:
```
from sklearn.model_selection import KFold
from sklearn.metrics import f1_score
```

In [32]:
```
def crossValidation(df,target,k=3,K=5):


    kf=KFold(n_splits=k,shuffle=True)

    models=[]
    f1_scores=[]

    best_score=None
    best_model=None
```

```
        #print(kf.get_n_splits(df))
        for train,test in kf.split(df):

            dftr=df.iloc[train]
            dfts=df.iloc[test]

            ada=AdaBoost(k=K,target_attr=target)

            ada.train(df=dftr)
            y_pred=ada.test(dfts)
            y_true=np.array(dfts[target],dtype=pd.Series)

           # print(y_true)

            yy_true=[]
            for i in range(len(y_true)):
                if(y_true[i]=='yes'):
                    yy_true.append(1)
                else:
                    yy_true.append(-1)


            fs=f1_score(y_true=yy_true,y_pred=y_pred)

            models.append(ada)
            f1_scores.append(fs)

            if(best_score==None or (fs>best_score)):
                best_model=ada
                best_score=fs

            print(fs)

        print("AVG: ",sum(f1_scores)/len(f1_scores))

        return best_model,best_score


In [33]: try_fold=[5,10,20]
         try_round=[5,10,20]

         for i in try_fold:
             for j in try_round:
                 print("TRYING fold = ",i," and Round = ",j)
                 print()
                 m,s=crossValidation(df=trainSet,k=i,K=j,target='y')
                 print()
```

```
TRYING fold =  5  and Round =  5

0.6315789473684211
0.6003937007874015
0.6320582877959927
0.6194368755676658
0.6138059701492538
AVG:  0.6194547563337469

TRYING fold =  5  and Round =  10

0.5949953660797034
0.6220984215413184
0.6452830188679246
0.6283595922150138
0.6087751371115173
AVG:  0.6199023071630956

TRYING fold =  5  and Round =  20

0.613531047265987
0.6116970278044104
0.6123222748815165
0.6606982990152194
0.5990867579908675
AVG:  0.6194670813916001

TRYING fold =  10  and Round =  5

0.5825242718446603
0.6142595978062156
0.6209523809523809
0.6245353159851301
0.6334519572953737
0.6092184368737475
0.6458333333333334
0.6329113924050633
0.6150943396226415
0.6139705882352942
AVG:  0.619275161435384

TRYING fold =  10  and Round =  10

0.6073500967117988
0.5891472868217055
0.6148148148148148
0.624087591240876
0.6564102564102564
```

```
0.635036496350365
0.6078799249530956
0.6368715083798883
0.6038461538461538
0.6165137614678899
AVG:  0.6191957890996844

TRYING fold =  10  and Round =  20

0.6421052631578948
0.5988700564971751
0.5973025048169556
0.6019417475728156
0.6206896551724138
0.6297709923664122
0.6052631578947368
0.6223908918406071
0.6539792387543253
0.6199261992619927
AVG:  0.6192239707335329

TRYING fold =  20  and Round =  5

0.6472727272727273
0.6353790613718412
0.5900383141762452
0.6093189964157706
0.6127946127946128
0.6119402985074627
0.6159695817490494
0.6022304832713755
0.5703422053231939
0.6183206106870229
0.6106870229007634
0.648
0.5845070422535211
0.633204633204633
0.6360424028268552
0.6759581881533101
0.6119402985074627
0.6412213740458015
0.6199261992619925
0.6184738955823293
AVG:  0.6196783974152986

TRYING fold =  20  and Round =  10

0.6385964912280702
```

```
0.6456140350877193
0.5498007968127491
0.5606060606060607
0.6124031007751938
0.6083333333333333
0.6520146520146521
0.6567164179104478
0.6691176470588235
0.6428571428571429
0.6153846153846154
0.5952380952380952
0.6219081272084807
0.6409266409266409
0.6109090909090908
0.5985401459854014
0.6564885496183206
0.5839416058394161
0.6293706293706294
0.5972222222222222
AVG:  0.6192994700193553

TRYING fold =  20  and Round =  20

0.6421404682274248
0.701219512195122
0.6413793103448275
0.5494505494505494
0.6324110671936758
0.6446886446886447
0.582089552238806
0.7295597484276728
0.7428571428571429
0.627177700348432
0.6188679245283019
0.6312056737588653
0.6315789473684211
0.606060606060606
0.6738351254480286
0.6621160409556314
0.5955882352941176
0.5813953488372093
0.628158844765343
0.5327510917030568
AVG:  0.6327265767345939
```