kmadden5-hw2-1

September 28, 2018

```
In [40]: import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt
         import sklearn
         from sklearn import metrics
         from sklearn.metrics import precision_score, recall_score, f1_score
         from pandas import Series, DataFrame
         import scipy
         from scipy import stats
         import math
```

```
In [2]: %matplotlib inline
```

```
In [3]: train = pd.read_csv("Dataset-football-train.txt", delimiter="\t")
```

```
In [4]: train.head(24)
```

Out[4]:

	ID	Date	Opponent	Is_Home_or_Away	\
0	1	9/5/15	Texas	Home	
1	2	9/12/15	Virginia	Away	
2	3	9/19/15	Georgia Tech	Home	
3	4	9/26/15	UMass	Home	
4	5	10/3/15	Clemson	Away	
5	6	10/10/15	Navy	Home	
6	7	10/17/15	USC	Home	
7	8	10/31/15	Temple	Away	
8	9	11/7/15	PITT	Away	
9	10	11/14/15	Wake Forest	Home	
10	11	11/21/15	Boston College	Away	
11	12	11/28/15	Stanford	Away	
12	13	9/4/16	Texas	Away	
13	14	9/10/16	Nevada	Home	
14	15	9/17/16	Michigan State	Home	
15	16	9/24/16	Duke	Home	
16	17	10/1/16	Syracuse	Home	
17	18	10/8/16	North Carolina State	Away	
18	19	10/15/16	Stanford	Home	
19	20	10/29/16	Miami Florida	Home	

```
20  21   11/5/16                 Navy           Home
21  22  11/12/16                 Army           Home
22  23  11/19/16         Virginia Tech           Home
23  24  11/26/16                  USC           Away

   Is_Opponent_in_AP25_Preseason Media Label
0                            Out   NBC   Win
1                            Out   ABC   Win
2                             In   NBC   Win
3                            Out   NBC   Win
4                             In   ABC  Lose
5                            Out   NBC   Win
6                             In   NBC   Win
7                            Out   ABC   Win
8                            Out   ABC   Win
9                            Out   NBC   Win
10                           Out   NBC   Win
11                            In   FOX  Lose
12                           Out   ABC  Lose
13                           Out   NBC   Win
14                           Out   NBC  Lose
15                           Out   NBC  Lose
16                           Out  ESPN   Win
17                           Out   ABC  Lose
18                            In   NBC  Lose
19                           Out   NBC   Win
20                           Out   CBS  Lose
21                           Out   NBC   Win
22                            In   NBC  Lose
23                            In   ABC  Lose
```

```
In [5]: test = pd.read_csv("Dataset-football-test.txt", delimiter="\t")

In [6]: test.head()

Out[6]:    ID     Date        Opponent Is_Home_or_Away Is_Opponent_in_AP25_Preseason  \
        0  25   9/2/17          Temple            Home                           Out
        1  26   9/9/17         Georgia            Home                            In
        2  27  9/16/17  Boston College            Away                           Out
        3  28  9/23/17  Michigan State            Away                           Out
        4  29  9/30/17      Miami Ohio            Home                           Out

          Media Label
        0   NBC   Win
        1   NBC  Lose
        2  ESPN   Win
        3   FOX   Win
        4   NBC   Win
```

Here we create the function to check the information gain.

```
In [7]: from collections import defaultdict
        def information_gain(column_x, column_y, yes_label, no_label):
            yes = 0
            no = 0
            for item in column_y:
                if item == yes_label:
                    yes += 1
                else:
                    no += 1
            try:
                hy1 = (-float(yes)/len(column_y))* math.log(float(yes)/len(column_y), 2)
            except:
                hy1=0
            try:
                hy2 = (-float(no)/len(column_y))*math.log(float(no)/len(column_y), 2)
            except:
                hy2=0
            H_Y = hy1 + hy2
            options = defaultdict(list)
            for index, item in enumerate(column_x):
                options[item].append(index)

            y_given_x = 0
            for item in options:
                y = 0
                n = 0
                for index in options[item]:
                    if column_y[index] == yes_label:
                        y += 1
                    else:
                        n += 1

                try:
                    y_given_x_yes = (float(y)/len(column_x))*(-float(y)/len(item))* (math.log(
                except:
                    y_given_x_yes = 0
                try:
                    y_given_x_no = (float(n)/len(column_x))*(-float(n)/len(item))* (math.log(f
                except:
                    y_given_x_no = 0
                y_given_x += y_given_x_yes + y_given_x_no
            information_gain = abs(H_Y - y_given_x)
            print(information_gain)
            return

In [8]: Label = list(train["Label"])
```

The dates are exact, so I decided to sort them into buckets based on month and year. This way if we actually split on date it is information that might be helpful in predicting. Otherwise, since there's no way the team would play two games on one day, the date would never be useful.

```
In [10]: i = 0
         for item in train["Date"]:
             if item[0] == "9" and item[len(item)-1] =="5":
                 train["Date"][i] = "sept_15"
             elif item[0] == "9" and item[len(item)-1] =="6":
                 train["Date"][i] = "sept_16"
             elif item[1] == "0" and item[len(item)-1] =="5":
                 train["Date"][i] = "oct_15"
             elif item[1] == "0" and item[len(item)-1] =="6":
                 train["Date"][i] = "oct_16"
             elif item[1] == "1" and item[len(item)-1] =="5":
                 train["Date"][i] = "nov_15"
             else:
                 train["Date"][i] = "nov_16"
             i += 1
```

```
/Users/kailee/anaconda/lib/python2.7/site-packages/ipykernel/__main__.py:4: SettingWithCopyWarn
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
/Users/kailee/anaconda/lib/python2.7/site-packages/ipykernel/__main__.py:8: SettingWithCopyWarn
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
/Users/kailee/anaconda/lib/python2.7/site-packages/ipykernel/__main__.py:12: SettingWithCopyWa
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
/Users/kailee/anaconda/lib/python2.7/site-packages/ipykernel/__main__.py:6: SettingWithCopyWarn
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
/Users/kailee/anaconda/lib/python2.7/site-packages/ipykernel/__main__.py:10: SettingWithCopyWa
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
/Users/kailee/anaconda/lib/python2.7/site-packages/ipykernel/__main__.py:14: SettingWithCopyWa
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
```

```
In [11]: train["Date"]
```

```
Out[11]: 0     sept_15
         1     sept_15
         2     sept_15
         3     sept_15
         4      oct_15
         5      oct_15
         6      oct_15
         7      oct_15
         8      nov_15
         9      nov_15
         10     nov_15
         11     nov_15
         12    sept_16
         13    sept_16
         14    sept_16
         15    sept_16
         16     oct_16
         17     oct_16
         18     oct_16
         19     oct_16
         20     nov_16
         21     nov_16
         22     nov_16
         23     nov_16
         Name: Date, dtype: object
```

I turn all of the feature columns into lists so that they are in the proper format to go into my information gain function.

```
In [12]: Media = list(train["Media"])
         H_A = list(train["Is_Home_or_Away"])
         Date = list(train["Date"])
         Preseason = list(train["Is_Opponent_in_AP25_Preseason"])
         Opponent = list(train["Opponent"])
```

```
In [13]: print("IG for Media is:")
         information_gain(Media, Label, "Win", "Lose")
         print("IG for H_A is:")
         information_gain(H_A, Label, "Win", "Lose")
         print("IG for Date is:")
         information_gain(Date, Label, "Win", "Lose")
         print("IG for Preseason is:")
         information_gain(Preseason, Label, "Win", "Lose")
         print("IG for Opponent is:")
         information_gain(Opponent, Label, "Win", "Lose")
```

```
IG for Media is:
3.5119220121
IG for H_A is:
```

```
2.52454807158
IG for Date is:
0.491375199749
IG for Preseason is:
5.92426380405
IG for Opponent is:
0.567158476583
```

Preseason has the highest information gain, so that is what we will branch on. This means that we need to sort all our other values into either branch 1 or branch 2 depending on whether they were in or out of the preseason.

```
In [15]: Media1 = []
         H_A1 = []
         Date1 = []
         Opponent1 = []
         Label1 = []
         Media2 = []
         H_A2 = []
         Date2 = []
         Opponent2 = []
         Label2 = []
         for index, item in enumerate(Preseason):
             if item == "In":
                 Media1.append(Media[index])
                 H_A1.append(H_A[index])
                 Date1.append(Date[index])
                 Opponent1.append(Opponent[index])
                 Label1.append(Label[index])
             else:
                 Media2.append(Media[index])
                 H_A2.append(H_A[index])
                 Date2.append(Date[index])
                 Opponent2.append(Opponent[index])
                 Label2.append(Label[index])
         In = pd.DataFrame()
         Out = pd.DataFrame()
         In["Media"] = Media1
         In["H_A"] = H_A1
         In["Date"] = Date1
         In["Opponent"] = Opponent1
         In["Label"] = Label1
         Out["Media"] = Media2
         Out["H_A"] = H_A2
         Out["Date"] = Date2
         Out["Opponent"] = Opponent2
         Out["Label"] = Label2
```

```
In [16]: In.head()
```

```
Out[16]:   Media   H_A     Date      Opponent Label
         0   NBC  Home  sept_15  Georgia Tech   Win
         1   ABC  Away   oct_15       Clemson  Lose
         2   NBC  Home   oct_15           USC   Win
         3   FOX  Away   nov_15      Stanford  Lose
         4   NBC  Home   oct_16      Stanford  Lose
```

```
In [17]: print("IG for Media in In is:")
         information_gain(Media1, Label1, "Win", "Lose")
         print("IG for H_A in In is:")
         information_gain(H_A1, Label1, "Win", "Lose")
         print("IG for Date in In is:")
         information_gain(Date1, Label1, "Win", "Lose")
         print("IG for Opponent in In is:")
         information_gain(Opponent1, Label1, "Win", "Lose")
```

```
IG for Media in In is:
0.453381877644
IG for H_A in In is:
0.44400137237
IG for Date in In is:
0.408691896551
IG for Opponent in In is:
0.428679360789
```

We will split on Media since this is the highest information gain.

```
In [18]: H_A_nbc = []
         H_A_abc = []
         H_A_fox = []
         Label_nbc = []
         Label_abc = []
         Label_fox = []
         Date_nbc = []
         Date_abc = []
         Date_fox = []
         Opponent_nbc = []
         Opponent_abc = []
         Opponent_fox = []
         for index, item in enumerate(Media1):
             if item == "NBC":
                 H_A_nbc.append(H_A1[index])
                 Date_nbc.append(Date1[index])
                 Opponent_nbc.append(Opponent1[index])
                 Label_nbc.append(Label1[index])
             elif item == "ABC":
```

```
            H_A_abc.append(H_A1[index])
            Date_abc.append(Date1[index])
            Opponent_abc.append(Opponent1[index])
            Label_abc.append(Label1[index])
        else:
            H_A_fox.append(H_A1[index])
            Date_fox.append(Date1[index])
            Opponent_fox.append(Opponent1[index])
            Label_fox.append(Label1[index])
```

Now we return to the Out branch.

```
In [19]: Out.head()
```

Out[19]:

	Media	H_A	Date	Opponent	Label
0	NBC	Home	sept_15	Texas	Win
1	ABC	Away	sept_15	Virginia	Win
2	NBC	Home	sept_15	UMass	Win
3	NBC	Home	oct_15	Navy	Win
4	ABC	Away	oct_15	Temple	Win

```
In [20]: print("IG for Media in Out is:")
         information_gain(Media2, Label2, "Win", "Lose")
         print("IG for H_A in Out is:")
         information_gain(H_A2, Label2, "Win", "Lose")
         print("IG for Date in Out is:")
         information_gain(Date2, Label2, "Win", "Lose")
         print("IG for Opponent in Out is:")
         information_gain(Opponent2, Label2, "Win", "Lose")
```

```
IG for Media in Out is:
2.49746609611
IG for H_A in Out is:
1.70140255558
IG for Date in Out is:
0.376915748979
IG for Opponent in Out is:
0.470218797116
```

We will split on Media since this is the highest information gain.

```
In [21]: H_A_nbc2 = []
         H_A_abc2 = []
         H_A_cbs2 = []
         H_A_espn2 = []
         Label_nbc2 = []
         Label_abc2 = []
         Label_cbs2 = []
```

```
Label_espn2 = []
Date_nbc2 = []
Date_abc2 = []
Date_cbs2 = []
Date_espn2 = []
Opponent_nbc2 = []
Opponent_abc2 = []
Opponent_cbs2 = []
Opponent_espn2 = []
for index, item in enumerate(Media2):
    if item == "NBC":
        H_A_nbc2.append(H_A2[index])
        Date_nbc2.append(Date2[index])
        Opponent_nbc2.append(Opponent2[index])
        Label_nbc2.append(Label2[index])
    elif item == "ABC":
        H_A_abc2.append(H_A2[index])
        Date_abc2.append(Date2[index])
        Opponent_abc2.append(Opponent2[index])
        Label_abc2.append(Label2[index])
    elif item == "CBS":
        H_A_cbs2.append(H_A2[index])
        Date_cbs2.append(Date2[index])
        Opponent_cbs2.append(Opponent2[index])
        Label_cbs2.append(Label2[index])
    else:
        H_A_espn2.append(H_A2[index])
        Date_espn2.append(Date2[index])
        Opponent_espn2.append(Opponent2[index])
        Label_espn2.append(Label2[index])
```

Now we calculate the information gain for these four branches: Out_NBC and Out_ABC. We don't have to worry about Out_CBS and Out_ESPN since they each only have one game contained within them, so there is no possibility of splitting, rather they are the final nodes.

```
In [22]: print(Date2)
```

```
['sept_15', 'sept_15', 'sept_15', 'oct_15', 'oct_15', 'nov_15', 'nov_15', 'nov_15', 'sept_16',
```

```
In [23]: print("IG for H_A in Out_NBC is:")
         information_gain(H_A_nbc2, Label_nbc2, "Win", "Lose")
         print("IG for Date in Out_NBC is:")
         information_gain(Date_nbc2, Label_nbc2, "Win", "Lose")
         print("IG for Opponent in Out_NBC is:")
         information_gain(Opponent_nbc2, Label_nbc2, "Win", "Lose")
```

```
IG for H_A in Out_NBC is:
1.56093787441
```

```
IG for Date in Out_NBC is:
0.240355884586
IG for Opponent in Out_NBC is:
0.321663178886
```

This means we will split on Home/Away for Out_NBC. Thus we will create Out_NBC_Home and Out_NBC_Away.

```
In [24]: print("IG for H_A in Out_ABC is:")
         information_gain(H_A_abc2, Label_abc2, "Win", "Lose")
         print("IG for Date in Out_ABC is:")
         information_gain(Date_abc2, Label_abc2, "Win", "Lose")
         print("IG for Opponent in Out_ABC is:")
         information_gain(Opponent_abc2, Label_abc2, "Win", "Lose")
```

```
IG for H_A in Out_ABC is:
0.584183719779
IG for Date in Out_ABC is:
0.552034063122
IG for Opponent in Out_ABC is:
0.57368877302
```

We will split on Opponent for Out_ABC, since all the games in Out_ABC are Away so it makes no sense to split on Home/Away. This means we will create Out_ABC_Virginia, Out_ABC_Temple, Out_ABC_Pitt, Out_ABC_Texas, and Out_ABC_NCState. However, these each only contain one game, so we do not need to calculate the information gain or split any further.

```
In [25]: print(Date_nbc2)
```

```
['sept_15', 'sept_15', 'oct_15', 'nov_15', 'nov_15', 'sept_16', 'sept_16', 'sept_16', 'oct_16'
```

Now, let us return to the In branch that is being split on Media. We have In_NBC, In_ABC, and In_FOX, however In_FOX only has one game so has no need for any new branches. Within In_ABC there are only two games and both of them are losses, so we also have no need to split.

```
In [26]: print("IG for H_A in In_NBC is:")
         information_gain(H_A_nbc, Label_nbc, "Win", "Lose")
         print("IG for Date in In_NBC is:")
         information_gain(Date_nbc, Label_nbc, "Win", "Lose")
         print("IG for Opponent in In_NBC is:")
         information_gain(Opponent_nbc, Label_nbc, "Win", "Lose")
```

```
IG for H_A in In_NBC is:
0.5
IG for Date in In_NBC is:
```

```
0.576617011622
IG for Opponent in In_NBC is:
0.62832077057
```

This means we will split on Opponent. Thus we will create In_NBC_GT, In_NBC_USC, In_NBC_Stanford, and In_NBC_VT. However, for each of these there is only one game, thus there is no reason to calculate the information gain after.

Now we return to the Out branch, and create the lists for Out_NBC_Home and Out_NBC_Away.

```
In [27]: Label_home = []
         Label_away = []
         Date_home = []
         Date_away = []
         Opponent_home = []
         Opponent_away = []
         for index, item in enumerate(H_A_nbc2):
             if item == "Home":
                 Date_home.append(Date_nbc2[index])
                 Opponent_home.append(Opponent_nbc2[index])
                 Label_home.append(Label_nbc2[index])
             else:
                 Date_away.append(Date_nbc2[index])
                 Opponent_away.append(Opponent_nbc2[index])
                 Label_away.append(Label_nbc2[index])

In [28]: print(Label_away)

['Win']
```

There is only one game in Out_NBC_Away, so we are done with this branch and do not need to calculate information gain. However, we do need to calculate information gain for Out_NBC_Home to decide what to split on next.

```
In [29]: print("IG for Date in Out_NBC_Home is:")
         information_gain(Date_home, Label_home, "Win", "Lose")
         print("IG for Opponent in Out_NBC_Home is:")
         information_gain(Opponent_home, Label_home, "Win", "Lose")

IG for Date in Out_NBC_Home is:
0.298659226585
IG for Opponent in Out_NBC_Home is:
0.349682813507
```

So now we will split on Opponent for Out_NBC_Home. However, this splits our tree into all individual games, so now there is no more information to be gained and our decision tree is done.

Now that we have split the tree until there is no information gain or all the features have been used, we can test our tree. For a visualization of the tree please see the PDF file. Below we will write the code to test our new data according to the decision tree branch rules.

```
In [32]: def predict_outcome(dataframe):
             predictions = []
             for index, row in dataframe.iterrows():
                 if row["Is_Opponent_in_AP25_Preseason"] == "In":
                     if row["Media"] == "Fox" or "ABC":
                         predictions.append("Lose")
                     elif row["Media"] == "NBC":
                         if row["Opponent"] == "Virginia Tech" or "Stanford":
                             predictions.append("Lose")
                         elif row["Opponent"] == "USC" or "Georgia Tech":
                             predictions.append("Win")
                         else:
                             predictions.append("Win")
                     else:
                         predictions.append("Win")
                 else:
                     if row["Media"] == "CBS":
                         predictions.append("Lose")
                     elif row["Media"] == "ESPN":
                         predictions.append("Win")
                     elif row["Media"] == "ABC":
                         if row["Opponent"] == "Virginia" or "Temple" or "PITT":
                             predictions.append("Win")
                         elif row["Opponent"] == "Texas" or "North Carolina State":
                             predictions.append("Lose")
                         else:
                             predictions.append("Win")
                     elif row["Media"] == "NBC":
                         if row["Opponent"] == "Texas" or "Army" or "UMass" or "Nevada" or "Wal
                             predictions.append("Win")
                         elif row["Opponent"] == "Michigan State" or "Duke":
                             predictions.append("Lose")
                         else:
                             predictions.append("Win")
                     else:
                         predictions.append("Win")
             return predictions

In [34]: predictions = predict_outcome(test)

In [51]: actual = list(test["Label"])

In [59]: precision_score(actual, predictions, pos_label="Win")

Out[59]: 1.0
```

```
In [60]: recall_score(actual, predictions, pos_label="Win")

Out[60]: 0.88888888888888884

In [61]: f1_score(actual, predictions, pos_label="Win")

Out[61]: 0.94117647058823528
```

kmadden5-hw2-2

September 28, 2018

```
In [3]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import sklearn
        from sklearn import metrics
        from sklearn.metrics import precision_score, recall_score, f1_score
        from pandas import Series, DataFrame
        import scipy
        from scipy import stats
        import math
```

```
In [4]: %matplotlib inline
```

```
In [5]: train = pd.read_csv("Dataset-football-train.txt", delimiter="\t")
```

```
In [93]: train.head()
```

Out[93]:

	ID	Date	Opponent	Is_Home_or_Away	Is_Opponent_in_AP25_Preseason	Media	Label
0	1	sept_15	Texas	Home	Out	NBC	Win
1	2	sept_15	Virginia	Away	Out	ABC	Win
2	3	sept_15	Georgia Tech	Home	In	NBC	Win
3	4	sept_15	UMass	Home	Out	NBC	Win
4	5	oct_15	Clemson	Away	In	ABC	Lose

```
In [6]: test = pd.read_csv("Dataset-football-test.txt", delimiter="\t")
```

Now we create the functions for information gain, split information, and gain ratio.

```
In [13]: from collections import defaultdict
         def information_gain(column_x, column_y, yes_label, no_label):
             yes = 0
             no = 0
```

```
            for item in column_y:
                if item == yes_label:
                    yes += 1
                else:
                    no += 1
            try:
                hy1 = (-float(yes)/len(column_y))* math.log(float(yes)/len(column_y), 2)
            except:
                hy1=0
            try:
                hy2 = (-float(no)/len(column_y))*math.log(float(no)/len(column_y), 2)
            except:
                hy2=0
            H_Y = hy1 + hy2
            options = defaultdict(list)
            for index, item in enumerate(column_x):
                options[item].append(index)

            y_given_x = 0
            for item in options:
                y = 0
                n = 0
                for index in options[item]:
                    if column_y[index] == yes_label:
                        y += 1
                    else:
                        n += 1

                try:
                    y_given_x_yes = (float(y)/len(column_x))*(-float(y)/len(item))* (math.log
                except:
                    y_given_x_yes = 0
                try:
                    y_given_x_no = (float(n)/len(column_x))*(-float(n)/len(item))* (math.log(
                except:
                    y_given_x_no = 0
                y_given_x += y_given_x_yes + y_given_x_no
            information_gain = abs(H_Y - y_given_x)
            return information_gain
```

In [38]:
```
def split_info(column_x):
    splitinfo = 0
    unique_values = set()
    for item in column_x:
        unique_values.add(item)
    for value in unique_values:
        count = 0
        for x in column_x:
```

```
                    if value == x:
                        count += 1
                try:
                    splitinfo -= (float(count)/len(column_x)) * (math.log(float(count)/len(co
                except:
                    continue
            return splitinfo
```

```
In [53]: def gain_ratio(informationgain, splitinformation):
            try:
                gain = float(informationgain)/splitinformation
            except:
                gain = 0
            print(gain)
            return
```

The dates are exact, so I decided to sort them into buckets based on month and year. This way if we actually split on date it is information that might be helpful in predicting. Otherwise, since there's no way the team would play two games on one day, the date would never be useful.

```
In [10]: i = 0
         for item in train["Date"]:
             if item[0] == "9" and item[len(item)-1] =="5":
                 train["Date"][i] = "sept_15"
             elif item[0] == "9" and item[len(item)-1] =="6":
                 train["Date"][i] = "sept_16"
             elif item[1] == "0" and item[len(item)-1] =="5":
                 train["Date"][i] = "oct_15"
             elif item[1] == "0" and item[len(item)-1] =="6":
                 train["Date"][i] = "oct_16"
             elif item[1] == "1" and item[len(item)-1] =="5":
                 train["Date"][i] = "nov_15"
             else:
                 train["Date"][i] = "nov_16"
             i += 1
```

```
/Users/kailee/anaconda/envs/py36/lib/python3.6/site-packages/ipykernel_launcher.py:4: SettingW
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  after removing the cwd from sys.path.
/Users/kailee/anaconda/envs/py36/lib/python3.6/site-packages/ipykernel_launcher.py:8: SettingW
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html

/Users/kailee/anaconda/envs/py36/lib/python3.6/site-packages/ipykernel_launcher.py:12: Setting
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  if sys.path[0] == '':
/Users/kailee/anaconda/envs/py36/lib/python3.6/site-packages/ipykernel_launcher.py:6: SettingW
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html

/Users/kailee/anaconda/envs/py36/lib/python3.6/site-packages/ipykernel_launcher.py:10: Setting
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  # Remove the CWD from sys.path while we load stuff.
/Users/kailee/anaconda/envs/py36/lib/python3.6/site-packages/ipykernel_launcher.py:14: Setting
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
```

I turn all of the feature columns into lists so that they are in the proper format to go into my information gain function.

```
In [11]: Label = list(train["Label"])
         Media = list(train["Media"])
         H_A = list(train["Is_Home_or_Away"])
         Date = list(train["Date"])
         Preseason = list(train["Is_Opponent_in_AP25_Preseason"])
         Opponent = list(train["Opponent"])
```

Now we will find the gain ratio for all the features.

```
In [39]: print("GR for Media is:")
         gain_ratio(information_gain(Media, Label, "Win", "Lose"), split_info(Media))
         print("GR for H_A is:")
         gain_ratio(information_gain(H_A, Label, "Win", "Lose"), split_info(H_A))
         print("GR for Date is:")
         gain_ratio(information_gain(Date, Label, "Win", "Lose"), split_info(Date))
         print("GR for Preseason is:")
         gain_ratio(information_gain(Preseason, Label, "Win", "Lose"), split_info(Preseason))
         print("GR for Opponent is:")
         gain_ratio(information_gain(Opponent, Label, "Win", "Lose"), split_info(Opponent))
```

```
GR for Media is:
2.2728038661523553
GR for H_A is:
2.6450734821292787
GR for Date is:
0.19008987542825642
GR for Preseason is:
```

```
6.802739132585862
GR for Opponent is:
0.1333979174226363
```

Preseason has the highest gain ratio, so that is what we will branch on. This means that we need to sort all our other values into either branch 1 or branch 2 depending on whether they were in or out of the preseason.

```
In [40]: Media1 = []
         H_A1 = []
         Date1 = []
         Opponent1 = []
         Label1 = []
         Media2 = []
         H_A2 = []
         Date2 = []
         Opponent2 = []
         Label2 = []
         for index, item in enumerate(Preseason):
             if item == "In":
                 Media1.append(Media[index])
                 H_A1.append(H_A[index])
                 Date1.append(Date[index])
                 Opponent1.append(Opponent[index])
                 Label1.append(Label[index])
             else:
                 Media2.append(Media[index])
                 H_A2.append(H_A[index])
                 Date2.append(Date[index])
                 Opponent2.append(Opponent[index])
                 Label2.append(Label[index])
         In = pd.DataFrame()
         Out = pd.DataFrame()
         In["Media"] = Media1
         In["H_A"] = H_A1
         In["Date"] = Date1
         In["Opponent"] = Opponent1
         In["Label"] = Label1
         Out["Media"] = Media2
         Out["H_A"] = H_A2
         Out["Date"] = Date2
         Out["Opponent"] = Opponent2
         Out["Label"] = Label2

In [41]: In.head()

Out[41]:   Media   H_A     Date      Opponent Label
         0   NBC  Home  sept_15  Georgia Tech   Win
```

```
1   ABC  Away   oct_15       Clemson  Lose
2   NBC  Home   oct_15           USC   Win
3   FOX  Away   nov_15      Stanford  Lose
4   NBC  Home   oct_16      Stanford  Lose
```

```
In [46]: print("GR for Media In Preseason is:")
         gain_ratio(information_gain(Media1, Label1, "Win", "Lose"), split_info(Media1))
         print("GR for H_A In Preseason is:")
         gain_ratio(information_gain(H_A1, Label1, "Win", "Lose"), split_info(H_A1))
         print("GR for Date In Preseason is:")
         gain_ratio(information_gain(Date1, Label1, "Win", "Lose"), split_info(Date1))
         print("GR for Opponent In Preseason is:")
         gain_ratio(information_gain(Opponent1, Label1, "Win", "Lose"), split_info(Opponent1))
```

```
GR for Media In Preseason is:
0.3288274626044878
GR for H_A In Preseason is:
0.4506584375036809
GR for Date In Preseason is:
0.18278414959221004
GR for Opponent In Preseason is:
0.1917233815272914
```

We will split on Home/Away since it is the highest gain ratio.

```
In [44]: Media_home = []
         Date_home = []
         Opponent_home = []
         Label_home = []
         Media_away = []
         Date_away = []
         Opponent_away = []
         Label_away = []
         for index, item in enumerate(H_A1):
             if item == "Home":
                 Media_home.append(Media1[index])
                 Date_home.append(Date1[index])
                 Opponent_home.append(Opponent1[index])
                 Label_home.append(Label1[index])
             else:
                 Media_away.append(Media1[index])
                 Date_away.append(Date1[index])
                 Opponent_away.append(Opponent1[index])
                 Label_away.append(Label1[index])
```

```
In [54]: print("GR for Media in In_Home is:")
         gain_ratio(information_gain(Media_home, Label_home, "Win", "Lose"), split_info(Media_
         print("GR for Date in In_Home is:")
```

```
        gain_ratio(information_gain(Date_home, Label_home, "Win", "Lose"), split_info(Date_hom
        print("GR for Opponent in In_Home is:")
        gain_ratio(information_gain(Opponent_home, Label_home, "Win", "Lose"), split_info(Oppo
```

```
GR for Media in In_Home is:
0
GR for Date in In_Home is:
0.28830850581104195
GR for Opponent in In_Home is:
0.31416038528480106
```

The highest gain ratio is Opponent, so we will split on that. Now we check for Away.

```
In [56]: print("GR for Media in In_Away is:")
         gain_ratio(information_gain(Media_away, Label_away, "Win", "Lose"), split_info(Media_a
         print("GR for Date in In_Away is:")
         gain_ratio(information_gain(Date_away, Label_away, "Win", "Lose"), split_info(Date_awa
         print("GR for Opponent in In_Away is:")
         gain_ratio(information_gain(Opponent_away, Label_away, "Win", "Lose"), split_info(Oppo
```

```
GR for Media in In_Away is:
0.4748908380134757
GR for Date in In_Away is:
0.27182162559524287
GR for Opponent in In_Away is:
0.27432227074380156
```

The highest gain ratio is Media so we will split on that. But first, we return to the Out branch.

```
In [45]: Out.head()
```

```
Out[45]:   Media   H_A     Date  Opponent Label
         0   NBC  Home  sept_15     Texas   Win
         1   ABC  Away  sept_15  Virginia   Win
         2   NBC  Home  sept_15     UMass   Win
         3   NBC  Home   oct_15      Navy   Win
         4   ABC  Away   oct_15    Temple   Win
```

```
In [47]: print("GR for Media in Out Preseason is:")
         gain_ratio(information_gain(Media2, Label2, "Win", "Lose"), split_info(Media2))
         print("GR for H_A in Out Preseason is:")
         gain_ratio(information_gain(H_A2, Label2, "Win", "Lose"), split_info(H_A2))
         print("GR for Date in Out Preseason is:")
         gain_ratio(information_gain(Date2, Label2, "Win", "Lose"), split_info(Date2))
         print("GR for Opponent in Out Preseason is:")
         gain_ratio(information_gain(Opponent2, Label2, "Win", "Lose"), split_info(Opponent2))
```

```
GR for Media in Out Preseason is:
1.7218353111111935
GR for H_A in Out Preseason is:
1.816442622529741
GR for Date in Out Preseason is:
0.1482471247085744
GR for Opponent in Out Preseason is:
0.12206599213447909
```

The highest gain ratio is Home/Away, so we will split on that.

```
In [48]: Media_home2 = []
         Date_home2 = []
         Opponent_home2 = []
         Label_home2 = []
         Media_away2 = []
         Date_away2 = []
         Opponent_away2 = []
         Label_away2 = []
         for index, item in enumerate(H_A2):
             if item == "Home":
                 Media_home2.append(Media2[index])
                 Date_home2.append(Date2[index])
                 Opponent_home2.append(Opponent2[index])
                 Label_home2.append(Label2[index])
             else:
                 Media_away2.append(Media2[index])
                 Date_away2.append(Date2[index])
                 Opponent_away2.append(Opponent2[index])
                 Label_away2.append(Label2[index])
```

```
In [49]: print("GR for Media in Out_Home is:")
         gain_ratio(information_gain(Media_home2, Label_home2, "Win", "Lose"), split_info(Media
         print("GR for Date in Out_Home is:")
         gain_ratio(information_gain(Date_home2, Label_home2, "Win", "Lose"), split_info(Date_
         print("GR for Opponent in Out_Home is:")
         gain_ratio(information_gain(Opponent_home2, Label_home2, "Win", "Lose"), split_info(Op
```

```
GR for Media in Out_Home is:
2.8827292687087667
GR for Date in Out_Home is:
0.14844250168055695
GR for Opponent in Out_Home is:
0.1301713511509548
```

The highest gain ratio is Media so we will split on that. But first we return to the Away branch to check the gain ratios there.

```
In [51]: print("GR for Media in Out_Away is:")
         gain_ratio(information_gain(Media_away2, Label_away2, "Win", "Lose"), split_info(Media
         print("GR for Date in Out_Away is:")
         gain_ratio(information_gain(Date_away2, Label_away2, "Win", "Lose"), split_info(Date_a
         print("GR for Opponent in Out_Away is:")
         gain_ratio(information_gain(Opponent_away2, Label_away2, "Win", "Lose"), split_info(Op
```

```
GR for Media in Out_Away is:
1.0772716011696248
GR for Date in Out_Away is:
0.20647107469050285
GR for Opponent in Out_Away is:
0.20964275582594497
```

The highest gain ratio is Media so we will split on that. For now though, we consider the In branch and the splits that we have determined must occur there. For In_Home, the gain ratio was highest for Opponent, but since there are no repeat opponents in that list, we will not need to calculate gain ratio for those branches, as there will be nothing left to split on. For In_Away, the highest gain ratio was Media, and there are two different branches that must be created, one for ABC and the other for FOX. However, all of the games in In_Away were loses, meaning we do not actually need to branch. So, the In side of the tree is complete. We look at the Out side again. We are splitting on Media for both Out_Home and Out_Away, and so we must sort our games into the correct lists. First we will sort for Home.

```
In [80]: Date_NBC = []
         Opponent_NBC = []
         Label_NBC = []
         Date_ESPN = []
         Opponent_ESPN = []
         Label_ESPN = []
         Date_CBS = []
         Opponent_CBS = []
         Label_CBS = []
         for index, item in enumerate(Media_home2):
             if item == "NBC":
                 Date_NBC.append(Date_home2[index])
                 Opponent_NBC.append(Opponent_home2[index])
                 Label_NBC.append(Label_home2[index])
             elif item == "ESPN":
                 Date_ESPN.append(Date_home2[index])
                 Opponent_ESPN.append(Opponent_home2[index])
                 Label_ESPN.append(Label_home2[index])
             else:
                 Date_CBS.append(Date_home2[index])
                 Opponent_CBS.append(Opponent_home2[index])
                 Label_CBS.append(Label_home2[index])
```

Since ESPN and CBS only have one game each, we will not need to calculate gain ratio for those or split. So we only calculate the gain ratio for NBC.

```
In [81]: print("GR for Date in Out_Home_NBC is:")
         gain_ratio(information_gain(Date_NBC, Label_NBC, "Win", "Lose"), split_info(Date_NBC))
         print("GR for Opponent in Out_Home_NBC is:")
         gain_ratio(information_gain(Opponent_NBC, Label_NBC, "Win", "Lose"), split_info(Oppon
```

```
GR for Date in Out_Home_NBC is:
0.12344443055879188
GR for Opponent in Out_Home_NBC is:
0.11031264567713367
```

The highest gain ratio is Date, so we will split on that. First we sort Out_Away for the Media branch though.

```
In [82]: Date_NBC2 = []
         Opponent_NBC2 = []
         Label_NBC2 = []
         Date_ABC2 = []
         Opponent_ABC2 = []
         Label_ABC2 = []
         for index, item in enumerate(Media_away2):
             if item == "NBC":
                 Date_NBC2.append(Date_away2[index])
                 Opponent_NBC2.append(Opponent_away2[index])
                 Label_NBC2.append(Label_away2[index])
             else:
                 Date_ABC2.append(Date_away2[index])
                 Opponent_ABC2.append(Opponent_away2[index])
                 Label_ABC2.append(Label_away2[index])
```

Since there is only one game in NBC2, we do not need to calculate gain ratio for that, since there is nothing else to split on. So we only calculate it for ABC2.

```
In [83]: print("GR for Date in Out_Away_ABC is:")
         gain_ratio(information_gain(Date_ABC2, Label_ABC2, "Win", "Lose"), split_info(Date_ABC
         print("GR for Opponent in Out_Away_ABC is:")
         gain_ratio(information_gain(Opponent_ABC2, Label_ABC2, "Win", "Lose"), split_info(Oppo
```

```
GR for Date in Out_Away_ABC is:
0.23774813024470418
GR for Opponent in Out_Away_ABC is:
0.24707430616942733
```

The highest gain ratio is Opponent, so we will split on that. However, since there are no repeat opponents in this list, then we are done for this branch. So now we go back to Out_Home_NBC and split on Date. We only need to create lists for branches that have more than one game, that is, sept15 and sept16.

```
In [92]: Date_NBC
```

```
Out[92]: ['sept_15',
          'sept_15',
          'oct_15',
          'nov_15',
          'sept_16',
          'sept_16',
          'sept_16',
          'oct_16',
          'nov_16']
```

```
In [87]: Opponent_sept15 = []
         Label_sept15 = []
         Opponent_sept16 = []
         Label_sept16 = []
         for index, item in enumerate(Date_NBC):
             if item == "sept_15":
                 Opponent_sept15.append(Opponent_NBC[index])
                 Label_sept15.append(Label_NBC[index])
             else:
                 Opponent_sept16.append(Opponent_NBC[index])
                 Label_sept16.append(Label_NBC[index])
```

In the sept15 list we only have wins, which means we are done. For the sept16 list since we have some wins and some losses, we will have to split on the last feature Opponents, but these are all individual games, so we are also done. Now the tree is complete! Please see the PDF file for a drawing.

Now we can construct a function to test our test data based on our constructed tree.

```
In [88]: Label_sept15
```

```
Out[88]: ['Win', 'Win']
```

```
In [90]: Opponent_sept16
```

```
Out[90]: ['Navy',
          'Wake Forest',
          'Nevada',
          'Michigan State',
          'Duke',
          'Miami Florida',
          'Army']
```

```
In [116]: def predict(dataframe):
              predictions = []
              for index, row in dataframe.iterrows():
                  if row["Is_Opponent_in_AP25_Preseason"] == "In":
                      if row["Is_Home_or_Away"] == "Home":
                          if row["Opponent"] == "Stanford" or "Virginia Tech":
                              predictions.append("Lose")
```

```
                    else:
                        predictions.append("Win")
                else:
                    predictions.append("Lose")
            else:
                if row["Is_Home_or_Away"] == "Home":
                    if row["Media"] == "CBS":
                        predictions.append("Lose")
                    elif row["Media"] == "ESPN":
                        predictions.append("Win")
                    elif row["Media"] == "NBC":
                        if row["Date"] == "sept_16":
                            if row["Opponent"] == "Nevada":
                                predictions.append("Win")
                            else:
                                predictions.append("Lose")
                        else:
                            predictions.append("Win")
                    else:
                        predictions.append("Win")
                else:
                    if row["Media"] == "ABC":
                        if row["Opponent"] == "North Carolina State" or "Texas":
                            predictions.append("Lose")
                        else:
                            predictions.append("Win")
                    else:
                        predictions.append("Win")
        return predictions
```

Now we do the same Date preprocessing for the test set that we did on the train set.

```
In [95]: i = 0
         for item in test["Date"]:
             print(item)
             if item[0] == "9" and item[len(item)-1] =="7":
                 test["Date"][i] = "sept_17"
             elif item[1] == "0" and item[len(item)-1] =="7":
                 test["Date"][i] = "oct_17"
             else:
                 test["Date"][i] = "nov_17"
             i += 1
```

```
/Users/kailee/anaconda/envs/py36/lib/python3.6/site-packages/ipykernel_launcher.py:14: SettingW
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
```

```
In [108]: test.head()
```

Out[108]:

	ID	Date	Opponent	Is_Home_or_Away	Is_Opponent_in_AP25_Preseason	Media	Label
0	25	nov_16	Temple	Home	Out	NBC	Win
1	26	nov_16	Georgia	Home	In	NBC	Lose
2	27	nov_16	Boston College	Away	Out	ESPN	Win
3	28	nov_16	Michigan State	Away	Out	FOX	Win
4	29	nov_16	Miami Ohio	Home	Out	NBC	Win

```
In [117]: predictions = predict(test)

In [118]: print(predictions)
```

```
['Win', 'Lose', 'Win', 'Win', 'Win', 'Lose', 'Lose', 'Win', 'Win', 'Lose', 'Win', 'Lose']
```

```
In [119]: actual = list(test["Label"])

In [121]: precision_score(actual, predictions, pos_label="Win")
```

```
Out[121]: 1.0
```

```
In [122]: recall_score(actual, predictions, pos_label="Win")
```

```
Out[122]: 0.7777777777777778
```

```
In [123]: f1_score(actual, predictions, pos_label="Win")
```

```
Out[123]: 0.8750000000000001
```

kmadden5-hw2-3

September 28, 2018

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
from sklearn import metrics
from sklearn.metrics import precision_score, recall_score, f1_score
from pandas import Series, DataFrame
import scipy
from scipy import stats
import math
import csv
from sklearn import feature_extraction
```

In [2]:
```
data_train = pd.read_csv("Dataset-football-train.txt", delimiter="\t")
```

In [3]:
```
data_test = pd.read_csv("Dataset-football-test.txt", delimiter="\t")
```

In [4]:
```
i = 0
for item in data_train["Date"]:
    if item[0] == "9":
        data_train["Date"][i] = "sept"
    elif item[1] == "0":
        data_train["Date"][i] = "oct"
    else:
        data_train["Date"][i] = "nov"
    i += 1
```

```
/Users/kailee/anaconda/envs/py36/lib/python3.6/site-packages/ipykernel_launcher.py:4: SettingW
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  after removing the cwd from sys.path.
/Users/kailee/anaconda/envs/py36/lib/python3.6/site-packages/ipykernel_launcher.py:6: SettingW
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html

/Users/kailee/anaconda/envs/py36/lib/python3.6/site-packages/ipykernel_launcher.py:8: SettingW
```

```
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
```

```
In [5]: i = 0
        for item in data_test["Date"]:
            if item[0] == "9":
                data_test["Date"][i] = "sept"
            elif item[1] == "0":
                data_test["Date"][i] = "oct"
            else:
                data_test["Date"][i] = "nov"
            i += 1
```

```
/Users/kailee/anaconda/envs/py36/lib/python3.6/site-packages/ipykernel_launcher.py:4: SettingWi
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  after removing the cwd from sys.path.
/Users/kailee/anaconda/envs/py36/lib/python3.6/site-packages/ipykernel_launcher.py:6: SettingWi
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html

/Users/kailee/anaconda/envs/py36/lib/python3.6/site-packages/ipykernel_launcher.py:8: SettingWi
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
```

```
In [6]: def prior_prob(column, label):
            label_count = 0
            for item in column:
                if item == label:
                    label_count += 1
            return (float(label_count)/len(column))
```

```
In [7]: def likelihood(column, label, predict_column, pos_label):
            count = 0
            for index, value in enumerate(column):
                if predict_column[index] == pos_label:
                    if value == label:
                        count += 1
            pos_count = 0
            for item in predict_column:
                if item == pos_label:
```

```
                pos_count += 1
        return float(count)/pos_count
```

In [8]:
```
likelihood(list(data_train["Media"]), "NBC", list(data_train["Label"]), "Win")
```

Out[8]:
```
0.7142857142857143
```

In [9]:
```
def evidence(column, label):
    label_count = 0
    for item in column:
        if item == label:
            label_count += 1
    return (float(label_count)/len(column))
```

In [10]:
```
def post_prob(prior, likelihood,evidence):
    return (prior*float(likelihood))/evidence
```

In [11]:
```
def train(dataframe, predict_column, pos_label, column_names):
    probabilities = {}
    pos_prior = prior_prob(predict_column, pos_label)
    for name in column_names:
        column = list(dataframe[name])
        values = set()
        for value in column:
            values.add(value)
        for unique_value in values:
            likeli = likelihood(column, unique_value, predict_column, pos_label)
            evi = evidence(column, unique_value)
            po_prob = post_prob(pos_prior, likeli, evi)
            probabilities[unique_value] = po_prob
    print(probabilities)
    return probabilities
```

In [12]:
```
data_train.head()
```

Out[12]:

	ID	Date	Opponent	Is_Home_or_Away	Is_Opponent_in_AP25_Preseason	Media	Label
0	1	sept	Texas	Home	Out	NBC	Win
1	2	sept	Virginia	Away	Out	ABC	Win
2	3	sept	Georgia Tech	Home	In	NBC	Win
3	4	sept	UMass	Home	Out	NBC	Win
4	5	oct	Clemson	Away	In	ABC	Lose

```
In [13]: predicting_column = list(data_train["Label"])

In [14]: data_train.drop(["Label", "ID"], axis=1, inplace=True)

In [15]: test_predict_column = list(data_test["Label"])

In [16]: data_test.drop(["Label", "ID"], axis=1, inplace=True)

In [17]: list_names = ["Date", "Opponent", "Is_Home_or_Away", "Is_Opponent_in_AP25_Preseason",

In [18]: train_output = train(data_train, predicting_column,"Win", list_names)

{'nov': 0.5, 'oct': 0.6250000000000001, 'sept': 0.6250000000000001, 'Syracuse': 1.0, 'Virginia


In [19]: def test(dataframe, column_names, train_output):
             predicted = []
             for index, row in dataframe.iterrows():
                 row_prob = 1
                 for name in column_names:
                     value = row[name]
                     try:
                         prob = train_output[value]
                     except:
                         prob = 0
                     row_prob = row_prob * prob
                 if row_prob >= .5:
                     predicted.append("Win")
                 else:
                     predicted.append("Lose")
             print(predicted)
             return predicted


In [20]: predicted = test(data_test, list_names, train_output)

['Lose', 'Lose', 'Lose', 'Lose', 'Lose', 'Lose', 'Lose', 'Lose', 'Lose', 'Lose', 'Lose', 'Lose


In [21]: precision_score(test_predict_column,predicted, pos_label="Lose")

Out[21]: 0.25

In [22]: recall_score(test_predict_column,predicted, pos_label="Lose")

Out[22]: 1.0

In [23]: f1_score(test_predict_column,predicted, pos_label="Lose")

Out[23]: 0.4
```

kmadden5-hw2-4

September 28, 2018

In [58]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
from pandas import Series, DataFrame
import scipy
from scipy import stats
from sklearn import tree
```

In [59]:
```
D = pd.read_csv("Dataset-film-data copy.csv")
```

In [60]:
```
Y = D["GENRE"].values
```

In [61]:
```
D.head()
```

Out[61]:

	FILM_ID	AVGRATING_WEBSITE_1	AVGRATING_WEBSITE_2	AVGRATING_WEBSITE_3	AVGRATING_WEBSITE_4	GENRE
0	f1	5.1	3.5	1.4	0.2	ACTION
1	f2	4.9	3.0	1.4	0.2	ACTION
2	f3	4.7	3.2	1.3	0.2	ACTION
3	f4	4.6	3.1	1.5	0.2	ACTION
4	f5	5.0	3.6	1.4	0.3	ACTION

In [62]:
```
D.drop(["GENRE","FILM_ID"],axis=1,inplace=True)
```

In [63]:
```
clf = tree.DecisionTreeClassifier()
dtree = clf.fit(D.values, Y)
```

In [64]:
```
import graphviz
dot_data = tree.export_graphviz(clf, out_file=None)
graph = graphviz.Source(dot_data)
graph.render("Films")
```

Out[64]:
```
'Films.pdf'
```