In play tennis price modeling

Part 2. The implementaion of the prediction algorithm.

I was forced to slightly change the algorithm to predict future prices after studying the data received from Betfair. We don't have information about who is now serving player. Thus, we will use only probability of the player A to win a point.

We will use a small class to describe any node in the tennis match graph.

```
class TreeNode:
    def __init__(self, data,parent = None,left=None, right=None):
        self.data = data      #contains current score
        self.left = left        #points to the next node after win
        self.right = right    #points to the next node after lose
        self.parent = parent    #points to the previous node
```

First of all we should create a function to calculate the probability of the player A to win the current game from any point. For simplicity, we will add 15 for the first and second points and 10 for other points. We will have scores like: 15:0, 30:15, 40:30, 50:40, 50:50, 60:50 ... The limit in 130 allows to avoid an infinite loop. This is a simple recursion function for calculation a probability to win a game from any point. We use the global variable probsumg to collect the total probability.

```
def prob_win_game(stnode,prob_point):
    #Calculate probability to win one game for player A from the current point
    global probsumg
    point_A = stnode.data[6]
    point_B = stnode.data[7]
    server = stnode.data[2]
    if point_B > 130 or  point_A > 130 : return
    if point_B > 40 and point_B - point_A >= 20:
```

```
        return
    if point_A > 40 and point_A - point_B >= 20:
        prob_multip = stnode.data[3]
        cnode = stnode
        while 1:
            pn = cnode.parent
            if pn == None: break
            prob_multip *= pn.data[3]
            cnode = pn
        probsumg += prob_multip
        return
    #player A win
    if point_A < 30:
        dlt = 15
    else:
        dlt = 10
     win_node = TreeNode([0,0, "A",prob_point,0,0,point_A+dlt,point_B],stnode)
    stnode.left = win_node
    prob_win_game(win_node,prob_serv,prob_return)
    #player B win
    if point_B < 30:
        dlt = 15
    else:
        dlt = 10
    lose_node = TreeNode([0,0, "A",1-prob_point,0,0,point_A,point_B+dlt],stnode)
    stnode.right = lose_node
    prob_win_game(lose_node,prob_serv,prob_return)
```

At the next step we will create a function for calculation probability to win a tai break if we know a probability to win a point. This is a recursion function with limit in 14 points for tai break score to avoid an endless loop. I use a wrapper for the recursion function in this case.

```
def wrap_prob_win_taibreak(stnode1,prob_point):
    probsumt = [0]
```

```
    if stnode1.data[6] > 7 :
        d = stnode1.data[6] - 7
        stnode1.data[6] = 7
        stnode1.data[7] -= d

    def prob_win_taibreak(stnode,prob_point):
        #Calculate probability to win in one game for player A from the current point
        point_A = stnode.data[6]
        point_B = stnode.data[7]
        server = stnode.data[2]
        if point_B > 14 or  point_A > 14 :
            return
        if (point_B > 6 and point_B - point_A > 1):
            return
        if (point_A > 6 and point_A - point_B > 1) :
            prob_multip = stnode.data[3]
            cnode = stnode
            while 1:
                pn = cnode.parent
                if pn == None: break
                prob_multip *= pn.data[3]
                cnode = pn
            probsumt[0] += prob_multip
            return
        #player A win
        dlt = 1
        win_node = TreeNode([0,0, "A",prob_point,0,0,point_A+dlt,point_B],stnode)
        stnode.left = win_node
        prob_win_taibreak(win_node,prob_point)
        #player B win
        lose_node = TreeNode([0,0, "A",1-prob_point,0,0,point_A,point_B+dlt],stnode)
        stnode.right = lose_node
        prob_win_taibreak(lose_node,prob_point)
    prob_win_taibreak(stnode1,prob_point)
    return probsumt[0]
```

Now we can define a probability to win any set if we know the probability to win a game and tai break. This recursion function collects all probabilities at the global variable probsum.

```
def prob_win_set(stnode,probg,prob_tai):
    #Calculate probability to win a set for player A from the current point
    #using probability to win one game - probg
    global probsum,recn
    game_A = stnode.data[4]   #game score for player A
    game_B = stnode.data[5]
    probg_curl = 1 - probg
```

```
    prob_tail = 1-prob_tai #5
    def back_prop(stnode):
        prob_multip = stnode.data[3]
        cnode = stnode
        while 1:
            pn = cnode.parent
            if pn == None: break
            prob_multip *= pn.data[3]
            cnode = pn
        return prob_multip
    if (game_A in [6,7] and game_A - game_B >= 2):
        probsum += back_prop(stnode)
         return
    if (game_B in [6,7] and game_B - game_A >= 2):
        return
    if game_A <= 6 and game_B <= 6 and game_A + game_B < 12:
        #Win the next game
        win_node = TreeNode([0,0,"B",probg,game_A+1,game_B],stnode)
        stnode.left = win_node
        prob_win_set(win_node,probg,prob_tai)
        #lose the next game
        lose_node = TreeNode([0,0,"B",probg_curl,game_A,game_B+1],stnode)
        stnode.right = lose_node
        prob_win_set(lose_node,probg,prob_tai)
    elif  game_A >= 6 and game_B >= 6:
        #Tai break
        win_node = TreeNode([0,0,"B",prob_tai,game_A+1,game_B],stnode)
        stnode.left = win_node
        probsum += back_prop(win_node)
         return
```

After that we can create a function for calculation probability of the player A to win a match if we know probability to win game and tai break.

```
def calc_prob(scA,scB,scsA,scsB,probg,probtai,typeMatch=3):
    #Find probability to win match if we know probability to win game and tai break
    #scA,scB  - score in the match
    # The score for current set -scsA,scsB, probg - probability to win game
    # score for previous set =pscsA,pscsB  ,pscsA=0,pscsB=0
    rset = typeMatch - (scA + scB)
    if scA > scB + rset: return 1     #player A won match
    if scB > scA + rset: return 0     #player B won match
    global probsum
    pointA = 0
    pointB = 0
    server = "A"
    stnode = TreeNode([scA,scB,server,1,scsA,scsB,pointA,pointB])
```

```
    #Calculate the probability to win the current set
     probsum = 0
     prob_win_set(stnode,probg,probtai)
    prob_set_cur = probsum
    #Calculate the probability to win the next set
    stnode.data[4] = 0
    stnode.data[5] = 0
    probsum = 0
    prob_win_set(stnode,probg,probtai)
    prob_set_v = probsum
    probsum = 0
    prob_sets(stnode,prob_set_cur,prob_set_v,3)
    prob_match = probsum
    return prob_match
```

Finally we can create a function for probability of the player A to win a match if we know probability to win a point.

```
def calc_prob_match_from_point2(scA,scB,scsA,scsB,pointA,pointB,prob_point):
    #Find probability to win match if we know probability to win a point
    #scA,scB  - score in the match
    # The score for current set -scsA,scsB, probg - probability to win game
    global probsumg
    server = "A"
    stnode = TreeNode([scA,scB,server,1,scsA,scsB,pointA,pointB])
    prob_p = [i*0.01 for i in range(101)]
    x = np.array(prob_p)
    #Calculate probability to win a full game
    y = np.array(points_game[(0,0)])
    f_game = interpolate.interp1d(x, y)
    prob_game = float(f_game(prob_point))
    #Probability to win full tai break
    y = np.array(points_tai[(0,0)])
    f_tai = interpolate.interp1d(x, y)
    prob_tai = float(f_tai(prob_point))
    scsAw = scsAl = scsA
    scsBw = scsBl = scsB
    scAw = scAl = scA
    scBw = scBl = scB
    if scsA == 6 and scsB == 6:
        #Probability to win the current tai break
        y = np.array(points_tai[(pointA,pointB)])
        f_tai = interpolate.interp1d(x, y)
        prob_curr_game = float(f_tai(prob_point))
        scAw = scA + 1
        scsAw = 0
        scsBw = 0
```

```
            scBl = scB + 1
            scsAl = 0
            scsBl = 0
        else:
            #Calculate probability to win the current game
            y = np.array(points_game[(pointA,pointB)])
            f_c_game = interpolate.interp1d(x, y)
            prob_curr_game = float(f_c_game(prob_point))
            #if player A won the current game
            scsAw = scsA + 1
            if scsAw >= 6 and scsAw - scsB > 1:
                scAw = scA + 1
                scsAw = 0
                scsBw = 0
            #if player B won the current game
            scsBl= scsB + 1
            if scsBl >= 6 and scsBl - scsA > 1:
                scBl = scB + 1
                scsAl = 0
                scsBl = 0
        #Calculate the probability to win the match if player A win the current game
        prob_match_w = calc_prob(scAw,scBw,scsAw,scsBw,prob_game,prob_tai)
        #Calculate the probability to win the match if player A lost the current game
        prob_match_l = calc_prob(scAl,scBl,scsAl,scsBl,prob_game,prob_tai)
        prob_match = prob_curr_game * prob_match_w + (1- prob_curr_game) * prob_match_l
        return prob_match
```

So we can calculate probabilities to win a match from any moment for any probability to win a point. But we need a reverse function. Really we know only probability to win match as 1/Price and we should define the probability to win a point. To do this we create a table with all possible scores in the match and range of probabilities to win a point 0,0.01,0.02....0.98,0.99,1.0 We will be able to select from the file prob_point_match.pkl an appropriate probability of the player A to win a point if we know the current score and current price. After that we will use the function prediction_point for calculation of the next two prices if the player A to win and to lose.

```
def prediction_point(scA,scB,scsA,scsB,pointA,pointB,price):
    #Predict the next prices for win and lose a point
    #scA,scB - the match score
    #scsA,scsB - the score in the current set
    #pointA,pointB - point score
    #price - the current price
    probr =  round(1.0/price,2)
    dct = joblib.load("prob_point_match.pkl")
```

```
if pointA == "Av" : pointA = 50
if pointB == "Av" : pointB = 50
if scsA == 6 and scsB == 6:
    if pointA > 7 :
        d = pointA - 7
        pointA = 7
        pointB -= d
prob_match = dct[(scA,scB,scsA,scsB,pointA,pointB)]
prob_game =[0] + [i*0.01 for i in range(30,71)] + [1.0]
x = np.array(prob_match)
y = np.array(prob_game)
f_match_game = interpolate.interp1d(x, y)
probp = float( f_match_game(1.0/price))
scAw,scBw,scsAw,scsBw,pointAw,pointBw = win_score(scA,scB,scsA,scsB,pointA,pointB)
scAl,scBl,scsAl,scsBl,pointAl,pointBl = lose_score(scA,scB,scsA,scsB,pointA,pointB)
probmw = calc_prob_match_from_point2(scAw,scBw,scsAw,scsBw,pointAw,pointBw,probp)
if probmw == 0: probmw = 0.01
pricew = round(1/probmw,2)
probml = calc_prob_match_from_point2(scAl,scBl,scsAl,scsBl,pointAl,pointBl,probp)
if probml == 0: probml = 0.01
pricel = round(1/probml,2)
return pricew,pricel
```