Dunya Kupasini 2014 Tahminleri

Projede kullanilan 4 Python dosyasi var:

`match_stats`: Mac istatistiklerini yukleyen kodlar.

`features`: Ham istatistik verileri ozelliklere (features) donduruyor, ki bu ozellikler yapay ogrenim modeline girilebilsin. Bu ozellikler onceki K macin verilerini ozetleme amacli yaratildilar, ki bu ozelliklere dayanarak bir sonraki maci tahmnin edebilelim.

`world_cup` Veriyi temizlemek ve modeli kurmak icin kullanilan yardimci kodlar.

`power`: Birbiriyle belli sayida mac yapmis takimlarin bir "guc siralamasini" hesaplamak.

Ozellik insasi

Sonraki mac tahmini icin onceki N macin ozet istatistiklerine bakiyoruz, N'in ne oldugu `history_size` ile tanimli.

```
import world_cup
import features
import match_stats
import pandas as pd
history_size = 3

game_summaries = features.get_game_summaries()
data = features.get_features(history_size)
```

Bu ozellikler, dedigimiz gibi, onceki K macin ozeti. Bu ozetlerin cogu ortalama, ve ortalamalar dakika bazli olarak alinmis. Ozetler dakika bazli cunku mac zamanini asan maclari da hesaba katmak icin boyle yaptik. Mesela mac basina yapilan pas degerini alsaydik, zamani asan bir macta normalden cok daha fazla olacakti, bu modeli bozardi.

Modelde kullanilacak ozellikler sunlardir:

`is_home`: Takim evinde mi, deplasmanda mi oynuyor. Futbolda bu degiskeninin cok onemli oldugu goruluyor.

`avg_points`: Onceki K macta kazanilan ortalama puan (galibiyet icin 3, esitlik icin 1, kayip icin 0.

`avg_goals`: Onceki K macta atilan averaj gol.

`op_average_goals`: Rakip tarafindan son K macta atilan averaj gol.

`pass_70/80`: Hucum sahasinin 30%-20%'sinde dakika basina verilen basarili pas.

`op_pass70/80`: Hucum sahasinin 30%-20%'sinde rakip tarafindan verilmis dakika bazinda basarili paslar.

`expected_goals`: Son K mactaki gol beklentisi, ki bu beklenti atilan sut ve ve sutun kaleden uzakligi baz alinarak hesaplanan bir sayi.

passes: Dakika basina atilan paslar.

bad_passes: Dakika bazinda verilen ama basarili olmayan paslar.

pass_ratio: Basarili paslarin orani.

corners: Dakika bazinda atilan kornerler.

fouls: Yapilan faul sayisi (dk bazli)

cards: Kirmizi ya da sari alinan kart ceza sayisi (mac basina).

shots: Dakika bazinda atilan sut.

op_*: Rakipler hakkindaki bazi tarihi istatistikler. Dikkat, bu "rakip" op_team_name'de gosterilen rakip degil, genel olarak bu takimin rakiplerinin ona karsi nasil oynadigini gostermeye calisan bir istatistik. Mesela op_corners bu takimin rakiplerinin dakika basina kac korner kazandigini gosteriyor.

*_op_ratio: Takimin istatistiklerinin rakiplerine olan orani [?]

Ozellik olmayan kolonlar

matchid: Macin id'si

teamid: Takimin id'si

op_teamid: Rakip takimin tekil id'si

team_name: Takimin ismi

op_team_name: Rakip takimin ismi

timestamp: Mac ne zaman oynandi

competitionid: Genel musabakayi gosteren kod (dunya kupasi, vs).

Hedef kolonlar:

Alttaki kolonlar tahmin edilmeye ugrasilabilecek olan kolonlar. Eger bilinen veri uzerinde tahmin yapmak istiyorsak, bu kolonlari tahmin oncesi disari atmaliyiz, bunu unutmayalim. Birkac hedef kolon var ama, biz sadece kazanilan puani tahmin etmeye ugrasacagiz, belki diger modeller diger kolonlari tahmin etmeye ugrasirlar, mesela atilan gol sayisi gibi.

points: Macin puan sonucu.

goals: teamid'deki takimin attigi gol sayisi.

op_goals: op_teamid ile gosterilen takimin attigi gol sayisi.

```
club_data = data[data['competitionid'] <> 4]
# Show the features latest game in competition id 4, which is the world cup.
print data[data['competitionid'] == 4].iloc[0]
```

```
matchid                                    731828
teamid                                        366
op_teamid                                     632
competitionid                                   4
seasonid                                     2013
is_home                                         0
team_name                             Netherlands
op_team_name                            Argentina
timestamp              2014-07-09 21:00:00.000000
goals                                           0
op_goals                                        0
points                                          1
avg_points                               2.333333
avg_goals                                1.333333
op_avg_goals                            0.3333333
pass_70                                 0.4720355
pass_80                                 0.1506976
op_pass_70                              0.2647796
op_pass_80                             0.07850102
expected_goals                           1.444374
op_expected_goals                       0.4114247
passes                                   3.834864
bad_passes                               1.013622
pass_ratio                              0.7655947
corners                                0.07099121
fouls                                   0.1262374
cards                                           1
shots                                   0.1552259
op_passes                                 3.38986
op_bad_passes                            1.024551
op_corners                             0.03467955
op_fouls                                0.1570661
op_cards                                 2.666667
op_shots                               0.09249659
goals_op_ratio                           1.333333
shots_op_ratio                           1.702273
pass_op_ratio                            1.025426
Name: 0, dtype: object
```

Atilan goller ve sonucu eksenlere alarak bir tablo yaratalim (crosstab).

```
import pandas as pd
print pd.crosstab(
    club_data['goals'],
    club_data.replace(
        {'points': {
            0: 'lose', 1: 'tie', 3: 'win'}})['points'])
```

```
points  lose  tie  win
goals
0        768  279    0
1        508  416  334
2        134  218  531
3         23   42  325
4          2    6  158
5          0    2   67
```

```
6            0     0    13
7            0     0     6
8            0     0     1
```

5'den fazla gol atmak tabii ki kazanmayi garantiliyor, hic atmamak 75% ihtimalle kaybedilecek demektir (bazen de beraberlik olur tabii!).

Not: Fakat tabloda 4 gol sonrasi kazanimlar direk artmiyor, niye? Cunku bu maclar uzatma sonrasi atilan penaltilardan geliyor, her iki takimda bu sirada cok gol atiyor, ama biri mutlaka kaybediyor [1].

Modeli egitmek

Veri tabanimizdaki klup verisini kullanarak (yani hic dunya kupasi verisi kullanmadan) egitecegiz. Bu kod `world_cup.py` icinde. Sonuc bir lojistik regresyon modeli olacak, ve sonra test verisi uzerinde tahmin yapacagiz. Regresyonunun Rsquared degerini gosterecegiz, ki bu egitim verisi uzerinden gosterilebilir. Rsquared modelin veriye ne kadar uydugunu gosteren bir rakamdir (ne kadar yuksekse o kadar iyi).

```
import world_cup
reload(world_cup)
import match_stats
pd.set_option('display.max_rows', 5000)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

# Don't train on games that ended in a draw, since they have less signal.
train = club_data.loc[club_data['points'] <> 1]
# train = club_data

(model, test) = world_cup.train_model(
     train, match_stats.get_non_feature_columns())
print "Rsquared: %0.03g" % model.prsquared
```

```
Rsquared: 0.149
```

Onemli ozellikleri secmek

Lojistik regresyon modelimiz regularizasyon kullaniyor; bu demektir ki daha cetrefil modeller cezalandiriliyor. Bu cezalandirmanin yan etkisi olarak biz hangi ozelliklerin daha onemli oldugunu gorebiliyoruz, cunku daha onemsiz olan ozellikler modelden atiliyorlar (katsayilari sifira iniyor).

Bu baglamda ozellikleri uce ayirabiliriz:

Pozitif ozellikler: Bu ozellikler mevcut ise takimin kazanma sansi yukseliyor.

Negative features: Tam tersi

Dropped features: Onemli olmayan ozellikler, ki bu ozellikler modele dahil edilirse asiri uygunluk (overfitting) durumu ortaya cikar.

```
def print_params(model, limit=None):
    params = model.params.copy()
```

```python
    params.sort(ascending=False)
    del params['intercept']

    if not limit:
        limit = len(params)

    print("Pozitif ozellikler")
    params.sort(ascending=False)
    print np.exp(params[[param > 0.001 for param in params]]).sub(1)[:limit]

    print("\nAtilan ozellikler")
    print params[[param  == 0.0 for param in params]][:limit]

    print("\nNegatif ozellikler")
    params.sort(ascending=True)
    print np.exp(params[[param < -0.001 for param in params]]).sub(1)[:limit]

print_params(model, 10)
```

```
Pozitif ozellikler
is_home              0.848337
pass_70              0.254729
expected_goals       0.169235
opp_op_corners       0.159163
op_passes            0.120319
opp_op_pass_80       0.095970
avg_goals            0.092000
opp_bad_passes       0.075657
opp_cards            0.068903
fouls                0.062809
dtype: float64

Atilan ozellikler
op_pass_70               0
opp_op_cards             0
op_bad_passes            0
opp_op_bad_passes        0
opp_op_fouls             0
corners                  0
pass_ratio               0
opp_corners              0
op_fouls                 0
opp_goals_op_ratio       0
dtype: float64

Negatif ozellikler
opp_pass_70             -0.203015
opp_expected_goals      -0.144740
op_corners              -0.137309
opp_op_passes           -0.107397
op_pass_80              -0.087566
opp_avg_goals           -0.084249
bad_passes              -0.070335
cards                   -0.064461
opp_fouls               -0.059097
opp_passes              -0.049240
```

```
dtype: float64
```

Klup verisi uzerinde tahmin

`predicted`: Takimin kazanma sansi (tahmin).

`points`: Gercekten ne oldu.

```
reload(world_cup)
results = world_cup.predict_model(model, test, match_stats.get_non_feature_columns())

predictions = world_cup.extract_predictions(results.copy(), results['predicted'])

print 'Dogru tahminler:'
print predictions[(predictions['predicted'] > 50) & (predictions['points'] == 3)][:5]
```

```
Dogru tahminler:
              team_name          op_team_name  predicted           expected    
8      Portland Timbers        Real Salt Lake  52.418756    Portland Timbers    Portland
42       Rayo Vallecano            Granada CF  60.862465      Rayo Vallecano      Rayo V
49  AtlÃ©tico de Madrid                 Getafe  64.383541  AtlÃ©tico de Madrid  AtlÃ©t
57      Colorado Rapids  Vancouver Whitecaps  51.836366     Colorado Rapids     Colorad
58          Real Madrid         Real Sociedad  64.100904         Real Madrid         Rea
```

```
print 'Yanlis tahminler:'
print predictions[(predictions['predicted'] > 50) & (predictions['points'] < 3)][:5]
```

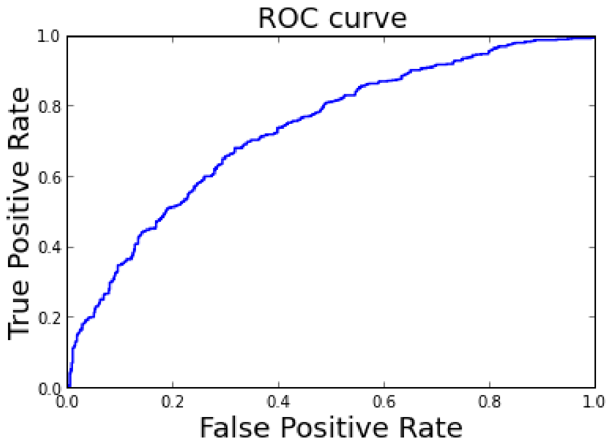
```
Yanlis tahminler:
                  team_name          op_team_name  predicted                 expected
1        Seattle Sounders FC  Vancouver Whitecaps  51.544963      Seattle Sounders FC  Va
2   New England Revolution        Real Salt Lake  63.950714  New England Revolution
3          Philadelphia Union             FC Dallas  54.213693      Philadelphia Union
14  New England Revolution      Montreal Impact  52.762065  New England Revolution
20         New York Red Bulls            Toronto FC  55.533969      New York Red Bulls
```

Tahminlerimizi kontrol etmek

Kontrol icin mesela hesabimizin rasgele tahminden ne kadar iyi oldugunu hesaplayabiliriz (lift) ya da AUC hesabi yapip ROC egrisini hesaplariz. AUC herhalde en iyisi, bu hesap cok ilginctir, 0.5 (rasgele sans) ve 1.0 arasindadir (mukemmel tahmin), ve bu hesap dengesiz veri setlerine karsi dayaniklidir. Mesela 0/1 etiketi tahmininde test setinde mesela yuzde 90 oraninda 1 olsa ve modelimiz surekli 1 tahmin etse, basit bir olcum bize modelimizin yuzde 90 basarili oldugunu soylerdi. AUC boyle durumlara karsi dayaniklidir, bize 0.5 sonucunu verir.

```
baseline = (sum([yval == 3 for yval in club_data['points']])
            * 1.0 / len(club_data))
y = [yval == 3 for yval in test['points']]
world_cup.validate(3, y, results['predicted'], baseline,
                   compute_auc=True)
plt.savefig('doc_en_01.png')
```

```
(3) Lift: 1.42 Auc: 0.738
```



Modelden eksik olan bir sey var; sonraki maci onceki birkac macin ozetinden tahmin etmeye ugrasiyoruz ama belki bazi takimlar onceki K macta cok zorlu rakiplerle ugrasmistir, bazilari cok kolay rakiplerle ugrasmistir. Bu durumda onceki maclarin istatistigi bize tum hikayeyi anlatmayacaktir.

Bu problemi cozmek icin ayri bir regresyon daha isletebiliriz. Bu regresyon bir guc siralamasi (power ranking) hesaplayabilir, bu hesap FIFA/CocaCola'nin enternasyonel takimlar icin yaptigi guc siralama hesabina benzer. ABD'de beyzbol ve Amerikan futbolu icin de benzer bir hesap yapiliyor.

Bu hesabi yaptiktan sonra -tek bir numerik sayi olacaktir, bazi takimlar icin daha yuksek bazi takimlar icin daha alcak olacaktir, ki bu rakam uzerinden siralama yapilabilsin-, onu bir ozellik olarak lojistik regresyon modeline dahil edebiliriz. Guc siralamasi esas olarak su tur irdelelerin modelimize dahilini mumkun kilar; eger A takimi B'yi yendiyse, B C'yi yendiyse, A buyuk ihtimalle C takimindan daha iyidir. Bu niye iyi? Cunku elimizde yapilabilecek tum maclarin kombinasyonu yok, mac verisi seyrek (sparse). Ama eldeki birkac mactan bir guc siralamasi hesaplayabiliyoruz iste. Bu bir avantaj.

Siralama hesabi yapildiktan sonra hizli bazi kontrolleri ciplak gozle yapabiliriz, mesela sonuca bakariz, eger Wiggan (zayif bir takim) 1.0 degeri almis, Chelsea (guclu bir takim) 0.0 degeri almis ise bir seyler yanlis demektir.

Tabii buna ragmen bazi takimlara hala uygun siralama veremeyebiliriz, mesela A,B'yi, B,C'yi yeniyor, sonra C A'yi yeniyor. Bu sekilde siralayamadigimiz durumda takimi 0.5 ile tam ortaya koyacagiz.

Ayrica enternasyonel takimlarin siralamasi cok gurultulu bir veri oldugu ve seyrek oldugu icin onu yuzdeliklere (quartiles) ayirarak gosterecegiz, yani siralamalar 0, .33, .66, or 1.0 olarak gozukecekler.

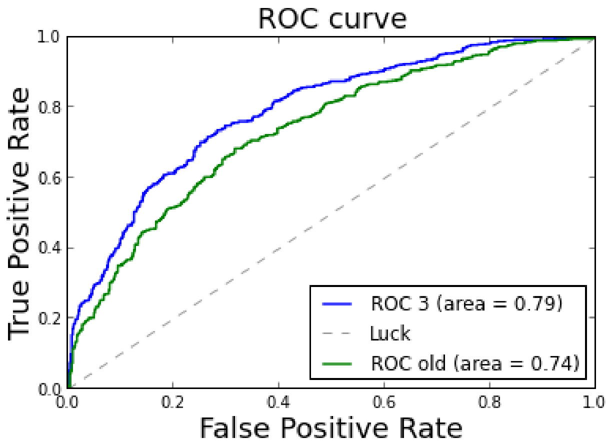
Siralamayi nihai modele dahil edince basari oranimizin onemli derecede arttigini gorecegiz.

```
import power
reload(power)
```

```
reload(world_cup)
def points_to_sgn(p):
  if p > 0.1: return 1.0
  elif p < -0.1: return -1.0
  else: return 0.0
power_cols = [
  ('points', points_to_sgn, 'points'),
]

power_data = power.add_power(club_data, game_summaries, power_cols)
power_train = power_data.loc[power_data['points'] <> 1]

# power_train = power_data
(power_model, power_test) = world_cup.train_model(
    power_train, match_stats.get_non_feature_columns())
print "\nRsquared: %0.03g, Power Coef %0.03g" % (
    power_model.prsquared,
    math.exp(power_model.params['power_points']))

power_results = world_cup.predict_model(power_model, power_test,
    match_stats.get_non_feature_columns())
power_y = [yval == 3 for yval in power_test['points']]
world_cup.validate(3, power_y, power_results['predicted'], baseline,
                   compute_auc=True, quiet=False)

print_params(power_model, 8)

plt.plot([0, 1], [0, 1], '--', color=(0.6, 0.6, 0.6), label='Luck')
# Add the old model to the graph
world_cup.validate('old', y, results['predicted'], baseline,
                   compute_auc=True, quiet=True)
plt.legend(loc="lower right")
plt.savefig('doc_en_02.png')
```

```
New season 2014
New season 2013
New season 2013
New season 2012
New season 2012
New season 2011
['Blackburn Rovers: 0.000', 'Real Betis: 0.000', 'D.C. United: 0.000', 'Celta de Vigo

Rsquared: 0.22, Power Coef 2.18
(3) Lift: 1.56 Auc: 0.791
    Base: 0.374 Acc: 0.708 P(1|t): 0.778 P(0|f): 0.667
    Fp/Fn/Tp/Tn p/n/c: 99/248/347/496 595/595/1190
Pozitif ozellikler
power_points      1.177169
is_home           0.787110
opp_op_corners    0.170848
expected_goals    0.058597
opp_cards         0.045538
pass_70           0.036267
avg_goals         0.035456
opp_avg_points    0.033857
dtype: float64
```

```
Atilan ozellikler
passes                  0
op_pass_80              0
op_expected_goals       0
opp_shots_op_ratio      0
bad_passes              0
pass_ratio              0
opp_pass_op_ratio       0
shots                   0
dtype: float64

Negatif ozellikler
opp_power_points      -0.540688
op_corners            -0.145918
opp_expected_goals    -0.055353
cards                 -0.043555
opp_pass_70           -0.034997
opp_avg_goals         -0.034242
avg_points            -0.032748
opp_fouls             -0.022867
dtype: float64
(old) Lift: 1.42 Auc: 0.738
```



Simdi dunya kupasini tahmin edelim!

Aynen klup verisinde yaptigimiz gibi dunya kupasi icin de benzer istatistikleri hesaplayabiliriz. Bu durumda elimizde hedefler olmayacak, yani kimin kazandigini bilemeyecegiz (aslinda bazi dunya kupasi maclarinin sonucunu biliyoruz, ama tahminlerimizi hicbir maci bilmiyormus gibi yapalim). Ve tekrar vurgulayalim: klup verisiyle egittigimiz modeli kullanarak dunya kupasini tahmin edecegiz. Yani model ve tahmin tamamen farkli takimlar uzerinden yapilacak!

`features.get_wc_features()` bize tum dunya kupasi maclari icin gereken ozellikleri yaratip dondurecektir.

```
import world_cup
import features
reload(match_stats)
reload(features)
```

```
reload(world_cup)
wc_data = world_cup.prepare_data(features.get_wc_features(history_size))
wc_labeled = world_cup.prepare_data(features.get_features(history_size))
wc_labeled = wc_labeled[wc_labeled['competitionid'] == 4]
wc_power_train = game_summaries[game_summaries['competitionid'] == 4].copy()
```

Ev sahasi avantaji

Klup verisi ile dunya kupasi verisi arasindaki bazi farklardan biri bu, dunya kupasi icin ev sahibi olmak ne demektir? Tek ev sahibi resmi olarak 2014 kupasina ev sahipligi yapan Brezilya midir? Belki diger Latin Amerika takimlarini da ev sahibi olarak gorebiliriz? Bazi modeller 'is home"u sadece Brezilya'ya vermis, ayni kitadaki diger takimlara da 'azicik' ev sahipligi vermis, cunku istatistiklere gore bu takimlar kendi kitalarinda daha iyi performans gostermisler, vs.

Biz daha degisik bir model kullanacagiz, ve bu model belki biraz subjektif.. Biz `is_home` ogesine 0.0 ila 1.0 arasinda bir deger atayacagiz, ki bu degerin buyuklugu o takimin taraftarlarinin hem sayi, hem de destek enerjisi uzerinden olculecek. Bunu yapmamizin sebebi ilk turlarda goruldugu uzere, taraftarinin daha iyi destekledigi takimlarin digerlerine gore daha iyi performans gostermesi. Mesela Sili'nin taraftari takimini muthis destekledi, Ispanya taraftari orali bile olmadi, Sili Ispanya macini Sili 2-0 kazandi. Bunun gibi pek cok mac gozlemledik, cogunda guney Amerika takimlari vardi, ama cok taraftar gonderen takimlar da vardi, mesela Meksika. Ya da ABD vardi, cok taraftari vardi ama sessizdiler, onlar daha dusuk skorlar aldilar.

```
import pandas as pd
wc_home = pd.read_csv('wc_home.csv')

def add_home_override(df, home_map):
  for ii in xrange(len(df)):
    team = df.iloc[ii]['teamid']
    if team in home_map:
        df['is_home'].iloc[ii] = home_map[team]
    else:
        # If we don't know, assume not at home.
        df['is_home'].iloc[ii] = 0.0

home_override = {}
for ii in xrange(len(wc_home)):
    row = wc_home.iloc[ii]
    home_override[row['teamid']] = row['is_home']

# Add home team overrides.
add_home_override(wc_data, home_override)
```

Dunya Kupasi Guc Siralamasi

Bu hesabin dunya kupasi verisi uzerinde yapilmasi lazim, cunku guc siralamasi o takimlarin arasindaki maclara dayanilarak yapilan bir hesap. Bu maclar ise, dunya kupasi takimlari baglaminda, oldukca seyrek cunku bazi takimlar bazi takimlarla neredeyse onyildir oynamamis. Cogu AVvrupa takimi mesela guney Amerika takimiyla oynamamis, Asyali takimlarla daha bile az oynamis. Klup bazinda kullandigimiz ayni nuamrayi burada da kullanabiliriz, ama basarisizliga hazir olmak lazim!

Hesap altta

```
# When training power data, since the games span multiple competitions,
# just set is_home to 0.5
#
# Otherwise when we looked at games from the 2010 world cup, we'd think
# Brazil was still at home instead of South Africa.

wc_power_train['is_home'] = 0.5
wc_power_data = power.add_power(wc_data, wc_power_train, power_cols)

wc_results = world_cup.predict_model(power_model, wc_power_data,
    match_stats.get_non_feature_columns())
```

```
New season 2013
New season 2009
New season 6
['Australia: 0.000', 'Serbia: 0.016', 'USA: 0.017', 'Cameroon: 0.035',
'Iran: 0.081', 'Croatia: 0.180', 'Nigeria: 0.204', "C\xc3\xb4te d'Ivoire:
0.244", 'Costa Rica: 0.254', 'Algeria: 0.267', 'Paraguay: 0.277',
'Honduras: 0.279', 'Slovakia: 0.281', 'Greece: 0.284', 'Switzerland:
0.291', 'Ecuador: 0.342', 'Uruguay: 0.367', 'Sweden: 0.386', 'Japan:
0.406', 'Mexico: 0.409', 'Chile: 0.413', 'Colombia: 0.438', 'England:
0.460', 'Belgium: 0.467', 'Ukraine: 0.470', 'Portugal: 0.487', 'Ghana:
0.519', 'South Korea: 0.532', 'France: 0.648', 'Spain: 0.736', 'Argentina:
0.793', 'Italy: 0.798', 'Brazil: 0.898', 'Netherlands: 0.918', 'Germany:
1.000']
```

Tahmin

Nihayet hazirlandigimiz ana geldik. Simdi dunya kupasi maclarini tahmin edelim. 4 kolon gosterecegiz:

`predicted`: Yuzde kac ihtimalle (ismi ilk gelen) takimin kazanacagi

`points`: Gercekten ne oldugu. Oynanmayan mac `NaN`. Dikkat, penalti atislarina giden maclar esitlik olarak gosterilecek.

Ama bir dakika! Bu sonuclar daha once gosterdiginiz (Google tahminleri kastediliyor) tahminlerden degisik! Bunun sebepleri sunlar: Bazi hatalari tamir ettik, yani kod degisti. Ilk model mesela uzayan maclar yuzunden kabaran istatistiklerin durumunu hesaba almiyordu.

Ikinci sebep, model sonu belli (deterministik) degil, egitim verisi icin verinin belli bir kismini rasgele olarak seciyoruz, bu sebeple sonuclar bir hesaptan digerine degisik cikabiliyor (ki bazen sonuclar cok degisik olabiliyor).

16. turu tahmin ederken mesela onceki 3 maci, ceyrek finaller icin onceki 4, yarifinaller icin 5, ve finaller icin onceki 6 maci kullandik [biz bu dokumanda onceki 3

maci kullandik, `history_size` parametresiyle oynayarak degisik sonuclar kontrol edilebilir, hatta yazarlarin bahsettigi rasgelelik modelden tamamen cikartilabilir].

```
pd.set_option('display.max_rows', 5000)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

wc_with_points = wc_power_data.copy()
wc_with_points.index = pd.Index(
    zip(wc_with_points['matchid'], wc_with_points['teamid']))
wc_labeled.index = pd.Index(
    zip(wc_labeled['matchid'], wc_labeled['teamid']))
wc_with_points['points'] = wc_labeled['points']

wc_pred = world_cup.extract_predictions(wc_with_points,
                                        wc_results['predicted'])

# Reverse our predictions to show the most recent first.
wc_pred.reindex(index=wc_pred.index[::-1])
# Show our predictions for the games that have already happenned.
print wc_pred
```

```
        team_name   op_team_name  predicted       expected         winner  points
0       Argentina        Germany  46.070814        Germany             NA     NaN
1     Netherlands         Brazil  42.833863         Brazil             NA     NaN
2     Netherlands      Argentina  48.641542      Argentina           draw       1
3         Germany         Brazil  44.011593         Brazil        Germany       3
4      Costa Rica    Netherlands  14.442625    Netherlands           draw       1
5         Belgium      Argentina  18.596031      Argentina      Argentina       0
6        Colombia         Brazil  23.890421         Brazil         Brazil       0
7         Germany         France  75.116349        Germany        Germany       3
8             USA        Belgium  32.400646        Belgium        Belgium       0
9     Switzerland      Argentina  19.272768      Argentina      Argentina       0
10        Algeria        Germany   5.926496        Germany        Germany       0
11        Nigeria         France   8.694729         France         France       0
12         Greece     Costa Rica  40.448104     Costa Rica           draw       1
13         Mexico    Netherlands  20.402491    Netherlands    Netherlands       0
14        Uruguay       Colombia  46.480264       Colombia       Colombia       0
15          Chile         Brazil  26.574916         Brazil           draw       1
16        Germany            USA  91.980986        Germany        Germany       3
17          Ghana       Portugal  49.051707       Portugal       Portugal       0
18    Switzerland       Honduras  60.223070    Switzerland    Switzerland       3
19         France        Ecuador  84.538857         France           draw       1
20      Argentina        Nigeria  88.491450      Argentina      Argentina       3
21  CÃ´te d'Ivoire         Greece  61.074502  CÃ´te d'Ivoire         Greece       0
22        Uruguay          Italy  32.685428          Italy        Uruguay       3
23        England     Costa Rica  63.457326        England           draw       1
24         Brazil       Cameroon  94.788074         Brazil         Brazil       3
25         Mexico        Croatia  78.020214         Mexico         Mexico       3
26          Spain      Australia  90.521542          Spain          Spain       3
27          Chile    Netherlands  28.342133    Netherlands    Netherlands       0
28       Portugal            USA  65.457259       Portugal           draw       1
29        Algeria    South Korea  17.376285    South Korea        Algeria       3
30          Ghana        Germany  14.588539        Germany           draw       1
31           Iran      Argentina   5.193843      Argentina      Argentina       0
```

32	Ecuador	Honduras	53.848926	Ecuador	Ecuador	3
33	France	Switzerland	78.659381	France	France	3
34	Costa Rica	Italy	24.836756	Italy	Costa Rica	3
35	Greece	Japan	44.355013	Japan	draw	1
36	England	Uruguay	61.012694	England	Uruguay	0
37	Croatia	Cameroon	40.212875	Cameroon	Croatia	3
38	Chile	Spain	42.624474	Spain	Chile	3
39	Netherlands	Australia	93.535889	Netherlands	Netherlands	3
40	Mexico	Brazil	20.372064	Brazil	draw	1
41	USA	Ghana	39.500993	Ghana	USA	3
42	Nigeria	Iran	53.813244	Nigeria	draw	1
43	Portugal	Germany	15.337884	Germany	Germany	0
44	Honduras	France	22.953848	France	France	0
45	Ecuador	Switzerland	59.987076	Ecuador	Switzerland	0
46	Japan	CÃ´te d'Ivoire	51.528885	Japan	CÃ´te d'Ivoire	0
47	Italy	England	68.767968	Italy	Italy	3
48	Costa Rica	Uruguay	45.347946	Uruguay	Costa Rica	3
49	Australia	Chile	19.487987	Chile	Chile	0
50	Netherlands	Spain	60.493928	Netherlands	Netherlands	3
51	Cameroon	Mexico	30.018950	Mexico	Mexico	0
52	Croatia	Brazil	6.268704	Brazil	Brazil	0
53	Spain	Netherlands	35.602227	Netherlands	Spain	3
54	Germany	Uruguay	76.467450	Germany	Germany	3
55	Spain	Germany	29.438134	Germany	Spain	3
56	Netherlands	Uruguay	71.342186	Netherlands	Netherlands	3
57	Spain	Paraguay	83.007655	Spain	Spain	3
58	Germany	Argentina	42.635127	Argentina	Germany	3
59	Ghana	Uruguay	41.784682	Uruguay	draw	1
60	Brazil	Netherlands	60.821972	Brazil	Netherlands	0
61	Portugal	Spain	23.464891	Spain	Spain	0
62	Japan	Paraguay	61.278000	Japan	draw	1
63	Chile	Brazil	24.459600	Brazil	Brazil	0
64	Slovakia	Netherlands	12.082967	Netherlands	Netherlands	0
65	Mexico	Argentina	17.626748	Argentina	Argentina	0
66	England	Germany	20.763176	Germany	Germany	0
67	Ghana	USA	71.310871	Ghana	Ghana	3
68	South Korea	Uruguay	45.148588	Uruguay	Uruguay	0
69	Brazil	Portugal	81.610878	Brazil	draw	1
70	Germany	Ghana	81.621494	Germany	Germany	3
71	Serbia	Australia	38.204905	Australia	Australia	0
72	CÃ´te d'Ivoire	Brazil	10.186423	Brazil	Brazil	0
73	Australia	Ghana	23.702414	Ghana	draw	1
74	Japan	Netherlands	10.773998	Netherlands	Netherlands	0
75	Serbia	Germany	4.731113	Germany	Serbia	3
76	Mexico	France	42.801515	France	Mexico	3
77	South Korea	Argentina	15.255040	Argentina	Argentina	0
78	Switzerland	Spain	18.747704	Spain	Switzerland	3
79	Portugal	CÃ´te d'Ivoire	65.031075	Portugal	draw	1
80	Paraguay	Italy	12.288896	Italy	draw	1
81	Australia	Germany	7.395354	Germany	Germany	0
82	Ghana	Serbia	83.682899	Ghana	Ghana	3
83	USA	England	34.763699	England	draw	1
84	France	Italy	28.651132	Italy	draw	1
85	Portugal	Germany	14.833907	Germany	Germany	0
86	France	Portugal	72.141913	France	France	3

87	Italy	Germany	33.364112	Germany	Italy	3
88	France	Brazil	22.742882	Brazil	France	3
89	Portugal	England	49.550454	England	draw	1
90	Ukraine	Italy	28.378865	Italy	Italy	0
91	Argentina	Germany	46.801014	Germany	draw	1
92	France	Spain	47.126654	Spain	France	3
93	Ghana	Brazil	9.144470	Brazil	Brazil	0
94	Ukraine	Switzerland	62.637340	Ukraine	draw	1
95	Australia	Italy	8.365416	Italy	Italy	0
96	Netherlands	Portugal	70.231295	Netherlands	Portugal	0
97	Ecuador	England	34.379086	England	England	0
98	Mexico	Argentina	29.233199	Argentina	Argentina	0
99	Sweden	Germany	10.914079	Germany	Germany	0

Kodlar

```python
"""
    Predicts soccer outcomes using logistic regression.
"""

import random
import math

import numpy as np
random.seed(987654321)
np.random.seed(987654321)
import pandas as pd
import pylab as pl
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
import statsmodels.api as sm


def _drop_unbalanced_matches(data):
    """  Because we don't have data on both teams during a match, we
         want to drop any match we don't have info about both teams.
         This can happen if we have fewer than 10 previous games from
         a particular team.
    """
    keep = []
    index = 0
    data = data.dropna()
    while index < len(data) - 1:
        skipped = False
        for col in data:
            if isinstance(col, float) and math.isnan(col):
                keep.append(False)
                index += 1
                skipped = True

        if skipped:
            pass
        elif data.iloc[index]['matchid'] == data.iloc[index+1]['matchid']:
            keep.append(True)
            keep.append(True)
```

```python
            index += 2
        else:
            keep.append(False)
            index += 1
    while len(keep) < len(data):
        keep.append(False)
    results = data[keep]
    if len(results) % 2 != 0:
        raise Exception('Unexpected results')
    return results


def _swap_pairwise(col):
    """ Swap rows pairwise; i.e. swap row 0 and 1, 2 and 3, etc.  """
    col = pd.np.array(col)
    for index in xrange(0, len(col), 2):
        val = col[index]
        col[index] = col[index + 1]
        col[index+1] = val
    return col


def _splice(data):
    """ Splice both rows representing a game into a single one. """
    data = data.copy()
    opp = data.copy()
    opp_cols = ['opp_%s' % (col,) for col in opp.columns]
    opp.columns = opp_cols
    opp = opp.apply(_swap_pairwise)
    del opp['opp_is_home']

    return data.join(opp)


def split(data, test_proportion=0.4):
    """ Splits a dataframe into a training set and a test set.
        Must be careful because back-to-back rows are expeted to
        represent the same game, so they both must go in the
        test set or both in the training set.
    """

    train_vec = []
    if len(data) % 2 != 0:
        raise Exception('Unexpected data length')
    while len(train_vec) < len(data):
        rnd = random.random()
        train_vec.append(rnd > test_proportion)
        train_vec.append(rnd > test_proportion)

    test_vec = [not val for val in train_vec]
    train = data[train_vec]
    test = data[test_vec]
    if len(train) % 2 != 0:
        raise Exception('Unexpected train length')
    if len(test) % 2 != 0:
```

```
        raise Exception('Unexpected test length')
    return (train, test)


def _extract_target(data, target_col):
    """ Removes the target column from a data frame, returns the target
        col and a new data frame minus the target. """
    target = data[target_col]
    train_df = data.copy()
    del train_df[target_col]
    return target, train_df


def _check_eq(value):
    """ Returns a function that checks whether the value equals a
        particular integer.
    """
    return lambda (x): int(x) == int(value)


L1_ALPHA = 16.0
def build_model_logistic(target, data, acc=0.00000001, alpha=L1_ALPHA):
    """ Trains a logistic regresion model. target is the target.
        data is a dataframe of samples for training. The length of
        target must match the number of rows in data.
    """
    data = data.copy()
    data['intercept'] = 1.0
    logit = sm.Logit(target, data, disp=False)
    return logit.fit_regularized(maxiter=1024, alpha=alpha, acc=acc, disp=False)


def validate(label, target, predictions, baseline=0.5, compute_auc=False,
             quiet=True):
    """ Validates binary predictions, computes confusion matrix and AUC.

      Given a vector of predictions and actual values, scores how well we
      did on a prediction.

      Args:
        label: label of what we're validating
        target: vector of actual results
        predictions: predicted results. May be a probability vector,
          in which case we'll sort it and take the most confident values
          where baseline is the proportion that we want to take as True
          predictions. If a prediction is 1.0 or 0.0, however, we'll take
          it to be a true or false prediction, respectively.
        compute_auc: If true, will compute the AUC for the predictions.
          If this is true, predictions must be a probability vector.
    """

    if len(target) != len(predictions):
        raise Exception('Length mismatch %d vs %d' % (len(target),
                                                       len(predictions)))
    if baseline > 1.0:
```

```
    # Baseline number is expected count, not proportion. Get the proportion.
    baseline = baseline * 1.0 / len(target)

zipped = sorted(zip(target, predictions), key=lambda tup: -tup[1])
expect = len(target) * baseline

(true_pos, true_neg, false_pos, false_neg) = (0, 0, 0, 0)
for index in xrange(len(target)):
    (yval, prob) = zipped[index]
    if float(prob) == 0.0:
        predicted = False
    elif float(prob) == 1.0:
        predicted = True
    else:
        predicted = index < expect
    if predicted:
        if yval:
            true_pos += 1
        else:
            false_pos += 1
    else:
        if yval:
            false_neg += 1
        else:
            true_neg += 1
pos = true_pos + false_neg
neg = true_neg + false_pos
# P(1 | predicted(1)) and P(0 | predicted(f))
pred_t = true_pos + false_pos
pred_f = true_neg + false_neg
prob1_t = true_pos * 1.0 / pred_t if pred_t > 0.0 else -1.0
prob0_f = true_neg * 1.0 / pred_f if pred_f > 0.0 else -1.0

# Lift = P(1 | t) / P(1)
prob_1 = pos * 1.0 / (pos + neg)
lift = prob1_t / prob_1 if prob_1 > 0 else 0.0

accuracy = (true_pos + true_neg) * 1.0 / len(target)

if compute_auc:
    y_bool =  [True if yval else False for (yval, _) in zipped]
    x_vec = [xval for (_, xval) in zipped]
    auc_value = roc_auc_score(y_bool, x_vec)
    fpr, tpr, _ = roc_curve(y_bool, x_vec)
    pl.plot(fpr, tpr, lw=1.5,
        label='ROC %s (area = %0.2f)' % (label, auc_value))
    pl.xlabel('False Positive Rate', fontsize=18)
    pl.ylabel('True Positive Rate', fontsize=18)
    pl.title('ROC curve', fontsize=18)
    auc_value = '%0.03g' % auc_value
else:
    auc_value = 'NA'

print '(%s) Lift: %0.03g Auc: %s' % (label, lift, auc_value)
if not quiet:
```

```
        print '    Base: %0.03g Acc: %0.03g P(1|t): %0.03g P(0|f): %0.03g' % (
            baseline, accuracy, prob1_t, prob0_f)
        print '    Fp/Fn/Tp/Tn p/n/c: %d/%d/%d/%d %d/%d/%d' % (
            false_pos, false_neg, true_pos, true_neg, pos, neg, len(target))


def _coerce_types(vals):
    """ Makes sure all of the values in a list are floats. """
    return [1.0 * val for val in vals]


def _coerce(data):
    """ Coerces a dataframe to all floats, and standardizes the values. """
    return _standardize(data.apply(_coerce_types))


def _standardize_col(col):
    """ Standardizes a single column (subtracts mean and divides by std
        dev).
    """
    std = np.std(col)
    mean = np.mean(col)
    if abs(std) > 0.001:
        return col.apply(lambda val: (val - mean)/std)
    else:
        return col


def _standardize(data):
    """ Standardizes a dataframe. All fields must be numeric. """
    return data.apply(_standardize_col)


def _clone_and_drop(data, drop_cols):
    """ Returns a copy of a dataframe that doesn't have certain columns. """
    clone = data.copy()
    for col in drop_cols:
        if col in clone.columns:
            del clone[col]
    return clone


def _normalize(vec):
    """ Normalizes a list so that the total sum is 1. """
    total = float(sum(vec))
    return [val / total for val in vec]


def _games(data):
    """ Drops odd numbered rows in a column. This is used when we
        have two rows representing a game, and we only need 1. """
    return data[[idx % 2 == 0 for idx in xrange(len(data))]]


def _team_test_prob(target):
```

```
    """ We predict both team A beating team B and team B beating
        team A. Use predictions in both directions to come up with
        an overall probability.
    """
    results = []
    for idx in range(len(target)/2):
        game0 = float(target.iloc[idx*2])
        game1 = float(target.iloc[idx*2+1])
        results.append(game0/(game0+game1))
    return results


def extract_predictions(data, predictions):
    """' Joins a dataframe containing match data with one
         containing predictions, returning a dataframe with
         team names, predicted values, and if available, the
         actual outcome (in points).
    """
    probs = _team_test_prob(predictions)
    teams0 = []
    teams1 = []
    points = []
    for game in xrange(len(data)/2):
        if data['matchid'].iloc[game*2] != data['matchid'].iloc[game*2+1]:
            raise Exception('Unexpeted match id %d vs %d', (
                                   data['matchid'].iloc[game * 2],
                                   data['matchid'].iloc[game * 2 + 1]))
        team0 = data['team_name'].iloc[game * 2]
        team1 = data['op_team_name'].iloc[game * 2]
        if 'points' in data.columns:
            points.append(data['points'].iloc[game * 2])
        teams0.append(team0)
        teams1.append(team1)
    results =  pd.DataFrame(
        {'team_name': pd.Series(teams0),
         'op_team_name': pd.Series(teams1),
         'predicted': pd.Series(probs).mul(100)},
         columns = ['team_name', 'op_team_name', 'predicted'])

    expected_winner = []
    for game in xrange(len(results)):
        row = results.iloc[game]
        col = 'team_name' if row['predicted'] >= 50 else 'op_team_name'
        expected_winner.append(row[col])

    results['expected'] = pd.Series(expected_winner)

    if len(points) > 0:
        winners = []
        for game in xrange(len(results)):
            row = results.iloc[game]
            point = points[game]
            if point > 1.1:
                winners.append(row['team_name'])
            elif point < 0.9:
```

```
                winners.append(row['op_team_name'])
            elif point > -0.1:
                winners.append('draw')
            else:
                winners.append('NA')
        results['winner'] = pd.Series(winners)
        results['points'] = pd.Series(points)
    return results


def _check_data(data):
    """ Walks a dataframe and make sure that all is well. """
    i = 0
    if len(data) % 2 != 0:
        raise Exception('Unexpeted length')
    matches = data['matchid']
    teams = data['teamid']
    op_teams = data['op_teamid']
    while i < len(data) - 1:
        if matches.iloc[i] != matches.iloc[i + 1]:
            raise Exception('Match mismatch: %s vs %s ' % (
                                matches.iloc[i], matches.iloc[i + 1]))
        if teams.iloc[i] != op_teams.iloc[i + 1]:
            raise Exception('Team mismatch: match %s team %s vs %s' % (
                                matches.iloc[i], teams.iloc[i],
                                op_teams.iloc[i + 1]))
        if teams.iloc[i + 1] != op_teams.iloc[i]:
            raise Exception('Team mismatch: match %s team %s vs %s' % (
                                matches.iloc[i], teams.iloc[i + 1],
                                op_teams.iloc[i]))
        i += 2


def prepare_data(data):
    """ Drops all matches where we don't have data for both teams. """
    data = data.copy()
    data = _drop_unbalanced_matches(data)
    _check_data(data)
    return data


def train_model(data, ignore_cols):
    """ Trains a logistic regression model over the data. Columns that
        are passed in ignore_cols are considered metadata and not used
        in the model building.
    """
    # Validate the data
    data = prepare_data(data)
    target_col = 'points'
    (train, test) = split(data)
    train.to_csv('/tmp/out3.csv')
    (y_train, x_train) = _extract_target(train, target_col)
    x_train2 = _splice(_coerce(_clone_and_drop(x_train, ignore_cols)))

    y_train2 = [int(yval) == 3 for yval in y_train]
```

```python
    model = build_model_logistic(y_train2, x_train2, alpha=8.0)
    return (model, test)


def predict_model(model, test, ignore_cols):
    """ Runs a simple predictor that will predict if we expect a team to
        win.
    """

    x_test = _splice(_coerce(_clone_and_drop(test, ignore_cols)))
    x_test['intercept'] = 1.0
    predicted =  model.predict(x_test)
    result = test.copy()
    result['predicted'] = predicted
    return result


"""
    Ranks soccer teams by computing a power index based
    on game outcomes.
"""

import numpy as np
from numpy.linalg import LinAlgError
import pandas as pd

import world_cup

def _build_team_matrix(data, target_col):
    """ Given a dataframe of games, builds a sparse power matrix.
        We expect the input data to have two back to back rows for
        each game. The first row will have information about the home
        team, the second row will have information about the away team.
        The matrix we compute will have columns representing teams and
        rows representing games. For each game, the home team will have
        a positive value that team's column. The away team will have a
        negative value in that column. Since home advantage is so
        important in soccer, we discount the home team by a certain
        margin. Note that we also have to be somewhat careful here,
        because for world cup data, we use values of is_home that are
        not binary (that is, they range between 0,0.0 and 1.0.
        The final column in the power matrix is a points value,
        computed as the difference between the target column for the
        home team and the target column for the away team.
    """
    teams = {}
    nrows = len(data) / 2
    for teamid in data['teamid']:
        teams[str(teamid)] = pd.Series(np.zeros(nrows))

    result = pd.Series(np.empty(nrows))
    teams[target_col] = result

    current_season = None
    current_discount = 2.0
```

```
    for game in xrange(nrows):
        home = data.iloc[game * 2]
        away = data.iloc[game * 2 + 1]
        if home['seasonid'] != current_season:
            # Discount older seasons.
            current_season = home['seasonid']
            current_discount *= 0.6
            print "New season %s" % (current_season,)

        home_id = str(home['teamid'])
        away_id = str(away['teamid'])
        points = home[target_col] - away[target_col]

        # Discount home team's performance.
        teams[home_id][game] = (1.0 + home['is_home'] * .25) / current_discount
        teams[away_id][game] = (-1.0 - away['is_home'] * .25) / current_discount
        result[game] = points

    return pd.DataFrame(teams)


def _build_power(games, outcomes, coerce_fn, acc=0.0001, alpha=1.0, snap=True):
    """ Builds power model over a set of related games (they
        should all be from the same competition, for example).
        Given a series of games and their outcome, builds a logistic
        regression model that computes a relative ranking for the teams.
        Returns a dict of team id to power ranking between 0 and 1.
        If snap is set, the rankings are bucketed into quartiles. This
        is useful bcause we may only have rough estimates of power
        rating and we don't want to get a false specificity.
    """
    outcomes = pd.Series([coerce_fn(val) for val in outcomes])
    games.to_csv('/tmp/out.csv')
    model = world_cup.build_model_logistic(outcomes, games,
        acc=acc, alpha=alpha)

    #print model.summary()
    params = np.exp(model.params)
    del params['intercept']
    params = params[params != 1.0]
    max_param = params.max()
    min_param = params.min()
    param_range = max_param - min_param
    if len(params) == 0 or param_range < 0.0001:
        return None

    params = params.sub(min_param)
    params = params.div(param_range)
    qqs = np.percentile(params, [20, 40, 60, 80])
    def _snap(val):
        """ Snaps a value to a quartile. """
        for idx in xrange(len(qqs)):
            if (qqs[idx] > val):
                return idx * 0.25
        return 1.0
```

```
    if snap:
        # Snap power data to rough quartiles.
        return params.apply(_snap).to_dict()
    else:
        return params.to_dict()


def _get_power_map(competition, competition_data, col, coerce_fn):
    """ Given the games in a competition and the target column
        describing the result, compute a power ranking of the teams.
        Since the 'fit' is likely to be fairly loose, we may
        have to try several times with different regularization and
        alpha parameters before we get it to converge.
        Returns a map of team id to power ranking.
    """
    acc = 0.000001
    alpha = 0.5
    while True:
        if alpha < 0.1:
            print "Skipping power ranking for competition %s column %s" % (
                competition, col)
            return {}
        try:
            games = _build_team_matrix(competition_data, col)
            outcomes = games[col]
            del games[col]
            competition_power = _build_power(games, outcomes, coerce_fn, acc,
                                             alpha, snap=False)
            if not competition_power:
                alpha /= 2
                print 'Reducing alpha for %s to %f due lack of range' % (
                    competition, alpha)
            else:
                return competition_power
        except LinAlgError, err:
            alpha /= 2
            print 'Reducing alpha for %s to %f due to error %s' % (
                competition, alpha, err)


def add_power(data, power_train_data, cols):
    """ Adds a number of power columns to a data frame.
        Splits the power_train_data into competitions (since those will
        have disjoint power statistics; for example, EPL teams don't play
        MLS teams (in regular games), so trying to figure out which team is
        stronger based on wins and losses isn't going to be useful.

        Each entry in cols should be a column name that will be used to
        predict, a function that wil evaluate the difference in that
        column between the two teams that played a game, and a final
        name that will be used to name the resulting power column.

        Returns a data frame that is equivalent to 'data' ammended with
        the power statistics for the primary team in the row.
    """
```

```python
    data = data.copy()
    competitions = data['competitionid'].unique()
    for (col, coerce_fn, final_name) in cols:
        power = {}
        for competition in competitions:
            competition_data = power_train_data[
                power_train_data['competitionid'] == competition]
            power.update(
                _get_power_map(competition, competition_data, col, coerce_fn))

        names = {}
        power_col = pd.Series(np.zeros(len(data)), data.index)
        for index in xrange(len(data)):
            teamid = str(data.iloc[index]['teamid'])
            names[data.iloc[index]['team_name']] = power.get(teamid, 0.5)
            power_col.iloc[index] = power.get(teamid, 0.5)
        print ['%s: %0.03f' % (x[0], x[1])
               for x in sorted(names.items(), key=(lambda x: x[1]))]
        data['power_%s' % (final_name)] = power_col
    return data

"""
    Turns raw statistics about soccer matches into features we use
    for prediction. Combines a number of games of history to compute
    aggregates that can be used to predict the next game.
"""

import pandas as pd

import match_stats

def get_wc_features(history_size):
    return pd.read_csv('results-20140714-123022.csv',sep=',')

def get_features(history_size):
    return pd.read_csv('results-20140714-123519.csv',sep=',')

def get_game_summaries():
    return pd.read_csv('results-20140714-124014.csv',sep=',')


def get_non_feature_columns():
    """ Returns a list of the columns that are in our features dataframe that
        should not be used in prediction. These are essentially either metadata
        columns (team name, for example), or potential target variables that
        include the outcome. We want to make sure not to use the latter, since
        we don't want to use information about the current game to predict that
        same game.
    """
    return ['teamid', 'op_teamid', 'matchid', 'competitionid', 'seasonid',
            'goals', 'op_goals', 'points', 'timestamp', 'team_name',
            'op_team_name']

def get_feature_columns(all_cols):
    """ Returns a list of all columns that should be used in prediction
```

```
        (i.e. all features that are in the dataframe but are not in the
        features.get_non_feature_column() list).
    """
    return [col for col in all_cols if col not in get_non_feature_columns()]
```

Kaynaklar

[1] http://googlecloudplatform.blogspot.de/2014/07/google-cloud-platform-
html

[2] https://github.com/GoogleCloudPlatform/ipython-soccer-predictions

[3] http://nbviewer.ipython.org/github/GoogleCloudPlatform/ipython-soccer
blob/master/predict/wc-final.ipynb

[4] http://sayilarvekuramlar.blogspot.com/2014/07/dunya-kupasini-tahmin-l
html