

Cok Boyutlu Regresyon ile Tavsiye

Tavsiye sistemlerinde eger tercih edilen nesnelerin sayisi cok fazla degilse ilginç bir metot cok boyutlu cikti regresyonu (multioutput regression -MR-) kullanmak. Tavsiye sistemleri bilindigi gibi cogunlukla bircok mumkun secenegi siralayarak en iyi secimleri en uste koymaya ugrasir, ve yine, cogunlukla bu sistemleri egitirken her secenek icin tek bir etiket (begenildi / begenilmedi) ile o etiketin bagli oldugu "kaynak" verisi arasinda iliski kurulmaya ugrasilir, ve her secenek icin ayri bir obje egitilir (SVD bir istisna, bu metot ayni anda tum kullanicilar ve tum secimleri gozonune alabilir).

MR kullanarak cok boyutlu kaynak verisiyle yine cok boyutlu cikti arasinda *ayni anda* bir iliski kurabiliriz, ve daha onemlisi bu iliskiye esnek (soft) bir sekilde yapabiliriz. Ayrica MR girdi ve cikti arasinda iliski kurdugu gibi, ciktilar arasinda da baglantilari bulabilir, eger her secenek icin ayri ayri tavsiye objeleri kullansaydik bu iliskiye ogrenemezdik. Mesela 40 yas ustü Arizona'dan gelen kisiler aksiyon filmi seviyor, ve ayni kisiler yine komedi filmi seviyor. MR aksiyon ile komedi ciktilari arasindaki iliskiye ogrenebilir.

Isler bir MR metotunun ornegini [2] yazisinda bulabiliriz, bu yazida karar agaci (decision tree) kullaniliyor, `scikit-learn` paketi karar agaclarinin ciktinin cok olabildigini destekliyor. [2]'deki ornegin amaci bir cemberi ogrenmek mesela; cember soyle olusturulmus, kaynak tek bir boyut - sirali sayilar, cikti ise iki boyutlu, $\cos(x)$ digeri $\sin(x)$. Bu dogal olarak bir cember ortaya cikartacaktır. [2]'de goruldugu gibi MR bu cemberi ogreniyor.

Ayni teknigi mesela Movielens 1M verisinde genre ogrenmek icin kullanabiliriz. Kaynak musterinin kisisel bilgileri, hedef ise cok boyutlu 0/1 olarak kodlanmış *tum genre'ler* olacak. Eger bir kisi hem komedi hem aksiyon sevmis ise matrisin o kisiye tekabül eden satirinda aksiyon ve komedi kolonu 1 olacak mesela. Bu sekilde tum musteriler kodlanacak, ve kaynak olarak kisisel veri alinacak.

Bu veriyi *Pivottlama* yazisinin dizininde bulabilirsiniz, zip dosyasini acmanız yeterli.

```
import pandas as pd
cols = ['user_id', 'movie_id', 'rating', 'timestamp']
ratings = pd.read_csv('../stat_pandas_ratings/ratings.dat', sep='::',
                      header=None, names=cols)
cols = ['movie_id', 'title', 'genres']
movies = pd.read_csv('../stat_pandas_ratings/movies.dat', sep='::',
                     header=None, names=cols)
cols = ['user_id', 'gender', 'age', 'occupation', 'zip']
users = pd.read_csv('../stat_pandas_ratings/users.dat', sep='::',
                    header=None, names=cols)

genre_iter = (set(x.split('|')) for x in movies.genres)
genres = sorted(set.union(*genre_iter))
dummies = pd.DataFrame(np.zeros((len(movies), len(genres))), columns=genres)
for i, gen in enumerate(movies.genres):
    dummies.ix[i, gen.split('|')] = 1
```

```

movies_windic = movies.join(dummies.add_prefix('Genre_'))
movies_windic = movies_windic.drop(['title', 'genres'], axis=1)

joined = ratings.merge(movies_windic, left_on='movie_id', right_on='movie_id')

y = joined.groupby('user_id').sum()
y = y.drop(['movie_id', 'rating', 'timestamp'], axis=1)
y[y > 0.0] = 1.0

```

Ciktiyi kodlarken Movielens verisinde bir kisi bir filme herhangi bir not vermis ise o filmin genre'sini "tercih edilir" olarak 1 ile isaretledik. Tabii ki o film icin not dusuk olabilir (ve bu durumu bir filtre kosulu olarak kullanabilirdik), ama bu, o kisinin o belirli filmi begenmedigini gosterir, o filmi sectiyse musterisi buyuk bir ihtimalle o genre'yi seviyordur. Bu hipotezi test edebilirsiniz, `y[y > 0.0]` yerine `y[y > 0.3]` secilirse basari oranini dusuyor. Yani bize faydali olacak degerli verileri ikinci filtre ile kaybediyoruz.

```

from sklearn.feature_extraction import DictVectorizer
def one_hot_dataframe(data, cols):
    vec = DictVectorizer()
    mkdict = lambda row: dict((col, row[col]) for col in cols)
    tmp = vec.fit_transform(data[cols].to_dict(outtype='records')).toarray()
    vecData = pd.DataFrame(tmp)
    vecData.columns = vec.get_feature_names()
    vecData.index = data.index
    data = data.drop(cols, axis=1)
    data = data.join(vecData)
    return data

X = users.copy()
X['occupation2'] = users['occupation'].map(lambda x: str(x))
X['zip2'] = users['zip'].map(lambda x: str(x)[0])
X['zip3'] = users['zip'].map(lambda x: str(x)[:2])
X = one_hot_dataframe(X, ['occupation2', 'gender', 'zip2', 'zip3'])
X = X.drop(['occupation', 'zip'], axis=1)
X = X.set_index('user_id')
X = X.ix[y.index]
X = X.reindex(y.index)
print X.shape, y.shape

(6040, 134) (6040, 18)

from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_auc_score
from sklearn.cross_validation import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import Lasso, Ridge, LinearRegression

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=1000)
clf = RandomForestRegressor(max_depth=3, n_estimators=5)

```

```

clf.fit(x_train,y_train)
y_pred = clf.predict(x_test)

fpr, tpr, thresholds = roc_curve(np.ravel(y_test), np.ravel(y_pred))
roc_auc = auc(fpr, tpr)
print 'Tree AUC', roc_auc

imps = pd.Series(list(clf.feature_importances_), index=X.columns)
imps = imps.order(ascending=False).head(20)
print 'important features'
print np.array(imps.index)

Tree AUC 0.844777794213
important features
['gender=F' 'age' 'gender=M' 'occupation2=10' 'zip3=46' 'zip2=3' 'zip3=37'
 'zip3=65' 'zip3=31' 'zip3=33' 'zip3=41' 'zip3=39' 'zip3=67' 'zip3=01'
 'zip3=20' 'zip3=24' 'zip3=95' 'zip3=08' 'zip3=09' 'zip3=03']

```

Test etmek için AUC olcusunu kullandık. Bir puruz vardı, AUC hem tahmin hem de test etiketlerini tek boyutlarda alır, bizim çıktımız çok boyutlu idi. Ne yapmalı? Biz de her iki matrisi (test etiketleri, tahminler) “düzlestirdik”, tek boyut haline getirdik. Nasıl olsa her iki matris aynı şekilde düzleştirildiği ve boyutları zaten aynı olduğu için sonuç iki vektör de aynı boyutta oldu. Ve bu iki vektör üzerinde AUC hesabı yaptık. Sonuç üstte görüldüğü gibi %84.

Ayrıca `RandomForestRegressor` objesi regresyonun önemli bulduğu kaynak öğelerini önem sırasına göre gösterebilir. Üstte cinsiyet (gender), yaş (age) genre seçiminde önemli görünüyor. Bu akla yatkın. Ayrıca meslekler (occupation) arasında 10 kodlu olan önemliymiş, bu kodun ilköğretim öğrencisine tekabül ettiğini gördük. İlginç.

Karsılaştırmak için basit bir alternatif kodladık, tüm beğenilerin ortalamasını aldık, ve bu senaryoda tahmin olarak herkes için sürekli aynı tahmini yaptık. Buradan gelen sonuç %83.

```

y_naive = np.array(y_train).mean(axis=0)
y_naive = pd.DataFrame([y_naive], index=range(y_test.shape[0]),
                        columns=list(y.columns))

fpr, tpr, thresholds = roc_curve(np.ravel(y_test), np.ravel(y_naive))
roc_auc = auc(fpr, tpr)
print 'Naive AUC', roc_auc

Naive AUC 0.833752370725

```

Evet, bu çok basit bir tahmin için yüksek gelebilir, fakat bir Netflix Veri Bilimcisi’nin söylediği gibi “popüleriteyi yenmek zordur”. Neyse ki üstteki MR metodu popüleriteyi yendi. Ayrıca bir şey daha yaptı, eğer tavsiye sistemi için popülerite kullansaydık herkes için sürekli aynı tavsiyeyi vermemiz gerekecekti. MR bazlı “kisielleştirilmiş” metod herkes için ayrı tavsiyeler verebilecektir.

http://upload.wikimedia.org/wikipedia/commons/2/24/ZIP_Code_zones.svg

http://scikit-learn.org/stable/auto_examples/tree/plot_tree_regression_multioutput.html