# به نام خدا

## پروژه پایانی درس داده کاوی
## تخمین قیمت مسکن


**سیدمصطفی احمدی**
**۹۴۳۱۰۵۳**


**تابستان ۹۸**

**فهرست**

# پیش پردازش

در این بخش سعی کردیم که از طریق تولید اتوماتیک داده‌های گم‌شده، تغییر نوع آنها و همچنین کاهش ابعاد، داده ها را تمیز کنیم.

## تولید داده های گم شده

برای تولید اتوماتیک داده‌های گم‌شده از imputer استفاده کردیم. اما آنچه که میخواستیم به شکل آماده در imputer ها موجود نبود. هدف ما این بود که در داده های عددی، اگر داده ای گم شده است جای آن میانگین را بگذارد و در داده های کتگوریکال، چیزی را که در اکثریت وجود دارد بگذارد. برای این کار یک کلاس ساختیم که از imputer اصلی ارث بری میکند. به این صورت:

```
class DataFrameImputer(TransformerMixin):

    def __init__(self):

    def fit(self, X, y=None):
        self.fill = pd.Series([X[c].value_counts().index[0]
                               if X[c].dtype == np.dtype('O') else X[c].mean() for c in X],
                              index=X.columns)
        return self

    def transform(self, X, y=None):
        return X.fillna(self.fill)
```

## تغییر نوع داده‌ها

داده‌های کتگوریکال که نوع آنها string باید میبود، در dataframe به شکل object ذخیره شده بودند. همچنین این داده‌ها حجم زیادی اشغال میکردند، برای اینکه در ذخیره سازی روی ram راحت تر باشیم، آنها را به شکل دستی encode کردیم. نحوه اینکد کردن هم در تکه کد زیر آمده است:

```
for i in imputed_train.columns:
    if imputed_train[i].dtype not in (int, float):
        imputed_train[i] = imputed_train[i].astype(str)
        temp = imputed_train[i].unique()
        temp_dict = {temp[i]: i for i in range(len(temp))}
        imputed_test[i] = imputed_test[i].map(temp_dict)
        imputed_train[i] = imputed_train[i].map(temp_dict)
```

همانطور که مشاهده میشود، یک دیکشنری از این استرینگ ها ساخته ایم و به هر استرینگ یک عدد نسبت داده ایم تا حجم مورد نیاز برای ذخیره سازی اش کمتر باشد.

## کاهش ابعاد

با توجه به این که هر چه ابعاد داده بیشتر باشند، نیاز به فضای ذخیره سازی افزایش می یابد و همچنین زمان مورد نیاز برای ساخت مدل نیز افزایش می یابد. داده ها ۸۰ ستون دارند که یکی مربوط به قیمت است و بدون احتساب آن میشود ۷۹ ویژگی! در نتیجه برای بهینه شدن فضای مورد استفاده و زمان مورد نیاز به ساخت مدل، باید طوری ابعاد را کاهش داد که اولا دقت مدل حاصل شده از داده های جدید اختلاف فاحش نسبت به دقت مدل حاصل از داده های قبلی نداشته باشد، ثانیا داده های جدید فضای کمتری برای ذخیره سازی و همچنین زمان کمتری برای ساخت مدل نیاز داشته باشند. برای کاهش ابعاد، روش های متنوعی وجود دارند که به اختصار به هر یک از آنها اشاره می کنیم.

روش MDS[1]: فرض میکنیم n آیتم در یک فضای d بعدی داده شده است و یک ماتریس n در n مجاورت بین آیتم های داده وجود دارد، MDS یک نمایش k بعدی از آن آیتم ها تولید میکند که فاصله بین نقاط در فضای جدید، تاثیر پذیرفته از مجاورت در داده های اصلی است.

روشPCA[2] : این الگوریتم یکی از محبوب ترین الگوریتم های موجود برای کاهش بعد به شمار می رود. این الگوریتم با انتخاب بزرگترین مقدار ویژه ماتریس کوواریانس و بیشینه کردن واریانس باقی مانده، اجزایی را می یابد که تصویرسازی را غیر وابسته میکنند.

روشICA[3] : این روش، یک روش مرتبه بالاتر است که پیش بینی های خطی را دنبال می کند که نه تنها به یکدیگر متصل نیستند، بلکه تقریبا به صورت آماری مستقل هستند.

روش منحنی ها، سطوح، چندوجهی های بنیادین[4]: در شرایطی که داده های اولیه دارای ساختار منحنی هستند، روش هایی مانند PCA به خوبی کار نمی کنند. در چنین شرایطی تقریب زدن داده منحنی توسط یک خط مستقیم، تقریب خوبی از داده اصلی نیست. برای چنین نوع داده ای، راه حل می تواند استفاده از منحنی، سطوح و چند وجهی باشد. منحنی ای که بتواند به خوبی فیت داده شود یکی از بخش های مهم آنالیز داده است. در این هنگام که برای داده ها یک منحنی بیابیم، ابعاد داده به شکل غیر خطی به یک بعد کاهش می یابد.

روشLLE[5] : این روش برای یادگیری چندوجهی های نزدیک به داده و تصویر داده به آنهاست. برای هر مورد، الگوریتم دنبال k همسایه نزدیک میگردد و یک مجموعه از وزن ها برای تقریب آن تولید میکند. این عمل به طور همزمان برای همه داده استفاده میشود. زمانی که وزن ها مشخص شدند، الگوریتم به دنبال نقاطی از ابعاد پایین تر میگردد. نقاط جدید از همسایگان خود به همان شیوه بازسازی می شوند.

روشIsomap[6] : اگر در روش MDS فاصله بین اجسام به شکل جغرافیایی اندازه گیری شوند، آنگاه به Isomap خواهیم رسید. در یک چند وجهی کوتاهترین مسیر در گراف همسایگی است که هر موجودیت را به k همسایه نزدیک خود متصل می کند.

در مقاله ای که در دانشگاه Kaunas تهیه شده است به طور مفصل این شش الگوریتم از نظر سرعت و دقت با یکدیگر در سه نوع داده ی تصادفی کلاستر نشده، تصادفی کلاستر شده و داده واقعی مالی گرفته شده از یک سایت معاملات سهام مقایسه شده اند.

در این مقاله یک معیار درستی و سرعت برای هر الگوریتم به صورت عدد 1 تا 6 نسبت داده شده است که 1 به معنای کمترین درستی/سرعت و 6 به معنای بیشترین آن میباشد.

---

[1] Multidimensional Scaling

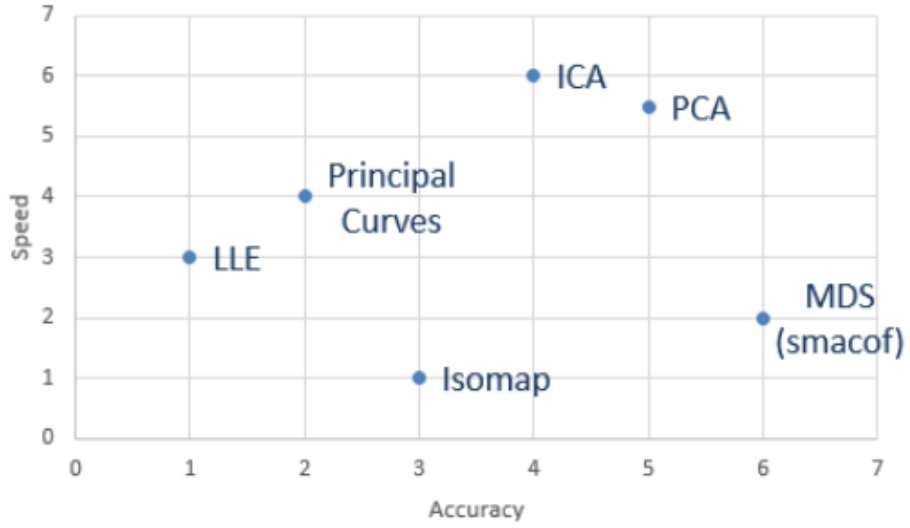
[2] Principal Components Analysis

[3] Independent Component Analysis

[4] Principal Curves, Surfaces and Manifolds

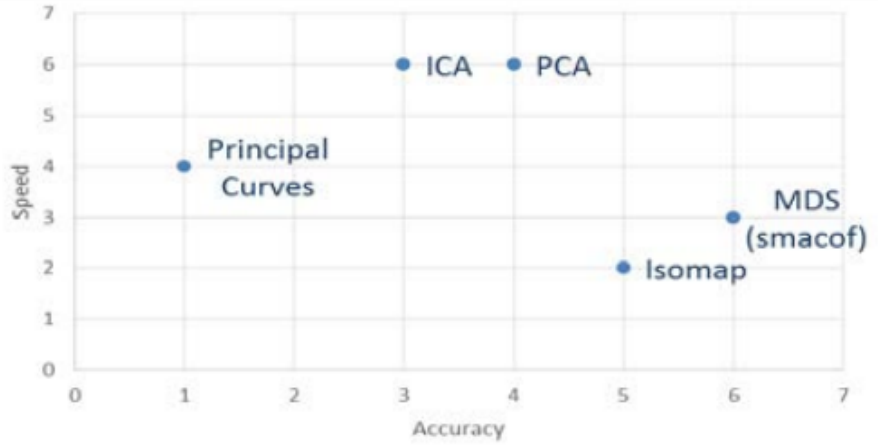
[5] Locally Linear Embedding

[6] Isometric Mapping




که قابل ملاحظه است مجموع امتیازات الگوریتم PCA در اینجا بالاترین نمره است.

در داده های واقعی مالی نیز نتایج به این شرح است:




که در اینجا نیز مشاهده میشود که باز هم الگوریتم PCA بالاترین مجموع نمرات را دارد.

به این دلیل که هم سرعت و هم دقت برای ما حائز اهمیت است (سرعت به دلیل تعداد رکوردهای بالایی که در اختیار داریم، و دقت هم به دلیل اینکه مدلی میخواهیم که کارایی معقولی از خود به نمایش بگذارد) ما نیز به همین نتایج بسنده کرده و PCA را به عنوان الگوریتم منتخب خود انتخاب می کنیم.

## مقایسه‌ی درخت تصمیم و جنگل تصادفی

در این قسمت داده های train را در شرایط مساوی به الگوریتم های decision tree و random forest دادیم. یعنی در این بخش به دنبال یک مدلی بودیم که در مسابقات محلی بتواند برنده شود و سپس برنده را در ککل سابمیت کنیم! البته نکته اینجاست که هیچ یک را تیون نکردیم و ساده ترین حالتشان را به اجرا گذاشتیم. نتایج به این شرح بودند:

| | With PCA | Without PCA |
|---|---|---|
| **Random Forest** | 25298.07134537762 | 19463.72684234542 |
| **Decision Tree** | 33210.72105187551 | 27419.49592420772 |
| **برنده** | rf | rf |

در جدول بالا منظور از اعداد داخل خانه‌ها، خطای مدل است. پس یعنی هرچه خطا کمتر باشد، مدل بهتری داریم.

## پارامتر تیونینگ

میدانیم که پس از انتخاب مدل، مهمترین بخش این است که پارامترهایش را به بهترین شکل ممکن طراحی کنیم. برای اینکار از RandomizedSearchCV استفاده کردیم که کار را برایمان راحت کند. پارامترهایی که به ذهنمان رسید میتوانند تیون شوند همراه با مقادیر مختلفشان به شرح زیر هستند:

```
n_estimators = [6, 60, 600, 1000]
min_samples_leaf = [2, 5, 10]
max_features = [0.1, 0.5, 0.9]
```

برای بدون pca بهترین پارامترها:

{'n_estimators': 60, 'min_samples_leaf': 2, 'max_features': 0.5}

با خطای: 17358.611790991014

برای با pca بهترین پارامترها:

{'n_estimators': 600, 'min_samples_leaf': 2, 'max_features': 0.9}

با خطای: 22874.913488223396

در نهایت دیدیم که بدون pca نتایج بهتری برایمان حاصل میشود و ما رندم فارست با پارامترهای n_estimators مساوی با ۶۰ و min_samples_leaf مساوی با ۲ و max_features مساوی با ۰.۵ را راهی مرحله ی بعد میکنیم.
اما توجه به این نکته ضروری است که گرچه ما pca را کنار گذاشتیم، اما بسته به کاربرد استفاده از pca میتواند بسیار مفید باشد. زمانی که test time و train time اهمیت بالایی دارند استفاده از این الگوریتم بسیار توصیه میشود، اما در اینجا چون این زمان ها برایمان مسئله نیست، میتوانیم آن را کنار بگذاریم.