به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

**هوش مصنوعی**

# تمرین کامپیوتری چهارم

| | |
|---|---|
| نام و نام خانوادگی | مهسا تاجیک |
| شماره دانشجویی | 810198126 |
| تاریخ ارسال گزارش | 14 خرداد |

## فاز اول : پیش پردازش و مشاهده‌ی داده

داده‌ای که در اختیار داریم دارای 9 ستون و 130443 سطر است که ستون ها عبارتند از :

```
(['brand', 'category', 'created_at', 'description', 'image_count',
  'mileage', 'price', 'title', 'year'],
 dtype='object')
```

بعضی از سطرها مقادیر ازدست‌رفته دارند که آن‌ها را از دیتافریم حذف کردیم و ابعاد داده به (120451,9) کاهش یافت.

سپس ستون‌هایی که داده‌های آن‌ها از نوع دسته‌ای هستند، یعنی ستون های brand, category , year, created_at را به مقادیر عددی کد کردیم.

ستون category شامل دو مقدار مختلف heavy و light است که زمانیکه ما مقادیر ازدست‌رفته را حذف می‌کنیم ، تمام ستون‌های شامل مقادیر heavy هم حذف می شوند و این ستون تنها یک مقدار خواهد داشت بنابراین اطلاعاتی به ما اضافه نمی‌کند و همانطور که انتظار داریم information gain آن هم صفر می‌شود.

ستون created_at شامل نام یکی از روزهای هفته و ساعت است که ساعت را از آن حذف کردیم و روزهای هفته را که مقادیر کتگوریکال داشتند به مقادیر عددی کد کردیم.

برای پیش پردازش روی ستون‌های description و title که متن فارسی دارند، از کتابخانه‌ی هضم استفاده می کنیم . ابتدا این دو ستون را normalize می کنیم و punctuationها را حذف می‌کنیم که شامل موارد زیر است:

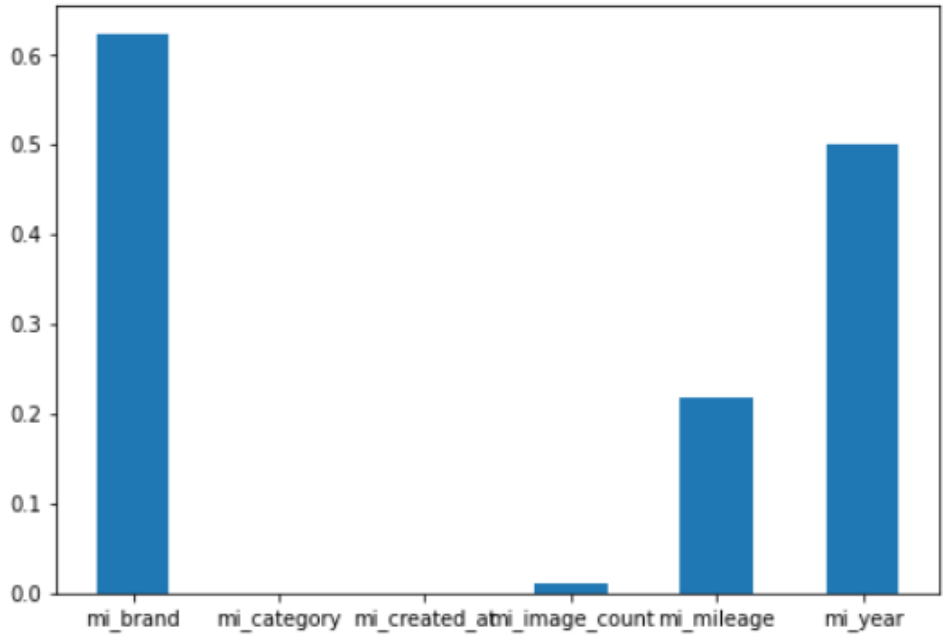
```
punctuations = !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~؛،,»«<>؟!
```

سپس با استفاده از countVectorizer برای هر کدام از این دو ستون، ماتریسی شامل کلمات تمام متن ها و تعداد تکرار هر کدام در هر سطر از دیتافریم تشکیل می دهیم. سپس این دو دیتافریم را با دیتافریم اصلی که ستون های title، description و category از آن حذف شده ادغام می‌کنیم.

پرسش ۱) از این نمودار میتوانیم برای انتخاب ویژگی‌ها استفاده کنیم زمانیکه تعداد ویژگی ها خیلی زیاد باشد. با استفاده از این نمودار مشخص می‌شود که کدامیک از ویژگی ها اطلاعات بیشتری به ما می‌دهند و بعضی از آن ها هیچ اطلاعاتی اضافه نمی‌کنند و می توانیم آن‌ها را حذف کنیم و سایز دیتا را

کاهش دهیم و در زمان و حافظه‌ی مصرفی صرفه جویی کنیم. در واقع وابستگی بین هر دو ویژگی را می توان محاسبه کرد و اگر مقدارش صفر شد یعنی دو ویژگی کاملا از هم مستقل هستند

```
[0.62419425 0.         0.         0.01149709 0.21863502 0.5001221 ]
```




پرسش ۲) با دستور data.dtypes تایپ ستون‌ها را می توانیم مشاهده کنیم :

```
brand           object
category        object
created_at      object
description     object
image_count      int64
mileage        float64
price            int64
title           object
year            object
```

ستون‌هایی که تایپ object دارند می توانیم ابتدا به تایپ category تبدیل کرده سپس با دستور cat.codes به مقادیر عددی کد کنیم :

```
data['brand'] = data['brand'].astype('category').cat.codes
data['category'] = data['category'].astype('category').cat.codes
data['year'] = data['year'].astype('category').cat.codes
data['created_at'] = data['created_at'].map(lambda x: x[:-5])
data['created_at'] = data['created_at'].astype('category').cat.codes
```

پرسش ۳) برای شمارش تعداد کلمات در ستون های متنی title , description از countVectorizer استفاده کردیم که متدهایی بنام min_df و max_df دارد که با تنظیم آن ها میتوان کلماتی که تعداد

تکرار خیلی کم یا تعداد تکرار زیادی دارند و در اکثر داکیومنت ها تکرار شدند یا همان stop word ها را حذف کرد. همچنین متدی بنام ngram_range دارد که می توانیم مشخص کنیم که چند کلمه پشت هم را در نظر بگیرد چون همانطور که میدانیم بعضی کلمات در جاهای مختلف معانی متفاوتی دارند زمانیکه کلمات قبل یا بعد آن ها را در نظر میگیریم به فهم معنی کلمه در آن جایگاه کمک می کند.

بعد از چندین بار تغییر این مقادیر، برای مقادیر زیر به جواب بهینه رسیدیم و آن ها را فیکس کردیم و در کل 36 ویژگی خواهیم داشت:

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer1 = CountVectorizer(min_df=0.1,max_df=0.9,ngram_range=(1,3))
desc_tokens = vectorizer1.fit_transform(data_description_arr)

vectorizer2 = CountVectorizer(min_df=0.1,max_df=0.9,ngram_range=(1,3))
title_tokens = vectorizer2.fit_transform(data_title_arr)

desc_df = pd.DataFrame(desc_tokens.toarray(),columns=vectorizer1.get_feature_names())
title_df = pd.DataFrame(title_tokens.toarray(),columns=vectorizer2.get_feature_names())

data = pd.concat([data,desc_df,title_df])
data = data.fillna(0)
data
```

پرسش ۴) یکی از روش‌های برخورد با مقادیر از دست رفته ، حذف آن هاست که ما هم از همین روش استفاده کردیم و سطرهایی که حتی یک مقدار nan در آن ها بود ، حذف کردیم. مزیت این روش این است که یک مدل robust داریم اما با حذف این مقادیر ، اطلاعاتی را از دست خواهیم داد و این روش زمانی بهتر است استفاده شود که سایز دیتاست به نسبت مقادیر از دست رفته به اندازه کافی بزرگ باشد.

روش دیگر جایگزین کردن یک مقدار ثابت بجای مقادیر ازدست رفته است مثلا می توانیم تمام مقادیر nan را با 0 ، 1 یا 1- پرکنیم. یا همچنین میتوانیم با میانگین یا میانه‌ی آن سطر پر کنیم یا اینکه روش روش های backwardfill یا forwardfill استفاده کنیم و با مقدار قبلی یا بعدی در سطر یا ستون ، آن خانه از دیتافریم را پر کنیم.

## فاز دوم : پیش‌بینی قیمت

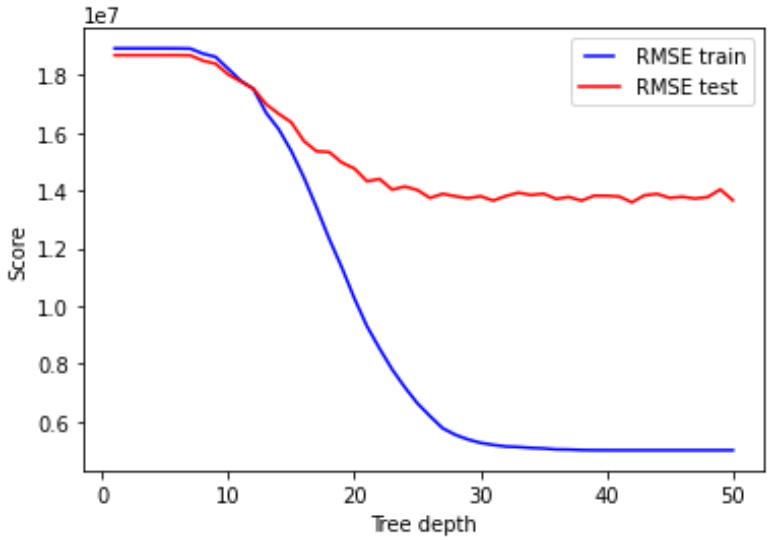
پارامتر n_neighbors مشخص می کند که برای طبقه بندی هر دیتا چندتا همسایه اطراف آن را باید برای تصمیم گیری در مورد کلاس این دیتا در نظر بگیریم که برای مقادیر 10 و 15 و 20 مقدار خطای rmse بصورت زیر است :

### KNN

| n_neighbors | RMSE Score |
|---|---|
| 20 | 17415425.85978448 |
| 15 | 17193196.469603904 |
| 10 | 17045764.18711257 |

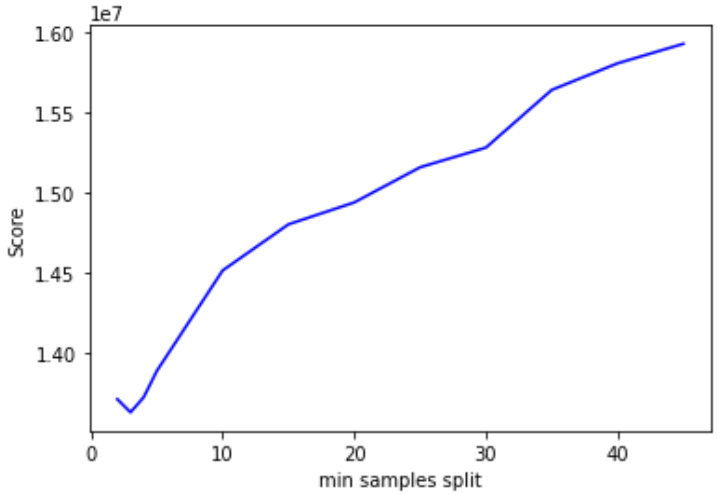
## DECISION TREE

- مقدار پارامتر `min_samples_split` را در حالت دیفالت یعنی 2 قرار دادیم و مقدار پارامتر `max_depth` را از 1 تا 50 تغییر دادیم و مقادیر خطای RMSE بصورت زیر می باشد. کمترین مقدار خطای RMSE حدودا برابر است با **13,600,000**




با دستور `tree._max_depth` حالت دیفالت را برای پارامتر `max_depth` میتوان پیدا کرد که برای درخت ما این مقدار برابر است با 41.

- مقدار `max_depth` را روی 42 فیکس میکنیم و مقدار پارامتر `min_samples_split` را برای مقادیر `[2,3,4,5,10,15,20,25,30,35,40,45]` تغییر می دهیم و مقادیر خطای `mse, rmse` را محاسبه میکنیم که تغییرات `rmse` بصورت زیر است:




بنابراین هرچقدر مقدار پارامتر min_samples_split کمتر باشد خطای کمتری داریم و کمترین مقدار در 3 اتفاق می افتد.

بصورت کلی که با تغییر مقادیر پارامترها خطای RMSE از حدود 13میلیون تا 18میلیون تغییر داشت.(در بعضی اجراها به خطای حدود 6 میلیون هم رسیدیم ولی در اجرای بعدی این مقدار مجددا افزایش پیدا کرد.)

## Linear regression

خطای این مدل حدودا 13 میلیون شد:

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression().fit(X_train, y_train)
coef = lr.coef_
intercept = lr.intercept_
y_pred = lr.predict(X_test)
lr.score(X_train, y_train)
mse = mean_squared_error(y_test,y_pred)
rmse = mean_squared_error(y_test,y_pred,squared=False)
rmse
```

```
13735635.807216046
```

پرسش ۵) به طور معمول 15 تا 20 درصد داده‌ها برای تست و بقیه برای آموزش در نظر گرفته می شود.

تقسیم بندی مختلف داده های آموزش و تست برای **Linear Regression :**

نتیجه برای 80 درصد داده ها برای آموزش و 20 درصد برای تست :

```
12845970.622448027
```

نتیجه برای 98 درصد داده ها برای آموزش و 2 درصد برای تست :

```
13735635.807216046
```

نتیجه برای 40 درصد داده ها برای آموزش و 60 درصد برای تست :

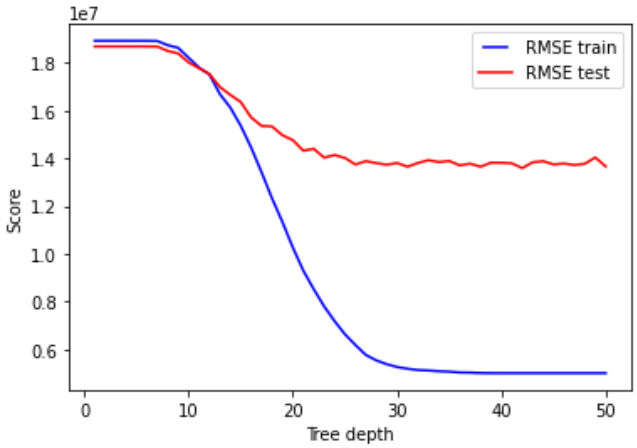
```
13860931.545627901
```

زمانیکه دیتای کافی برای آموزش نداریم و 40 درصد را برای آموزش در نظر میگیریم مدل به خوبی یاد نمیگیرد و underfitting اتفاق می افتد زمانیکه که 98 درصد داده ها را برای آموزش و دو درصد را برای تست میگیریم ، مدل بیش از حد آموزش میبیند و قابلیت generalization را از دست می دهد و overfitting اتفاق می افتد در هر دو حالت خطا بیشتر از زمانیست که 80 درصد برای آموزش و 20 درصد برای تست در نظر گرفتیم.

پرسش ۶)

بیشترین عمق ، برای decision tree بین 1تا 50 تغییر دادیم.نمودار خطای داده های اموزش و تست بصورت زیر است:

## Decision tree



## فاز سوم : استفاده از مدل‌های تجمیعی

پرسش ۷) مقادیر خطای ترین و تست برای random forest به ازای max_depth های 2 ، 5 و 10 بصورت زیر است:

```
Rmse_train Score: 18925575.094937544
Rmse_test Score: 18689445.033343326
Rmse_train Score: 18925575.230869036
Rmse_test Score: 18689445.172420606
Rmse_train Score: 18922040.329567824
Rmse_test Score: 18689379.429148108
```