

بسم الله الرحمن الرحيم

پروژه ۴ درس هوش مصنوعی  
دکتر فدایی و دکتر یعقوب زاده

مهندی وجہی

۸۱۰۱۰۱۵۵۸

## فهرست

4.....	مقدمه
5.....	آشنایی با مجموعه داده .....
5.....	خواندن داده ها.....
6.....	بررسی مقادیر جدول .....
7.....	بررسی همبستگی بین ویژگی ها .....
8.....	نمودار های hexbin و scatter .....
10.....	سایر بررسی .....
12.....	پیش پردازش داده ها .....
12.....	پر کردن داده های از دست رفته .....
14.....	حذف ستون ها .....
15.....	انواع داده ها .....
16.....	نرم الگوریتم استاندارد کردن داده ها .....
17.....	پیش پردازش داده های دسته ای .....
18.....	تقسیم بندی داده ها برای آموزش مدل.....
20.....	سایر روش های پیش پردازش .....
23.....	آموزش، ارزیابی و تنظیم .....
23.....	انواع مدل های یادگیری ماشین .....
23.....	مدل های با ناظر .....
24.....	مدل های بدون ناظر .....
24.....	مدل های تقویتی .....
25.....	مدل های نیمه ناظری .....
26.....	رگرسیون .....
26.....	رگرسیون خطی .....
26.....	پیاده سازی رگرسیون خطی مرتبه اول .....
27.....	روش های مختلف ارزیابی مدل .....
27.....	RSS .....
27.....	MSE .....
27.....	RMSE .....
28.....	R <sup>2</sup> score .....
28.....	ارزیابی مدل رگرسیون خطی مرتبه ۱ .....
31.....	طبقه بندی .....
31.....	کردن درخت تصمیم Prune .....

31	موارد استفاده از درخت تصمیم
32	تفاوت KNN با سایر مدل ها در آموزش
32	neighbor nearest one
32	روش های سنجش فاصله در KNN
32	پیش بینی دهک خانه با استفاده از KNN و درخت تصمیم
33	برآورد کردن بهترین پارامتر ها
34	رسم درخت تصمیم
35	Underfitting و Overfitting
36	روش های Ensemble
36	چرایی استفاده و کاربرد
36	روش bagging
37	روش boosting
37	جنگل تصادفی
37	Bootstrapping
38	تعداد درخت های تصمیم در جنگل تصادفی
38	موارد استفاده و عدم استفاده جنگل تصادفی
38	واریانس در جنگل تصادفی
38	پیاده سازی جنگل تصادفی
38	شرح هایپر پارامتر ها
39	پیدا کردن بهترین هایپر پارامتر
41	XGBoost
41	Gradient Boosting
42	عملکرد
42	پارامتر ها
43	پیاده سازی
45	SVM
45	ساخت مدل
45	ارزیابی مدل ها
46	روش محاسبه هایپر پارامتر ها
46	پیاده سازی
48	ارزیابی
50	نتیجه گیری
51	راه های بهبود و توسعه
52	منابع

## مقدمه

در این پروژه ما قصد داریم قیمت خانه را در شهر بوستون آمریکا پیش‌بینی کنیم. هدف از این پروژه آشنایی با نحوه انجام یک پروژه یادگیری ماشین است. در ابتدا پروژه و در بخش اول ما با داده‌ها و ساختار کلی آنها آشنا می‌شویم در ادامه به پیش‌پردازش داده‌های می‌پردازیم که در این مرحله باید داده‌های را آماده سازی کنیم این کار شامل پر کردن مقادیر از دست رفته جدول و همچنین حذف بعضی ویژگی‌ها و نرم‌الایز کردن آنها می‌شود در آخر این قسمت نیز داده‌ها برای آموزش مدل به ۳ قسمت تقسیم بنده می‌کنیم. در بخش سوم به ساخت مدل‌های یادگیری ماشین و ارزیابی آنها می‌پردازیم از جمله آنها می‌توان به رگرسیون خطی و پلینومیال، KNN و SVM اشاره کرد. در آخر هم هریک از مدل‌ها را ارزیابی می‌کنیم و با یکدیگر مقایسه می‌کنیم.



## آشنایی با مجموعه داده

### خواندن داده ها

در این مرحله لازم است مجموعه داده DataSet.xlsx را وارد کنیم. این مجموعه داده شامل اطلاعاتی درباره خانه ها در شهر بوستون است. ما از این مجموعه داده برای پیش بینی قیمت خانه های این شهر استفاده می کنیم و این داده ها در ۳ بخش test,train,validation مورد استفاده قرار می گیرد. لازم به ذکر است مشخص نیست این داده ها از کجا به دست آمده. در ادامه ویژگی ها و ستون های موجود در فایل به همراه توضیحات آن آمده.

نام ستون	توضیحات
CRIM	per capita crime rate by town.
ZN	proportion of residential land zoned for lots over 25,000 sq.ft.
INDUS	proportion of nonretail business acres per town
CHAS	Charles River dummy variable (= 1 if tract bounds river; 0 (otherwise)
NOX	nitric oxides concentration (parts per 10 million).
RM	average number of rooms per dwelling.
AGE	proportion of owner-occupied units built prior to 1940
DIS	weighted distances to five Boston employment centers.
RAD	index of accessibility to radial highways
TAX	full-value property-tax rate per \$10,000
PTRATIO	pupil-teacher ratio by town
B	.Bk – 0.63)^2 where Bk is the proportion of blacks by town)1000
LSTAT	% lower status of the population
MEDV (Target)	Median value of owner-occupied homes in \$1000s

جدول زیر نمونه ای از داده ها است.

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3	222.0	18.7	NaN	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

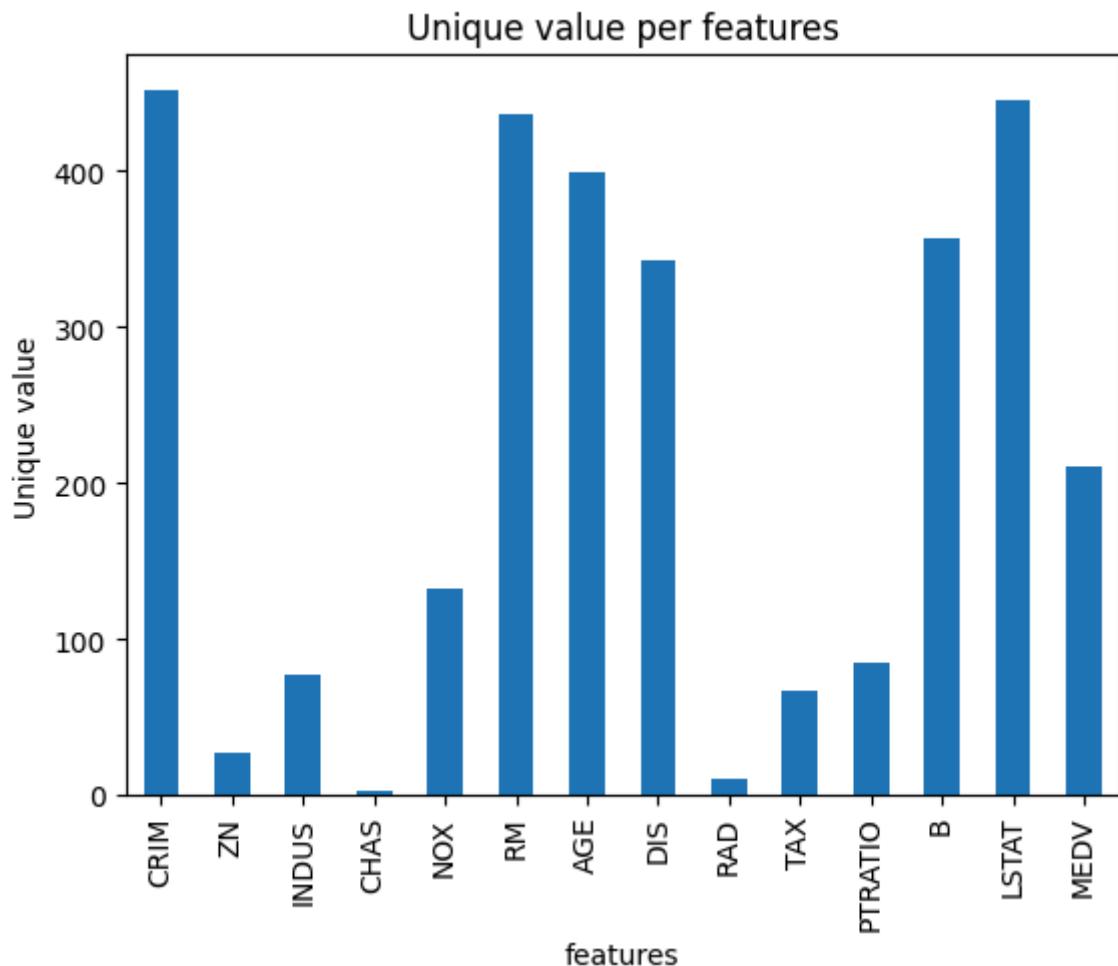
## بررسی مقادیر جدول

حال به بررسی دقیق تر داده های جدول می پردازیم.

	missing_count	missing_ratio	<class 'pandas.core.frame.DataFrame'>						
			RangeIndex: 506 entries, 0 to 505						
			Data columns (total 14 columns):						
#	Column	Non-Null Count	Dtype						
0	CRIM	506 non-null	float64						
1	ZN	506 non-null	float64						
2	INDUS	506 non-null	float64						
3	CHAS	480 non-null	float64						
4	NOX	506 non-null	float64						
5	RM	506 non-null	float64						
6	AGE	506 non-null	float64						
7	DIS	479 non-null	float64						
8	RAD	506 non-null	int64						
9	TAX	506 non-null	float64						
10	PTRATIO	506 non-null	float64						
11	B	486 non-null	float64						
12	LSTAT	506 non-null	float64						
13	MEDV	452 non-null	float64						
dtypes: float64(13), int64(1)									
memory usage: 55.5 KB									

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE
count	506.000000	506.000000	506.000000	480.000000	506.000000	506.000000	506.000000
mean	1.269195	13.295257	9.205158	0.175000	1.101175	15.679800	58.744660
std	2.399207	23.048697	7.169630	0.380364	1.646991	27.220206	33.104049
min	0.000000	0.000000	0.000000	0.000000	0.385000	3.561000	1.137000
25%	0.049443	0.000000	3.440000	0.000000	0.449000	5.961500	32.000000
50%	0.144655	0.000000	6.960000	0.000000	0.538000	6.322500	65.250000
75%	0.819623	18.100000	18.100000	0.000000	0.647000	6.949000	89.975000
max	9.966540	100.000000	27.740000	1.000000	7.313000	100.000000	100.000000

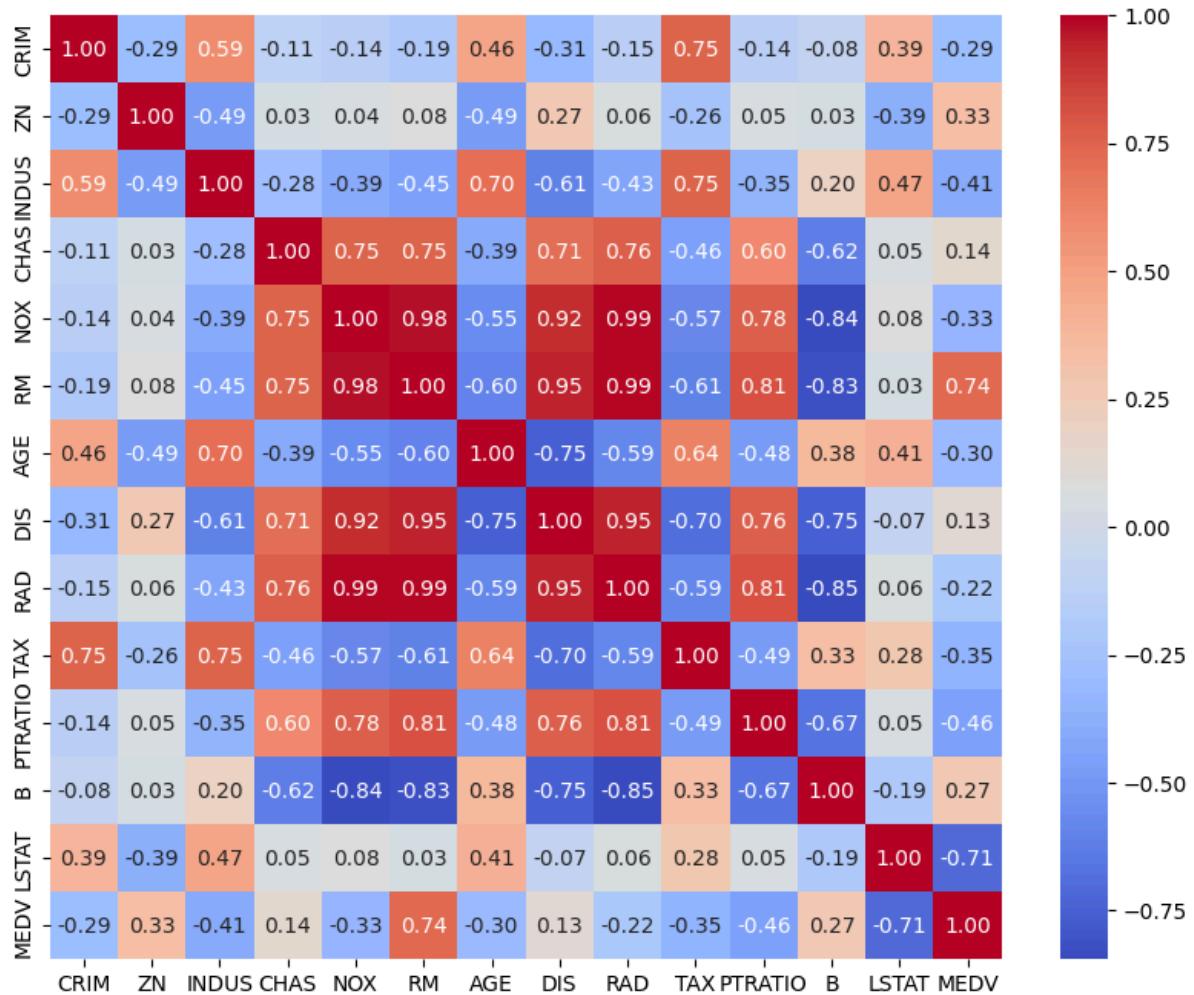
	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
count	479.000000	506.000000	506.000000	506.000000	486.000000	506.000000	452.000000
mean	6.211663	78.063241	339.317787	42.614980	336.820947	11.537806	23.750442
std	6.527286	203.542157	180.670077	87.585243	121.174519	6.064932	8.808602
min	1.129600	1.000000	20.200000	2.600000	0.320000	1.730000	6.300000
25%	2.425900	4.000000	254.000000	17.000000	370.415000	6.877500	18.500000
50%	3.917500	5.000000	307.000000	18.900000	390.885000	10.380000	21.950000
75%	6.341400	24.000000	403.000000	20.200000	395.630000	15.015000	26.600000
max	24.000000	666.000000	711.000000	396.900000	396.900000	34.410000	50.000000



همانطور که مشاهده می شود ستون CHAS مقادیرش از همه کمتر است دلیل این موضوع این است که بر اساس توضیحات ارائه شده همراه فایل می توان دید که این ستون یک ویژگی باینری است. مشاهده می کنید که بعضی از ستون ها تقریباً برای هر خانه مقدار منحصر به فرد دارند به عنوان مثال، CRIM, RM, AGE, DIS, B, LSTAT این وضعیت را دارند. این موضوع نشان دهنده این است که این ستون ها از نوع عددی هستند و در مقابل ستون های ZN, CHAS, RAD به احتمال بالای متغیر های دسته ای هستند.

## بررسی همبستگی بین ویژگی ها

همبستگی یا کورولیشن همانطور که از نامشان پیداست نشان دهنده میزان وابستگی ۲ ویژگی به یکدیگر هستند البته باید توجه کرد که این تابع فقط وابستگی خط را نشان می دهد. هیت مپ همبستگی در واقع با محاسبه همبستگی بین هر جفت داده و نمایش آنها در یک هیت مپ به ما کمک می کند که درک خوبی از روابط به ستون ها در داده هایمان پیدا کنیم. هیت مپ همبستگی ویژگی ها را رسم می کنیم.

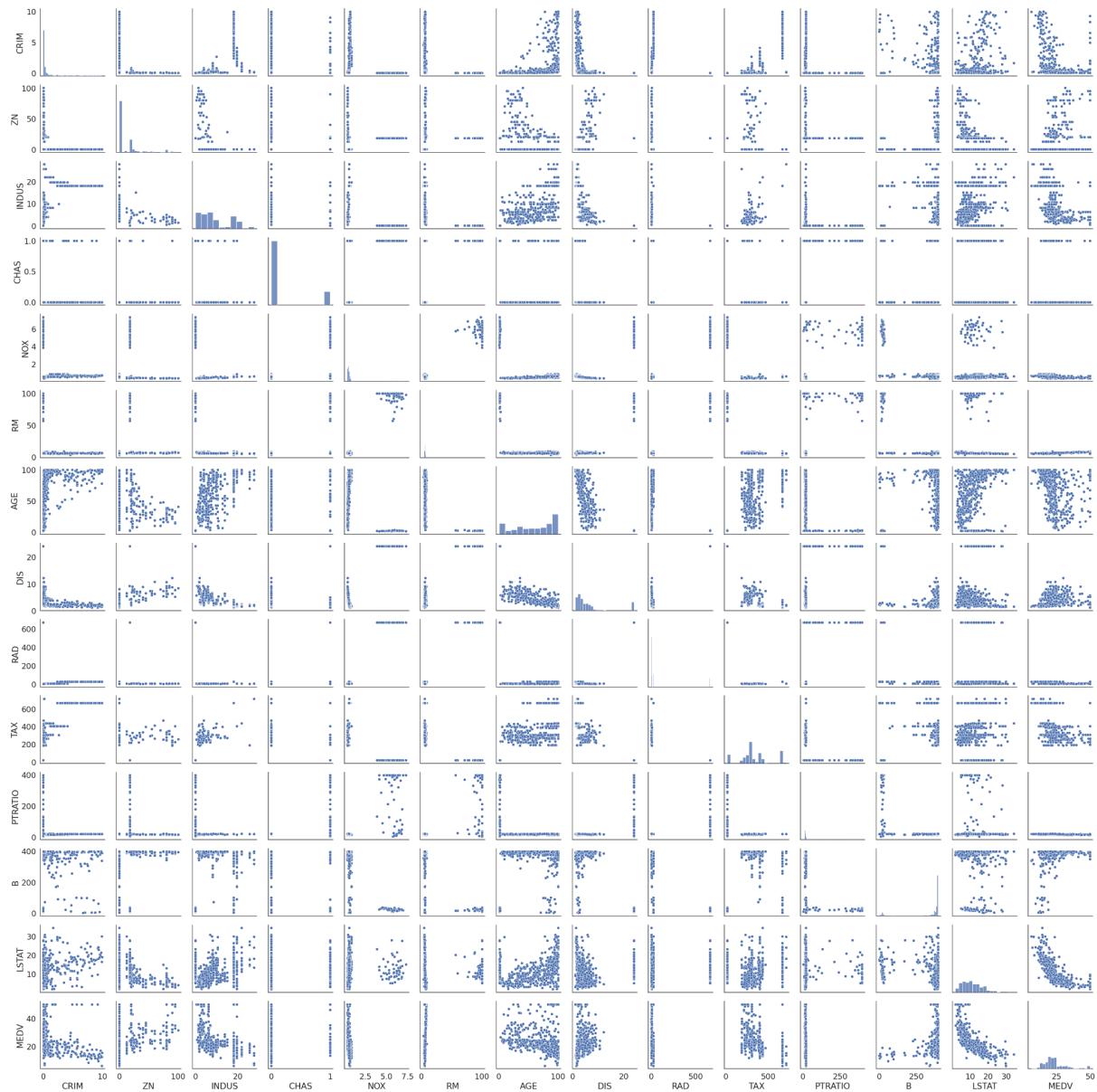


مشاهده می شود که بعضی موارد وابستگی زیادی دارد و در ادامه می شود بعضی موارد که همبستگی تقریباً ۱ دارند را حذف کرد.

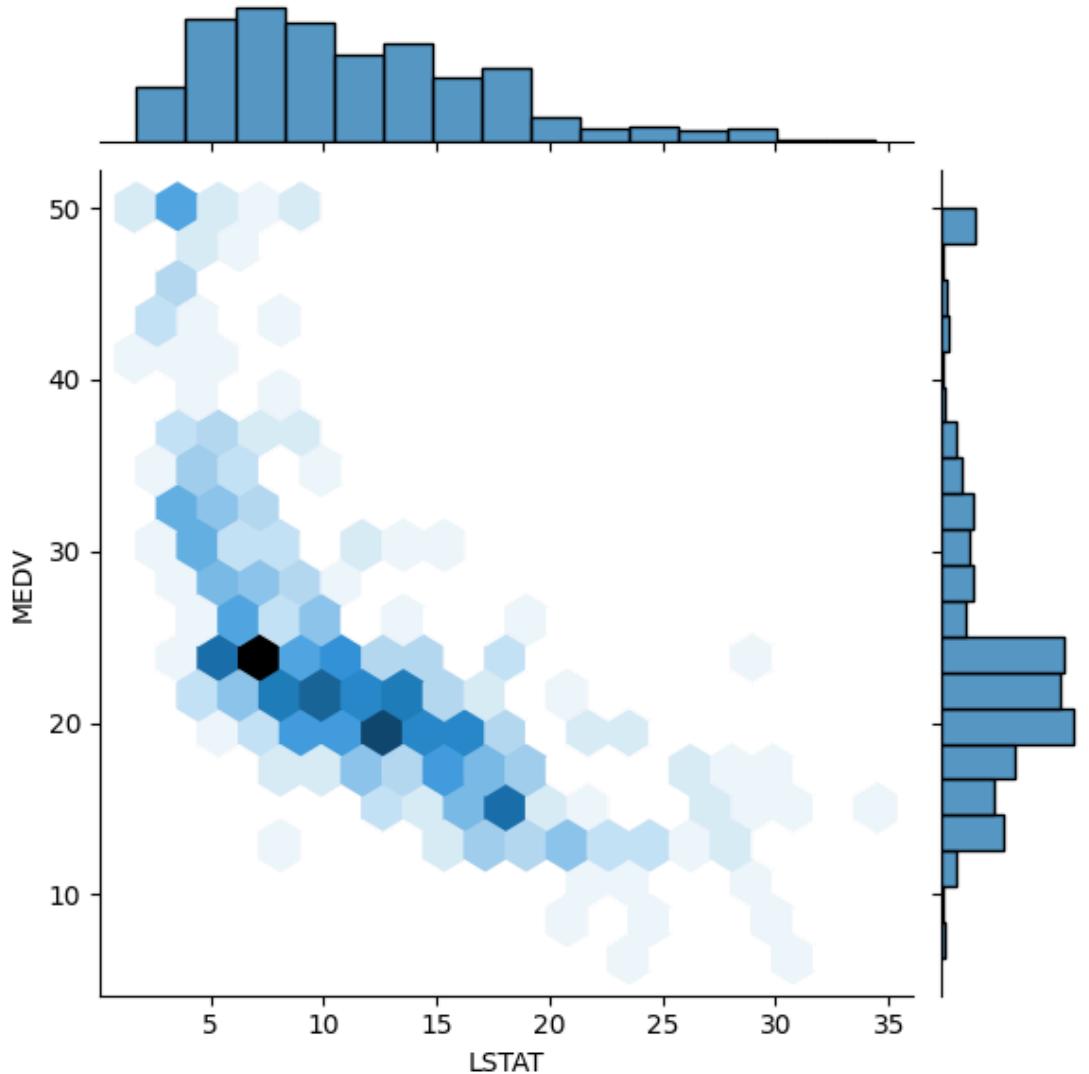
حال به ستون هدف نگاه می کنیم مشاهده می شود که این ستون با RM که تعداد اتاق های هر خانه است همبستگی ۷۵ درصدی دارد دلیل این موضوع نیز واضح است با افزایش تعداد اتاق های خانه قیمت خانه افزایش پیدا می کند. همچنین با LSTAT که وضعیت پایین در جمعیت را نشان می دهد همانطور که انتظار می روند همبستگی منفی قوی ای دارد. رابطه آن با PTRATIO, INDUS جای بررسی دارد.

## نمودار های hexbin و scatter

نمودار scatter برای نمایش داده های استفاده می شود به این صورت که در حالت ۲ بعدی ویژگی انتخاب می شود و به ازای هر داده یک نقطه روی نمودار نمایش داده می شود. این نمودار در درک رابطه بین دو ویژگی، کشف داده ها پرت، روند داده ها، همبستگی مثبت یا منفی بودن ویژگی ها، میزان همبستگی آنها، خطی بودن یا نبودن همبستگی به ما کمک می کند. می توان این نمودار را برای هر دو ویژگی رسم کرد.

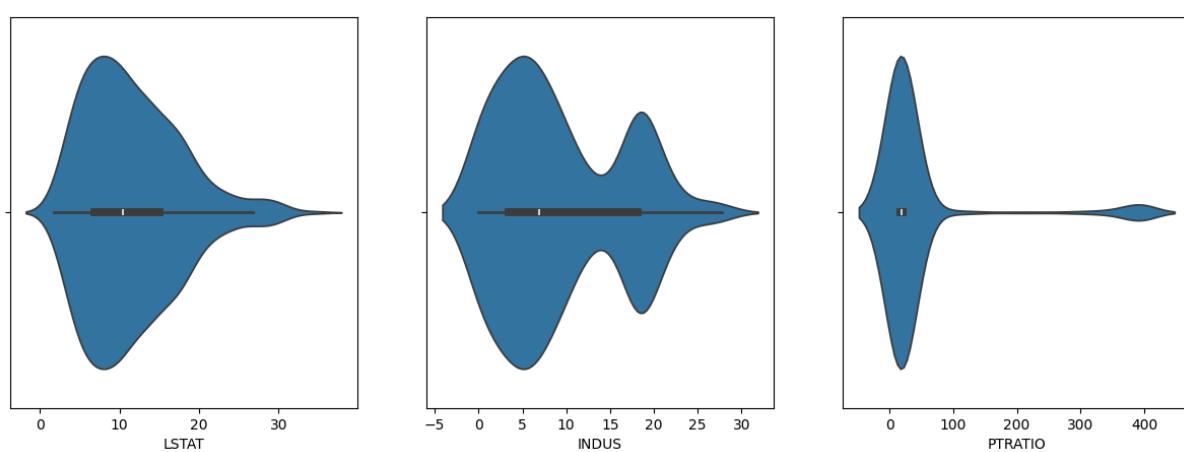


مثلا از نمودار های بالا می توان متوجه شد که رابطه ستون هدف و RM غیر خطی و از نوع توانی منفی است. نمودار hexbin در واقع جدول را به خانه های شش ضلعی تقسیم می کنه و خانه ها را بر اساس تعداد داده هایی که در آن قرار دارند تنظیم رنگ می کنند این موضوع در داده هایی با حجم بالا که نقاط scatter روی هم می افتد کاربردی است زیرا درک بهتری از تراکم به ما می دهد. نمودار زیر برای RM و ستون هدف است.



## سایر بررسی

می توان برای داده ها نمودار ویالون را رسم کرد که نشان دهنده توزیع داده ها است و در خوبی از داده برای ما ایجاد می کند.



همانطور که مشاهده می شود نمودار LSTAT چولگی به راست دارد. همچنین نمودار INDUS بایمودال است و ۲ قله دارد. نمودار PTRATIO هم به ما می گوید که این داده اسکیل درستی ندارد و باید درست شود. همچنین می شود پیروی داده ها از توزیع نرمال را نیز بررسی کرد. مورد دیگری که به ذهن می آید این است که از آنجا که منبع داده های مشخص نیست، قانون بنفورد را برای داده ها بررسی کنیم و به صحت آنها بیشتر مطمئن شویم. رسم نمودار هیستوگرام نیز می تواند درک خوبی از هر ستون به ما بدهد.

## پیش پردازش داده ها

### پر کردن داده های از دست رفته

به این موضوع در بخش کتبی اشاره شد. برای تکمیل بودن گزارش از توضیحات همان بخش استفاده می کنیم.

بهتر است اول با انواع داده های از دست رفته آشنا شویم.



- **Missing Completely At Random - MCAR**: در این حالت احتمال از دست رفتن داده برای همه داده ها یکسان است و ارتباطی با گروه یا موارد موثری بر تحلیل ندارد و الگویی ندارد در واقع این موضوع کاملاً تصادفی است. این مشکل به می تواند به دلیل خطای سیستم یا خطای انسانی. مثلًا در یک کتابخانه کتابدار فراموش می کند زمان تحویل کتاب را در سامانه ثبت کند.
- **Missing At Random - MAR**: در این حالت ما می توانیم الگویی برای داده های از دست رفته ارائه کنیم. مثلًا فرض کنید به ما مجموعه داده ای داده اند که حاوی سن و جنسیت است در این داده افراد به جنسیت خود پاسخ داده اند اما برخی از زنان به سن خود پاسخ نداده اند. در این حالت در صورت تخمین یا حذف داده های می تواند باعث شود نتیجه ی ما اریب شود. ما می توانیم در آخر و بعد تحلیل خود تعداد این موارد و رابطه آنها را توضیح دهیم.
- **Missing Not At Random - MNAR**: در این حالت عدم وجود داده های تصادفی نیست. یعنی گروه خاصی از جامعه به سوال خاصی پاسخ نداده اند یا داده ای ندارم از آنها و ارتباطی هم بین سایر

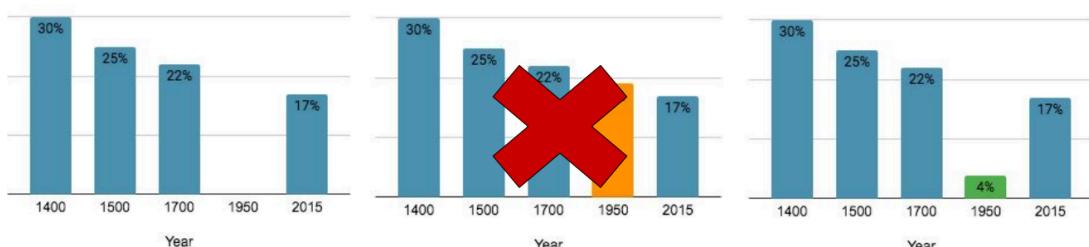
موارد با آن نمی توانیم پیدا کنیم. مثلا در پرسشنامه افراد فقیر از پاسخ به سوال مربوط به درآمد خودداری می کنند.

می توان از روش های زیر برای حل این مشکل استفاده کرد البته که این موارد بیشتر برای نوع MCAR و سپس MAR جواب می دهد و برای MNAR تنها می باید بعد ارائه مدل خود این موضوع را مذکور شویم:

1. **حذف ردیف ها دارای داده خالی:** راحت ترین کار این است که ما ردیف های حاوی داده خالی را حذف کنیم این کار در بعضی مواقع مناسب است اما مشکلاتی دارد. مثلا می تواند باعث شود که ما حجم بسیاری از داده های خود را از دست بدهیم و این موضوع باعث می شود داده کمی برای تعلیم مدل خود داشته باشیم. مشکل دیگر این است که خالی بودن بعضی ویژگی ها برای بعضی از ردیف های می تواند به دلیل وجود ویژگی خاصی در بین آنها باشد و در واقع آنها نماینده بخش خاصی از جامعه باشد که با حذف داده ما آن ها را از دست می دهیم و این موضوع می تواند باعث کاهش دقت مدل شود. به عنوان مثال فرض کنید ما می خواهیم با مدل افراد را بر اساس میزان درآمد خود رفتار شناسی کنیم در این داده های ممکن است مثلا افراد فقیر به سوال هایی مانند میزان خوشحالی خانواده خود جواب ندهد و ما با حذف این ردیف های عملابخش فقیر را از داده های خود حذف کرده ایم.

2. **جایگزینی با میانگین یا میانه:** در این روش که این روش نیز نسبتا راحت است ما داده های خالی را با میانگین یا میانه و یا هر آماره ای که نشان دهنده متوسط جامعه است جایگزین می کنیم و با این کار تاثیر مقدار جایگذاری را در مدل کم می کنیم اما این موضوع می نمی تواند در همه جا مناسب باشد.

3. **تخمین آنها با استفاده از ردیف هایی که در باقی موارد مشابه هستند:** در این روش ما با الگویی برای آن ردیف پیدا می کنیم و داده های از دست رفته آن را با استفاده از آن تقریب می زنیم البته این روش در همه جا مناسب نیست. به عنوان مثال تصویر زیر که از اسلاید های درس علوم داده گرفته شده به خوبی نشان دهد مشکل این روش است:



China's Share of Worldwide GDP  
<https://www.businessinsider.com/history-of-chinese-economy-1200-2017-2017-1>

روش های مختلفی برای این کار وجود دارد که در ادامه به برخی از آنها اشاره می کنیم:

a. **استفاده از مدل های یادگیری عمیق:** با این روش می توانیم با در نظر گرفتن روابط پیچیده بین داده ها مقدار داده را تخمین بزنیم ولی نکته قابل توجه این است که نتیجه حاصل قابل تفسیر و توجیه نیست.

- b. استفاده از مدل های دسته بندی و رگرسیون: می توانیم با انتخاب یکسری ویژگی هایی با همبستگی بالا و استفاده از رگرسیون یا دسته بندی بین ویژگی هدف و ویژگی با همبستگی با مقدار آن خانه را تقریب بزنیم.
- c. استفاده از KNN: استفاده از این روش باعث می شود که داده های کنار هم را پیدا کنیم و با استفاده از آنها این مقدار را تقریب بزنیم.
4. استفاده از داده ها بدون ایجاد تغییر: در این حالت ما می توانیم از مدل هایی استفاده کنیم که به داده های خالی حساس نباشد مانند درخت تصمیم این روش برای MNAR ها احتمالا مناسب تر هستند.

در نهایت با توجه به توضیحات بالا و خواسته سوال تصمیم گرفتیم از روش پر کردن با آماره های میانه، مد و میانگین و همچنین روش KNN استفاده کنیم. برای این کار کلاس زیر را به این صورت پیاده سازی می کنیم که در هنگام ساخته شدن دیتا فریم و تعداد همسایه های KNN را دریافت کند و سپس با فراخوانی متود fit و تعیین روش مدنظر دیتا فریم را طبق توضیحات کتابخانه sklearn پر کند و برگرداند همچنین دیتا فریم در یک متغیر ذخیره می شود که در دفعات بعد صرفا از آن استفاده کنیم.

```
class Imputer:
    def __init__(self, df, n_neighbors=5):
        self.df = df
        self.n_neighbors = n_neighbors
        self.imputer = pd.DataFrame()
        self.df_filled = df
    def fit(self, method) -> pd.DataFrame:
        if method == 'knn':
            self.imputer = KNNImputer(n_neighbors=self.n_neighbors)
        else:
            self.imputer = SimpleImputer(strategy=method)
        out = self.imputer.fit_transform(df)
        self.df_filled = pd.DataFrame(out, columns=df.columns)
        return self.df_filled
```

نتیجه استفاده از هر کدام از موارد در کد آمده و از ذکر آن در اینجا به علت طولانی شدن خودداری می کنیم.

## حذف ستون ها

در حالت کلی وقتی ستون هایی را حذف می کنند که یا ارتباطی به متغیر حذف ما نداشته باشد یا داده انحصاری و بی اهمیت مثل نام باشد یا با یک متغیر دیگر یکسان باشد مثلا همبستگی بالای ۹۵ درصد داشته باشد زیرا عملا ستون دیگر همان اطلاعات را به ما می دهد. در حالت کلی در این شرایط می توان از راه های زیر استفاده کرد:

- نگهداشتن فقط یکی از آنها: راحت ترین راه این است که ما تنها از یکی از آنها استفاده کنیم

2. استفاده از ابعاد اصلی (PCA): در این روش ما ویژگی‌هایی که همبستگی به یکدیگر دارند را کاهش می‌دهیم.

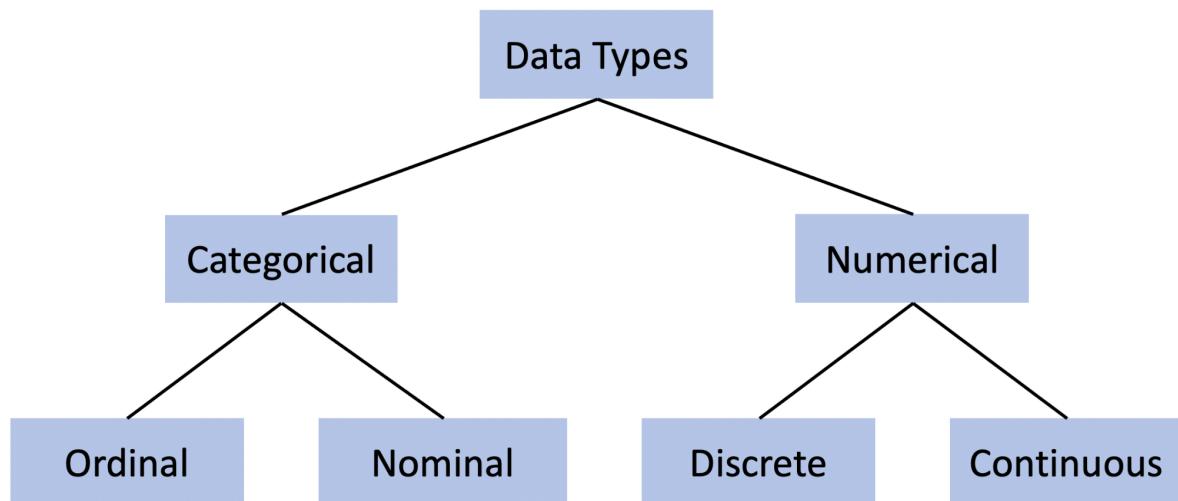
3. داده‌ها را تغییر ندهیم: در بعضی موارد اهمیت آن دسته از ویژگی‌ها برای ما زیاد است و می‌خواهیم مدل به آنها توجه بیشتری بکند در این حالت ما تغییری در آنها ایجاد نمی‌کنیم.

4. استفاده از مدل‌هایی که استقلال ویژگی‌ها شرط آنها نیست: برخی مدل‌ها به شرط استقلال ویژگی‌ها توسعه داده شده‌اند. به عنوان مثال مدل naive bayes شرط استقلال دارد پس استفاده از آن در این نوع داده‌ها ریسک بالا و احتمالاً خطای بالایی داشته باشد.

5. استفاده از Random Forests و Bagging: در این روش ما مدل‌هایی با ویژگی‌های متفاوت آموزش می‌دهیم در آخر بین آنها رای گیری می‌کنیم. در این روش ما ویژگی‌های داری همبستگی را در مدل‌های مختلف قرار می‌دهیم.

در اینجا به نظر می‌آید که ستون RAD با DIS همبستگی بالای ۹۵ درصد دارد بنابراین آن ۳ ستون را حذف می‌کنیم.

## انواع داده‌ها



به صورت کلی داده‌ها به ۲ درسته عددی و دسته‌ای تقسیم می‌شوند. همانطور که از نام داده‌های عددی پیداست این نوع داده‌های عدد هستند مثلاً قیمت خانه یا تعداد سیب‌های درون یک سبد را متغیر عددی می‌گویند اولی عددی پیوسته و دومی عددی گسسته است. داده‌های دسته به داده‌هایی می‌گویند از تعدادی دسته تشکیل شده‌اند مثلاً زن یا مرد بودن دسته‌ای غیر ترتیبی است و مثلاً متوسط، خوب، خیلی خوب داده دسته‌ای ترتیبی است. ما برای پیش‌بینی دسته هر داده از دسته بندی استفاده می‌کنیم و همچنین برای پیش‌بینی مقدار هر داده عددی از رگرسیون استفاده می‌شود. همانطور که قبل از بخش تعداد داده‌های غیر یکسان توضیح دادیم به نظر می‌آید ZN, CHAS, RAD داده‌های دسته‌ای هستند. حال مقادیر آنها را چاپ می‌کنیم.

```

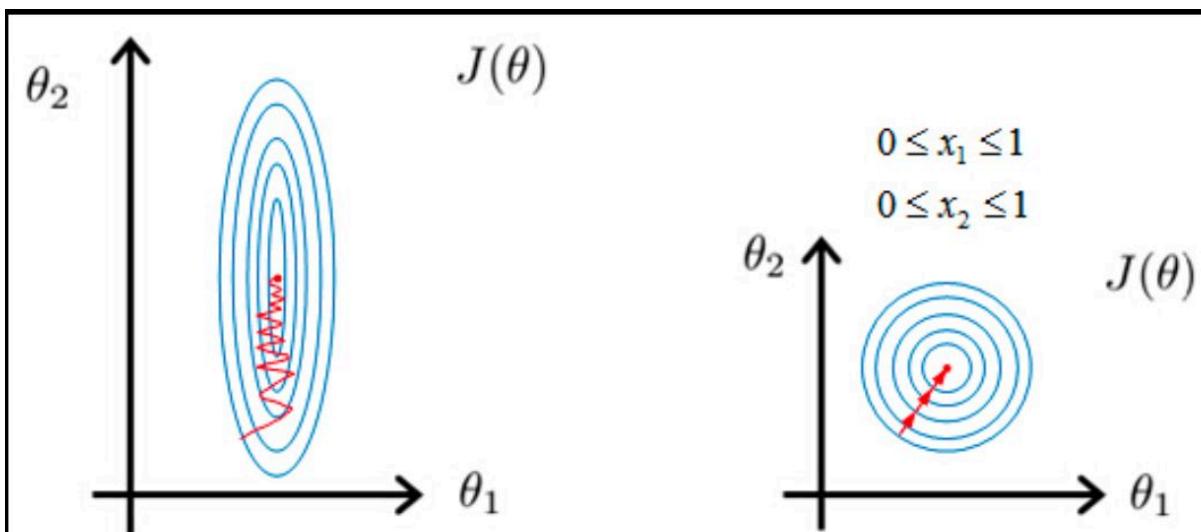
ZN [ 18.     0.    12.5   75.    21.    90.    85.   100.    25.   17.5   80.
28.
45.    60.    95.   82.5   30.    22.    20.    40.    55.    52.5   70.    34.
33.    35.   18.1]
CHAS [ 0.  nan  1.]
RAD [  1   2   3   5   4   8   6   7  24  666]

```

به نظر می آید که با توجه به مقادیر ZN متغیر عددی است. متغیر CHAS واضح دسته ای است. ولی برای RAD به علت مقدار عجیب ۶۶۶ کمی تردید وجود دارد ولی به نظر می آید که بشود دسته ای در نظر گرفت. باقی ستون ها همه عددی هستند.

## نرمال و استاندارد کردن داده ها

نرمال سازی داده های به ما کمک می کند که مقیاس داده هایمان را یکسان و در بازه منفی ۱ تا ۱ قرار دهیم. این موضوع در مدل های یادگیری ماشین باعث می شود که همگرایی مدل سریع تر و همچنین بازدهی آن افزایش پیدا کند. همچنین به دلیل یکسان شدن مقیاس ها نمودار ها قابلیت تفسیر بیشتر پیدا می کند و این موضوع باعث می شود بتوانیم تحلیل های آماری بهتری انجام دهیم. استاندارد کردن نیز کاری مشابه انجام می دهد منتها دیگر داده در بازه منفی یک و یک محدود نیست و در عوض میانگین داده ها را صفر می کند و واریانس را ۱ و این موضوع به خصوص برای تطبیق داده روی توزیع نرمال مفید است. به تفاوت کارکرد گرادیان کاهشی بعد و قبل از نرمال کردن توجه کنید.



نرمال کرد به صورت زیر انجام می شود:

$$result = \frac{data - min}{max - min}$$

استاندارد کردن به صورت زیر:

$$result = \frac{data - mean}{std}$$

در این پروژه استفاده از این ابزارها مفید خواهد بود این موضوع قبلاً و در زمان رسم نمودار ویالونی نیز به دلیل فشردگی نمودار اشاره شده بود. کلاس استاندارد یا نرمال کردن به شکل زیر است به دلیل سادگی از توضیح آن صرف نظر می‌کنیم.

```
class DataScaler:
    def __init__(self, df):
        self.df = df
        self.scaler = StandardScaler()
        self.norm = MinMaxScaler()
        self.df_standardized = df.copy()

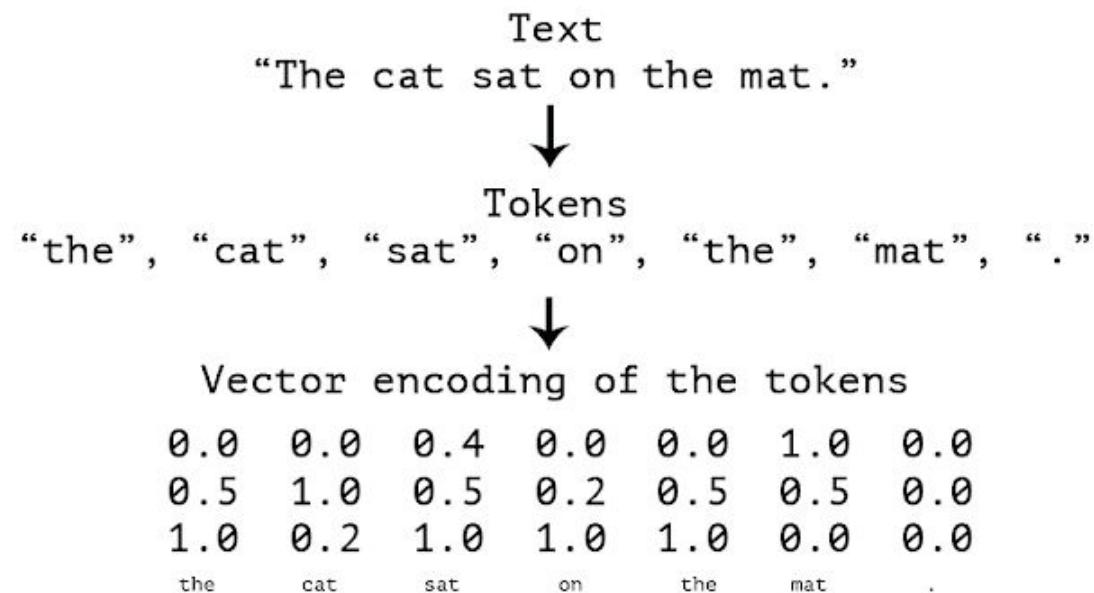
    def standardize(self) -> pd.DataFrame:
        self.df_standardized[self.df.columns] =
        self.scaler.fit_transform(self.df)
        return self.df_standardized

    def normalizing(self) -> pd.DataFrame:
        self.df_standardized[self.df.columns] =
        self.norm.fit_transform(self.df)
        return self.df_standardized
```

## پیش پردازش داده‌های دسته‌ای

برای پیش پردازش داده‌ها دسته‌ای روش‌های مختلفی وجود دارد که در ادامه به آنها اشاره می‌کنیم:

- **تکنایز کردن متن‌ها:** برای پردازش داده‌های متنی می‌توانیم آنها را تکنایز کنیم و سپس به عنوان یک لیست از آن استفاده کنیم. می‌شود از آن bag of word نیز درست کرد که برای تحلیل‌های مفید هستند. همچنین به روش آنها را رمز کنیم که مدل بتواند روابط آنها را متوجه بشود.



- **نسبت دادن یک عدد به هر دسته:** این موضوع به خصوص در داده‌های دسته‌ای ترتیبی خوب

عمل می کند ما می توانیم داده ها را کدگذاری کنیم تصویر زیر این موضوع را نشان می دهد.

The diagram illustrates the process of One-Hot-Encoding. On the left, a table shows raw data with columns: Make, Model, Year, Engine Fuel Type, Engine HP, and Engine Cylinders. An arrow points to the right, where the same data is shown in a transformed format. The transformed table includes the original columns plus additional columns for each categorical variable, where each category is represented by a binary value (0 or 1) indicating its presence in the row. For example, 'Engine Fuel Type' is expanded into 'premium' and 'unleaded (required)' columns, both of which are 1 for the first row and 0 for the others.

	Make	Model	Year	Engine Fuel Type	Engine HP	Engine Cylinders
0	BMW	Series M	2011	premium unleaded (required)	335.0	6.0
1	BMW	Series 1	2011	premium unleaded (required)	300.0	6.0
2	BMW	Series 1	2011	premium unleaded (required)	300.0	6.0
3	BMW	Series 1	2011	premium unleaded (required)	230.0	6.0
4	BMW	Series 1	2011	premium unleaded (required)	230.0	6.0

	Make Encoded	Model Encoded	Year	Engine HP	Engine Cylinders
0	4	1	2011	335.0	6.0
1	4	0	2011	300.0	6.0
2	4	0	2011	300.0	6.0
3	4	0	2011	230.0	6.0
4	4	0	2011	230.0	6.0

روش دیگری که وجود دارد این است که ما ستون دسته ای خود را به تعداد دسته ها بالا ببریم و هر ستون نشان دهنده این باشد که داده در آن دسته هست یا نه و این موضوع به صورت ۰ و ۱ نمایش داده شود.

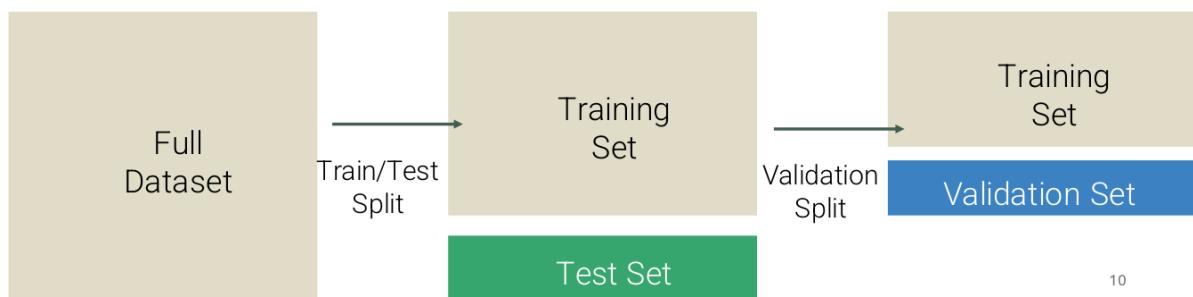
This diagram shows the transformation of a dataset for the 'Transmission Type' column. On the left, a table has columns: Engine Cylinders and Transmission Type. The 'Transmission Type' column contains repeated values 'MANUAL'. An arrow points to the right, where the data is transformed into a binary matrix. The transformed table includes the original columns plus additional columns for each transmission type: AUTOMATED\_MANUAL, AUTOMATIC, DIRECT\_DRIVE, MANUAL, and UNKNOWN. The 'MANUAL' row in the original data is now represented by a vector of 1s in the 'MANUAL' and 'UNKNOWN' columns, while all other transmission types are represented by 0s in those columns.

	Engine Cylinders	Transmission Type
0	6.0	MANUAL
1	6.0	MANUAL
2	6.0	MANUAL
3	6.0	MANUAL
4	6.0	MANUAL

	Engine Cylinders	AUTOMATED_MANUAL	AUTOMATIC	DIRECT_DRIVE	MANUAL	UNKNOWN
0	6.0	0	0	0	1	0
1	6.0	0	0	0	1	0
2	6.0	0	0	0	1	0
3	6.0	0	0	0	1	0
4	6.0	0	0	0	1	0

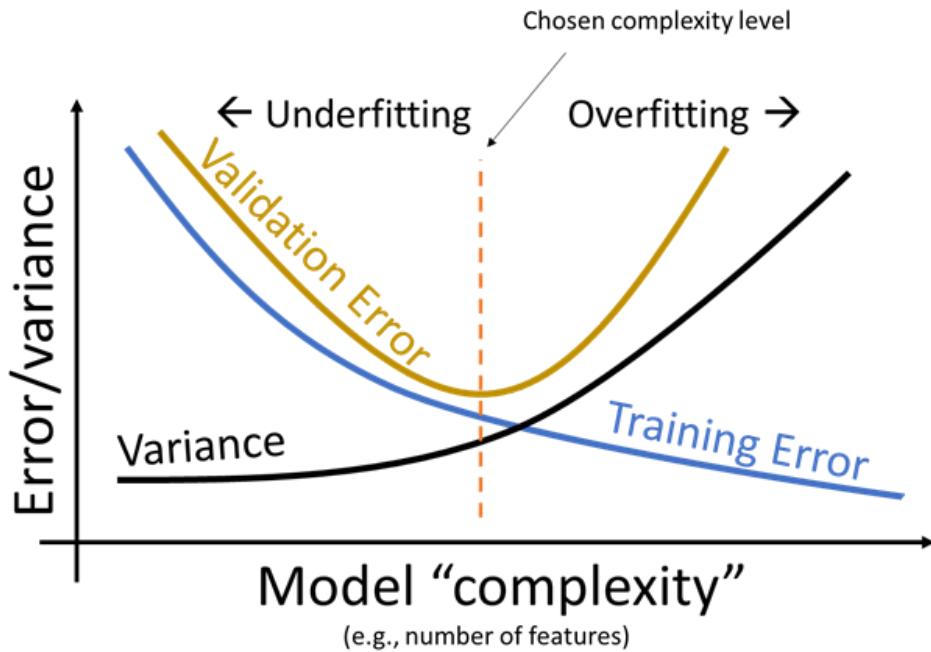
## تقسیم بندی داده ها برای آموزش مدل



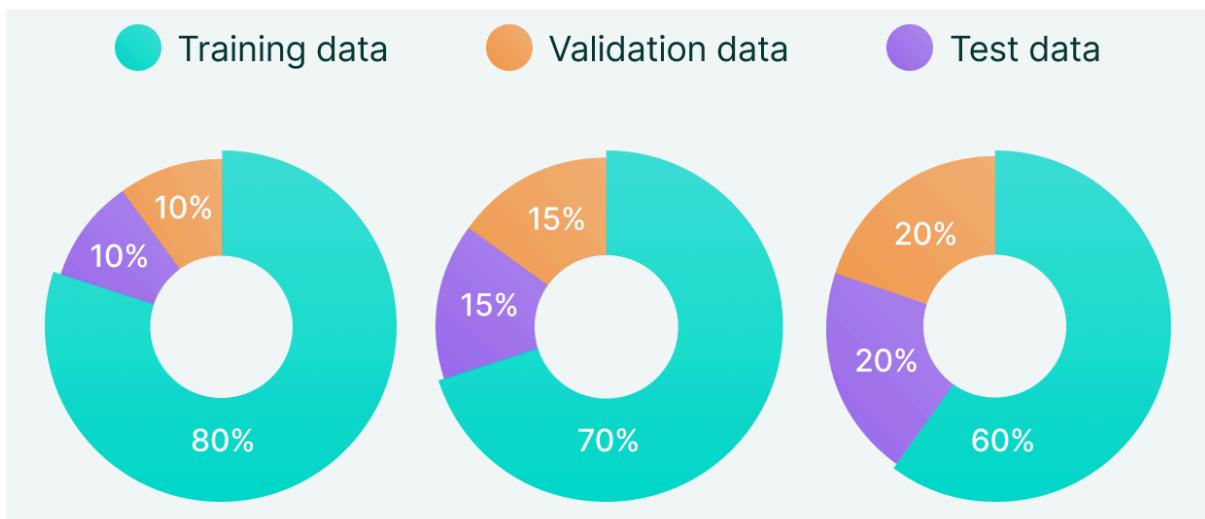
10

در پروژه های یادگیری ماشین ما لازم داریم که با تعدادی داده مدل خود را آموزش دهیم و سپس آن را ارزیابی کنیم پس ما باید داده هی خود را به ۲ قسمت تست و train تقسیم کنیم. مشکلی که وجود دارد این است که مدل ما اکثرا نیاز به تنظیم های پارامتر دارد برای این که ما هایپر پارامتر ها را تنظیم کنیم می توانیم از داده های تست استفاده کنیم اما مشکلی وجود دارد این است که در صورتی که این کار را انجام بدیم ممکن است مدل ما روی داده های تست فیت شود و دقت غیر واقعی دریافت کنیم برای همین بخشی از داده ها را مجدد جدا می کنیم و هایپر پارامتر ها اختصاص می دهیم به این بخش از داده ها داده های validation می گویند. ما این داده ها را برای سنجش کیفیت مدل در زمان طراحی آن استفاده می کنیم و

مدل را تا وقتی آموزش می دهیم یا هایپر پارامتر ها را جوری تعیین می کنیم که مقدار ارور آن مینیمم شود به نمودار زیر توجه کنید.



این که چه میزان از داده ها را به هر کدام اختصاص دهیم به حجم داده ها و نوع پروژه ما وابسته است به صورت کلی می توان از ترکیب های تصویر زیر استفاده کرد.



لازم به ذکر است که معمولاً قبل از تقسیم داده ها آنها را بُر می زند. در بعضی موارد تقسیم بندی خوشه ای نیز کاربرد دارد.

در این پروژه با توجه به این که حجم داده ها کم است به نظر می آید ترکیب ۱۰-۸۰-۱۰ مناسب باشد. حال کلاسی را پیاده سازی می کنیم که داده ها را جدا کند. ابتدا در هنگام تعریف کلاس دیتا فریم و ستون هدف و seed را دریافت می کند سپس ستون هدف را از جدول جدا می کند و آنها را ذخیره می کند. هنگام استفاده از متدهای جدادسازی ابتدا مجموع داده های تست و validation را جدا می کند و سپس به

نسبت گفته شده به ۲ قسمت test, validation تقسیم می کند. توابعی هم در رابطه با random seed قرار داده شده.

```
class DataSplitter:
    def __init__(self, df, target_var, random_state=4):
        self.X = df.drop(target_var, axis=1)
        self.y = df[target_var]
        self.random_state = random_state

    def split(self, test_size=0.1, validation_size=0.1):
        X_train, X_test, y_train, y_test = train_test_split(self.X, self.y,
                                                            test_size=test_size + validation_size,
                                                            random_state=self.random_state)

        X_val, X_test, y_val, y_test = train_test_split(X_test, y_test,
                                                        test_size=test_size / (test_size + validation_size),
                                                        random_state=self.random_state)

        return {
            'X': {'train': X_train, 'val': X_val, 'test': X_test},
            'y': {'train': y_train, 'val': y_val, 'test': y_test}
        }

    def get_random_state(self):
        return self.random_state

    def change_random_state(self):
        self.random_state += 1
        return self.random_state
```

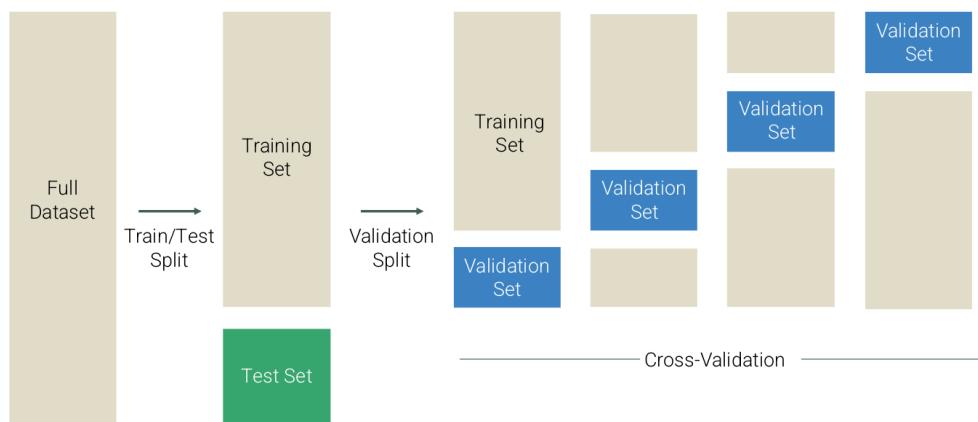
## سایر روش های پیش پردازش

1. **بررسی دقیق دلیل وجود نویز:** در بعضی مواد ممکن است داده های نویز نباشد و داده های پرت باشند در این موارد باید بررسی کرد دلیل این موضوع را پیدا کرد و سپس برای آنها اقدام کرد.
2. **حذف داده های نویز:** راحت ترین کار حذف داده های نویز است در صورتی که این موارد زیاد نیستند می توان آنها برای حذف کرد که باعث انحراف مدل نشوند اما اگر داده های نویز زیاد هستند می تواند حذف آنها منجر به این شوند که مدل ما نتواند با داده هایی به اندازه مناسب آموزش ببیند.
3. **بازتولید داده ها(Autoencoders):** می توان با استفاده از مدل های شبکه عصبی داده ها را بازتولید کرد و نویز آنها را کاهش داد این موضوع به خصوص در پردازش تصویر و آموزش مدل های تصویری می تواند کارآمد باشد.
4. **تبديل داده نویز به مقادیر قابل قبول:** در بعضی موارد تنها قالب داده متفاوت است و ما می توانیم به آسانی آنها را به قالب درست تبدیل کنیم به عنوان مثال در بخشی از داده های ما تاریخ به صورت میلادی و در باقی موارد شمسی است به آسانی می توان تاریخ های میلادی را به شمسی تبدیل کرد و از آنها استفاده کرد.

5. نگهدارشتن و افزایش داده های نویز: داده های نویز همیشه بد نیستند و این موضوع باعث می شود مدل روی داده های آموزشی فیت نشوند و فقط الگوی داده ها را یاد بگیرد در بعضی موارد که داده های ما کم هستند با استفاده از روش هایی از جمله ایجاد نویز داده های جدیدی بازتولید می کنیم که مدل بتواند با داده های بیشتر آموزش ببینید.

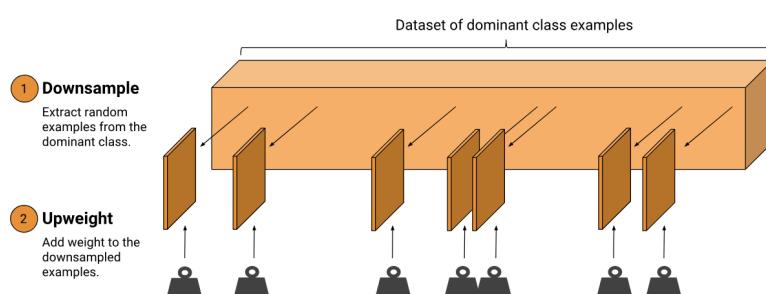
6. استفاده از ابعاد اصلی (PCA): با استفاده از این تکنیک می توان ویژگی ها را کاهش داد و ویژگی های نویز را حذف کرد. این روش واریانس داده ها را در هر بعد می سنجد و بر طبق آن ابعاد اصلی را انتخاب می کند.

7. **Cross Validation**: در این روش ما برای بررسی مدل خود داده های آموزشی را به قسمت هایی تقسیم می کنیم یکی را برای ارزیابی استفاده می کنیم و با مابقی آموزش می دهیم. این کار را به صورت چرخشی تکرار می کنیم.



8. متعادل سازی داده: برای این راه های مختلفی وجود دارد که به بعضی از آنها اشاره می کنیم:  
a. **تولید داده جدید**: برای این کار می توان مقداری به داده های قبلی نویز اضافه کرد یا آنها را مدل های عصبی بازتولید کرد به طوری که تعداد داده های دسته ها به یکدیگر نزدیک شوند.

b. **Upweight و DownSampling**: در این روش ما نسبت داده ها را تغییر می دهیم مثلا اگر در مثال سرطان ما ۰.۱ درصد نمونه سرطانی داریم آن را به عنوان مثال به ۵ درصد می رسانیم. برای این کار می توان از دسته کوچک تر نمونه تولید کرد یا از دسته بزرگ تر داده حذف کنیم. در آخر هم وزن داده های دسته بزرگ تر را زیاد می کنیم.

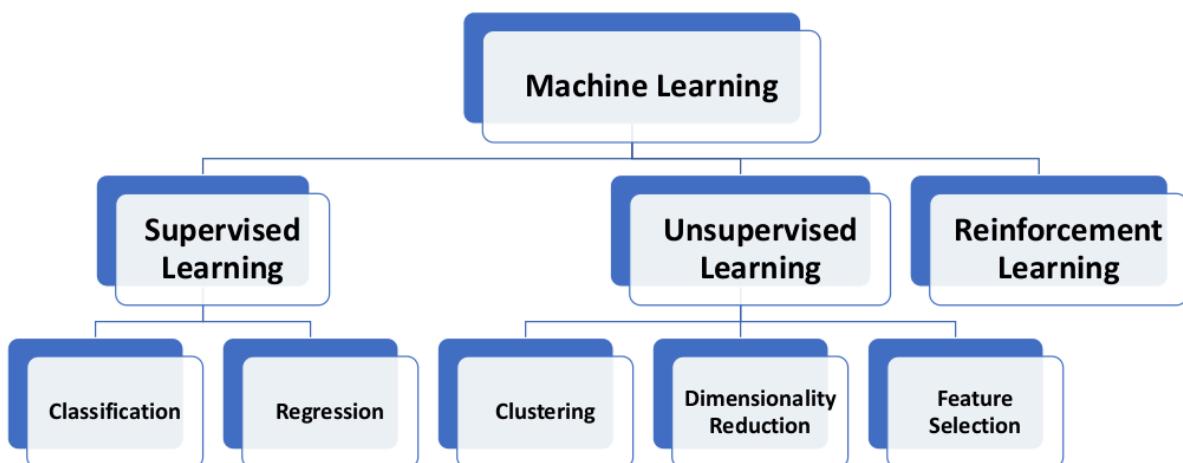




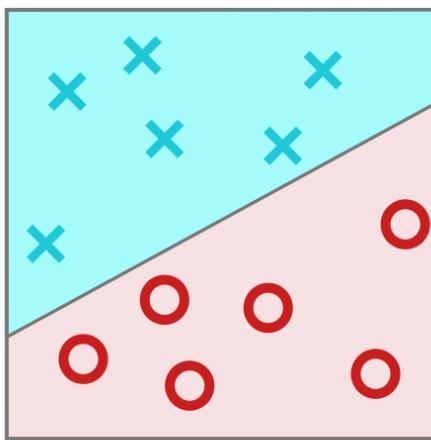
# آموزش، ارزیابی و تنظیم

## انواع مدل های یادگیری ماشین

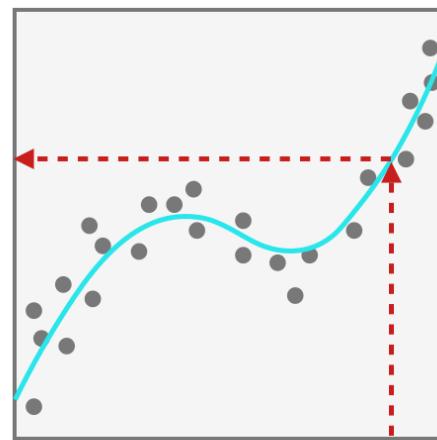
تقسیم بندی وظایف هر یک از انواع مدل های یادگیری ماشین:



## مدل های با ناظارت



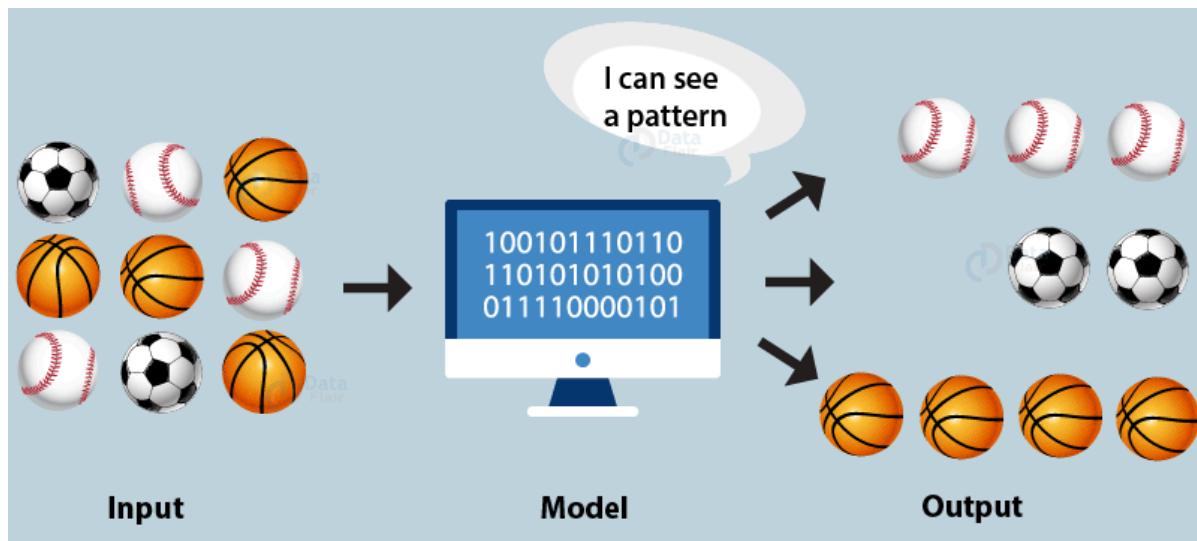
Classification



Regression

این مدل ها به این صورت عمل می کنند ما تعدادی داده برچسب خورده و یا تعدادی مسئله حل شده و با پاسخ به آن می دهم و می خواهیم که مدل با یادگیری الگوی آن بتواند مسائل بعدی را حل کند. اکثر کارهایی که این مدل ها انجام می دهند شامل تقسیم بندی داده دسته ای یا پیش بینی مقدار عددی در داده های عددی است. از کاربردهای آن میتوان به سیستم های احراز هویت، تشخیص محصول سالم و خراب و تحلیل تصاویر پزشکی نام برد.

## مدل های بدون ناظارت



برعکس روش یادگیری با ناظارت در این روش داده های ما برقسب خورده نیستند. در این مدل ها ما مجموعه داده ها را به مدل می دهیم و می خواهیم طبق الگو آنها را تقسیم بندی کند یا فیچر های جدیدی برای ما تولید کند. از این مدل ها می توان برای کاهش ابعاد و تعداد فیچر ها نیز استفاده کرد. از کاربرد های آن می توان به خوشه بندی متون در سایت های خبری یا نظرات محصولات و موارد مشابه استفاده کرد.

## مدل های تقویتی

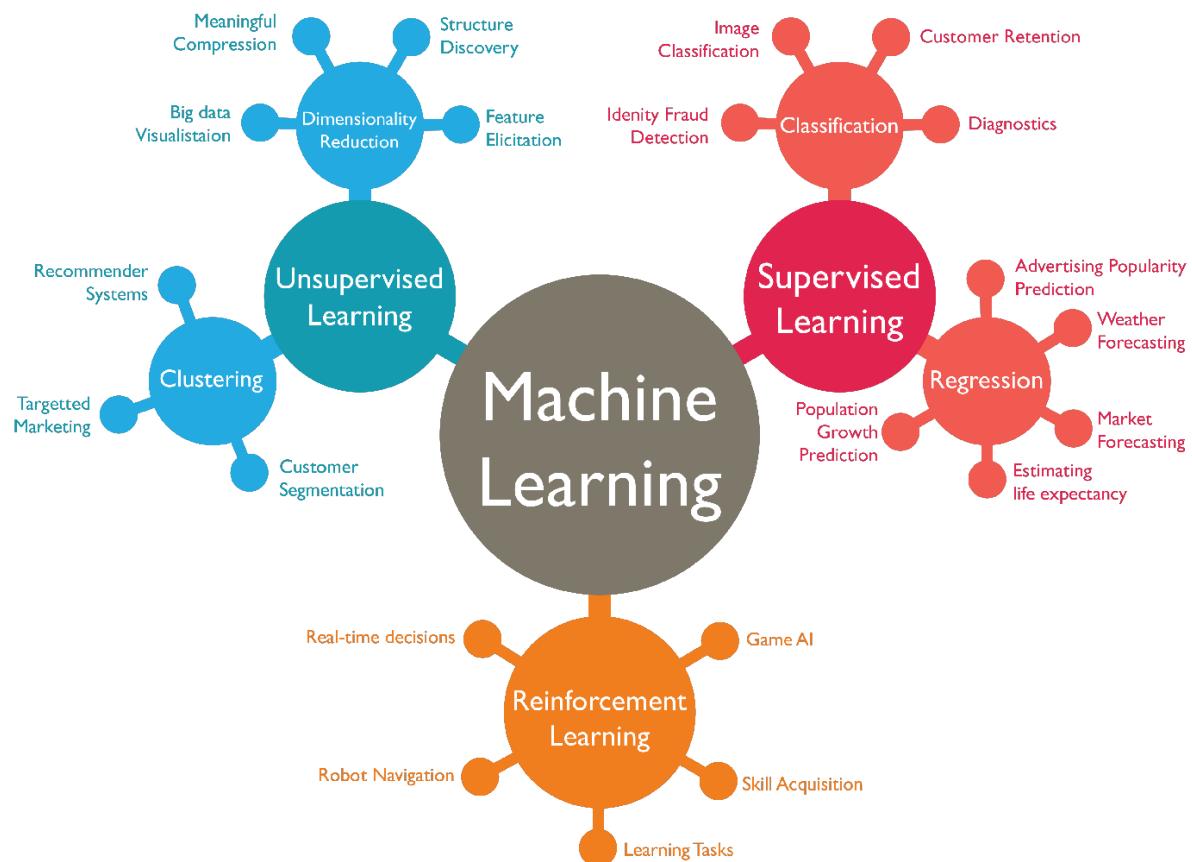


کارکرد این مدل ها به این صورت است که ما یکسری عملکرد برای مدل خود تعریف می کنیم و این مدل با تعامل محیط و با استفاده از تابع پاداشی که برای آن تعریف کردیم می تواند متوجه شود که چقدر عملکردش مطلوب است و آن را بهبود بخشد. این مدل ها می توانند در بازی های رایانه ای روابط ها مورد استفاده قرار گیرد.

## مدل های نیمه ناظارتی

در این نوع از مدل ها ما بخشی از داده ها را برچسب می زنیم و انتظار داریم مدل بتواند آموزش ببیند. این مدل ها مشابه مدل های ناظارتی هستند و ماموریت های مشابهی را می توانند انجام دهند.

در ادامه بعضی از کاربردهای هر نوع از این داده های آمده:



## رگرسیون

به مدل هایی که از یک یا چند متغیر عددی به پیش بینی یک یا چند متغیر عددی دیگر می پردازد رگرسیون می گویند این مدل ها در واقع سعی در پیدا کردن تابعی دارند که از روی متغیر های داده شده متغیر های هدف را بدست بیاورد. ما در این مسئله سعی در پیدا کردن خطی داریم که کمترین فاصله را از داده هایمان داشته باشد. در حالت کلی همانطور که توضیح داده شد مسائل مدل های یادگیری با نظارت به دو دسته رگرسیون و دسته بندی تقسیم می شود. در روش های تقسیم بندی با به پیش بینی داده های دسته ای می پردازیم در واقع در این روش سعی می کنیم مرز بین دسته ها را پیدا کنیم و با آن داده های بعدی را پیش بینی کنیم. اما در رگرسیون ما داده عددی را پیش بینی می کنیم.

### رگرسیون خطی

در این حالت ما سعی داریم رابطه بین متغیر ها را با استفاده از یک معادله خط نشان دهیم در واقع ما سعی در به دست آوردن پارامتر های مسئله زیر داریم.

$$y^{\wedge} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

In this equation:

- $\hat{y}$  is the predicted value.
- $n$  is the number of features.
- $x_i$  is the  $i^{\text{th}}$  feature value.
- $\theta_j$  is the  $j^{\text{th}}$  model parameter, including the bias term  $\theta_0$  and the feature weights  $\theta_1, \theta_2, \dots, \theta_n$ .

### پیاده سازی رگرسیون خطی مرتبه اول

روابط این مدل در صورت پروژه آمده و صرفا باید آن را پیاده سازی کنیم.

$$\hat{\alpha} = \frac{\sum(x_i - \hat{x})(y_i - \hat{y})}{\sum(x_i - \hat{x})^2}$$

معادل  $x_{\text{i\_x\_bar}} = \text{input} - \text{np.mean(input)}$  می باشد. بنابراین عبارت  $y_{\text{i\_y\_bar}} = \text{output} - \text{np.mean(output)}$  را می توان به صورت ضرب ماتریسی و به شکل زیر نوشت:

$$\alpha = (x_{\text{i\_x\_bar.T}} @ y_{\text{i\_y\_bar}}) / (x_{\text{i\_x\_bar.T}} @ x_{\text{i\_x\_bar}})$$

بنا هم پیاده سازی اش نکته ای ندارد و به صورت زیر است:

$$\beta = \hat{y} - \hat{\alpha}\hat{x}$$

```
beta = np.mean(output) - (alpha * np.mean(input))
```

تابع prediction مقدار پیش‌بینی شده را براساس ورودی حساب می‌کند. برای این کار صرفاً کافیست ورودی را در معادله خط خود قرار دهیم.

```
def get_regression_predictions(input, intercept, slope):
    predicted_values = intercept + slope * input
    return predicted_values
```

در صورت پروژه خواسته شده پیش‌بینی کنیم کدام ویژگی عملکرد بهتری دارد. با توجه به همبستگی منفی بالای LSTAT این متغیر انتخاب مناسبی برای پیش‌بینی باشد.

## روش‌های مختلف ارزیابی مدل

RSS

این معیار در واقع مجموع مربعات اختلاف‌ها را نشان می‌دهد این معیار به تنها‌ی معنای خاصی ندارند زیرا بازه‌ای نمی‌شود برای آن متصور شد ولی می‌شود برای مقایسه مدل‌ها استفاده شود.

$$RSS = \sum_{i=1}^n (\hat{y}_i - y_i)^2$$

MSE

در این مدل در واقع ما متوسط مربعات خطای داریم که برعکس معیار قبلی به تنها معنا دارد و می‌توان به تنها و هم در مقایسه از آن استفاده کرد.

$$MSE = \frac{1}{n} RSS$$

RMSE

این معیار جذر گرفته شده معیار قبلی است دلیل این موضوع این است که واحد معیار قبلی مربع مقادیر مسئله است ولی این معیار برابر واحد مسئله است این موضوع باعث می‌شود که درک بهتر در مقایسه داشته باشیم.

$$RMSE = \sqrt{MSE}$$

## R2 score

در واقع یک شاخص است که نشان میدهد تغییرات متغیر وابسته(خروجی) چقدر توسط متغیر ورودی (مستقل) توضیح داده میشوند.

R Squared همیشه بین صفر و یک است و هر چقدر نسبت total variance RSS به کمتر باشد نشان میدهد پراکندگی نسبت به پیشیبینی از پراکندگی نسبت به میانگین بسیار کمتر است و مدل خوبی فیت شده است.

R Squared به صورت زیر محاسبه میشود:

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \bar{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

هرچه R Squared به یک نزدیکتر باشد یعنی تغییرات متغیر وابسته توسط متغیر مستقل به خوبی پیشیبینی شده است.

حال یکی از بدی های این مدل اینست که هر چه predictor های بیشتری به مدل اضافه کنیم هر چند نتواند آن داده را به خوبی پیش بینی کند باعث افزایش مقدار R Squared می شود در حالی که Adj R Squared با اضافه شدن پیش بینی کننده جدید تنها زمانی افزایش می یابد که عبارت جدید مدل را بالاتر از آنچه که با احتمال به دست می آید افزایش دهد و زمانی کاهش می یابد که یک پیش بینی کننده مدل را کمتر از آنچه به طور تصادفی پیش بینی می شود، افزایش دهد.

## ارزیابی مدل رگرسیون خطی مرتبه ۱

برای تمامی ستون ها مدل را آموزش می دهیم و سپس مقایسه می کنیم.  
برای این کار گام های زیر را طی می کنیم:

1. آموزش مدل و به دست آوردن شبیه خط و عرض از مبدا

2. پیش بینی داده های تست

3. ارزیابی مدل با معیار های بخش قبل

4. رسم نمودار scatter برای مقایسه مقدار پیش بینی شده و واقعی

تکه کد زیر دقیقاً موارد بالا را برای هر ویژگی انجام می دهد:

```
designated_feature_list = splitted_data['X']['train'].columns

fig, axes = plt.subplots(2, len(designated_feature_list) // 2,
figsize=(20,10))

result = dict()
for i, feature in enumerate(designated_feature_list):
    alpha, beta = linear_regression(splitted_data['X']['train'][feature],
splitted_data['y']['train'])
    predicted_values =
```

```

get_regression_predictions splitted_data['X']['val'][feature], beta, alpha)

actual_values = splitted_data['y']['val']
result[feature] = {
    'MSE': get_mean_squared_error(predicted_values, actual_values),
    'RSS': get_residual_sum_squares(predicted_values, actual_values),
    'RMSE': get_root_mean_squared_error(predicted_values, actual_values),
    'R2 score': get_r2_score(predicted_values, actual_values)
}

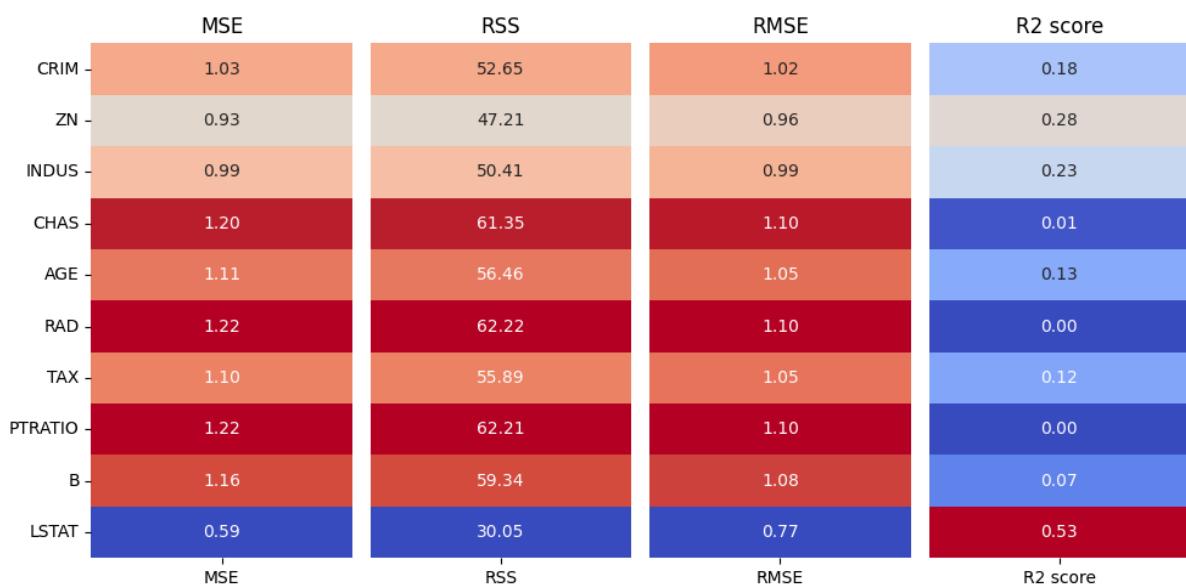
axes[i%2, i//2].plot([-2,2],[-2,2],linestyle='--', label='y=x')
axes[i%2, i//2].scatter(splitted_data['y']['val'],predicted_values)
axes[i%2, i//2].set_title(f'{feature}')

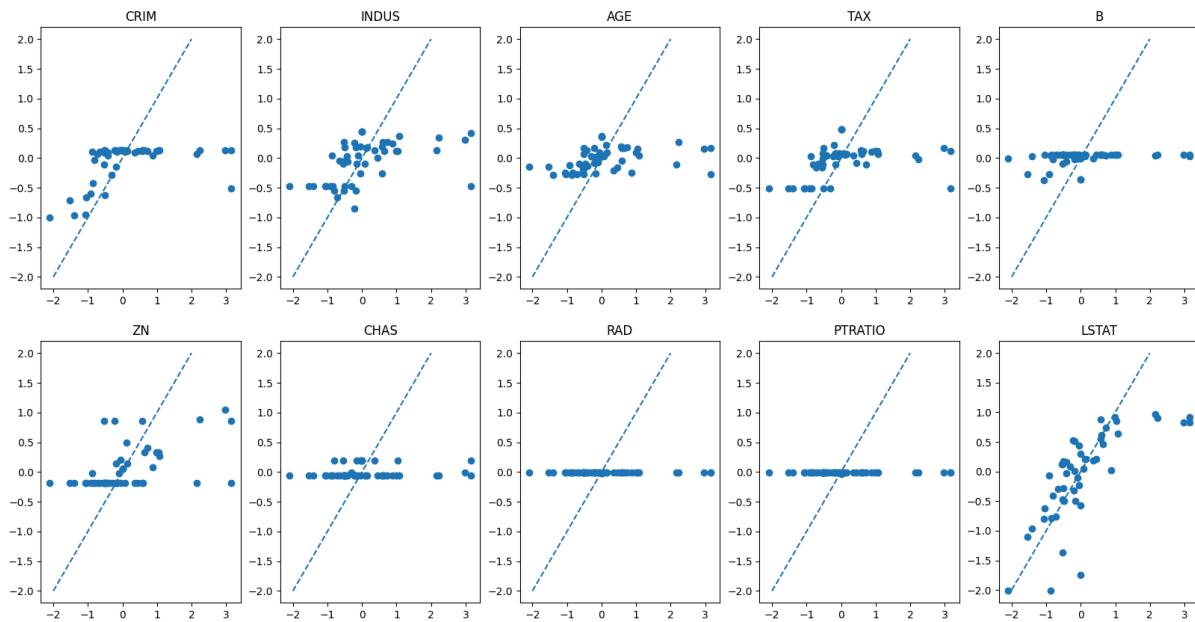
result_df = pd.DataFrame(result).transpose()
result_df

```

نتایج به شکل زیر است:

	MSE	RSS	RMSE	R2 score
CRIM	1.032271	52.645832	1.016007	0.178138
ZN	0.925628	47.207026	0.962096	0.276866
INDUS	0.988406	50.408723	0.994186	0.228057
CHAS	1.202898	61.347823	1.096767	0.014117
AGE	1.106987	56.456350	1.052135	0.131386
RAD	1.219945	62.217219	1.104511	0.001249
TAX	1.095908	55.891295	1.046856	0.119641
PTRATIO	1.219808	62.210199	1.104449	0.001078
B	1.163544	59.340763	1.078677	0.070819
LSTAT	0.589136	30.045943	0.767552	0.530622



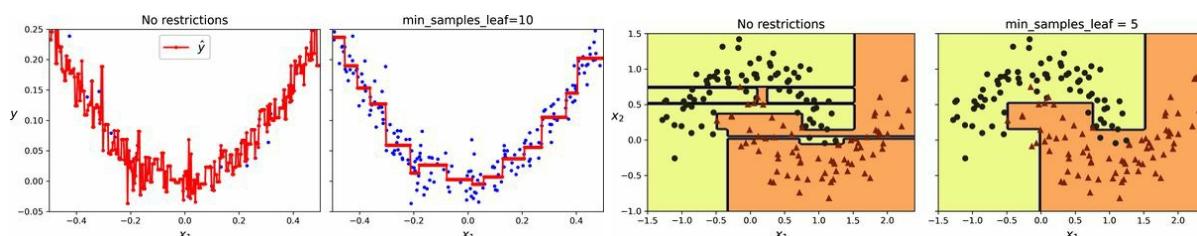


با نگاهی به جدول می توان متوجه شد که با اختلاف بسیار زیادی LSTAT بهترین پیش‌بینی را دارد و سپس ZN این موضوع را نمودارها نیز تایید می کنند. اما در کل نتیجه خوب نیست و دقت ۵۸٪ مطلوب نیست دلیل این موضوع نیز واضح است زیرا اگر تنها با یک ویژگی و به صورت خطی بتوان مقادیر هدف را پیش‌بینی کرد دیگر اصلا نیازی به یادگیری ماشین و روش‌های پیچیده نیست 😊.

## طبقه بندی

### کردن درخت تصمیم Prune

کردن درخت تصمیم به این معنی است که ما بعد از تشکیل درخت تصمیم حداقل تعداد داده های موجود در هر برگ را تعیین کنیم و برگ هایی که این شرایط را ندارند حذف کنیم. این موضوع باعث جلوگیری از اورفیت شدن مدل روی داده ها می شود. دو تصویر زیر این موضوع را در تسك رگرسیون و دسته بندی نشان می دهد.



مزایا:

- افزایش سرعت یادگیری و پاسخ دهی
- کاهش پیچیدگی مدل و اورفیت نشدن

معایب:

- تعیین پارامتر هرس
- احتمال آندرفیت شدن

### موارد استفاده از درخت تصمیم

در حالت کلی درخت تصمیم در موارد زیر مزایای بیشتری نسبت به سایر مدل ها دارد:

- **اهمیت تفسیر پذیری مدل:** در بعضی موارد از جمله موارد پزشکی ما نمی خواهیم تمامی روند را به عهده سیستم بگذاریم و در واقع می خواهیم به عنوان دستیار از آن استفاده کنیم در این شرایط مدل هایی مثل شبکه های عصبی و یادگیری عمیق که تفسیر پذیری پایین دارند مناسب نیستند و بر عکس درخت تصمیم که تفسیر پذیری بالایی دارد کاربرد دارد.
- **عملکرد مناسب در داده های غیر خطی:** همانطور که در تصویر بخش قبل هم دیدید درخت تصمیم می تواند الگوهای غیر خطی مانند سهمی با گستته سازی پیشینی کند.
- **استفاده در مجموعه داده ناقص:** در صورتی که بخشی از داده ها از دست رفته باشد و موجود نباشد درخت تصمیم از جمله مدل هایی است که می توان از آن استفاده کرد و مدل را با آن آموزش داد و استفاده کرد.
- **قابلیت استفاده هم در مسائل دسته بندی و هم رگرسیون**

## تفاوت KNN با سایر مدل‌ها در آموزش

در KNN عملاً ما مرحله آموزش ندارم فقط مدل داده‌ها را به خاطر می‌سپاردم و در هنگام پاسخ به سوال فاصله را از نقاط حساب می‌کنم. به این الگوریتم‌ها الگوریتم تنبل می‌گویند. تفاوت دیگری که وجود دارد در این الگوریتم پارامتری را محاسبه نمی‌کند. به دلیل موارد بالا و نوع خاص این الگوریتم حساسیت پایینی به داده‌های پرت دارد.

### neighbor nearest one

این الگوریتم نزدیک ترین داده مربوط به خود را پیدا می‌کند و داده همان برچسب را می‌زند.  
مزایا:

- سرعت بالاتر در مقایسه با  $k$  ها بالاتر
- سادگی پیاده‌سازی و درک آن

معایب:

- حساس به داده پرت
- سرعت پایین در صورتی که داده آموزشی حجم بالایی داشته باشد

## روش‌های سنجش فاصله در KNN

روش‌های مختلفی برای این کار وجود دارد که به بعضی از آنها اشاره می‌کنیم

- فاصله اقلیدسی

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

- فاصله منهتن

$$d = |x_1 - x_2| + |y_1 - y_2|$$

- فاصله همینگ: شمارش تعداد ستون‌های غیر یکسان

- فاصله مینکوفسکی:

$$d = (\|x_1 - x_2\|^p + \|y_1 - y_2\|^p)^{\frac{1}{p}}$$

## پیش‌بینی دهک خانه با استفاده از KNN و درخت تصمیم

ابتدا دهک‌ها گفته شده را به دست می‌آوریم:

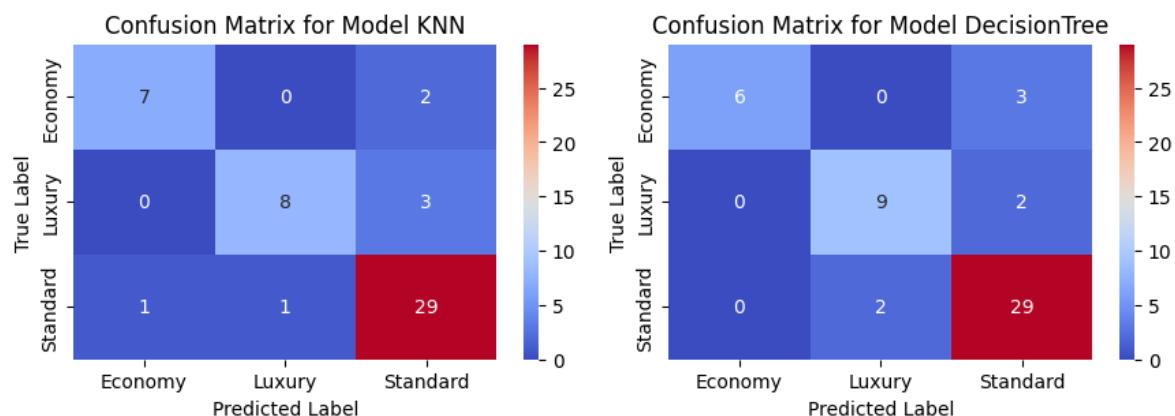
```
luxury_threshold = df['MEDV'].quantile(0.8)
economy_threshold = df['MEDV'].quantile(0.2)
```

سپس با یک شرط ساده ستون جدید را تشکیل می‌دهیم:

```
'Category' = df['MEDV'].apply(
    lambda x: 'Economy' if x <= economy_threshold else (
        'Luxury' if x >= luxury_threshold else 'Standard'))
```

مدل ها را آموزش می دهیم و نتایج را با کلاس نوشته شده Evaluator که در ادامه توضیح داده می شود محاسبه می کنیم.

	accuracy	f1 micro	recall micro	precision micro	f1 macro	recall macro	precision macro	f1 weighted	recall weighted	precision weighted
tree	0.862745	0.862745	0.862745	0.862745	0.831841	0.787227	0.907407	0.858063	0.862745	0.874728
knn	0.862745	0.862745	0.862745	0.862745	0.838612	0.813511	0.872277	0.860261	0.862745	0.864587



با کمی بازی با پارامتر ها در نهایت برای درخت تصمیم محدودیت حداقل ۴۰ نمونه در هر شکست را تنظیم کردیم و همسایه های KNN را روی ۷ گذاشتیم.

## برآورد کردن بهترین پارامتر ها

در این مرحله باید با استفاده از کتابخانه GridSearchCV بهترین پارامتر ها را پیدا کنیم برای این کار باید بازه جست و جو را برای هر پارامتر تعیین کنیم و سپس مدل را اجرا کنیم. این مدل از cross-validation استفاده می کند که بالاتر توضیح داده شد. بنابراین برای این مدل داده های آموزشی و validation را ادغام می کنیم.

این کار به صورت زیر انجام می شود:

```
from sklearn.model_selection import GridSearchCV
DT_param_range = {
    'criterion': ['gini', 'entropy', 'log_loss'],
    'min_samples_split': range(2, 100, 1),
    'max_depth': range(1, 15, 1)
}

DT_clf = GridSearchCV(DecisionTreeClassifier(), DT_param_range, cv=5)
DT_clf.fit(splitted_data['X'][['cross_train']],
splitted_data['y'][['cross_train']])
```

```
print("DT", DT_clf.best_params_)
```

نتایج به صورت زیر است:

```
KNN {'metric': 'manhattan', 'n_neighbors': 5, 'weights': 'distance'}
DT {'criterion': 'gini', 'max_depth': 10, 'min_samples_split': 4}
```

حال مدل ها را ارزیابی و با موارد دستی مقایسه می کنیم:

	accuracy	f1 micro	recall micro	precision micro	f1 macro	recall macro	precision macro	f1 weighted	recall weighted	precision weighted
DT_GS	0.784314	0.784314	0.784314	0.784314	0.750571	0.728283	0.781145	0.782844	0.784314	0.788077
KNN_GS	0.745098	0.745098	0.745098	0.745098	0.684779	0.645455	0.767053	0.734128	0.745098	0.756984
DT	0.843137	0.843137	0.843137	0.843137	0.801263	0.761616	0.879630	0.833601	0.843137	0.852941
KNN	0.784314	0.784314	0.784314	0.784314	0.717230	0.676768	0.872863	0.761997	0.784314	0.824661

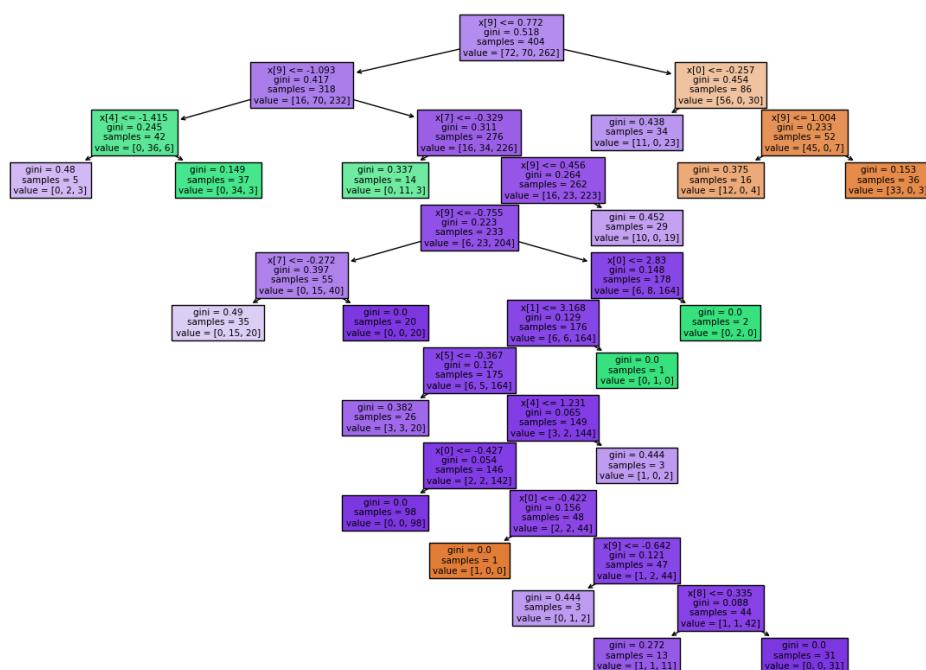
مشاهده می کنید که مدل ها دستی حدودا ۶٪ عملکرد بهتری دارند. دلیل این موضوع overfitting است. بهتر است به نتایج رو داده های آموزشی نگاهی بیندازیم:

	accuracy	f1 micro	recall micro	precision micro	f1 macro	recall macro	precision macro	f1 weighted	recall weighted	precision weighted
DT_GS	0.960396	0.960396	0.960396	0.960396	0.952737	0.959235	0.947052	0.960567	0.960396	0.961216
KNN_GS	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
DT	0.849010	0.849010	0.849010	0.849010	0.800357	0.758328	0.865066	0.843057	0.849010	0.852399
KNN	0.856436	0.856436	0.856436	0.856436	0.809323	0.781363	0.855461	0.850736	0.856436	0.856258

واضح است که مدل ما اورفیت کرده و در KNN ما صد درصد تشخیص درست داریم.

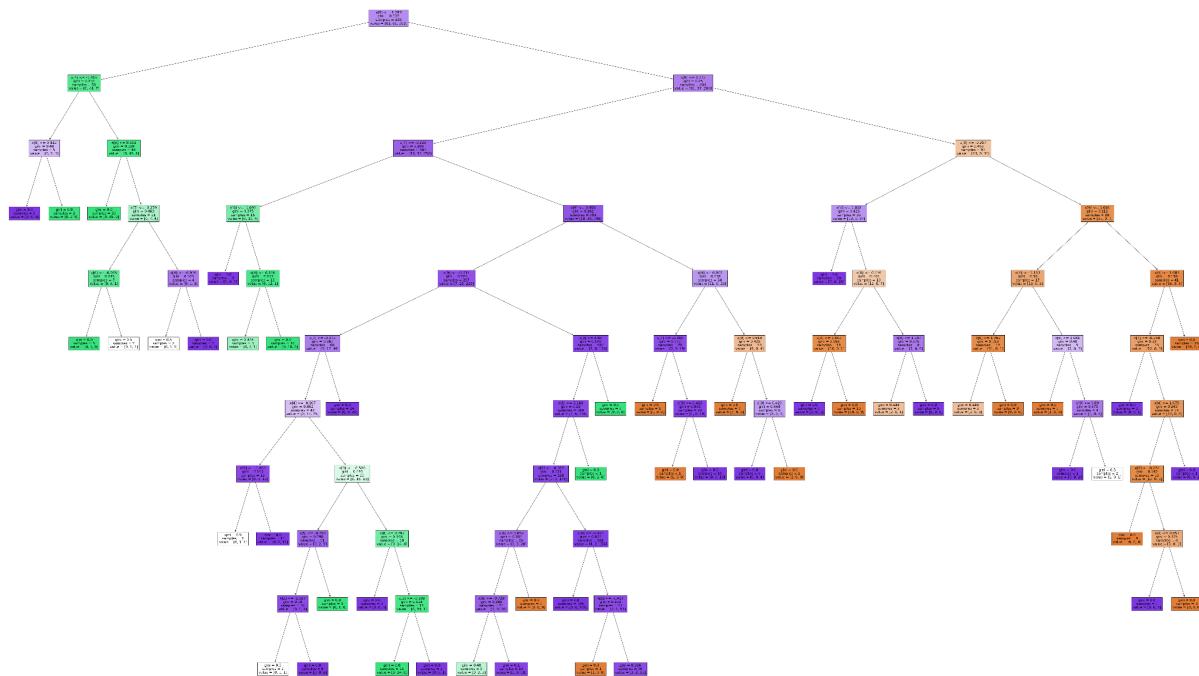
## رسم درخت تصمیم

درخت های تصمیم را اول برای مدل دستی خود رسم می کنیم:



با توجه به اینکه **gini** در بعضی جاها صفر شده پس احتمال اورفیت وجود دارد اما در حالت کلی بد به نظر نمی آید.

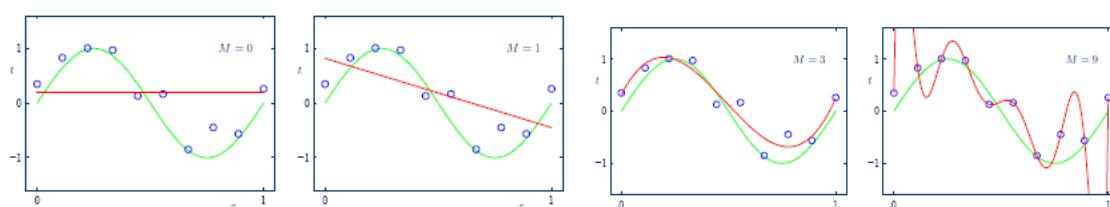
حال درخت مدل GSCV را بررسی می کنیم:



واضح است که این درخت برای ۴۰۰ داده اورفیت کامل است.

## Underfitting و Overfitting

برای درک این دو مفهوم در نظر بگیرید که یک دانش آموز می خواهد برای امتحان آماده شود. اگر دانش آموز برای آزمون ریاضی شروع به حفظ مسائل کتاب بکند به جای این که فرمول ها اثبات ها را یاد بگیرد شبیه به این است که مدل ما اورفیت کرده و اگر دانش آموز نتواند به اندازه کافی نمونه سوال ببیند و روش های حل را یاد نگیرد شبیه به این است که مدل آندرفیت کرده. اورفیت زمانی رخ می دهد که ما پیچیدگی مدل را بالا می بریم یا مثلا در همین درخت تصمیم محدودیتی در نظر نمی گیریم یا زمانی که هایپر پارامتر ها را به صورت دقیق روی داده ها تنظیم کرده ایم و آنر فیت زمانی رخ می دهد که یا ما داده های کمی داشته باشیم یا مدل بیش از اندازه ساده باشد (مانند رگرسیون خطی پیاده سازی شده) یا هایپر پارامتر ها به درستی تنظیم نشده باشند. مثال زیر این موضوع را در رگرسیون نشان می دهد. (a) آندر فیت و (b) اور فیت)



(a) 0'th order polynomial    (b) 1'st order polynomial    (c) 3'rd order polynomial    (d) 9'th order polynomial

# روش های Ensemble

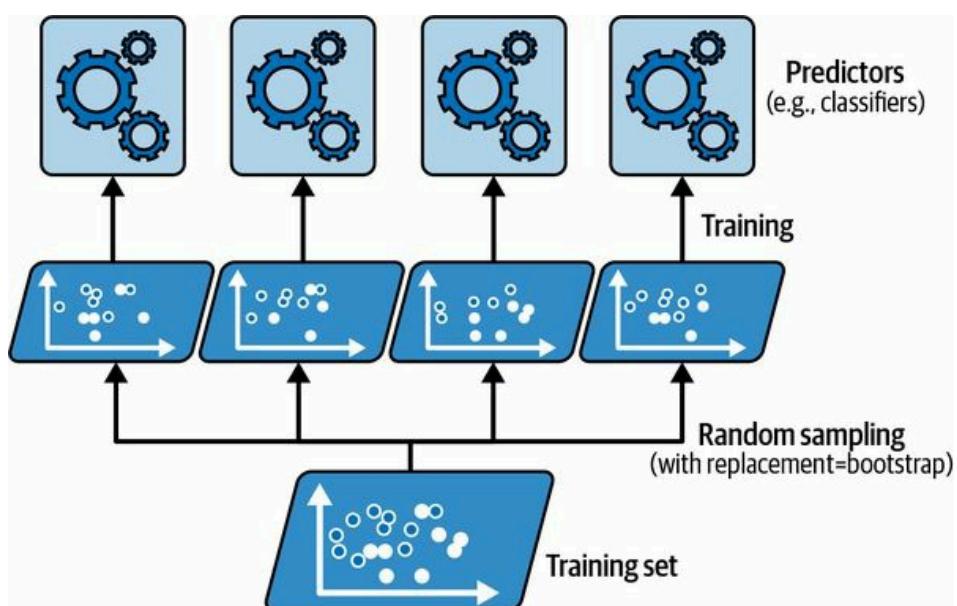
## چرایی استفاده و کاربرد

(Aurélien Géron - Hands-On Machine Learning 7) برگرفته از فصل 7

فرض کنید یک سوال پیچیده یادگیری ماشین را از صدها دانشجو در رشته ها (مانند فیچر ها) و دانشگاه (مانند ردیف های داده) های مختلف (مثل مهندسی و علوم کامپیوتر، ریاضی و آمار...) بپرسید یا استاد در بسیاری از موارد جمعیت جواب بهتری ارائه می دهد. به این موضوع خرد جمعی می گویند. در بسیاری از بخش و جوامع انسانی از این موضوع استفاده می شود پس چرا ما از این موضوع که هزاران سال آزمایش شده استفاده نکنیم. مثلاً فرض کنید ما می خواهیم از درخت تصمیم استفاده کنیم به جای ایجاد یک درخت تصمیم چند درخت تصمیم با زیر مجموعه ای از فیچر ها و داده ها می سازیم و سپس از نظر هر کدام برای پاسخ به سوال برآیند می گیریم به این موضوع جنگل تصادفی می گویند که از این دسته مدل ها هست. در مسابقات یادگیری ماشین اکثر تیم های برنده از چندین روش و مدل گروهی برای پیش بینی استفاده کردند. مزیت دیگری که وجود دارد این است که این مدل ها می توانند به صورت موازی آموزش ببینند و این باعث افزایش سرعت و استفاده بهینه از منابع می شود.

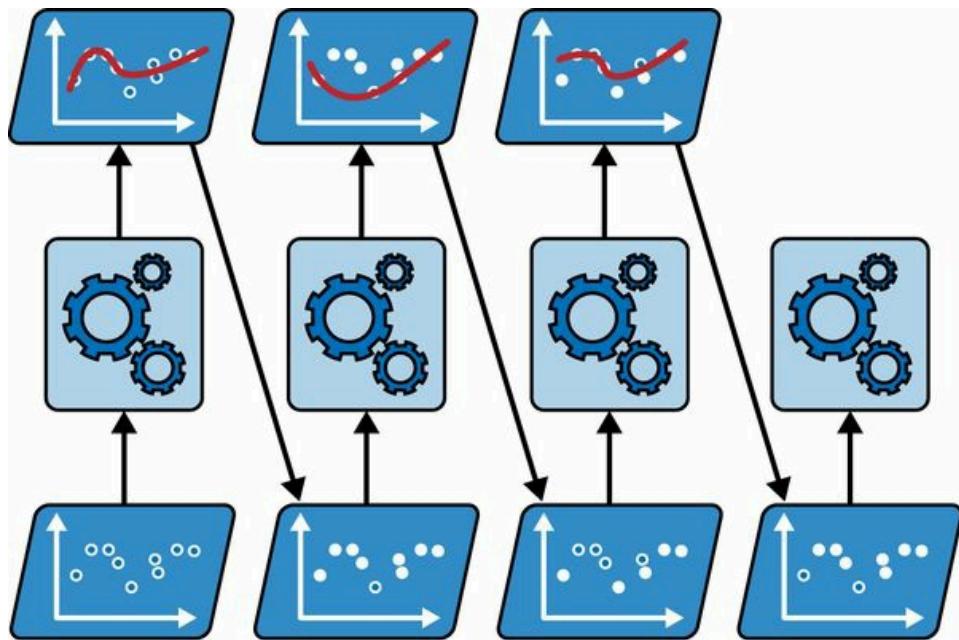
## روش bagging

هر مدل را با زیر مجموعه ای تصادفی ای که از داده های آموزشی گرفته شده آموزشی می دهیم. مدل های استفاده شده می توانند یکسان باشد یا خیلی متفاوت بسته به مسئله باید این را تعیین کنیم. حال اگر این نمونه برداری برای مدل های یکسان با جایگزینی باشد bagging نامیده می شود و در غیر این صورت pasting. در نهایت مدل ها که آموزش دیدند برای پیش بینی از آنها رای گیری می کنیم. روش های رگیری نیز مختلف است که از گفتن آن صرف نظر می کنیم.



## روش boosting

به طور کلی به روش ها که سعی می کنند از چندین یادگیرنده ضعیف و با ترکیب آنها یک یادگیرنده قوی درست کنند روش های boosting گفته می شوند. از روش های محبوب آن می توان به Adaboost برای دسته بندی و تقویت گرادیان برای رگرسیون اشاره کرد. الگوریتم Adaboost به این صورت عمل می کند که بعد از آموزش مدل اول در مدل دوم با افزایش وزن داده هایی که توسط مدل اول به خوبی یاد گرفته نشده اند تمرکز مدل را روی آنها بیشتر می کنیم و به همین صورت مدل های بعدی در پی بهتر کردن مدل های قبلی هستند.



## جنگل تصادفی

جنگل تصادفی در واقع چندین درخت تصمیم هستند که با روش bagging در کنار هم مدل بزرگتر و بهتری را تشکیل می دهند. در این مدل ما تقریباً همه هایپر پارامتر های درخت تصمیم را داریم.

## Bootstrapping

همانطور که در توضیح داده شد bagging اصطلاحاً به نمونه برداری با جایگذاری از نمونه ها گفته می شود. این موضوع در واقع شبیه همان مثال دانشجویان از دانشگاه های مختلف است که این موضوع باعث می شود بخشی از اطلاعات آنها یکسان باشد و نمونه ها مشترکاتی با یکدیگر داشته باشند و از آن طرف بخشی از داده های هر کدام متفاوت با یکدیگر هست که ایجاد تنوع می کند. این موضوع باعث افزایش بازده مدل و کاهش واریانس و پیچیدگی آن می شود.

## تعداد درخت های تصمیم در جنگل تصادفی

به طور کلی تعداد درخت های تصادفی در جنگل نباید نه انقدر کم باشد که عملاً کاربرد جنگل را از دست دهد نه انقدر زیاد باشد که عملاً حجم زیادی از درخت ها تکراری یا بی کاربرد باشند. طبق تحقیقی که کردیم این عدد حدود چند صد تا درخت (در مقاله Bagging Predictors این عدد ۱۰۰ پیشنهاد شده بود و در کتاب Aurélien Géron در مثال ها عدد ۵۰۰ قرار داده بود) است. البته بسته به حجم داده و پیچیدگی آنها متغیر است.

## موارد استفاده و عدم استفاده جنگل تصادفی

ممکن در در مواردی که حجم داده های ما زیاد هست استفاده از این روش مناسب نباشد زیرا حافظه زیادی را برای ذخیره درخت ها لازم دارم که البته به علت قابلیت توزیع در شبکه می توان بین سیستم ها توزیع کرد. این مدل به این علت که سعی می کند ساده باشد احتمالاً برای تشخیص وابستگی های جزئی مناسب نیست. اگر کلاس های داده نامتعادل باشند باید پیش پردازش هایی که قبل اشاره کردیم را انجام دهیم در غیر این صورت ممکن است به بسیاری از درخت ها اصلاً نمونه ای از کلاس دوم نرسد و این موضوع از معایب این روش است. این مدل در مقابل داده های نویز به خوبی عمل می کند بنابراین اگر داده هایمان نویز داشته باشند احتمالاً استفاده از این روش خوب باشد. همچنین می توان از این روش برای داده هایی که بخشی از آن از دست رفته است استفاده کنیم.

## واریانس در جنگل تصادفی

با توجه به توضیحات ارائه شده واضح است که استفاده از جنگل تصادفی باعث کاهش واریانس و پیچیدگی مدل می شود زیرا ما در آن تعداد زیادی مدل ساده داریم با واریانس پایین.

## پیاده سازی جنگل تصادفی

### شرح هایپر پارامتر ها

طبق خواسته پروژه ابتدا به توضیح هایپر پارامتر های می پردازیم:

**n\_estimators int, default=100**

تعداد درخت های تصمیم را مشخص می کند.

**criterion{"gini", "entropy", "log\_loss"}, default="gini"**

در درخت تصمیم ما الگوریتم های مختلفی داشتیم که میزان خلوص هر گره را نشان می داد. در این بخش می شود الگوریتم مورد نظر را انتخاب کرد

**max\_depth int, default=None**

حداکثر عمقی که هر درخت می تواند داشته باشد را مشخص می کند.

**min\_samples\_split int or float, default=2**

حداقل تعداد داده موجود را مشخص می کند که درخت بعد از آن دیگر تقسیم نشود.

**min\_samples\_leaf int or float, default=1**

حداقل تعداد داده موجود در هر برگ را مشخص می کند

**min\_weight\_fraction\_leaffloat, default=0.0**

مشابه پارامتر قبلی است با این تفاوت که به صورت نسبی و نسبت داده های هر برگ با کل داده ها مشخص می شود.

**max\_features{"sqrt", "log2", None}, int or float, default="sqrt"**

حداکثر تعداد فیچر های مورد استفاده در هر شکست را نشان می دهد.

**max\_leaf\_nodesint, default=None**

حداکثر تعداد برگ ها را مشخص می کند

**min\_impurity\_decreasefloat, default=0.0**

حداقل میزان کاهش ناخالصی برای تقسیم شدن هر گره.

**bootstrap bool, default=True**

بوت استرپ و جایگذاری در نمونه گیری از داده ها باشد یا خیر.

**oob\_scorebool or callable, default=False**

از داده های انتخاب نشده برای ارزیابی هر مدل استفاده بشود یا خیر.

**n\_jobsint, default=None**

تعداد پردازش های موازی را نشان می دهد.

**random\_stateint, RandomState instance or None, default=None**

تصادفی بودن نمونه برداری را نشان می دهد.

**verbose int, default=0**

مشخص می کند که مدل جزئیات را log بکند یا خیر.

**warm\_start bool, default=False**

از پاسخ های قبلی برای بهبود جواب ها استفاده کند یا خیر؟

**class\_weight{"balanced", "balanced\_subsample"}, dict or list of dicts, default=None**

یک لیست یا دیکشنری برای نسبت دادن وزن نمونه ها می گیرد.

**ccp\_alpha non-negative float, default=0.0**

حداقل پیچیدگی لازم برای هرس درخت را دریافت می کند.

**max\_samplesint or float, default=None**

حداکثر تعداد نمونه ها را مشخص می کند.

**monotonic\_cstarray-like of int of shape (n\_features), default=None**

کنترل میزان یکنواختی در طبقه بندی را مشخص می کند

## پیدا کردن بهترین هایپر پارامتر

ابتدا لازم است محدوده جست و جو را مشخص کنیم. با توجه به این که زمان اجرا طولانی می شود. نمی توانیم طول بازه را بزرگ بگیریم. سپس محدوده پارامتر ها و درخت تصادفی را به GSCV می دهیم تا جست و جو را شروع کند.

```

from sklearn.ensemble import RandomForestClassifier
RF_param_range = {
    'n_estimators': range(100,600,200),
    'min_samples_split': range(2, 50, 10),
    'max_depth': range(3, 7, 1)
}

RF_clf = GridSearchCV(RandomForestClassifier(), RF_param_range, cv=5)
RF_clf.fit(splitted_data['X']['cross_train'],
splitted_data['y']['cross_train'])
print("RF", RF_clf.best_params_)

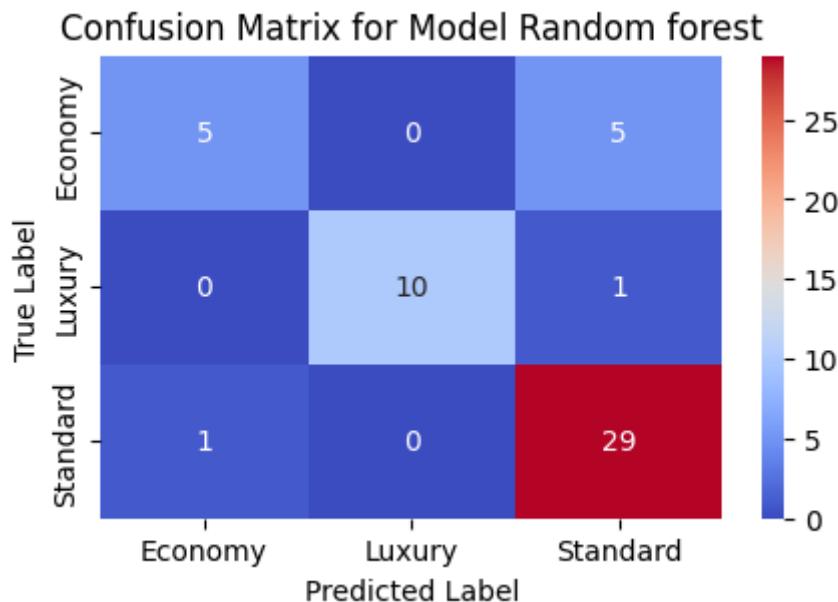
```

نتیجه به صورت زیر است:

```
RF {'max_depth': 6, 'min_samples_split': 2, 'n_estimators': 500}
```

حال مدل را به کلاس سنجش کفیت اضافه می کنیم تا نتیجه را مقایسه کنیم. نتیجه روی داده تست به

صورت زیر است:



	accuracy	f1 micro	recall micro	precision micro	f1 macro	recall macro	precision macro	f1 weighted	recall weighted	precision weighted
DecisionTree	0.843137	0.843137	0.843137	0.843137	0.801263	0.761616	0.879630	0.833601	0.843137	0.852941
KNN	0.784314	0.784314	0.784314	0.784314	0.717230	0.676768	0.872863	0.761997	0.784314	0.824661
DT_GS	0.784314	0.784314	0.784314	0.784314	0.750571	0.728283	0.781145	0.782844	0.784314	0.788077
KNN_GS	0.745098	0.745098	0.745098	0.745098	0.684779	0.645455	0.767053	0.734128	0.745098	0.756984
Random forest	0.862745	0.862745	0.862745	0.862745	0.823230	0.791919	0.887302	0.852851	0.862745	0.866480

همانطور که می بینید مقداری بهبود عملکرد نسبت به سایر مدل ها داشتیم.

نتیجه روی داده های تست به شکل زیر است:

	accuracy	f1 micro	recall micro	precision micro	f1 macro	recall macro	precision macro	f1 weighted	recall weighted	precision weighted
DecisionTree	0.849010	0.849010	0.849010	0.849010	0.800357	0.758328	0.865066	0.843057	0.849010	0.852399
KNN	0.856436	0.856436	0.856436	0.856436	0.809323	0.781363	0.855461	0.850736	0.856436	0.856258
DT_GS	0.960396	0.960396	0.960396	0.960396	0.952737	0.959235	0.947052	0.960567	0.960396	0.961216
KNN_GS	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
Random forest	0.930693	0.930693	0.930693	0.930693	0.912688	0.885438	0.947743	0.929529	0.930693	0.933429

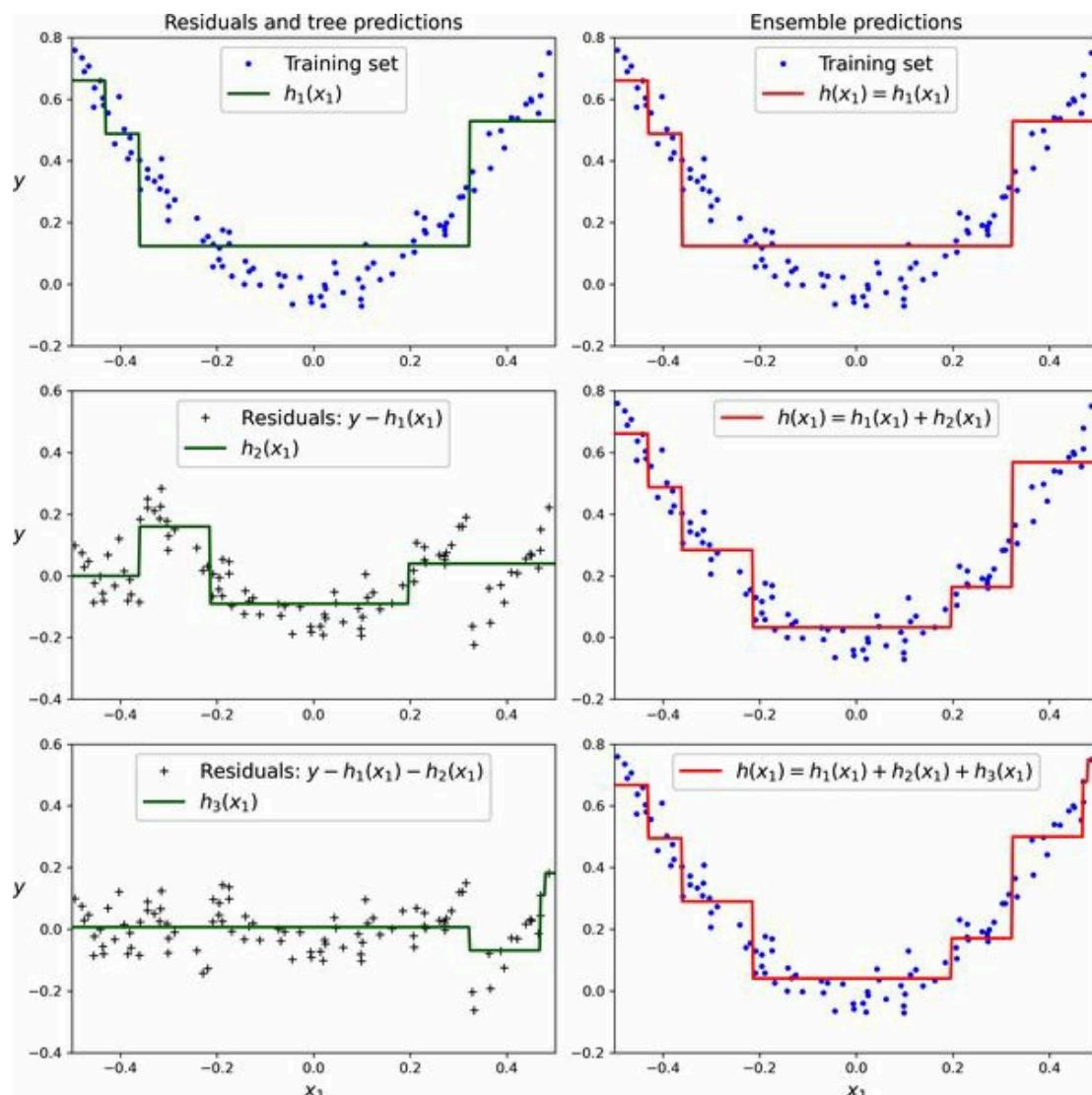
به نظر یک مقدار اورفیت است اما خیلی هم بد نیست.

## XGBoost

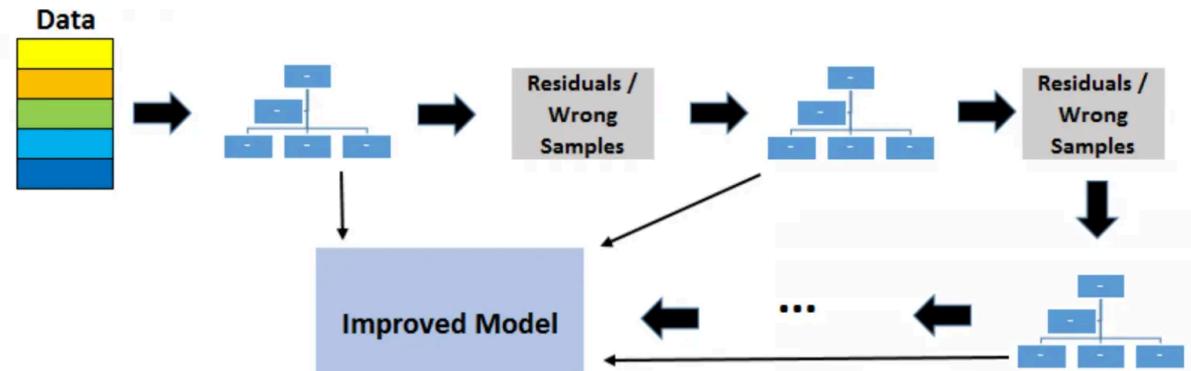
این مدل در واقع نوعی از Gradient Boosting است. پس بهتر است ابتدا کمی با این مدل ها آشنا شویم.

### Gradient Boosting

در این مدل ها سعی می شود که در هر مرحله پیشینی مدل را بهبود ببخشند و با تمرکز بر روی خطاهای مدل قبلی مدل نهایی را بهبود بخشنند. تصویر زیر به خوبی نشان دهنده نحوه عملکرد این مدل ها هست.



به مثال بالا دقیق کنید در این مثال ابتدا سعی شده رگرسیون با یک مدل ساده درخت تصمیم حل شود. سپس در بخش بعدی سعی شده رگرسیونی ساده برای اختلاف رگرسیون بخش قبل با داده ها ارائه شود. این عمل به طور متوالی تکرار شده و می توانید بهبود عملکرد مدل را برای آن بینند. برای درک عملکرد مدل به تصویر زیر توجه کنید.



تفاوت مدل گرادیان بهبود یافته با درخت تصمیم این است که در این مدل ما در واقع چندین درخت تصمیم داریم که مکمل یکدیگر هستند اما در مدل درخت تصمیم ما فقط یک درخت داریم. تفاوت این مدل با boosting tree این است که به طور کلی به هر مدلی از درخت تصمیم که به صورت پیاده شده درخت تصمیم می گویند و این مدل در واقع نوع خاصی از آن است که با گرادیان boosting کاهشی کار می کند.

## عملکرد

حالا که با Gradient Boosting آشنا شدیم به عملکرد XGBoost می پردازیم. در این مدل موارد گفته شده در بالا به صورت بهبود یافته پیاده شده و قابلیت پردازش چند هسته و توزیع شده دارد و حتی رو آپاچی اسپارک، کوبرنتیز و موارد مشابه قابل اجرا است همچنان تنظیمات پیشرفته تری هم دارد و در عمل عملکردی بهتر. نقطه قابل توجه دیگر این است که در بسیاری از زبان ها پشتیبانی می شود.

## پارامتر ها

### Learning Rate (eta)

میزان تاثیر هر درخت در پیش‌بینی نهایی را نشان می دهد و در مقادیر کم مدل دقیق تر است.

### Max Depth

حداکثر عمل درخت را مشخص می کند.

### Gamma

موثر در میزان تقسیم شدن درخت است. به ازای مقادیر بالاتر تقسیمات درخت موثر تر هستند و کمتر.

### Subsample

اندازه نمونه ها را برای هر درخت تعیین می کند.

### Colsample Bytree

درصد ویژگی هایی که برای هر درخت استفاده می شوند را مشخص می کند.

lambda (L2 regularization term) and alpha (L1 regularization term)

میزان موثر بودن L1, L2 را مشخص می کند.

min\_child\_weight

حداقل وزن هر گره درخت

scale\_pos\_weight

برای تنظیم تاثیر کلاس ها برای پروژه هایی که کلاس ها نامتعادل هستند.

## پیاده سازی

پارامتر ها را محدوده گذاری می کنیم و مدل را اجرا می کنیم.

```
XGB_param_range = {'nthread':[4],
                   'objective':['binary:logistic'],
                   'learning_rate': np.arange(0.04, 0.06, 0.002),
                   'max_depth': [6,7,8],
                   'min_child_weight': [11],
                   'silent': [1],
                   'subsample': [0.8],
                   'colsample_bytree': [0.7],
                   'n_estimators': [100], #number of trees, change it to 1000 for
better results
                   'missing':[-999],
                   'seed': [4],
                   'verbosity': [0]
                  }

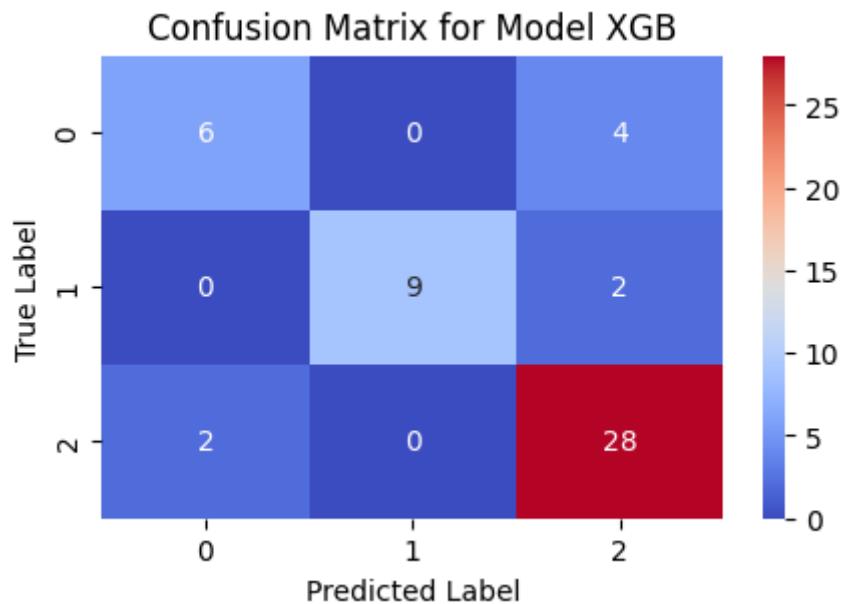
XGB_clf = GridSearchCV(XGBClassifier(), XGB_param_range, n_jobs=5, cv=5)
XGB_clf.fit(splitted_data['X']['cross_train'],
cat_to_num(splitted_data['y']['cross_train']))
print("XGB", XGB_clf.best_params_)
```

فقط مدل داده دسته ای به صورت عدد دریافت می کرد پس یک تبدیل ساده زدیم و به برچسب ها عدد نسبت دادیم.

نتیجه به صورت زیر است:

```
XGB {'colsample_bytree': 0.7, 'learning_rate': 0.0500000000000001, 'max_depth': 6,
'min_child_weight': 11, 'missing': -999, 'n_estimators': 100, 'nthread': 4,
'objective': 'binary:logistic', 'seed': 4, 'silent': 1, 'subsample': 0.8,
'verbosity': 0}
```

ارزیابی می کنیم روی داده های تست:



	accuracy	f1 macro	f1 micro	f1 weighted	precision macro	precision micro	precision weighted	recall macro	recall micro	recall weighted
XGB	0.843137	0.813889	0.843137	0.839542	0.857843	0.843137	0.847174	0.783838	0.843137	0.843137

نتیجه روی داده های تست:

	accuracy	f1 macro	f1 micro	f1 weighted	precision macro	precision micro	precision weighted	recall macro	recall micro	recall weighted
XGB	0.923267	0.90546	0.923267	0.922659	0.920902	0.923267	0.923316	0.891957	0.923267	0.923267

به دلیل ضعف سیستم نمی شد روی محدوده زیادی بررسی را انجام داده و دقت به دست آمده کمی پایین تر از انتظار است.

# SVM

## ساخت مدل

مانند قبل مدل را درست می کنیم.

```
svm_rbf = SVC(kernel='rbf')
svm_rbf.fit(splitted_data['X']['cross_train'],
splitted_data['y']['cross_train'])
svm_linear = SVC(kernel='linear')
svm_linear.fit(splitted_data['X']['cross_train'],
splitted_data['y']['cross_train'])
```

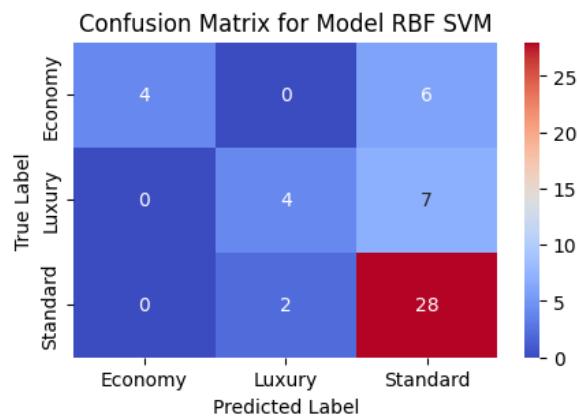
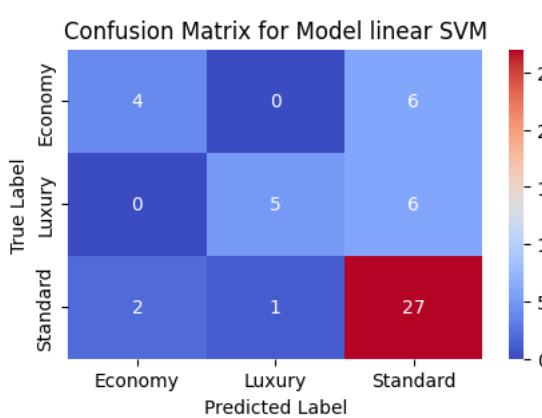
## ارزیابی مدل ها

نتیجه روی داده آموزشی:

	accuracy	f1 micro	recall micro	precision micro	f1 macro	recall macro	precision macro	f1 weighted	recall weighted	precision weighted
DecisionTree	0.849010	0.849010	0.849010	0.849010	0.800357	0.758328	0.865066	0.843057	0.849010	0.852399
KNN	0.856436	0.856436	0.856436	0.856436	0.809323	0.781363	0.855461	0.850736	0.856436	0.856258
DT_GS	0.900990	0.900990	0.900990	0.900990	0.883561	0.898088	0.872958	0.901863	0.900990	0.905109
KNN_GS	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
RBF SVM	0.853960	0.853960	0.853960	0.853960	0.810522	0.780885	0.850107	0.850234	0.853960	0.853378
linear SVM	0.863861	0.863861	0.863861	0.863861	0.827769	0.803290	0.858454	0.861376	0.863861	0.863208

نتیجه روی داده تست:

	accuracy	f1 micro	recall micro	precision micro	f1 macro	recall macro	precision macro	f1 weighted	recall weighted	precision weighted
DecisionTree	0.843137	0.843137	0.843137	0.843137	0.801263	0.761616	0.879630	0.833601	0.843137	0.852941
KNN	0.784314	0.784314	0.784314	0.784314	0.717230	0.676768	0.872863	0.761997	0.784314	0.824661
DT_GS	0.705882	0.705882	0.705882	0.705882	0.666716	0.680808	0.657432	0.700982	0.705882	0.699306
KNN_GS	0.745098	0.745098	0.745098	0.745098	0.684779	0.645455	0.767053	0.734128	0.745098	0.756984
RBF SVM	0.705882	0.705882	0.705882	0.705882	0.610250	0.565657	0.783198	0.677504	0.705882	0.741591
linear SVM	0.705882	0.705882	0.705882	0.705882	0.623615	0.584848	0.730769	0.685272	0.705882	0.717697



دقت ۷۰ درصد روی داده تست مطلوب نیست و با توجه به نتیجه روى داده آموزشی ما اورفیت داریم. در نهایت می توان گفت که مدل درخت تصمیم بالاترین دقیقیت را دارد و سپس مدل XGB در رتبه دوم قرار می گیرد و باقی مدل ها دقیقیت زیر ۸۰ درصد دارند که خیلی مطلوب نیست مدل SVM نیز خوب عمل نکرد شاید با پیش پردازش بهتر بتوان دقیقیت مدل SVM را بهتر کرد.

## روش محاسبه های پردازشی پارامتر ها

روش GS مقادیر مناسب را در بازه به طور قطعی تعیین می کند اما مشکل اورفیت و زمان زیاد پردازش برای آن هست. در روش RS بررسی پردازشی پارامتر ها به صورت شانسی است این موضوع سرعت جست و جو را بالاتر می برد و احتمال اورفیت را کاهش می دهد. با توجه به این سیستم من ضعیف است و مهارت لازم برای کار با GS را ندارم روش RS را ترجیح می دهم.

## پیاده سازی

مدل را به شکل زیر پیاده سازی می کنیم:

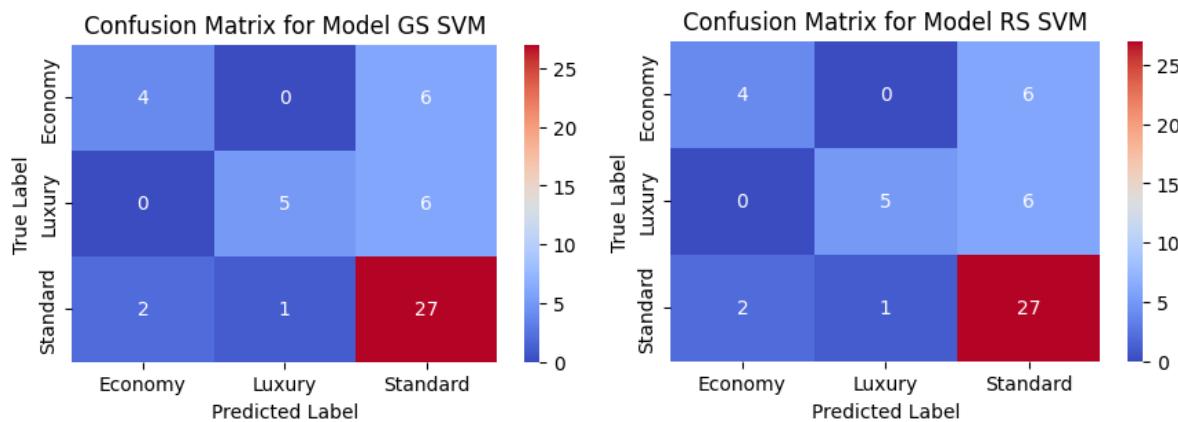
```
SVM_param_range = {
    "kernel": ('linear', 'rbf'),
    "degree": range(1,10,1),
    "gamma": range(0,10,1),
}

SVM_rs = RandomizedSearchCV(SVC(), SVM_param_range, cv=5)
SVM_rs.fit(splitted_data['X'][['cross_train']],
splitted_data['y'][['cross_train']])
print("SVM_rs", SVM_rs.best_params_)
SVM_clf = GridSearchCV(SVC(), SVM_param_range, cv=5)
SVM_clf.fit(splitted_data['X'][['cross_train']],
splitted_data['y'][['cross_train']])
print("SVM", SVM_clf.best_params_)
```

```
SVM_rs {'kernel': 'linear', 'gamma': 7, 'degree': 8}
SVM_gs {'degree': 1, 'gamma': 0, 'kernel': 'linear'}
```

نتایج روی داده تست و آموزش:

	accuracy	f1 micro	recall micro	precision micro	f1 macro	recall macro	precision macro	f1 weighted	recall weighted	precision weighted
DecisionTree	0.843137	0.843137	0.843137	0.843137	0.801263	0.761616	0.879630	0.833601	0.843137	0.852941
KNN	0.784314	0.784314	0.784314	0.784314	0.717230	0.676768	0.872863	0.761997	0.784314	0.824661
DT_GS	0.705882	0.705882	0.705882	0.705882	0.666716	0.680808	0.657432	0.700982	0.705882	0.699306
KNN_GS	0.745098	0.745098	0.745098	0.745098	0.684779	0.645455	0.767053	0.734128	0.745098	0.756984
Random forest	0.882353	0.882353	0.882353	0.882353	0.842713	0.803030	0.944444	0.870894	0.882353	0.901961
RBF SVM	0.705882	0.705882	0.705882	0.705882	0.610250	0.565657	0.783198	0.677504	0.705882	0.741591
linear SVM	0.705882	0.705882	0.705882	0.705882	0.623615	0.584848	0.730769	0.685272	0.705882	0.717697
RS SVM	0.705882	0.705882	0.705882	0.705882	0.623615	0.584848	0.730769	0.685272	0.705882	0.717697
GS SVM	0.705882	0.705882	0.705882	0.705882	0.623615	0.584848	0.730769	0.685272	0.705882	0.717697
	accuracy	f1 micro	recall micro	precision micro	f1 macro	recall macro	precision macro	f1 weighted	recall weighted	precision weighted
DecisionTree	0.849010	0.849010	0.849010	0.849010	0.800357	0.758328	0.865066	0.843057	0.849010	0.852399
KNN	0.856436	0.856436	0.856436	0.856436	0.809323	0.781363	0.855461	0.850736	0.856436	0.856258
DT_GS	0.900990	0.900990	0.900990	0.900990	0.883561	0.898088	0.872958	0.901863	0.900990	0.905109
KNN_GS	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
Random forest	0.930693	0.930693	0.930693	0.930693	0.912409	0.885305	0.948109	0.929424	0.930693	0.933625
RBF SVM	0.853960	0.853960	0.853960	0.853960	0.810522	0.780885	0.850107	0.850234	0.853960	0.853378
linear SVM	0.863861	0.863861	0.863861	0.863861	0.827769	0.803290	0.858454	0.861376	0.863861	0.863208
RS SVM	0.863861	0.863861	0.863861	0.863861	0.827769	0.803290	0.858454	0.861376	0.863861	0.863208
GS SVM	0.863861	0.863861	0.863861	0.863861	0.827769	0.803290	0.858454	0.861376	0.863861	0.863208



با وجود تغییر پارامتر ها نتایج هیچ تغییری نکرد.

## ارزیابی

ارزیابی مدل‌ها در قسمت مربوط به خودشان آمده در این قسمت به توضیح کلاس ارزیابی می‌پردازیم. کلاس Evaluator مدل‌ها را با معیارهای گفته شده ارزیابی می‌کند. ابتدا مدل‌ها را می‌گیرد تا در زمان گفته شده آنها ارزیابی کند. البته قابلیت اضافه کردن مدل در ادامه نیز وجود دارد. کلاس یک متوجه محاسبه معیارهای داده‌های تست را می‌گیرد و سپس با استفاده ازتابع کتابخانه ای آنها را برای تمام روش‌های میانگین گیری حساب می‌کند و سپس در یک دیکشنری می‌ریزد در نهایت دیکشنری به یک دیتا فریم پانداس تبدیل می‌شود و برگردانده می‌شود. متوجه مربوط به ماتریس درهم ریختگی است که ابتدا ماتریس حساب می‌شود و در یک هیئت مپ نمایش داده می‌شود در نهایت اگر لازم باشد ماتریس‌ها را برمی‌گرداند.

کد آن به صورت زیر است:

```
class Evaluator:
    def __init__(self, models:dict):
        self.models = models

    def calculate_evaluators(self, test_data, test_label):
        predict = {i: self.models[i].predict(test_data) for i in
self.models.keys()}
        result = dict()
        for i in self.models.keys():
            result[i] = dict()
            result[i]['accuracy'] = accuracy_score(test_label, predict[i])
            for j in ['micro', 'macro', 'weighted']:
                result[i][f'f1 {j}'] = f1_score(test_label,
predict[i], average=j)
                result[i][f'recall {j}'] = recall_score(test_label,
predict[i], average=j)
                result[i][f'precision {j}'] = precision_score(test_label,
predict[i], average=j)

        return pd.DataFrame(result).transpose()

    def show_confusion_matrix(self, test_data, test_label, return_val=False):
        predict = {i: self.models[i].predict(test_data) for i in
self.models.keys()}
        out = dict()

        for i in self.models.keys():
            result = confusion_matrix(test_label, predict[i])

            plt.figure(figsize=(5, 3))
            sns.heatmap(result, annot=True, cmap="coolwarm", fmt='d',
yticklabels=self.models[i].classes_,
xticklabels=self.models[i].classes_)
```

```
plt.title(f'Confusion Matrix for Model {i}')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()

out[i] = result

if return_val:
    return out

def add_or_change_module(self, model:dict):
    for i in model.keys():
        self.models[i] = model[i]
```

## نتیجه گیری

در این پروژه با روش های مختلف یادگیری ماشین آشنایی شدیم و ارتباط آنها به یکدیگر را متوجه شدیم همچنین با روش ها و درسر های تنظیم های پارامتر ها آشنا شدیم. راه های مختلف پیش پردازش را یاد گرفتیم. اما در نهایت به دقت مناسبی نرسیدیم دلیل این موضوع کم بودن داده ها و عدم تسلط در تنظیم های پارامتر ها می باشد.

## راه های بهبود و توسعه

- افزایش اطلاعات و نگاهی دقیق تر در بازار مسکن
- افزایش اطلاعات و نگاهی دقیق تر به وضعیت بوستون
- تنظیم بهتر پارامتر ها برای تمامی مدل ها
- ایجاد داده جدید
- ذخیره نتایج تست ها در کلاس Evaluator

## منابع

- اسلاید های درس علوم داده - دکتر یعقوب زاده و دکتر بهرک
- اسلاید های درس یادگیری ماشین - دکتر توسلی پور
- [6.4. Imputation of missing values – scikit-learn 1.4.2 documentation](#)
- [Normalization vs Standardization – GeeksforGeeks](#)
- [Train Test Validation Split: How To & Best Practices \[2023\]](#)
- Aurélien Géron - Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow\_ Concepts, Tools, and Techniques to Build Intelligent Systems-OReilly Media (2022)
- [Logistic Regression vs K Nearest Neighbors in Machine Learning – GeeksforGeeks](#)
- [1.6. Nearest Neighbors – scikit-learn 1.4.2 documentation](#)
- Bagging Predictors - LEO BREIMAN - Statistics Department, University of California, Berkeley), CA 94720
- [sklearn.model\\_selection.GridSearchCV – scikit-learn 1.4.2 documentation](#)
- [sklearn.tree.DecisionTreeClassifier – scikit-learn 1.4.2 documentation](#)
- [Random Forest Algorithm in Machine Learning – GeeksforGeeks](#)
- [sklearn.ensemble.RandomForestClassifier – scikit-learn 1.4.2 documentation](#)
- [An Introduction to Gradient Boosting Decision Trees - Machine Learning Plus](#)
- [ML | XGBoost \(eXtreme Gradient Boosting\) – GeeksforGeeks](#)
- [xgboost with GridSearchCV | Kaggle](#)