دانشگاه صنعتی شریف

دانشکده مهندسی هوافضا

گزارشکار پروژه ماشین لرنینگ ترمودینامیک-۲

نگارش

محمد متین وحیدی

استاد

محمدرضا مراد

فروردین 1403

1 فراهم کردن محیط کدنویسی و مقدمات کار

از نظر اکثر صاحب نظران و متخصصان حوضه کامپیوتر بهترین روش برای یادگیری کدنویسی، دست به کار شدن است. پس قبل از انجام هر کاری و حتی عمیق تر کردن دانش خود در ماشین لرنینگ ابتدا باید محیط کدنویسی آماده شود تا همزمان با یادگیری هر الگوریتم آن را در پایتون امتحان کنید . در ابتدا وارد محیط(Pycharm) میشوید و یک پروژه تشکیل میدهید. سپس کتابخانه های لازم را import میکنید. من این بخش کار را کاملا طبق ویدوای که از جلسه اول آموزش ضبط شده بود رفتم. اگر نیاز به تغییر احساس شود بعدا اعمال خواهد شد.

اولین مشکل بر خودم. بعد از تلاش برای نصب کتابخانه های ذکر شده در ویدیو به ارور زیر برخوردم:

Could not find a version that satisfies the requirement (from versions:) No matching distribution found for

بعد از گشتن دنبال راه حل فهمیدم که ویندوز اجازه دسترسی را وقتی که به اینترنت عمومی (مثل هات اسپات تلفن همراه) برای نرم افزار ها محدود کرده. بعد از انجام تغییرات لازم در فایروال همه چیز به درستی پیش رفت.

```
df = pd.read_csv("gt_2012.csv")
df.head(10)
```

دو خط بالا را به کد خود اضافه کردم تا مطمئن شوم فایل میتواند داده ها را بخواند و به من نشان دهد. کد بدون هیچ اروری ران شد اما هیچ داده ای هم به من نشان نداد. متوجه شدم باید برنامه را داخل فایلی با فرمت (.ipynb) که مخصوص jupyter notebook است بنویسم. یک فایل جدید ساختم و همه کد را به آن منتقل کردم دوباره ارور:

Jupyter server process exited with code 1

...

Jupyter command jupyter-notebook not found.

متوجه نمیشدم که مشکل کجاست. حدود دو ساعت تمام اینترنت را زیر و رو کردم و نتوانستم مشکل را حل کنم. مدام همین ارور را میگرفتم. میتوانستم با کمک cmd وارد jupyter Lab بشوم ولی از طریق تکست ادیتور نمیشد که نمیشد. بالاخره خسته شدم و تصمیم گرفتم که وقت دیگری به سراغش بیایم.

فردا که تکست ادیتور را باز کردم و برنامه ران کردم به طرز معجزه آسایی درست شده بود. حال یک قدم به هدف خود نزدیک تر شده بودم:

ModuleNotFoundError: No module named 'pandas'

کار کردن با این نوت بوک از کار نکردن با آن بیش تر ضرر دارد. دوباره کار را به روزی دیگر موکول میکنم.

فردا که فایل را باز میکنم و اجرا میکنم به امید اینکه درست شده باشد با این ارور مواجه میشم:

Run Error:
Unexpected character ('' (code 60)): expected a valid value (JSON String, Number, Array, Object or token 'null', 'true' or 'false') at [Source: (String)"

عالی شد. یک ارور جدید. داخل ترمینال را که نگاه میکنم میبینم api با هر بار اجرا کردن برنامه 302 مینویسد. با توجه به آشنایی مختصری که با نرم افزارهای تحت وب دارم سریع متوجه میشوم که مشکلی در ارتباط بین تکست ادیتور و سرور نوت بوک وجود دارد. کد را که در اینترنت سرچ میکنم نوشته علت آن انتقال موقتی سرور به مکانی دیگر است. از بخش
Settings\Language & Framework\Jupyter\Jupyter servers\Configured Server
ادرس سرور ژوپیتر را اصلاح میکنم. درست شد. دوباره همان ارور روز قبل را گرفتم. برای رفع ارور باید سرور ژوپیتر در همان virtual environment ای که قبلا ماژول ها را نصب کرده بودم اجرا می کردم. ولی سرور در conda
اجرا شده بود. برای venv خود یک karnel ساختم و karnel موجود در سرور را به گزینه مورد نظر خود تغییر دادم بعد از چند بار ری استارت کردن karnel مشکل حل شد.

```
df.describe()
```

خط بالا را اضافه میکنم تا اطلاعاتی کلی راجع به داده ها از جمله تعداد، میانگین، پراکندگی، ماکسیمم، مینیمم و ... نمایش داده شود.

```
cdf = df[['AT', 'AP', 'CO', 'NOX']]
cdf.head(5)
```

میتوان ستون ها را فیلتر کرد تا فقط داده هایی که میخواهیم در یک متغییر ذخیره و نمایش داده شود. چهار ستون انتخاب شده‌اند تا فقط اثبات شود که هیچ مشکلی در فرآیند پیش نخواهد آمد.

```
viz = cdf[['AT', 'AP', 'CO', 'NOX']]
viz.hist()
plt.show()
```

برای اینکه درکی از پراکندگی داده ها داشته باشیم کدهای بالا به کارمان می آید.

```
plt.scatter(cdf.AP, cdf.CO, color = 'red')
plt.xlabel('AT')
plt.ylabel('CO')
plt.show()
```

برای اینکه روابط بین داده ها کشف شود میتوان نمودار یک متغییر نسبت به دیگری را رسم کرد.

```
msk = np.random.rand(len(df)) < 0.8
train = cdf[msk]
test = cdf[~msk]
```

داده ها را به صورت رندوم به نسبت 20/80 به دو بخش تقسیم میکنیم با 80 درصد آموزش میدهیم وبا بقیه تست میکنیم.

به یاد می آورم که باید چهار فایل دیتا را داخل یک فایل متحد کنم. همه فایل ها را داخل یک پوشه جدید می ریزم و در command prompt با کمک دستور:

Open *.csv merged-data.csv

همه فایل ها را داخل merged-data می ریزم. کد را تغییر میدهم تا فایل جدید را بخواند و کل برنامه را اجرا میکنم بعد از سلولی که رسم نمودار را در آن خواسته بودم ارور زیر را دریافت میکنم:

TypeError: 'value' must be an instance of str or bytes, not a float

از چت جی پی تی علت ارور را میپرسم یکی از اشکالاتی که پیشنهاد میدهد وجود دیتای خالی یا به اصطلاح null داخل dataFrame است. کد زیر را مینویسم تا ببینم آیا دیتای null ای وجود دارد یا خیر:

```
df.isnull().values.any()
```

جواب بله است. با سرچ در نت به این نتیجه میرسم که اگر بعد از خواندن دیتا این خط کد را اضافه کنم دیگر نباید دیتای پوچ داشته باشم:

```
df = pd.read csv("merged-data.csv")
df = df.dropna()
df.isnull().values.any()
```

ارور رفع میشود ولی هروقت نمودار را رسم میکنم مدت زیادی طول میکشد که نتیجه را نشان دهد و وقتی که میدهد شکلی نامعقولی ست. یا دیتای جدید برای کامپیوتر بیش از حد سنگین است و یا در فرآیند merge کردن مشکلی پیش آمده.

یک کپی از فایل سال 2014 میگیرم و بصورت دستی ردیف های بقیه فایل ها را کپی و پیست میکنم. جواب داد!

PRE-PROCESSING برای CO

1.1 نگاهی اولیه به داده های CO

قبل از هرکاری باید نمودار co را نسبت به تمام متغییر های دیگر ببینم تا حدس بزنم که کدام یک از این متغییر ها تاثیر بیش تری میگذارد و چرا. برای اینکار کد زیر را در cell-5 مینویسم:

```
indx = 0

for name, data in df:
    indx += 1

    if indx == 10:
        break

    plt.subplot(3, 3, indx + 1)
    plt.scatter(data, df.CO, color = 'red')
    plt.xlabel(name)
    plt.ylabel('CO')
    plt.show()
```

این کد قرار است که در تمام ستون های دیتا فریممان پیمایش کند و داده های هر ستون را در data و اسم ستون را در name ذخیره سازی کند. شرط برای این است که CO و NOX نصب به خودشان رسم نشوند.

ValueError: x and y must be the same size

از دیباگر استفاده میکنم. مشخص میشود که در اولین پیمایش name = ‘A’ و data = ‘T’ نوشته شده. مشکل مشخص شد. ظاهرا پایتون کل دیتافریم را که یک کلاس ساخته شده درون pandas برای ذخیره داده است را به صورت یک string بزرگ تشخیص میدهد. اما احتمالا یک راهی برای پیمایش درون آن وجود دارد. اما من ترجیح میدهم که همانند یک غارنشین روش زیر را انتخاب کنم:

```
colNames = ['AT', 'AP', 'AH', 'AFDP', 'GTEP', 'TIT', 'TAT', 'CDP', 'TEY']
indx = 0

while indx < 9:
    indDataName = colNames[indx]
    depData = df.CO

    plt.subplot(3, 3, indx + 1)
    plt.scatter(df[indDataName], depData, color = 'red')
    plt.xlabel(indDataName)
    plt.ylabel('CO')
    plt.show()

    indx += 1
```

و جواب هم میدهد. اما به جای اینکه نمودار ها بزرگ و سه در سه چیده شوند کوچک و در یک ستون قرار گرفته اند. مشکل واضح است در هر دوره ای که حلقه اجرا میشود نمودار هم نمایش داده میشود. پس به همین خاطر پشت سر هم در ترمینال نمایش داده میشوند. plt.show() را از حلقه بیرون می آورم جایگیری نمودار ها درست است اما خیلی ریز هستند و نمیتوان برداشتی از آن داشت. داخل اینترنت سرچ میکنم که چطور میشه اندازه نمودار را تغییر داد به این نتیجه میرسم که با اضافه کردن این خط کد قبل حلقه باید مسئله حل شود. ومیشود

```
plt.figure(figsize=(10, 10))
```

همانطور که در "fig-1" مشاهده میکنید از نظر من بین CO و AH, AT, AP رابطه نموداری معناداری وجود ندارد. فقط به نظر میرسد که حول بعضی مقادیر این متغییر ها تجمع داده بیش تر است. که این احتمالا باید به خاطر پراکندگی خود متغییر های وابسته باشد و نه چیز دیگری.

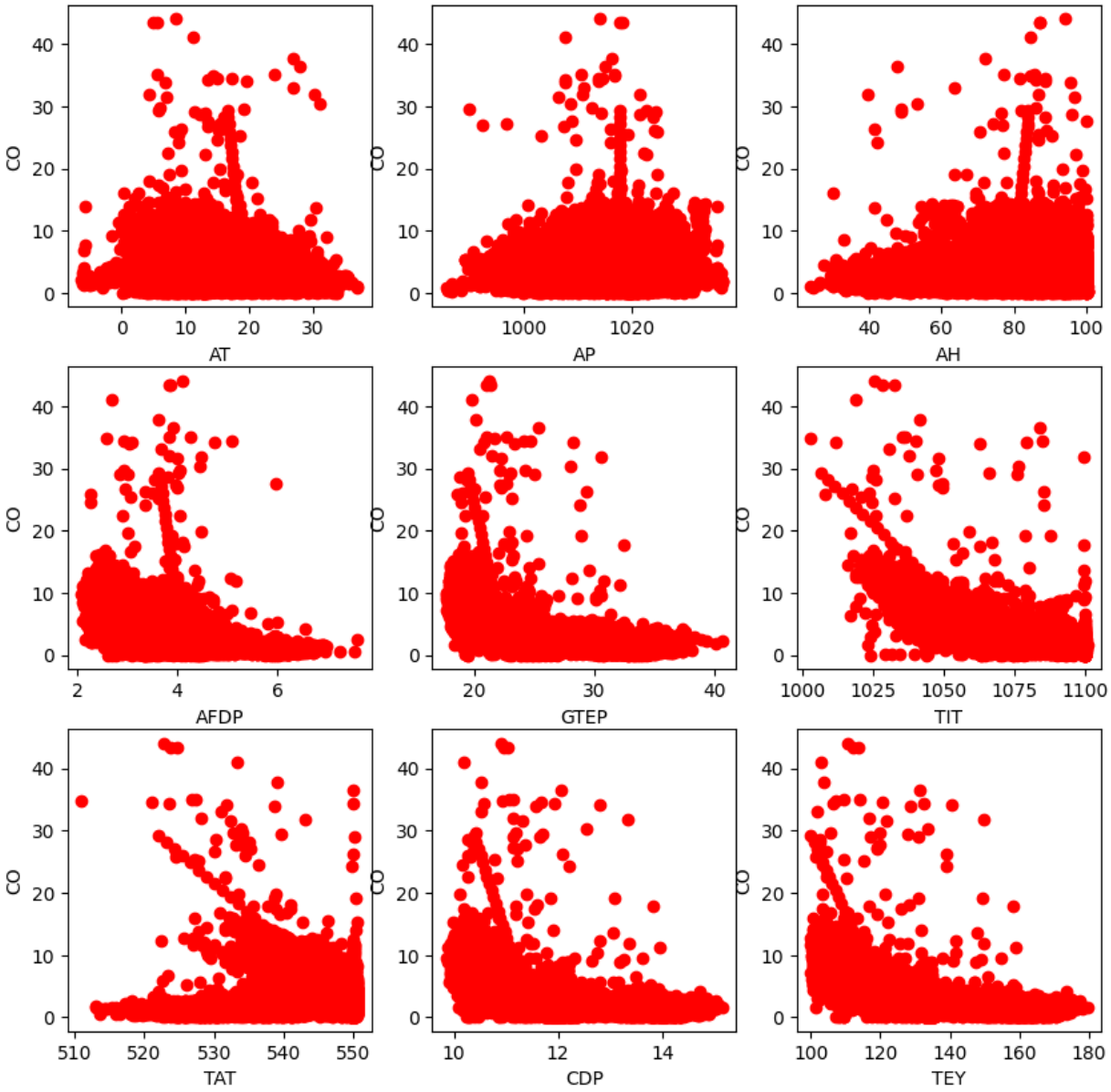

Fig-1 نمودار تغییرات CO نسبت به تمام متغییر های دیگر

Cell-3 را حذف میکنم چون دست و پاگیر است و کاربردی ندارد. به جایش کد زیر را مینویسم تا هر وقت بخواهم بتوانم پراکندگی داده های مورد نظر خود را ببینم:

```
dataHist = ['AT', 'AP', 'AH', 'AFDP', 'GTEP', 'TIT', 'TAT', 'TEY', 'CDP'
]
cdf = df[dataHist]
viz = cdf[dataHist]
```

```
viz.hist()
plt.show()
```

الان با هر بار عوض کردن dataHist میتوان پراکندگی داده های مورد نظر را دید و حتی مقایسه کرد.

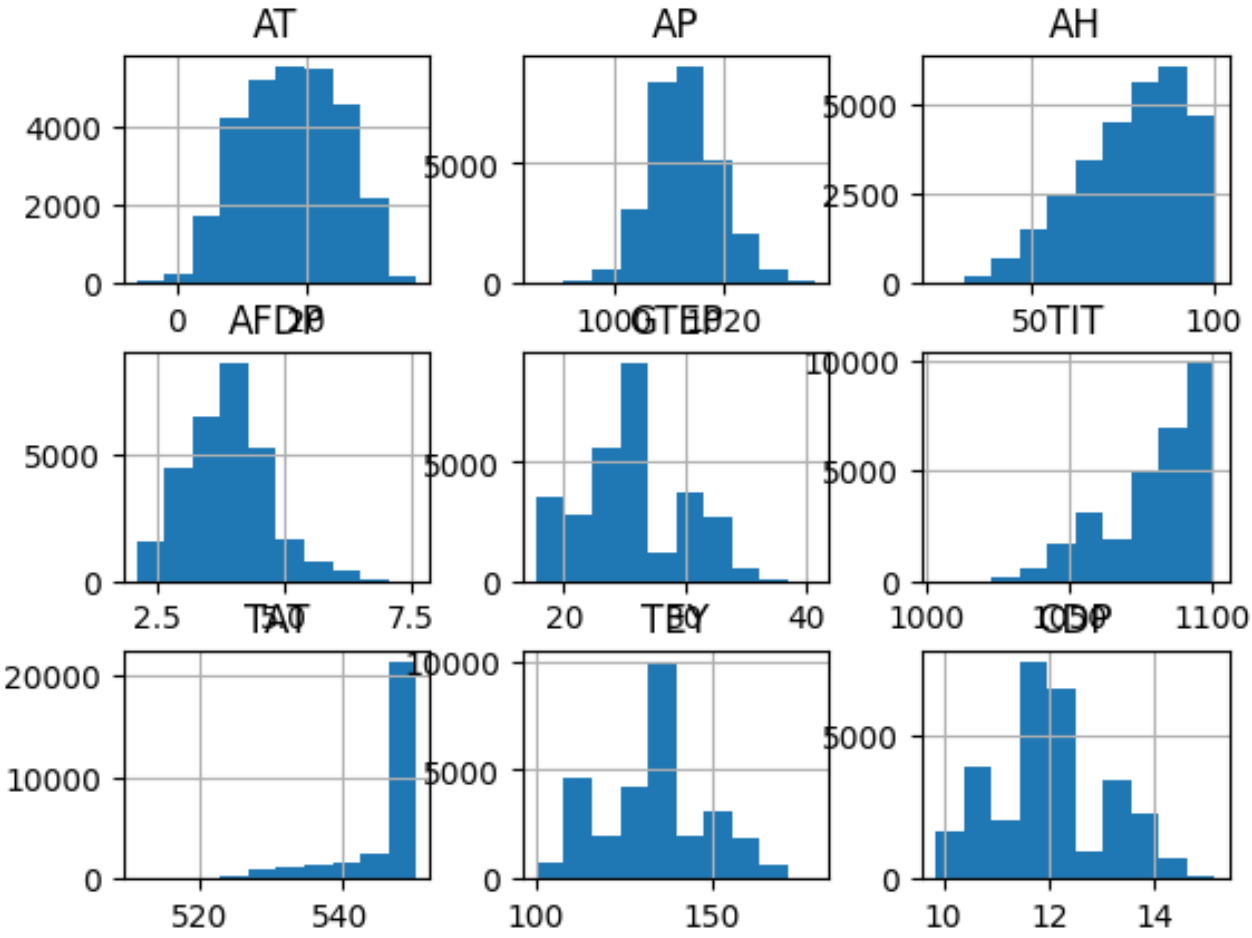

Fig-2 نمودار پراکندگی AH, AT, AP

پراکندگی داده های "fig-2" تقریبا مشابه نمودار های "fig-1" است و هر جا که داده بیش تری وجود دارد مقدار بیشتری هم دیده شده. البته باید توجه کرد که برای AH نمودار گاز حول مقدار 10 بیش تر و متراکم تر است.

واضح ترین رابطه متعلق به TIT است که هرچه روی ایکس جلوتر میرویم مقدار متغییر وابسته کمتر میشود. از نظر من این نمودار میتواند نمودار ساده خطی باشد اما اگر بخواهیم وسواس به خرج بدهیم نشانه های منحنی درجه دو یا حتی بالاتر را هم میتوان در آن دید.

یک رابطه بین متغییر های GTEP ,AFDP,TEY ,CDP با CO وجود دارد که هر چه بیش تر میشوند میزان CO تولیدی کم شده است و منحنی نمودار شبیه نمودار 1/x، تابع نمایی نزولی یا حتی چند جمله ای هایی با درجات بالاتر از 2 است.

نمودار پراکندگی و CO برای متغییر TAT بررسی میکنم متوجه میشوم که با اینکه بیشترین تراکم حوالی 550 است اما در نمودار CO کمترین مقادیر مربوط به همین ناحیه است. از طرفی مقادیر کمتر دو رفتار متناقض نشان میدهند که یک بصورت ثابت و دیگری خطی نزولی است.

1.2 انتخاب الگوریتم مناسب

واضحا مسئله ای که با آن مواجه هستیم از جنس classification, reinforcement learning و clustering نیست. البته با جستجو در اینترنت به این نتیجه میرسم که از برخی روش های غیرمستقیم میتوان الگوریتم های classification را برای پیش بینی مقادیر پیوسته بکار برد که ممکن است چندان بازدهی نداشته باشد. از طرفی میتوان با کمک clustering داده ها را به چند دسته تقسیم کرد و برای هر دسته از داده ها به صورت جداگانه از regression استفاده کرد.

Neural Network در نگاه اول به موضوع ما بی ربط به نظر می رسد اما با جستجوی بیش تر میبینم که فردی از آن برای پیش بینی قیمت مسکن در کالیفرنیا استفاده کرده پس من هم میتوانم الگوریتمی مشابه بکار ببرم.

Ensemble learning به سه دسته bagging, stacking, boosting تقسیم بندی میشود. به طوری کلی این دسته از الگوریتم ها برای تصمیم گیری استفاده میشود اما در stacking داده ها به چند الگوریتم متفاوت داده میشود و با ترکیب نتایج همه این داده ها به نتیجه بهتری رسید. از آنجایی که محدودیتی در انتخاب این مدل ها نیست میتوان چند مدل regression که بهترین بازدهی را داشتند با هم ترکیب کرد.

و در انتها راحت ترین و دردسترس ترین روش استفاده از regression است. برای اینکه بیش تر با الگوریتم های مختلف این دسته آشنا شوم کلید واژه most popular regression algorithms را سرچ میکنم.
(https://www.onlinemanipal.com/blogs/popular-regression-algorithms-in-machine-learning)

1- Linear Regression

مشهور ترین مدل ماشین لرنینگ که اول از همه آموخته میشود. این الگوریتم در یک کلام سعی میکند که داده ها را به خط راست فیت کند. این الگوریتم یک خط با مقادیر فرضی شیب و عرض از مبدا و مقدار تابع MSE(mean squared error) را مینیمم میکند.

2- Ridge Regression

این الگوریتم یک ترم به نام Penalty term را به تابع MSE اضافه میکند و مینیمم تابع جدید را بدست می آورد. ایده اصلی پشت آن این است که اگر مقدار تابع را کمی تغییر دهیم تا برای داده train بهترین مقدار را نشان ندهد، ممکن است باعث شود که مدل برای رنج بزرگ تری از داده های تست جواب بهتری تولید کند یا به اصطلاح جنرال تر شود. این مهم با کاهش شیب نمودارحاصل

میشود. برای سیستم های که بین متغییر های مستقل روابط وجود دارد این الگوریتم بهتر جواب خواهد داد.

3- Lasso Regression

ایده پشت آن مشابه الگوریتم قبلی است اما Penalty term متفاوتی دارد

4- Neural Network Regression

همانند همه شبکه های عصبی از یک لایه ورودی و یک لایه خروجی و تعدادی لایه پنهان در بینشان تشکیل شده. هر لایه از تعدادی نورون تشکیل شده که از همه نورون های لایه قبل ورودی میگیرد و به همه نورون های لایه بعدی خروجی میدهد. هر نورون یک بایاس و برای هر ورودی یک وزن دارد. همه ورودی ها در وزن خود ضرب و حاصل جمعشان با بایاس خرووجی نورون را تایین میکند. این وزن ها و بایاس ها روی هم تمام متغییر های مدل هستند که توسط تابع optimizer باید مینیمم شوند. Optimizer هم از یک تابع دیگر به نام loss برای سنجیدن عملکرد مدل استفاده میکند که برای رگرسیون همان MSE بهترین انتخاب است.

5- Decision Tree Regression

این الگوریتم که کاربرد اصلی آن در مسائل Classification است از تعدادی Node و تعدادی Leaf تشکیل شده. داخل هر Node یک شرط ساده وجود دارد مثل ($X_0 < 1$) که اگر هر داده این شرط را داشته باشد به راست و اگر نداشته باشد چپ میرود. در انتها فقط داده های خالص به Leaf خواهد رسید. منظور از داده خالص مجموعه داده هایی‌اند که متعلق به یک دسته‌اند. الگوریتم سعی میکند برای هر نود شرط را به گونه‌ای تایین کند که بیشترین خلوص داده بدست آید. اگر در هرکدام از شاخه‌ها داده خالص شد فبها وگرنه یک Node دیگر داریم که باید شرط برایش تایین کرد. در نهایت فضای دامنه به تعدادی ناحیه تقسیم میشود که معلوم است هر دامنه متعلق به کدام دسته است. برای پیش بینی نگاه میکنیم به اینکه داده جدید در کدام ناحیه است.برای تایین مقادیر پیوسته مانند قبل عمل میکنیم فقط این بار نواحی بر اساس مقدار متغییر وابسته مقایسه میشوند.

6- Random Forest Regression

این الگوریتم زیرمجموعه Ensemble learning به حساب می‌آید. در این الگوریتم به جای اینکه یک درخت(Decision Tree) را آموزش بدهیم تعداد بیش تری از آن ها را آموزش میدهیم. داده هر درخت یا با زیرمجموعه‌ای از مجموعه داده ها یا با تعداد محدودی متغییر مستقل آموزش داده میشود. در نهایت داده تست به همه مدل ها ارائه میشود و میانگین همه نتایج (برای رگرسیون) یا اکثریت جواب (برای دسته بندی) به عنوان جواب نهایی اعلام میشود.

7- Polynomial Regression

با روش هایی از همان الگوریتم رگرسیون خطی میتوان برای فیت کردن منحنی های مختلف به داده ها استفاده کرد.

سایتی که در بالا ذکر شده حدود 10 الگوریتم محبوب را معرفی میکند ولی توضیحات زیادی نمیدهد. به همین خاطر برای درک عمیق تر به سایر منابع هم متوسل شدم اما از این منبع به عنوان راهنما استفاده کردم. همان طور هم که معلوم است طیف گسترده‌ای از روش ها را پوشش داده که به نظرم کفایت میکند. البته از تعدادی الگوریتم فاکتور گرفتم تا کار راحت تر شود.

با توجه به مشاهدات و تحقیقات به این نتیجه میرسم که برای CO به ترتیب از این روش ها استفاده کنم:

(ترتیب از راحت ترین تا سخت ترین از لحاظ پیاده سازیست)

1- رگرسیون خطی ساده با TIT
2- رگرسیون خطی چندگانه با 'TEY', 'CDP', 'GTEP', 'AFDP'
3- رابطه خطی چندگانه با 'TEY', 'CDP', 'GTEP', 'AFDP' این بار به کمک Ridge و Lasso
4- فیت کردن تابع نمایی تک متغییره به کمک رگرسیون خطی برای 'TEY', 'CDP', 'GTEP', 'AFDP', 'TIT' هرکدام به صورت جداگانه
5- فیت کردن تابع نمایی چند متغییره به کمک رگرسیون خطی برای 'TEY', 'CDP', 'GTEP', 'AFDP', 'TIT'
6- فیت کردن تابع 1/x تک متغییره به کمک رگرسیون خطی برای 'TEY', 'CDP', 'GTEP', 'AFDP', 'TIT' هرکدام به صورت جداگانه
7- فیت کردن تابع 1/x چند متغییره به کمک رگرسیون خطی برای 'TEY', 'CDP', 'GTEP', 'AFDP', 'TIT'
8- فیت کردن تابع خطی درجه دو و سه تک متغییره برای 'TEY', 'CDP', 'GTEP', 'AFDP', 'TIT" هرکدام به صورت جداگانه
9- فیت کردن تابع خطی درجه دو و سه چند متغییره برای 'CDP', 'GTEP', 'AFDP', 'TIT'
10-استفاده از شبکه عصبی
11- استفاده از Decision Tree برای کل متغییر هایی که ظاهرا رابطه‌ای با CO دارند به صورت یکجا
12-استفاده از Random Forest برای همه متغییر ها

از آنجایی که من تخصصی در این زمینه ندارم اساس کار در ادامه کار آزمایش و خطا خواهد بود. ولی انتظار می‌رود که هرچه پیچیدگی بیشتر شود نتایج بهتری هم بدست بیاید.

2 PROCESSING برای CO

2.1 رابطه خطی بین CO-TIT

با کمک ویدیو های ضبط شده برنامه را مینویسم.

```
from sklearn import linear_model
regr = linear_model.LinearRegression()
x = np.asanyarray(train[['TIT']])
y = np.asanyarray(train[['CO']])
regr.fit(x, y)

x = np.asanyarray(test[['TIT']])
y = np.asanyarray(test[['CO']])

print(regr.score(x,y))

plt.plot(x, regr.predict(x))
plt.scatter(x, y, color='red')
plt.show()
```

از آنجایی که فرآیند آموزش و تست مدل با هم مرتبط هستند و برای هر مدل جدید باید تسست را تغییر داد ترجیح میدهم که هردو را داخل یک سلول بنویسم و همزمان اجرا کنم. از آنجایی که مدل تک متغییره است میتوان داخل یک صفحه دو بعدی آن را دید پس سه خط آخر را اضافه میکنم که بعد از محاسبات منحنی بدست آمده به همراه داده های تست نمایش داده شوند.

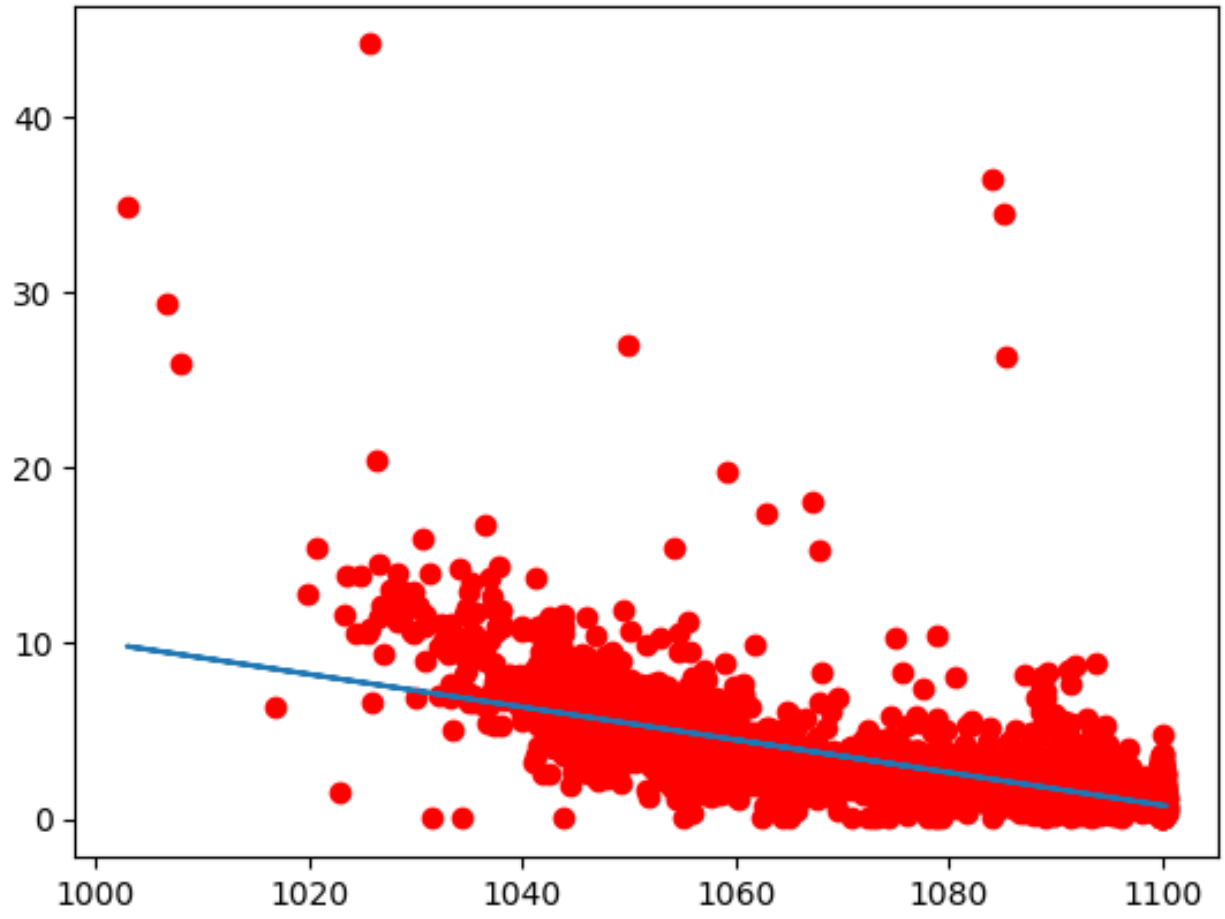

Fig-3 نمایش رابطه خطی بین CO-TIT

0.49 به عنوان معیار دقت تابع در پیشبینی رفتار بدست می آید. هرچه این عدد بیش تر شود یعنی مدل بهتر عمل میکند. برای اینکه از خطا هایی احتمالی پرهیز شود چند بار دیگر test و train را تایین میکنم (Cell-5) و با داده های جدید مدل را تعلیم میدهم. به ترتیب (0.526، 0.528، 0.518) تایین میشود. میتوان یک حلقه نوشت و داخل آن صدبار این کد را اجرا کرد و از همه اعداد میانگین گرفت تا عددی نسبتا دقیق بدست آید ولی من به همین داده ها اکتفا میکنم و میتوان گفت که مدل با دقت 50 درصد به بالا قابلیت پیش بینی دارد.

علاوه بر این موضوع حال که داده ها کمتر شده‌اند شکل منحنی با وضوح بیش تری دیده میشود. پس میتوان امیدوار بود که در ادامه از این متغییر حتی نتایج بهتری هم بگیریم.

2.2 رگرسیون خطی چندگانه با 'TIT', 'AFDP', 'GTEP', 'CDP', 'TEY

با وجود اینکه رابطه بین این متغییر ها شباهت چندانی به خط راست ندارد اما چندان بی ربط هم نیست. شاید سیستم ما ترکیبی متغییر با شد.

```
from sklearn import linear_model
regr = linear_model.LinearRegression()
x = np.asanyarray(train[['TEY', 'CDP','AFDP', 'GTEP', 'TIT']])
y = np.asanyarray(train[['CO']])
regr.fit(x, y)

x = np.asanyarray(test[['TEY', 'CDP', 'AFDP', 'GTEP' , 'TIT']])
y = np.asanyarray(test[['CO']])

print(regr.score(x,y))
plt.plot(test[['CDP']], regr.predict(x))
plt.show()
```

با کمک ویدیوهای جادی میفهمم که تفاوت چندانی بین کد رگرسیون ساده و چندگانه وجود ندارد. پس کد را از سلول بالا کپی میکنم با اعمال چند تغییر کوچک کد آماده است. نتیجه اصلا خوب نشد. به طور میانگین بعد از چهار بار اجرا 0.574 عددی است که میگیرم. حتی شصت درصد هم در نتایج ظاهر شد. خیلی خوبه. اما نموداری که داخل تصویر میبینم شبیه نقاشی یک بچه چهار ساله خط خطی است. احتمالا تصویر کردن یک نمودار چند بعدی روی صفحه دو بعدی این نتیجه را میدهد.

تصمیم میگیرم کمی هوشمندانه تر عمل کنم. داخل سلول بالا به ترتیب همه متغییر ها را میگذارم و میزان خطی بودن هرکدام را به ترتیب بررسی میکنم. میخواهم آنهایی که خطی تراند جدا کنم و بار دیگر فقط با آن ها تلاش بکنم. اگر حتی یک متغییر رابطه بدی داشته باشد میتواند کل مدل را خراب کند.

TIT: 0.563، GTEP: 0.339، AFDP: 0.233، CDP: 0.375، TEY: 0.396

در مقام مقایسه معلوم است که در AFDP این رابطه کمرنگ ترین و در TEY پررنگ ترین است. دوبار کد بالا را اجرا میکنم یک بار با همه به جز AFDP بار دیگر فقط با TEY و TIT . برای اولی 0.58 و برای دومی 0.53

که منطقی هم هست چون امتیاز متغییر ها تقریبا نزدیک به هم است پس با کم کردن متغییر بیش تر از اینکه از خطایمان کم شود از پیچیدگی و در نتیجه دقت مدل کم شده.

2.3 رابطه خطی چندگانه با 'AFDP', 'GTEP', 'CDP', 'TEY این بار به کمک Ridge و Lasso

داخل یکی از سایت ها خواندم که اگر بین متغییرهای مستقل رابطه خطی وجود داشته باشد باعث کاهش دقت مدل میشود ولی با کمک این دو الگوریتم میتوان خطا را کاهش داد. ممکن است اینجا هم همین باشد. اجرای این دو الگوریتم داخل پایتون به شدت آسان است و فقط کافی است یک خط را عوض کرد.

```
regr = linear_model.Ridge()
```

برای Lasso هم همین است. سلول جدیدی ایجاد نمیکنم تا شلوغ کاری نشود. با توجه به تجربیات قسمت قبل این سری فقط برای همه دیتاها باهم کد را اجرا میکنم.

نتیجه برای Ridge و Lasso به ترتیب 0.565 و 0.524 که ضعیف تر از الگوریتم رگرسیون خطی است.

2.4 فیت کردن تابع نمایی تک متغییره به کمک رگرسیون خطی برای 'TEY', 'CDP', 'GTEP', 'AFDP', 'TIT' هرکدام به صورت جداگانه

وقتی که داشتم برای اینکه چگونه میتوان یک رابطه غیر خطی مثل تابع نمایی را میتوان به دیتا ست کرد متوجه شدم که تابع بخصوصی برای اینکار وجود ندارد اما همیشه میتوان y را به گونه‌ای تغییر داد که متغییر مستقل یک ترکیب خطی از متغیر جدید Y باشد. مثلا در این مورد اگر به جای y لگاریتم طبیعی آن را بگذاریم در نهایت پس از انجام محاسبات به رابطه ($y = e^{ax+b}$) میرسیم.

```
from sklearn import linear_model
from sklearn.metrics import r2_score

indeVars = ['AFDP', 'GTEP', 'TIT', 'TAT', 'CDP', 'TEY']
testResults = []
regr = linear_model.LinearRegression()
def func(y):
    return np.log(y)

plt.figure(figsize=(30, 10))

for var in indeVars:
    x = np.asanyarray(train[[var]])
    y = np.asanyarray(train[['CO']])
    logY = func(y)

    regr.fit(x, logY)

    x = np.asanyarray(test[[var]])
    y = np.asanyarray(test[['CO']])
    logY = func(y)
```

```
    testResults.append(round(r2_score(logY, regr.predict(x)), 3))

    plt.subplot(2, 3, indeVars.index(var) + 1)
    plt.plot(x, regr.predict(x))
    plt.scatter(x, logY, color='red')
    plt.xlabel(var)
    plt.ylabel('CO')

print(testResults)
plt.show()
```

این بار یک حلقه نوشتم تا همه نتایجی که میخواهم در آن واحد بدست بیاید. ابتدا نام همه متغییر های وابسته را داخل یک لیست میریزم. تابع func برای این تعریف شده که بعدا بتوانم به راحتی از همین کد برای اجرای سایر توابع غیر خطی استفاده کنم. پایین تر مدل رگرسیون را تایین کردم و اندازه قابی که درآن نمودار ها نمایش داده خواهد شد. حلقه روی لیست پیمایش میکند و هر بار یک مدل برای متغییر تعلیم داده، نتیجه تست مدل را ذخیره کرده و نمودار خط بهمراه لگاریتم داده‌های تست را در یک subplot قرار میدهد.
نتیجه تست در نهایت در یک لیست با همان ترتیب داده های تست در خروجی به همراه نمودار نمایش داده میشود:

[0.263, 0.366, 0.485, 0.069, 0.412, 0.425]

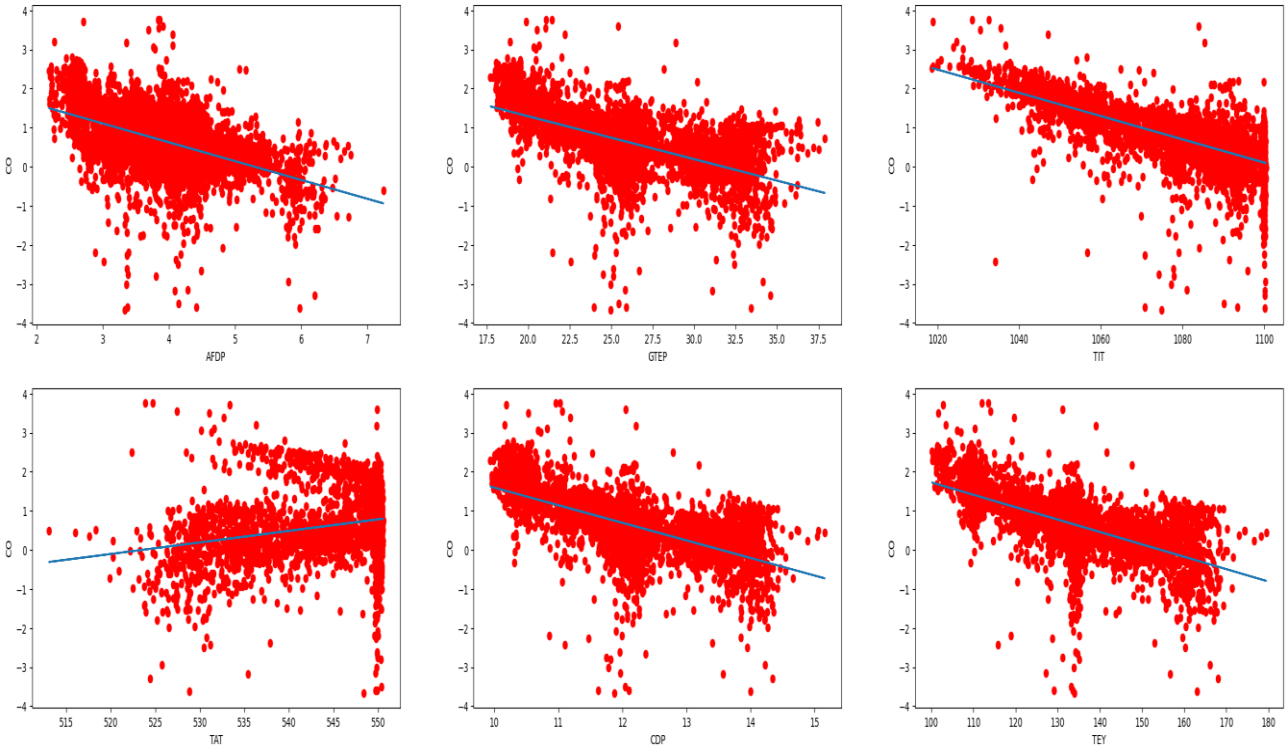

Fig-4 نمودار رابطه نمایی بین متغییرهای مستقل و CO

با وجود اینکه حدس اولیه‌ام بر این بود که توابع نمایی بهترین جواب را باید فراهم کنند اما بهترین نتیجه این قسمت از نتایج قسمت های قبلی کمتر شد!

2.5 فیت کردن تابع 1/x تک متغییره به کمک رگرسیون خطی برای 'AFDP', 'GTEP', 'CDP', 'TEY' هرکدام به صورت جداگانه 'TIT

کد برای این مدل آماده بود پس تصمیم گرفتم تغییر کوچکی در ترتیب بدهم. الان فقط کافی است که داخل func به جای لگاریتم طبیعی تابع معکوس y را بنویسیم. بعد از تبدیل نمودار تابع به (y = 1/ax + b) تبدیل خواهد شد.

[-0.0, 0.001, 0.0, 0.0, 0.001, 0.001]

نتیجه افتضاح شد! برخی از نتیجه صفر شده که یعنی نمودار ما نمیتواند رفتار را از یک حدس رندوم بهتر پیش بینی کند. البته که منفی هم داشتیم که یعنی بدتر از آن. به هر حال به نظر یک چیزی درست نیست وقتی به نمودار رسم شده نگاه میکنم به نظرم رفتار داده ها بسیار عجیب است. به جز چند عدد داده که دور از خط افتاده اند بقیه همه دور خط جمع‌اند.

البته که درستش هم همین است. اعداد یا بزرگ تر از یک اند که معکوسشان بین صفر و یک می افتد و یا کوچک ترند که از یک تا بینهایت جا میگیرند. و این نشان میدهد که تبدیل y به معکوسش کار عاقلانه ای نیست. در حقیقت اگر هیچ مقدار بین صفر و یکی نباشد قطعا میتوان یک خط با دقت خوبی از آن رد کرد! و این هیچ ربطی به منحنی نمودار ندارد.

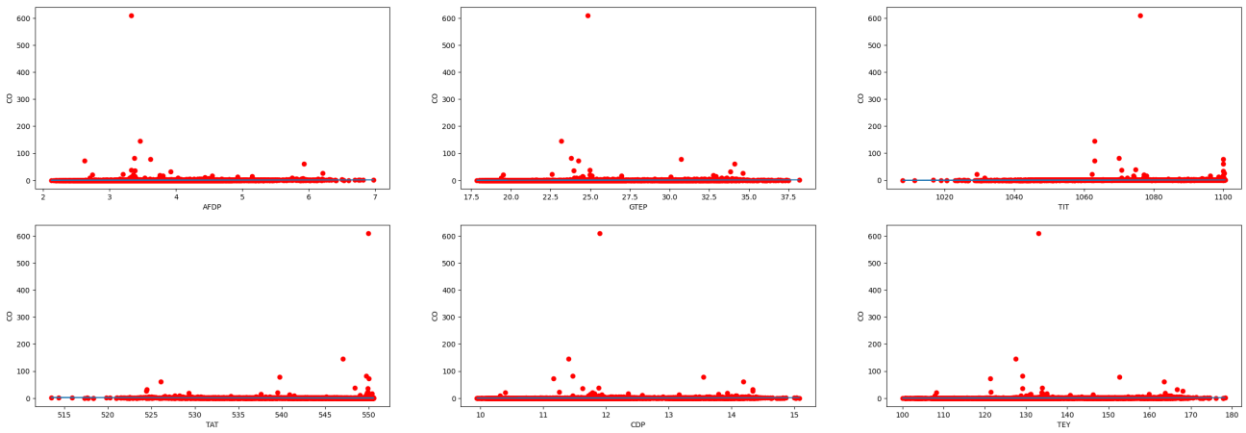

Fig-5 نمودار اولین تابع معکوس

پس راه حل چیست؟ دوباره جستجو در اینترنت را آغاز میکنم. از این سایت به آن سایت حتی از چت جی پی تی هم کمک میگیرم اما به نتیجه نمیرسم. تقریبا راه حلی برای مسئله وجود ندارد. چت جی پی تی هم راه حلی قبلی را پیشنهاد میدهد. کمی تفکر میکنم. با خود میگویم:« من به دنبال این هستم که ترکیب خطی y با 1/x را پیدا کنم. پس شاید بهتر است به جای تبدیل y ، x را به 1/x تبدیل کرد.»

[0.292, 0.41, 0.55, 0.004, 0.41, 0.444]

بله نتیجه همانی شد که میخواستم اکنون حتی میتوان دید که خط چطور جریان را دنبال میکند. البته یک مشکلی وجود دارد برخی خطوط در وسط ضخیم تر از طرفین به نظر میرسند. اما من فرصت اهمیت دادن بدان

را ندارم. با وجود اینکه این نمودار با دقت 55 درصد توانست رفتار را پیش بینی کند اما هنوز روش های خطی با یک و چند متغییر بهتر عمل کردند.

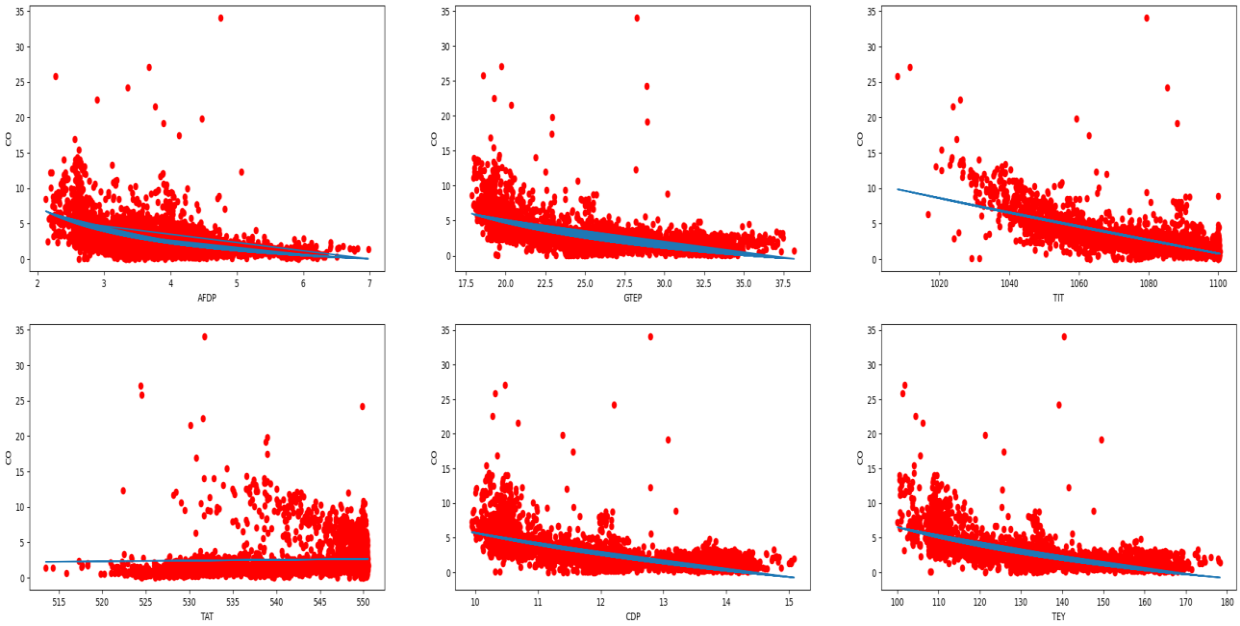

Fig-6 نمودار حقیقی تابع معکوس

یکی از ایرادات تابع 1/x این است که نمیتوان از آن به گونه ای استفاده کرد که منحنی های مختلف با رفتارهای مختلف را پیش بینی کرد. شکل منحنی تقریبا ثابت است و زیاد تغییر نمی کند. اما در نقطه مقابل استفاده از چند جمله‌ای های درجه بالا یا توابع سینوس و کسینوسی این انعطاف پذیری را در اختیار ما میگذارد که هر نوع داده‌ای را پیش بینی کنیم. شاید به همین خاطر است که بیشتر منابع داخل اینترنت راجع به این توابع بود. به همین دلیل از آموزش مدل چند متغییره این تابع صرف نظر میکنم.

2.6 فیت کردن تابع نمایی چند متغییره به کمک رگرسیون خطی برای 'TEY', 'CDP', 'GTEP', 'AFDP', 'TIT'

از نتایج قسمت 2.4 میتوان فهمید که رابطه نمایی TAT بسیار ضعیف است پس آن را وارد محاسبات خود نمیکنیم. نتیجه به طور میانگین 0.524 شد که با وجود اینکه از همه نمایی های به تنهایی بهتر بود اما بازهم بهترین روشی نبود که میشد از آن استفاده کرد.

2.7 فیت کردن تابع خطی درجه دو و سه تک متغییره برای "TIT', 'AFDP', 'GTEP', 'CDP', 'TEY هرکدام به صورت جداگانه

میتوان با کمک همان الگوریتم رگرسیون خطی چند متغییره توابع درجه 2 و 3 را به تابع ربط داد. این مهم با کمک PolynomialFeatures حاصل میشود. ایده این است که به جای متغییر وابسته دوم و سوم بیاییم و توان های دو و سه همان متغییر اولی را بگذاریم.

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn import linear_model
from sklearn.metrics import r2_score

indeVars = ['AFDP', 'GTEP', 'TIT', 'CDP', 'TEY']
testResults = []
degree = 2

plt.figure(figsize=(30, 10))

for Data in indeVars:
    x = np.asanyarray(train[[Data]])
    y = np.asanyarray(train[['CO']])


    poly = PolynomialFeatures(degree=degree, include_bias=False)
    polyX = poly.fit_transform(x.reshape(-1, 1))

    regr = linear_model.LinearRegression()
    regr.fit(polyX, y)

    x = np.asanyarray(test[[Data]])
    y = np.asanyarray(test[['CO']])
    sortedX = np.sort(x, axis=0)


    polyX = poly.fit_transform(x)
    sortedPoly = poly.fit_transform(sortedX)

    testResults.append(round(r2_score(y, regr.predict(polyX)), 3))

    plt.subplot(2, 3, indeVars.index(Data) + 1)
    plt.plot(sortedX, regr.predict(sortedPoly))
    plt.scatter(x, y, color='red')
    plt.xlabel(Data)
    plt.ylabel('CO')

print(testResults)
plt.show()
```

مانند قبل یک indeVars تعریف شده که تمام دیتای مستقل که قرار است نمودار برحسب آن رسم شود در آن ذخیره شده. testResults نتایج تست هر مرحله را ذخیره سازی میکند و degree درجه نمودار را تایین میکند.

Plt.figure() اندازه قابی که در آن نمودارها نمایش داده میشود را برحسب اینچ تایین میکند.

حلقه for روی indeVars پیمایش میکند و هر دوره اسم متغییر وابسته را در Data نگه میدارد. با کمک همین اسم x و y برای تست و آموزش استخراج میشود. PolynomialFeatures تابعیست که متغییر وابسته را گرفته و به صورت ماتریسی از توان 1،2،0 و.... در می‌آورد. سپس polyX توسط رگرسیون به y فیت میشود.

بعد از اجرا کردن متوجه میشوم که این بار هم نتیجه به صورت خط خطی ظاهر شده. حدس اولیه ام بر این است که داده ها داخل دیتای اولیه ترتیب مشخصی ندارند به همین علت وقتی نمودار x را رسم میکنم ممکن است خط ابتدای بازه به انتها متصل کند و برگردد وسط و دوباره و دوباره. در نتیجه اگر بخواهم یک خم پیوسته داشته باشم یا باید یک آرایه از اعداد پشت سرهم تشکیل دهم و پلات کنم یا همین x ای که الان دارم را قبل از نمایش مرتب کنم. از روش دوم استفاده میکنم و نمودار درست میشود اما میزان خطای مدل ها به طرز قابل توجهی افزایش پیدا میکند.

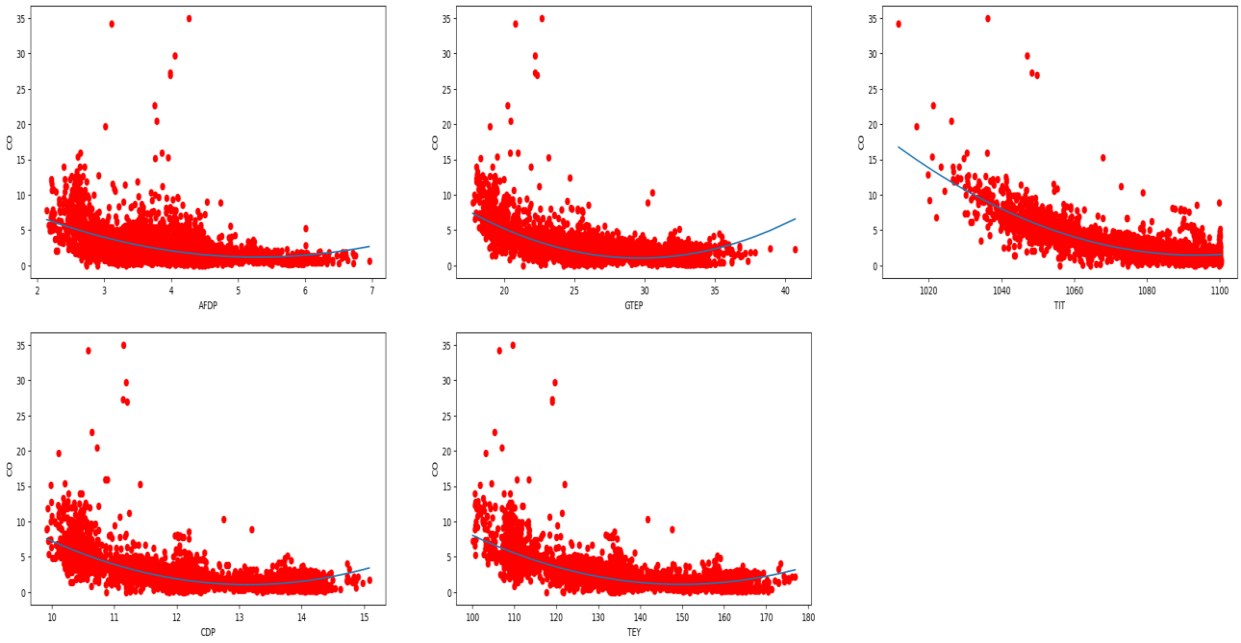

fig-7 نمودار درجه 2 تک متغییره CO

مشکل از آنجایی ست که x و y با هم رابطه یک به یک دارند که این رابطه فقط از روی ترتیب آرایه ها شکل گرفته و وقتی من x را sort میکنم و y را نه ویا حتی هر دو را به صورت جداگانه sort میکنم نتیجه این میشود که مدل به دیتا فیت نمیشود. به همین خاطر sortedX را میسازم که بتوانم با کمک آن نمودار را رسم کنم.

[0.247, 0.419, 0.609, 0.435, 0.455]

نتایج بدست آمده بسیار امیدوار کنندست علی الخصوص برای TIT خط به زیبایی جریان داده ها را دنبال میکند. تابع درجه دو مانند نعل اسب است و وقتی نقطه بحرانی را رد میکند تغییر رفتار میدهد. اما ظاهر داده

ها یک رفتار اکیدا نزولی را برمیتابد. به همین خاطر شاید اگر از چند جمله‌ای درجه 3 استفاده کنیم بتوانیم بهترین نتیجه را بگیریم.

[0.261, 0.432, 0.628, 0.442, 0.468]

نتایج بدست آمده برای یک مجموعه داده های تست و آموزش پس میتوان با هم دیگر مقایسه کرد و دید که مقدار اندکی بهبود داشته ایم. اما سوالی که پیش می‌آید این است که آیا این بهبود مفید بوده یا باعث overfitting شده است؟ همان طور که قبلا گفتم از آنجایی که توابع درجه 3 رفتار اکید دارند گزینه مناسب تری به نظر میرسند. البته تولید مقدار منفی co منطقی نیست و از طرفی اینکه برای مقادیر بالا دوباره افزایش میزان co داشته باشیم هم با عقل جور در نمی آید. از این دیدگاه بهترین رفتار باید از نمایی سر بزند. از طرفی میتوان اینگونه استدلال کرد که مقادیر بسیار بالا برای متغییر های مستقل غیرممکن است پس نباید نگران این بود که در مقادیر بالاتر چه رفتاری را شاهد خواهیم بود.

من استدلال دوم را میپذیرم و از نظرم بهترین مدل تا به اینجا همین مدل چند جمله ای درجه 3 بوده.

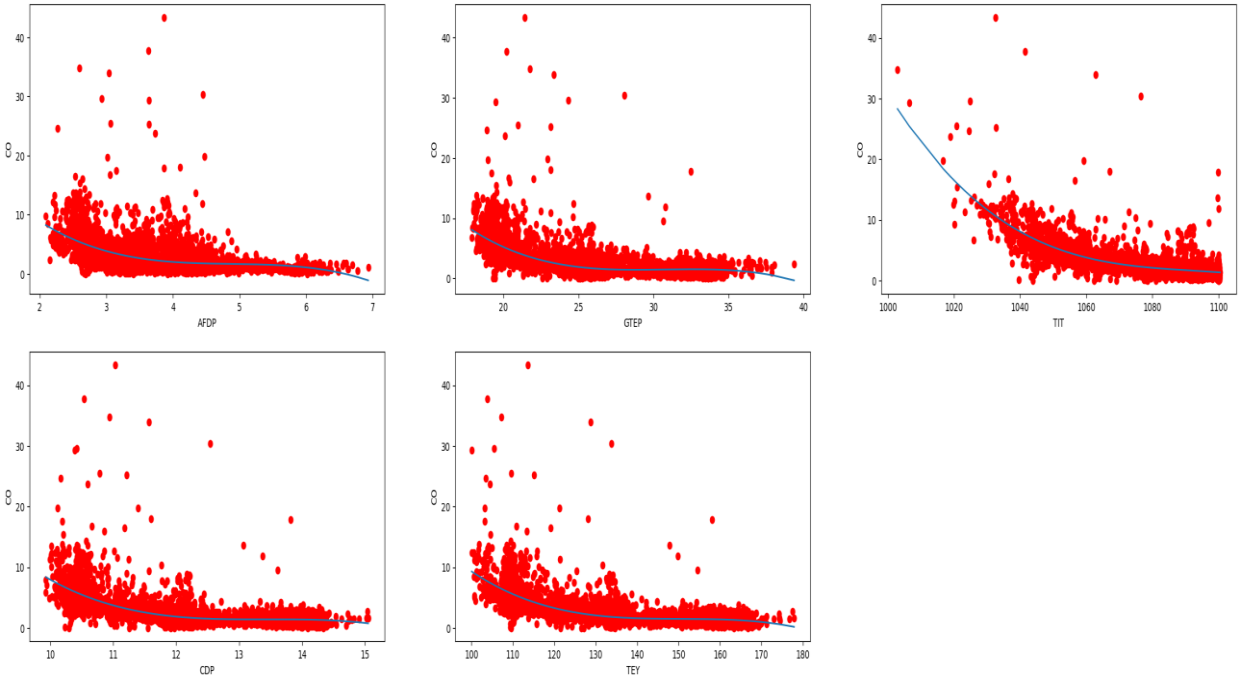

Fig-8 نمودار درجه 3 تک متغییره برای CO

2.8 فیت کردن تابع خطی درجه سه چند متغییره برای "TIT', 'AFDP', 'GTEP', 'CDP'

از آنجایی که رابطه TEY با CO خیلی ضعیف تر از بقیه است تصمیم میگیرم که در محاسباتم نادیده بگیرمش.

از طرفی درجه 3 در تک متغییره کار بهتری انجام داده بود پس معقول تر این است که فقط همین مدل را امتحان کنم.

0.667 نتیجه کار شد. حتی من 70 درصد هم توانستم از این مدل بگیرم.

2.9 استفاده از شبکه عصبی

من یک متخصص شبکه عصبی نیستم ولی اطلاعات پایه‌ای در این زمینه دارم و میدانم که مدل های مختلفی برای طراحی یک شبکه عصبی وجود دارد. بهترین کاری که به ذهنم رسید این بود که در اینترنت گشتم و سایتی را پیدا کردم که توانسته بود مسئله مشابهی را حل کند. همان داده ها را به طور کامل کپی کردم و داخل محیط پایتون اجرا کردم مدت زیادی طول نکشید که نتیجه 0.342 را گرفتم.

داخل هر شبکه عصبی تعدادی لایه وجود دارد و هر لایه تعدادی نورون دارد. هر نورون یک تابع activation دارد. این تابع تایین میکند که مقداری که نورون بدست می آورد اگر از یک حدی کمترباشد وارد محاسبات لایه بعدی نشود. از آنجایی که این جزو مباحث تخصصی است و با کمی تحقیق مشخص شد که تابع relu طرفدار زیادی دارد و برای لایه خروجی حتما باید linear باشد من این قسمت را تغییر ندادم.

تعداد لایه ها در ابتدا سه‌تا بود. یک لایه ورودی، یک لایه پنهان و یک لایه خروجی. لایه ورودی و خروجی هرکدام یکی و پنهان 32 نورون داشتند. برای ورودی من 5 نورون گذاشتم که هرکدام به یکی از متغییر ها مربوط شود.

'TIT', 'TEY', 'CDP', 'GTEP', 'AFDP' ویژگی مشترکشان این است که همگی رابطه معنی داری با CO داشتند.

برای لایه آخر هم همان یک نورون باید باشد چون یک خروجی بیش تر نداریم. (در مسائل classification چند نورون در لایه خروجی قرار دارد که هرکدام به یک دسته خاص تعلق دارند.) اما لایه پنهان جایی است که میتوان دست برد و تغییر ایجاد کرد.

بعد از اینکه شبکه عصبی تشکیل شود داخل model ذخیر میشود. Model به همراه یک optimizer و loss کامپایل میشود. چون مسئله ما از جنس رگرسیون است و در نهایت هم قرار است دقت تابع با MSE سنجیده شود بهترین تابع برای اینکه در هر دوره میزان خطا محاسبه شود همین تابع است. با وجود اینکه optimizer های مختلفی ارائه‌شده بازهم به خرد جمعی اتکا میکنم و تابع adam که از ابتدا داخل کد بوده و در منابع مختلف دیگر هم مورد استفاده قرار گرفته را انتخاب میکنم.

پس از compile مدل توسط ()fit. آموزش داده میشود. متغییرهای وابسته و مستقل به خورد دستگاه داده میشود. تعداد دفعاتی که مدل روی تمام داده ها عملیات انجام میدهد با epcho و گنجایش دسته هایی که پس از انجام عملیات دستگاه بازمیگردد و متغییر های نورون ها را از سر نوع تعریف میکند با batch-size نشان داده شده.

طبعا هرچه epcho بیش تر باشد جواب نهایی مدل دقیق تر است و زمان بیش تری هم برای رسیدن به نتیجه طول میکشد. هرچه batch-size بزرگ تر باشد مموری بیش تری برای انجام عملیات لازم است اما دقت انجام کار در هر دوره بیش تر و زمان انجام کل عملیات کم تر میشود. از طرفی هرچه batch-size کمتر باشد در نهایت نتیجه بهتری خواهیم گرفت.

در ابتدا من batch-size=10 و epcho=100 را مقادیر کمی تایین میکنم تا به سرعت جواب بدست بیاید. این کار برای آن است که معماری های مختلف را با هم دیگر مقایسه کنم و ببینم از کدام میتوانم نتایج بهتری بگیرم. نکته اینجاست که در این شرایط اختلافات بین عددهایی که یک مدل در دو نوبت متفاوت با همان مجموعه داده میگیرد بسیار زیاد است و تقریبا با عقل جور در نمی‌آید.

پس تصمیم گرفتم که epcho را زیاد کنم و با وجود اینکه زمان زیادی طول میکشد اما از نتایج آزمایشم بتوانم بهتر یاد بگیرم. یکی از آزمایشاتی که انجام دادم batch-size = 30000 در مقابل 10 بود برای epcho=1000 . یکی تقریبا تمام داده هایم را پوشش میداد و دیگری حتی از یک صدم مجموعه داده ها خیلی کم تر بود. نکته جالب این بود که مجموعه بزرگ تر با خطای بسیار زیاد شروع میکرد و در هر مرحله خطا را کاهش میداد. ولی وقتی به اواخر کار میرسید این مقدار کاهش خطا کمتر و کمتر میشد تا جایی که دیگر به سختی قابل احساس بود. اما batch-size کم در نقطه مقابل رفتاری رفت وبرگشتی داشت در یک قدم خطا زیاد میشد قدم بعدی کم. و این از الگوی خاصی پیروی نمیکرد. نکته جالب اینجاست که رفتار رندوم مقدار کم گاهی اوقات بهتر جواب میداد. پس تصمیم گرفتم batch-size را در حدود 100 فیکس کنم که مقداری از رفتار رفت و برگشتی کم تر شود و مقدار زیادی از آن تغییرات کم در انتهای کار.

یکی از نتایج ارزشمندی که بدست آوردم این بود که اضافه کردن لایه بیش تر از اضافه کردن نورون بازدهی را افزایش میدهد در حالی که اضافه کردن نورون به طرز قابل توجهی فرایند را کند میکند. از طرفی اضافه کردن لایه یک محدودیتی دارد و از 5 عدد به بالا لایه‌های بیش تر دردی دوا نمیکنند. تعداد نورون ها هم از 200 تا به بالا زیاد بازدهی ندارد.

تعداد epcho هم بعد از آزمایشات فراوان روی 9000 فیکس میکنم عددهای بالاتر تغییر چندانی را حاصل نمیکند.

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.layers import Dense, Input
from time import time

start = time()
model = tf.keras.Sequential([
    Input(shape=(5,)),
    Dense(32, activation='relu'),
    Dense(64, activation='relu'),
    Dense(128, activation='relu'),
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(1, activation='linear')
])

X_train = np.array(train[['TIT', 'TEY', 'CDP', 'GTEP', 'AFDP']])
Y_train = np.array(train[['CO']])
X_test = np.array(test[['TIT', 'TEY', 'CDP', 'GTEP', 'AFDP']])
Y_test = np.array(test[['CO']])
```

```
model.compile(optimizer='adam', loss='mean_squared_error')

model.fit(X_train, Y_train, epochs=9000, batch_size=300)

test_loss = model.evaluate(X_test,  Y_test)
end =time()

print(end-start)
print(f"Test Loss: {test_loss:.4f}")
```

نتیجه کار شد این ترکیبی که احتمالا بهترین کارایی را دارد. با این مدل توانستم 0.66 بگیرم. با توجه به سختی کار و مدت زمان طولانی که برای آموزش مدل نیاز است () نتیجه خوبی نبود. اما شاید اگر یک فرد متخصص با این مسئله سروکار داشت میتوانست نتایج بهتری بگیرد.

2.10 استفاده از DECISION TREE برای کل متغییر هایی که ظاهرا رابطه‌ای با CO دارند به صورت یکجا

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score

colNames = ['AFDP', 'GTEP', 'TIT', 'TAT', 'CDP', 'TEY']
testResults = []

for Data in colNames:
    trainX = train[[Data]]
    trainY = train[['CO']]
    testX = test[[Data]]
    testY = test[['CO']]

    regr = DecisionTreeRegressor(random state = 0)
    regr.fit(trainX, trainY)

    testNum = r2_score(testY, regr.predict(testX))
    testResults.append(round(testNum, 3))

print(testResults)
```

از همان اول هم امیدی به این الگوریتم نداشتم ولی میخواستم ببینم نتیجه چه خواهد شد.

[0.274 ,0.308 ,0.29- ,0.547 ,0.157 ,0.169-]

3 PREE-PROCESSING برای NOX

انجام کار برای NOX به مراتب آسان تر است. فقط کافی است برگردم به کدهای قبلی و تمام CO ها را جایگزین کنم. نمودار رسم کنم ببینم کدام الگوریتم مناسب است و جواب را بدست بیاورم. نمودار را که رسم میکنم هیچ چیزی مشخص نیست. انگار هیچ ارتباطی وجود ندارد. با توجه به تجربیات قبلی ابتدا همه نمودار ها را در قاب 20 در 20 به صورت جداگانه رسم میکنم. برای اینکه بازهم بتوانم الگو را راحت تر پیدا کنم فقط 20

درصد داده ها را پلات میکنم تا خلوت تر باشد. این کار را چندبار تکرار میکنم تا مطمئن شوم به بخش بخصوصی از مجموعه کل داده ها نگاه نمی اندازم.
واضح است که AP فقط پراکندگی داده را نشان میدهد و هیچ خمی را نشانگر نیست. برای AH همین نظر را دارم ولی با قطعیت کمتری. برای AT یک جریان بسیار پراکنده منحنی شکل وجود دارد. AFDP شکل تابع درجه دو رو به بالا دارد اما مانند AT خیلی پراکنده است. GTEP, CDP, TEY, TAT مثل دسته هایی از خطوط عمود بر محور متغییر وابسته است.
برای اینکه ببینیم آیا واقعا این رفتار یک الگو است و یا فقط پراکندگی طبیعی داده ها را نشان میدهد به یک نکته باید توجه کرد. اگر بین متغییر x و y هیچ رابطه ای نباشد پس هرجا x چگال تر است تعداد نقاط باید بیش تر باشد و نقاط باید در راستای محور y پراکنده شده باشند. چون احتمال اینکه مقادیر کمتر شایع y در این نواحی حضور داشته باشند بیش تر از بقیه قسمت هاست.
با توجه به استدلال صفحه قبل به نمودار hist داده ها نگاه می اندازیم. برای AH و TAT این نمودار از چپ به راست زیاد میشود و در اوج ناگهان تمام میشود. برای NOX این نمودار تقریبا یک منحنی توضیع نرمال حول 60 است.
اگر به شکل نمودار Scatter این دو تابع نگاه کنیم مانند یک مثلث است که قاعده آن سمت چپ قرار دارد. این همان چیزیست که میگفتم.
برای CDP و TIT و GTEP نمودار پراکندگی سه پیک دارد. داخل نمودار داده ها هم دقیقا سه خط عمودی مشاهده میشود. پس احتمالا ارتباطی بین این متغییر ها نیست.

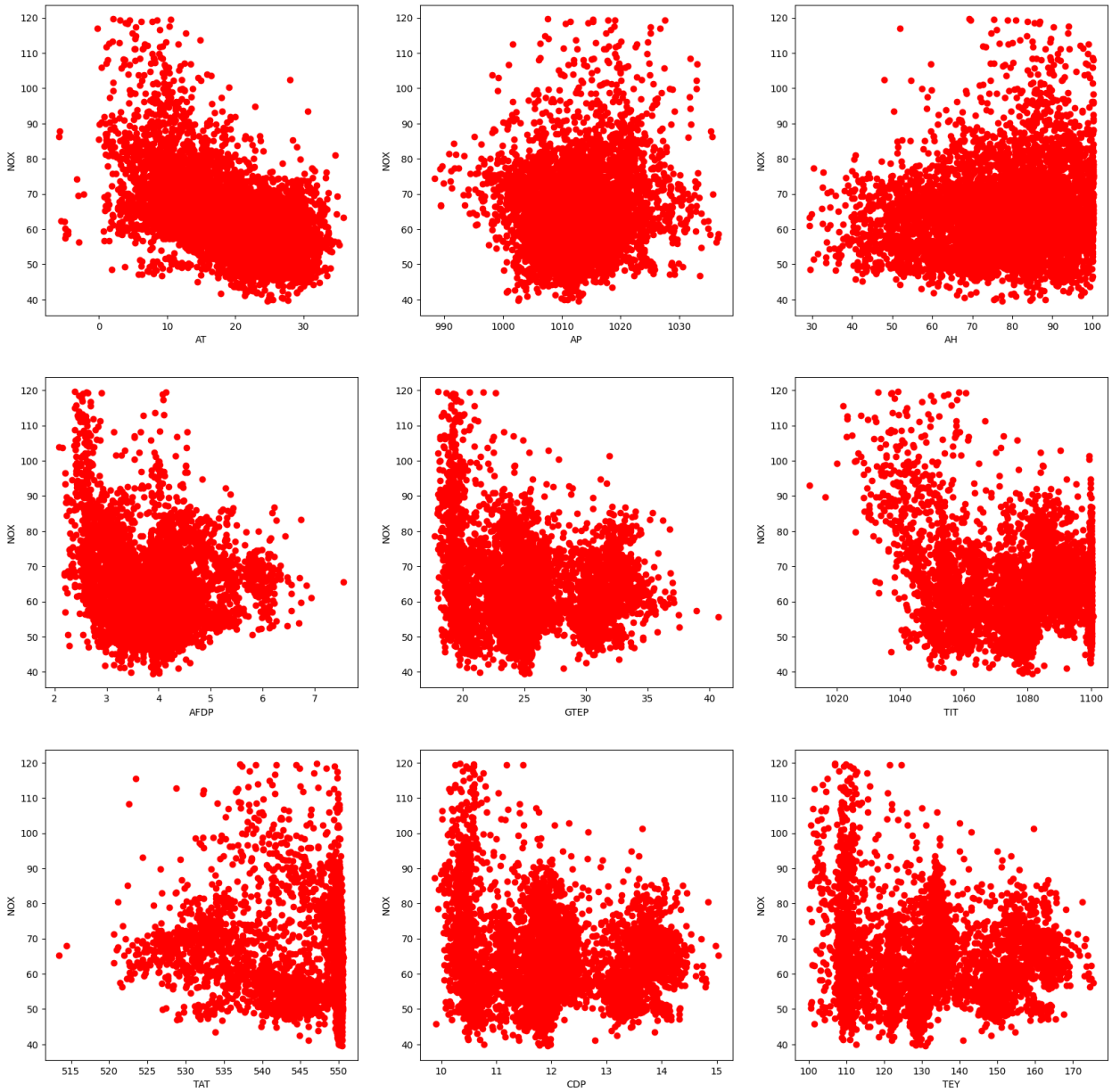

Fig-9 نمودار NOX نسبت به تمام متغییر های وابسته فقط 20% داده ها

تراکم داده برای TIT در انتهای بازه بیش تر است. اما به نسبت داده ها در این ناحیه پراکندگی کمتری دارند. علاوه برآن اینکه مقادیر بزرگ تر NOX در مقادیر کمتر TIT رخ داده میتواند معنی دار باشد.

برای AFDP بین 2.5 و 5 نوک قله بیش ترین داده هاست پس باید در این قسمت خط عمودی ببینیم. اما نسبتا در این نقطه تراکم داده حول یک نقطه مشخص داریم. ضمنا اینکه مقادیر زیادی از NOX های کلان در ابتدای بازه رخ داده اند هم نمیتواند بی ربط باشد. با این حال ارتباط بین AFDP نسبت به AT و TIT کمتر است.

به نظر می رسد که سیستم بسیار رندوم است و به سختی میتوان رفتار آن را پیش بینی کرد. مسلما نمیتوانم خطی به این نمودار ها فیت کنم. پس ترجیه می دهم که از شبکه عصبی برای پیش بینی رفتار تابع استفاده کنم.

4 PROCESSING برای NOX

تمام کد را از بالا کپی و داخل سلول جدیدی پیست میکنم. این کار را به این خاطر انجام میدهم چون آموزش دادن یک شبکه عصبی کار زمان بریست و اگر خواستم یک مدل از NOX آموزش بدهم بهتر است که مدلی که برای CO آموزش داده ام را نابود نکنم.
الان فقط کافی است متغییر های مستقل و وابسته را تایین کنم. مدل را فقط بر اساس AT و TIT و AFDP که رابطه مشهودی با NOX دارند آموزش میدهم. درغیر این صورت قطعا overfit کرده ام.
پس از آموزش و تست به نتیجه عجیبی میرسم. تابع MSE مقدار 52.2657 را نشان میدهد که برای این کار خطای فاجعه آمیزی است. اما تابع R2-SCORE خطای 0.69 را نشان می دهد که خطای مناسبیست. علت این تفاوت در نحوه اندازگیری خطای این دو تابع نهفته است.
تابع MSE اختلاف داده واقعی با مقدار پیش بینی شده را به توان دو میرساند و همه مقادیر را برای تمام نقاط جمع میزند. مقدار زیاد این تابع نشان دهنده این است که پراکندگی داده ها حول جوابی که ما بدست می آوریم زیاد است و این مدل به خوبی نمیتواند عدد دقیقی پیش بینی کند.
از طرف دیگر تابع R2-SCORE مقدار واریانس جواب را با واریانس میانگین تمام نقاط میسنجد و به ما میگوید که پیش بینی ما چقدر بهتر از میانگین عمل میکند.
در کل جوابی که بدست آمده عدد دقیقی به ما نمی دهد که این از همان ابتدا مشهود بود. اما به مراتب بهتر از این است که برای شرایطی که هیچ اطلاعاتی از آن نداریم بخواهیم پیش بینی را به میانگین داده ها واگذار کنیم.