به نام خدا

Shahid Beheshti University

دانشکده ریاضی و علوم کامپیوتر

(School of Mathematics and Computer Science)

داده کاوی

(Data Mining)

دکتر فراهانی

(Dr. Farahani)

تمرین شماره سه

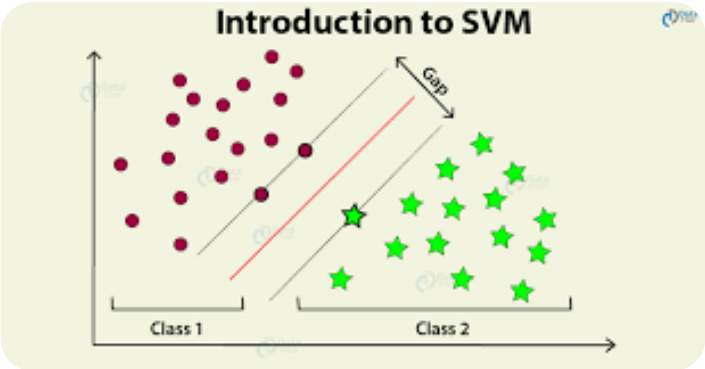
3rd Computer Assignment

محمدرضا بالاخانیان

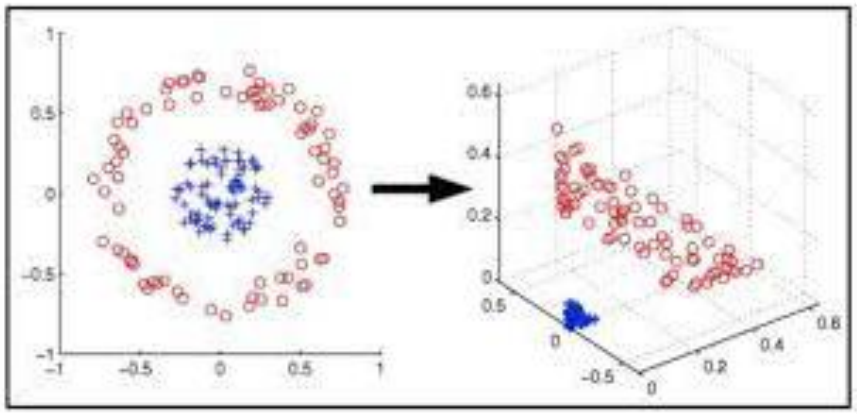
۹۹۴۲۲۲۰۲

بهار ۱۴۰۰-۹۹

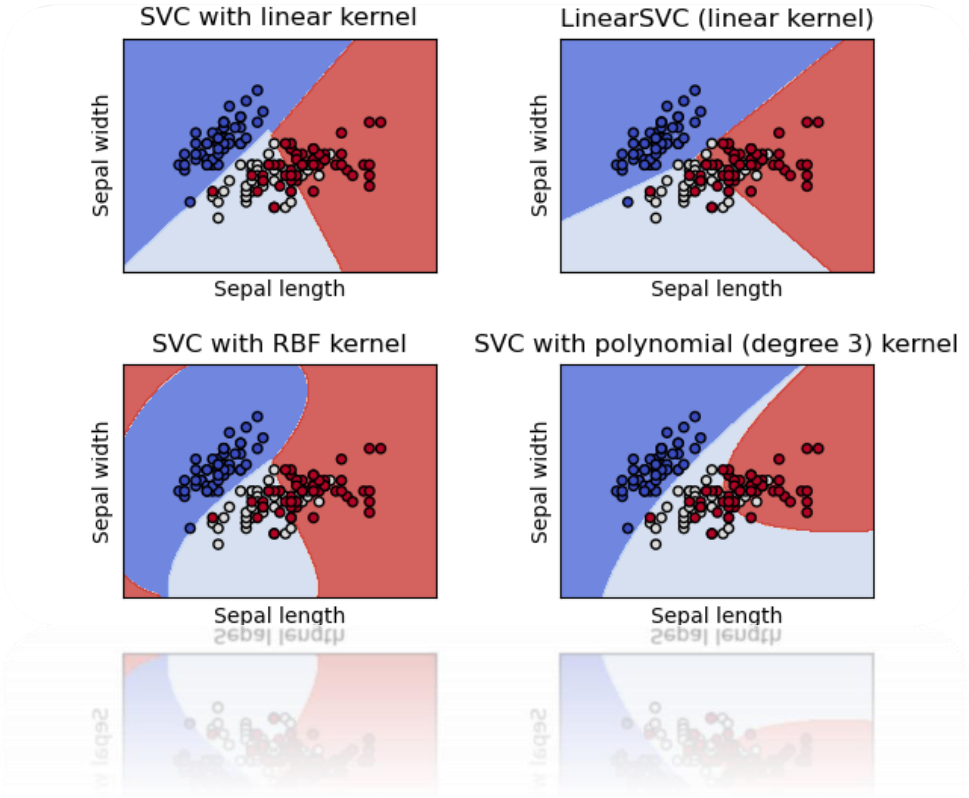
۱. هدف ماشین‌های بردار پشتیبان در واقع ایجاد یک هایپرپلین(یا خط در حالت دوبعدی) بین دادگان است؛ بطوری‌که بتواند دو کلاس را به صورت کلی از یک دیگر جدا نماید؛ یعنی چیزی همانند تصویر زیر:



اما اگر دادگان به این راحتی و توسط یک خط مستقیم قابل تفکیک نباشد، همانند تصویر زیر، چه باید کرد؟



می‌بینید که دادگان توسط یک خط معمولی قابل تکیک نیست. برای همین کاری که می‌کنند دادگان را به بعد بالاتری تبدیل می‌کنند تا بتوانند آن را با یک صفحه از یکدیگر تفکیک نمایند که کاری بسیار زمان‌بر است. اینجاست که ایده کرنل مطرح می‌شود که برای تفکیک دادگانی به صورت بالا، کاملاً مستقیم عمل می‌نمایند و دادگان را به فضای با ابعاد بیشتر منتقل می‌کنند.



هر یک از کرنل‌ها به روش خاص (فرمول متفاوتی برای ضرب) عمل می‌کنند. و هر یک کاربرد متفاوتی دارند:

۱.۱. چندجمله‌ایی بیشتر برای پردازش تصویر کاربرد دارد چرا که می‌تواند برای اعمال توابع برروی تصاویر مورد استفاده‌قرارگیرد.

۱.۲. کرنل گوسی و **RBF** نیز زمانی مورد استفاده است که اطلاعات قبلی درباره دادگان وجو ندارد.

۱.۳. کرنل سیگموید و تانژانت هیپربولیک نیز به جهت ماهیت این توابع کاربرد فراوانی در شبکه‌های عصبی دارند.

۱.۴. توابع بسل نیز در زمان‌هایی به کار می‌روند که نمی‌خواهیم ضرب داخلی انجام شود.

۱.۵. توابع **spline** خطی نیز در دادگان تنک قابل استفاده‌است.

$$k(x,y) = 1 + xy + xy\ min(x,y) - \frac{x+y}{2}\ min(x,y)^2 + \frac{1}{3}\min(x,y)^3$$

۱.۶. تابع آنوا را نیز می‌توان در رگرسیون مورد استفاده قرار داد چه را که همانند آنوا در رگرسیون‌های خطی می‌باشد.

منبع:

https://data-flair.training/blogs/svm-kernel-functions/

۲. برای این کار از کتابخانه سایکیت استفاده می‌نماییم.

```
dataset = pd.read_csv("train.csv")

X = dataset.drop(columns=["price_range"])
y = dataset.price_range

X_train, X_test, y_train, y_test = train_test_split(X, y)

clf = SVC()
clf.fit(X_train, y_train)
accuracy_score(predict:=clf.predict(X_test),y_test) * 100

95.0

predict

array([2, 1, 0, 1, 3, 1, 0, 1, 2, 3, 1, 1, 0, 1, 1, 3, 1, 2, 3, 0, 1, 3,
       1, 2, 1, 0, 0, 2, 3, 3, 2, 0, 0, 3, 1, 0, 3, 2, 2, 1, 1, 2, 1, 0,
       1, 1, 1, 3, 3, 2, 3, 3, 1, 1, 1, 0, 3, 0, 1, 2, 0, 0, 3, 3, 2, 1,
       3, 2, 2, 2, 2, 0, 0, 0, 3, 0, 2, 2, 3, 2, 0, 2, 3, 2, 0, 1, 3, 3,
       0, 2, 3, 0, 2, 2, 2, 3, 2, 2, 1, 2, 3, 2, 0, 1, 2, 2, 3, 1, 3, 2,
```

۳. در پرسش قبلی از کرنل **rbf** استفاده‌شد که در ادامه از پارامترهای مختلفی استفاده‌می‌کنیم:

۳.۱. سیگموید: دقت: ۱۸.۶

دقت بسیار پایین‌تر بوده‌است که علت آن را می‌توان چندکلاسه بودن دادگان دانست چرا که در دامنه ۰ تا ۱ بوده و برای ۲ کلاس مناسب است. البته برای دو کلاسه دقت حدود ۵۰ درصد بوده است که چرایی آن عدم توزیع مناسب دادگان در دو کلاس می‌باشد.

۳.۲. کرنل خطی: دقت: ۹۵.۴

اگرچه دقت آن بسیار بالاست اما سرعت آن بسیار پایین است که در توضیحات سوال یک گفته شد و در اینجا بطور میانگین ۲۰۰ برابر کندتر از دیگر کرنل‌ها می‌باشد.

۳.۳. درجه در کرنل چندجمله‌ایی : دقت:۹۵

این پارامتر تنها برای کرنل چندجمله‌ایی مورد استفاده قرار می‌گیرد؛ و هرچقدر بیشتر باشد اورفیت خواهدشد. بنابراین باید از یک درجه بیشتر منجر به کاهش دقت در تست شود که در اینجا در ۸ بیشترین دقت را داشته و پس از آن دقت کاهش می‌یابد.

۳.۴. پارامتر بعدی **decision_function_shape** می‌باشد که مشخص می‌نماید مقایسه بصورت یک با یک باشد یا یک با بقیه؟ که در اینجا هردو را آزمودیم و در عمل تفاوتی مشاهده نشد.

```
{'o': {'poly': 95.22, 'rbf': 95.06, 'sigmoid': 18.039999999999996}, 'r': {'poly': 95.54, 'rbf': 94.60000000000001, 'sigmoid': 18.059999999999995}}
```

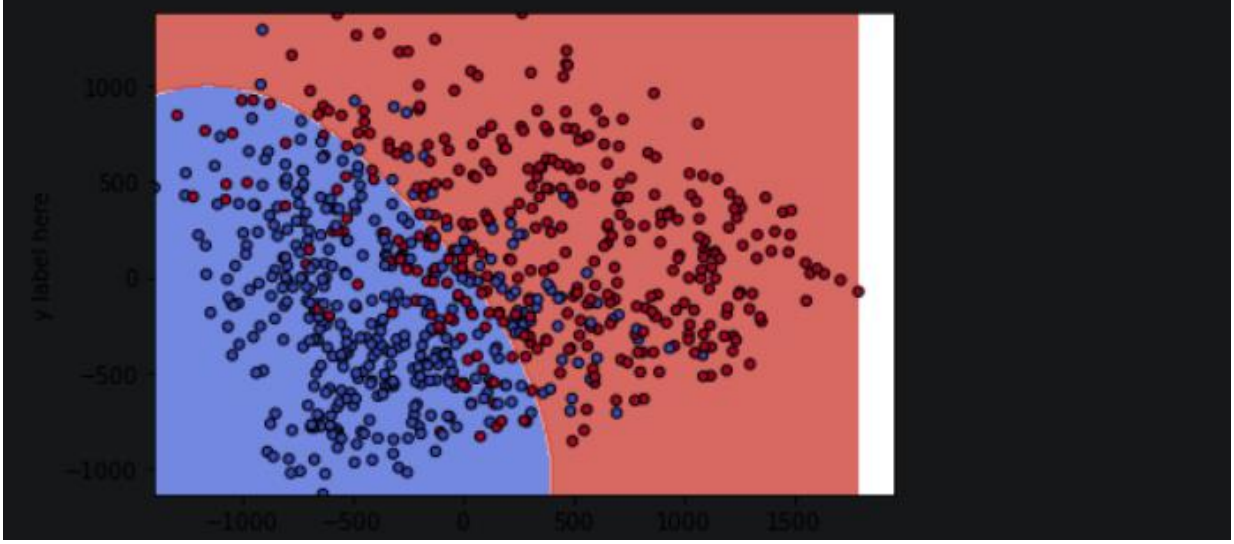
۳.۵. پارامتر دیگری که بررسی شد **break_ties** می‌باشد که در کرنل‌های مختلف و با شکل تصمیم متفاوت، تاثیر خاصی ایجاد نکرد و دقت به صورت میانگین برابر بود.

۴. این که حاشیه نرم باشد یا سخت توسط پارامتر **C** مشخص می‌گردد که هرچقدر بیش‌تر باشد به معنی سختی بیشتر و تفکیک بیشتر دادگان است و هرچقدر کمتر باشد سخت‌گیری کمتری انجام می‌شود و تفکیک چندان با دقت نیست: (توجه به علت تعداد بالای ویژگی‌ها از **PCA** برای کاهش ابعاد به تعداد ۲ جهت مشاهده‌پذیری استفاده‌نمودیم.)

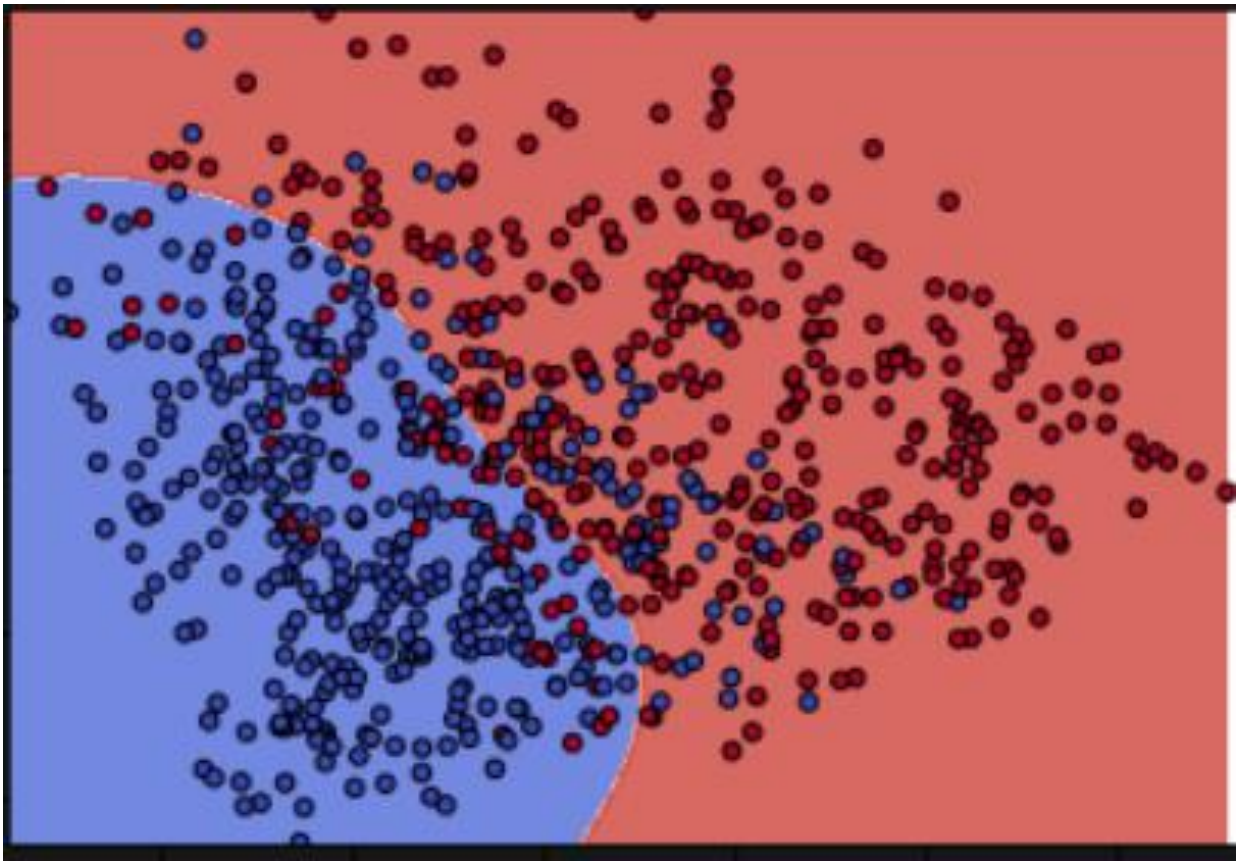
در ابتدا میزان پارامتر را برابر ۰.۰۱ قرار دادیم:

```
clf = SVC(kernel='rbf',C=0.01).fit(X_train, y_train)
fig, ax = plt.subplots()
title = ('Decision surface of linear SVC ')
X0, X1 = X_train[:, 0], X_train[:, 1]
xx, yy = make_meshgrid(X0, X1)
plot_contours(ax, clf, xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)
ax.scatter(X0, X1, c=y_train, cmap=plt.cm.coolwarm, s=20, edgecolors='k')
ax.set_ylabel('y label here')
ax.set_xlabel('x label here')
```

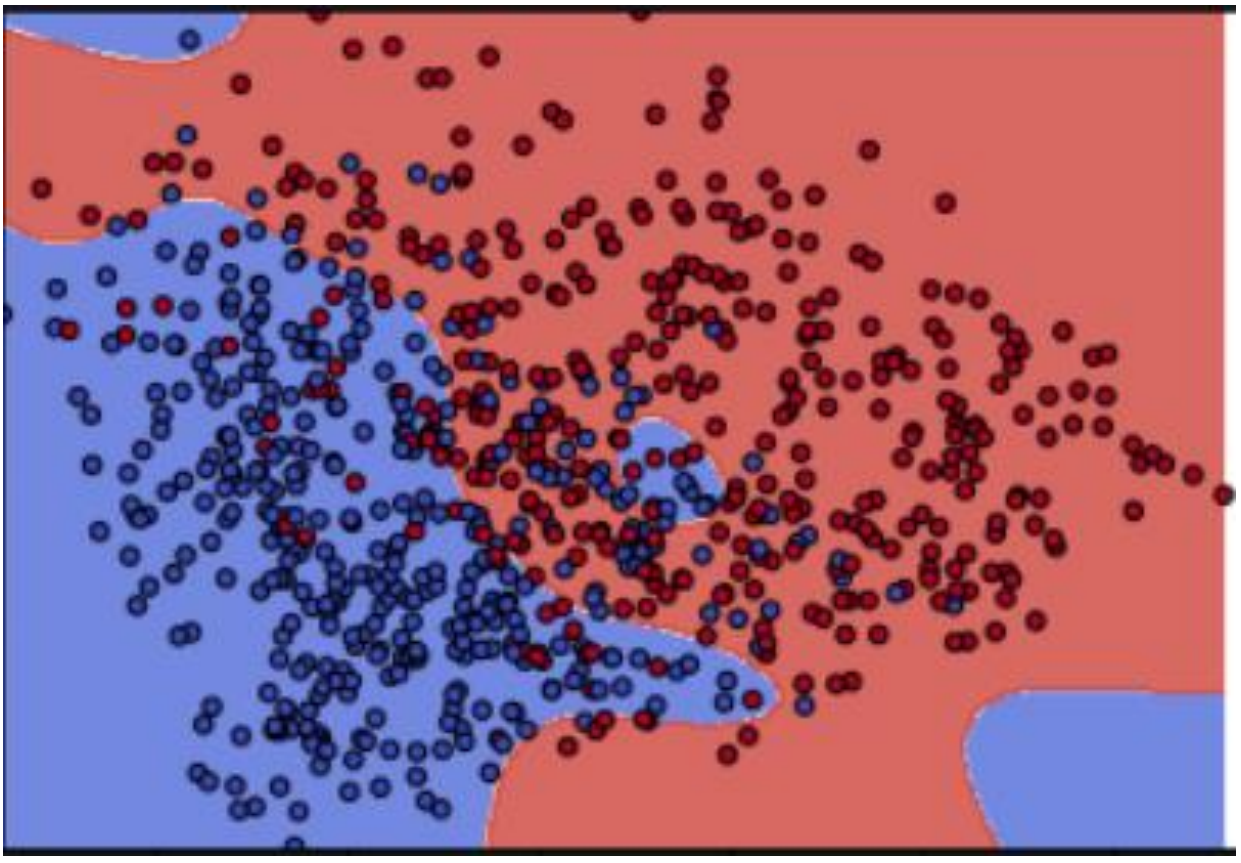
```
Text(0.5, 0, 'x label here')
```



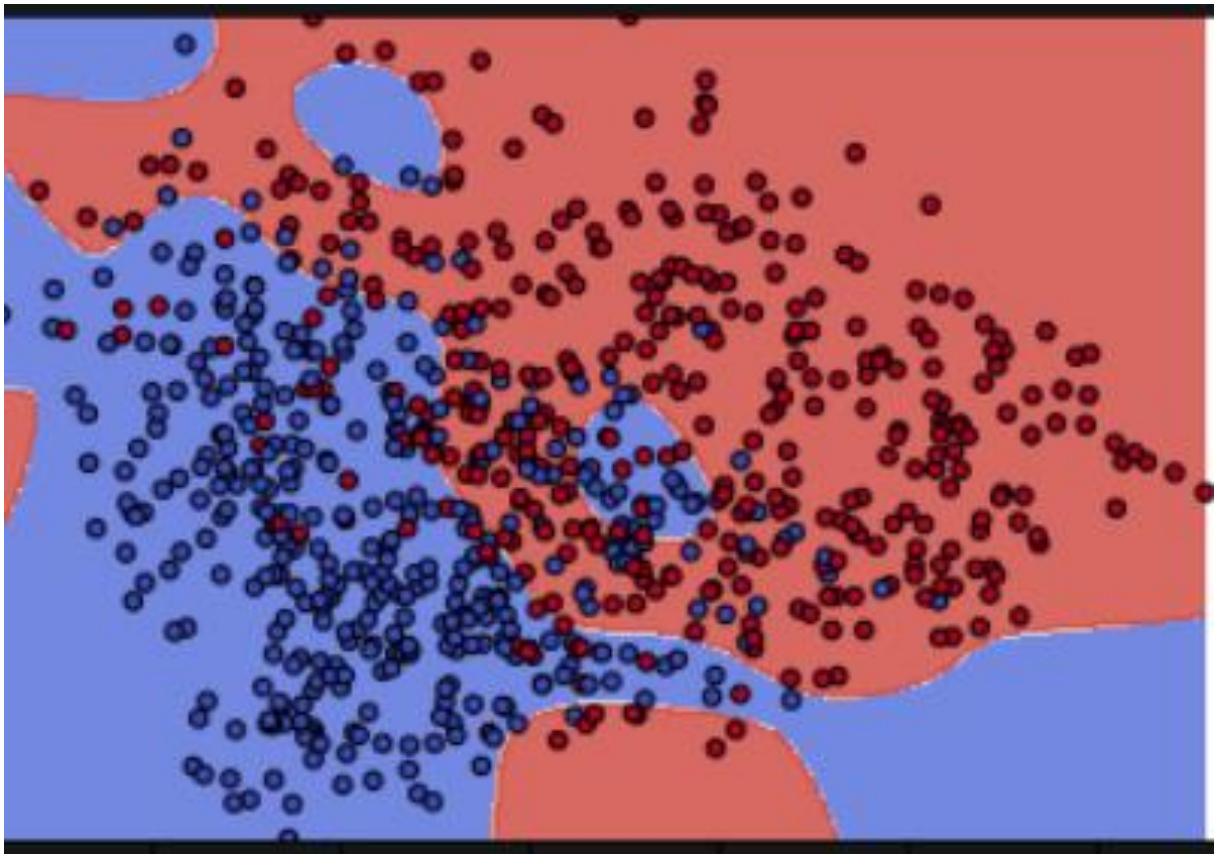
می‌بینید که تفکیک‌پذیری چندان بالا نیست و برای همین از مقدار ۱ برای پارامتر استفاده می‌نماییم:



باز هم همانند قبل است و



و شیوه تفکیک بسیار با قبل متفاوت است و می‌تواند بهتر تشخیص دهد.



در اینجا که تلاش‌کردیم میزان حاشیه را کاهش دهیم و کاملا به دادگان وفق دهیم؛ نتیجه بالا حاصل شد که دارای دقت بیشتری در دادگان آموزش خواهدبود. و تفکیک آن نیز بیش‌تر است.

۵.

الف) برای این کار از قطعه کد زیر استفاده‌شد:

```
def categorize(dataset, feature_name: str, num: int = None,bins = None):
        ds = dataset.copy()
        _max = ds[feature_name].max()
        _min = ds[feature_name].min()
        if type(num) == int:
            _bins = np.linspace(_min,_max,num+1)
            _lbls = list(range(num))
        else:
            _bins = [_min] + bins + [_max]
            _lbls = list(range(len(_bins)-1))
        return pd.cut(ds[feature_name],_bins,labels=_lbls).cat.codes
```

در این قطعه کد یکی از پارامترهای **num** (اگر بخواهیم طول بین‌ها مساوی باشد) یا **bin** (اگر بخواهیم بین‌ها دلخواه باشند) را تعیین کرده و سپس با استفاده از کمینه و بیشینه لبه‌های بین‌ها را مشخص کرده و دادگان را تقسیم می‌کنیم.

Cut دادگان را تقسیم کرده و **cat.code** آن ها را به اعداد ترتیبی کدگذاری می‌نماید.

ب) برای این کار به سادگی می‌توان از قطعه کد زیر استفاده نمود:

```
dataset = pd.get_dummies(dataset,columns =["n_cores"])
```

و از آنجایی که تنها ستون تعداد هسته‌ها داری ویژگی شبه طبقه‌بندی‌شده بود؛ تنها این ستون تفکیک شد. کاربرد این ویژگی در مواردی است که ما می‌خواهیم از کلاس‌بندی‌هایی مانند ماشین بردار پشتیبان استفاده نماییم که تنها می‌توانند دو کلاس را با یک‌دیگر مقایسه کنند؛ لذا چند کلاس را به حالات دو کلاسه تبدیل کرده و سپس از این الگوریتم‌ها استفاده می‌کنیم.

ج) علت استفاده از تبدیل لگاریتم نزدیک‌کردن بیشتر توزیع به حالت نرمال می‌باشد. و همچنین می‌توان از آن جهت نمایش میزان درصد تغییرات نیز استفاده‌نمود.

د)

dataset["sc_a"] = dataset.sc_h * dataset.sc_w

ه) تبدیل باکس‌کاکس جهت نرمال‌سازی خطاها می‌باشد که می‌تواند منجر به ساخت مدل بهتر شود.

۶. نتایج به صورت زیر حاصل شد:

همانطور که مشاهده می‌گردد بین مساوی حجم باطری تاثیر منفی در طبقه‌بندی دارد؛ درحالی‌که در سایر موارد بهبود کوچکی داشته‌است. (چرایی تاثیر منفی را می‌توان در عدم تاثیر مستقیم حجم باطری در قیمت دانست.)

```
for i in range(len(ss)):
    print(rem(orig_ds_cols,dd[i]+["price_range"],ss[i]))
    X = dataset[rem(orig_ds_cols,dd[i]+["price_range"],ss[i])]
    X_train, X_test, y_train, y_test = train_test_split(X, y)
    clf = SVC()
    t = time()
    clf.fit(X_train, y_train)
#     print(f"\n{time() - t}")
    print(acc:=accuracy_score(predict:=clf.predict(X_test),y_test) * 100)
```

```
['blue', 'clock_speed', 'dual_sim', 'fc', 'four_g', 'int_memory', 'm_dep', 'mobile_wt', 'pc', 'px_height', 'px_width', 'ram', 'sc_h', 'sc_w', 'talk_time', 'three_g', 'touch_screen', 'wifi', 'bp_1']
87.2
['blue', 'clock_speed', 'dual_sim', 'fc', 'four_g', 'int_memory', 'm_dep', 'mobile_wt', 'pc', 'px_height', 'px_width', 'ram', 'sc_h', 'sc_w', 'talk_time', 'three_g', 'touch_screen', 'wifi', 'bp_2']
91.60000000000001
['battery_power', 'blue', 'clock_speed', 'dual_sim', 'fc', 'four_g', 'int_memory', 'm_dep', 'mobile_wt', 'pc', 'px_height', 'px_width', 'ram', 'sc_h', 'sc_w', 'talk_time', 'three_g', 'touch_screen', 'wifi', 'n_cores_1', 'n_cores_2', 'n_cores_3', 'n_cores_4', 'n_cores_5', 'n_cores_6', 'n_cores_7', 'n_cores_8']
97.6
['battery_power', 'blue', 'clock_speed', 'dual_sim', 'fc', 'four_g', 'int_memory', 'm_dep', 'mobile_wt', 'pc', 'px_height', 'px_width', 'ram', 'talk_time', 'three_g', 'touch_screen', 'wifi', 'sc_a']
96.0
['battery_power', 'blue', 'clock_speed', 'dual_sim', 'fc', 'four_g', 'int_memory', 'm_dep', 'pc', 'px_height', 'px_width', 'ram', 'sc_h', 'sc_w', 'talk_time', 'three_g', 'touch_screen', 'wifi', 'mobile_wt_log']
96.8
['battery_power', 'blue', 'clock_speed', 'dual_sim', 'fc', 'four_g', 'int_memory', 'm_dep', 'pc', 'px_height', 'px_width', 'ram', 'sc_h', 'sc_w', 'talk_time', 'three_g', 'touch_screen', 'wifi', 'mobile_wt_boxcox']
98.0
```

```
# for i in range(len(ss)):
print("all columns")
X = dataset[[x for x in dataset.columns.to_list() if x not in ["price_range"]]]
X_train, X_test, y_train, y_test = train_test_split(X, y)
clf = SVC()
t = time()
clf.fit(X_train, y_train)
#     print(f"\n{time() - t}")
print(acc:=accuracy_score(predict:=clf.predict(X_test),y_test) * 100)
```

```
all columns
98.0
```

۷. از تفاوت این روش‌ها:

- نحوه هرس کردن آن‌ها می‌باشد به طوری که **ID3** ندارد و **C4.5** پیش‌هرس و **CART** پس‌هرس دارد.
- تفاوت مهم دیگر در معیار مورد استفاده آن‌هاست چرا که **ID3** از آنتروپی و **C4.5** ضریب گین و **CART** جینی استفاده می‌نماید.
- سوم در برخورد با نوع داده‌هاست **ID3** برای داده‌های دسته‌ایی و **C4.5** ویژگی‌های گسسته و **CART** ویژگی و هدف گسسته مورد استفاده می‌باشد.

یعنی روش **C4.5** دادگان پیوسته را با تقسیم‌بندی به صورت دسته‌ایی درمی‌آورد (از طریق استفاده از شرط) و سپس دقت‌ها را محاسبه کرده و براساس دقت بیشتر پیش‌روی می‌نماید و روش **CART** بسیار مشابه است و تنها این که از رگرسیون استفاده کرده و امکان داشتن هدف گسسته را می‌دهد.

منابع:

https://medium.com/@abedinia.aydin/survey-of-the-decision-trees-algorithms-cart-c۴-۵-id۳-۹۷df۸۴۲۸۳۱cd

https://medium.datadriveninvestor.com/tree-algorithms-id۳-c۴-۵-c۵-۰-and-cart-۴۱۳۳۸۷۳۴۲۱۶۴

۸. داریم:

```
dataset = pd.read_csv("train.csv")
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()
X = dataset.drop(columns=["price_range"])
y = dataset.price_range
X_train, X_test, y_train, y_test = train_test_split(X, y)
clf.fit(X_train,y_train)
accuracy_score(predict:=clf.predict(X_test),y_test)
```

```
0.834
```

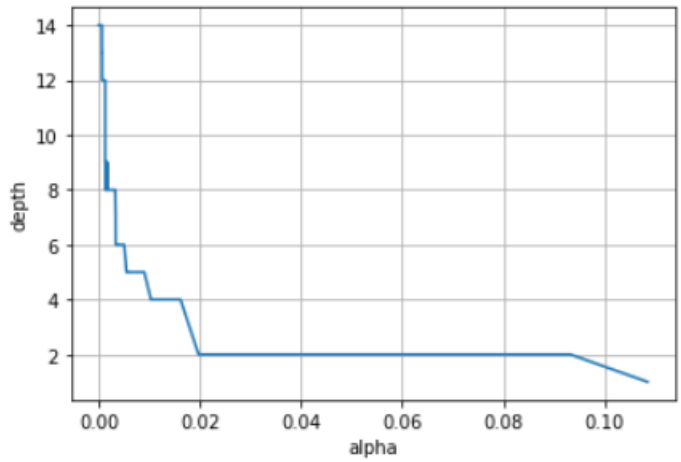
۹. با افزایش حداکثر عمق، میزان تفکیک داده‌ها بیشتر شده و بالتبع امکان انتخاب برای درخت بهتر می‌شود و بنابراین دقت افزایش می‌یابد اما از یک‌جایی به بعد با افزایش حداکثر عمق، تعداد نمونه‌ها در شاخه‌ها و برگ‌ها کاهش یافته و درخت برروی دادگان آموزش اورفیت می‌شود و دقت کاهش می‌یابد(این اتفاق در اینجا رخ نخواهدداد چرا که در این درخت امکان عمق بیش‌از ۱۳ وجود ندارد(با توجه به شرایط ویژگی‌ها).)

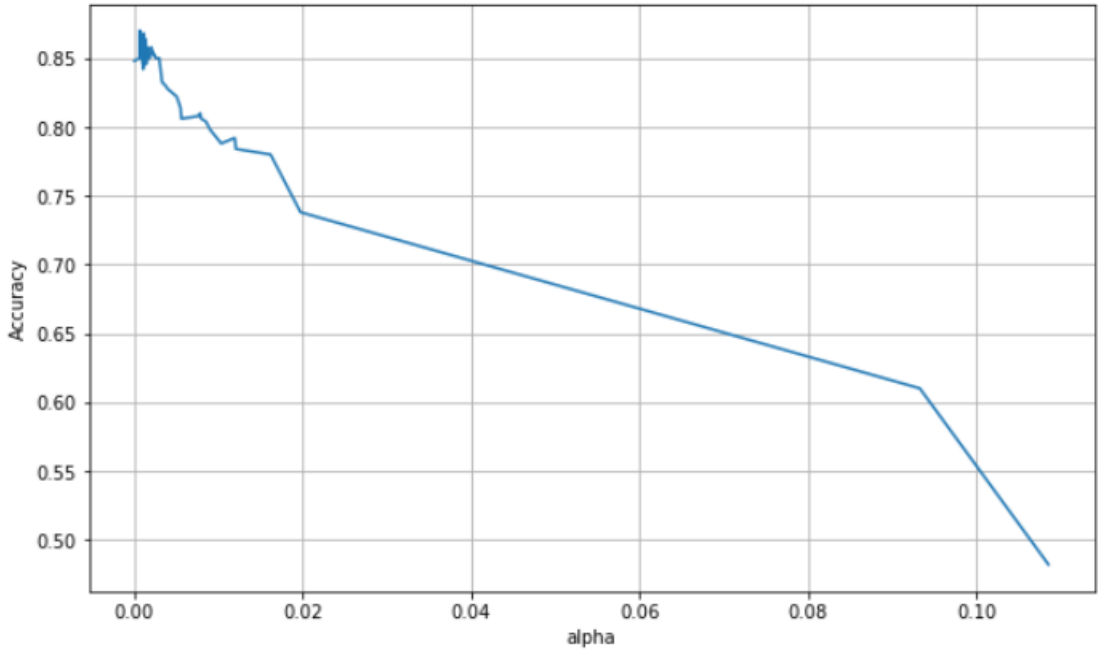
تعداد نمونه‌ها در برگ یا شاخه نیز همانند عمق عمل می‌نماید یعنی هرچقدر تعداد نمونه‌ها کم‌تر باشد احتمال تفکیک بیش‌تر است اما نباید زیر کم باشد (مثلا یک) که در این صورت اورفیت می‌شود و دقت نزولی خواهدشد.

پارامتر آخری که بررسی خواهدشد **min_weight_fraction_leaf** می‌باشد یعنی نسبت توزیع به چه شکل باشد که اگر ۰.۵ باشد هر برگ حداقل نیمی از نمونه‌های والد خود را خواهد داشت. این مورد نیز با کاهش بیش از حد منجر به اورفیت خواهدشد. اما از آن‌جا که کمینه نمونه ها را بررسی می‌کند و نه عمق، در عمق‌های به مراتب بالاتری اورفیت خواهدشد.

۱۰. همانطور که در پرسش‌های پیشین گفته شد یکی از معایب افزایش عمق تاثیر مستقیم آن بر بیش‌برازش شدن است؛ لذا پس (و گاهی اوقات پیش از ساخت درخت تصیم) اقدام به هرس می‌کنند که باعث حذف گره‌های ضعیف‌تر(که نمی‌توانند تفکیک مناسبی داشته‌باشند مثلا تعداد کمی نمونه دارند.) برای طبقه‌بندی می‌شود و در نهایت منجر به کاهش پیچیدگی و افزایش دقت می‌گردد.(یعنی بیش‌برازش را از بین می‌برد.)

۱۱. ابتدا نتایج را مشاهده می‌کنیم:





می‌بینید که با افزایش سطح هرس، عمق کاهش یافته و در ابتدا مقداری دقت افزایش می‌یابد اما با افزایش واریانس دقت نیز کاهش می‌یابد. برای این کار از کد زیر استفاده‌شده‌است:

```
clf = DecisionTreeClassifier()
# get ccp alphas
path = clf.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas = path.ccp_alphas
# get classifiers
clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(ccp_alpha=ccp_alpha)
    clf.fit(X_train, y_train)
    clfs.append(clf)
# get tree depth
tree_depths = [clf.tree_.max_depth for clf in clfs]
# get test accuracy
acc_scores = [accuracy_score(y_test, clf.predict(X_test)) for clf in clfs]
# plot depth per alpha
plt.plot(ccp_alphas[:-1], tree_depths[:-1])
plt.grid()
plt.xlabel("alpha")
plt.ylabel("depth")
# plot depth per alpha
plt.figure(figsize=(10,  6))
plt.grid()
plt.plot(ccp_alphas[:-1], acc_scores[:-1])
plt.xlabel("alpha")
plt.ylabel("Accuracy")
```

ابتدا آلفاهای ممکن را برای سطح هرس به دست آورده (که براساس مسیر درخت می‌باشد.) و سپس با توجه به هر آلفا یک بار برروی مجموعه دادگان اجرا کرده و عمق درخت را به دست می‌آوریم و همچنین برروی مجموعه دادگان آزمایش نیز بررسی می‌کنیم.

۱۲. ابتدا جنگل تصادفی را اجرا می‌کنیم:

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier()
clf.fit(X_train,y_train)
accuracy_score(clf.predict(X_test),y_test)
```

```
0.91
```

البته قابل پیش‌بینی می‌باشد که دقت کمی افزایش می‌یابد چرا که جنگل تصادفی چند درخت را با هم‌دیگر ترکیب کرده (به طوری‌که باياس افزایش می‌یابد) یعنی ترکیب را به طوری انجام می‌دهد که از اورفیت جلوگیری نماید. و گره‌ها محدود به چند نمونه با تشابه بالا نباشد. (جنگل تصادفی همانند بگینگ به صورت موازی، درخت‌ها را با یک‌دیگر ترکیب می‌نماید.)

۱۳. دلایل آن را می‌توان به طور خلاصه به صورت زیر بیان‌کرد:

الف) استفاده و توضیح آن حتی برای کسانی که در این زمینه تخصص ندارند ساده می‌باشد.

ب) امکان انتخاب ویژگی را به ما می‌دهند.

پ) برخلاف روش‌های یادگیری عمیق نیازی به تلاش‌های چندین‌باره جهت رسیدن به پاسخ بهینه نیست و با تلاش‌های کمتری می‌توان به آن رسید.

ت) می‌توان از احتمالات شرطی نظیر بیز استفاده کرد یا با روش‌های دیگر ترکیب نمود.

ث) با روابط غیرخطی نیز به خوبی کار می‌نماید.

۱۴. در ادامه دو روش ریپر و یک قاعده بیاموز را شرح می‌دهیم:

ریپر: که مخفف هرس افزایشی متوالی جهت کاهش خطا می‌باشد. و ۱. برای داده‌های غیربالانس مناسب می‌باشد و ۲. از بیش‌برازش جلوگیری می‌نماید.

روش کار: فرض نمایید که دو کلاس داشته‌باشیم. ریپر کلاسی را به عنوان پیش‌فرض درنظر می‌گیرد که نمونه‌های بیشتری (بالای ۵۰ درصد) دارا باشد و تلاش می‌کند قواعدی را جهت تشخیص کلاس دیگر می‌یابد.

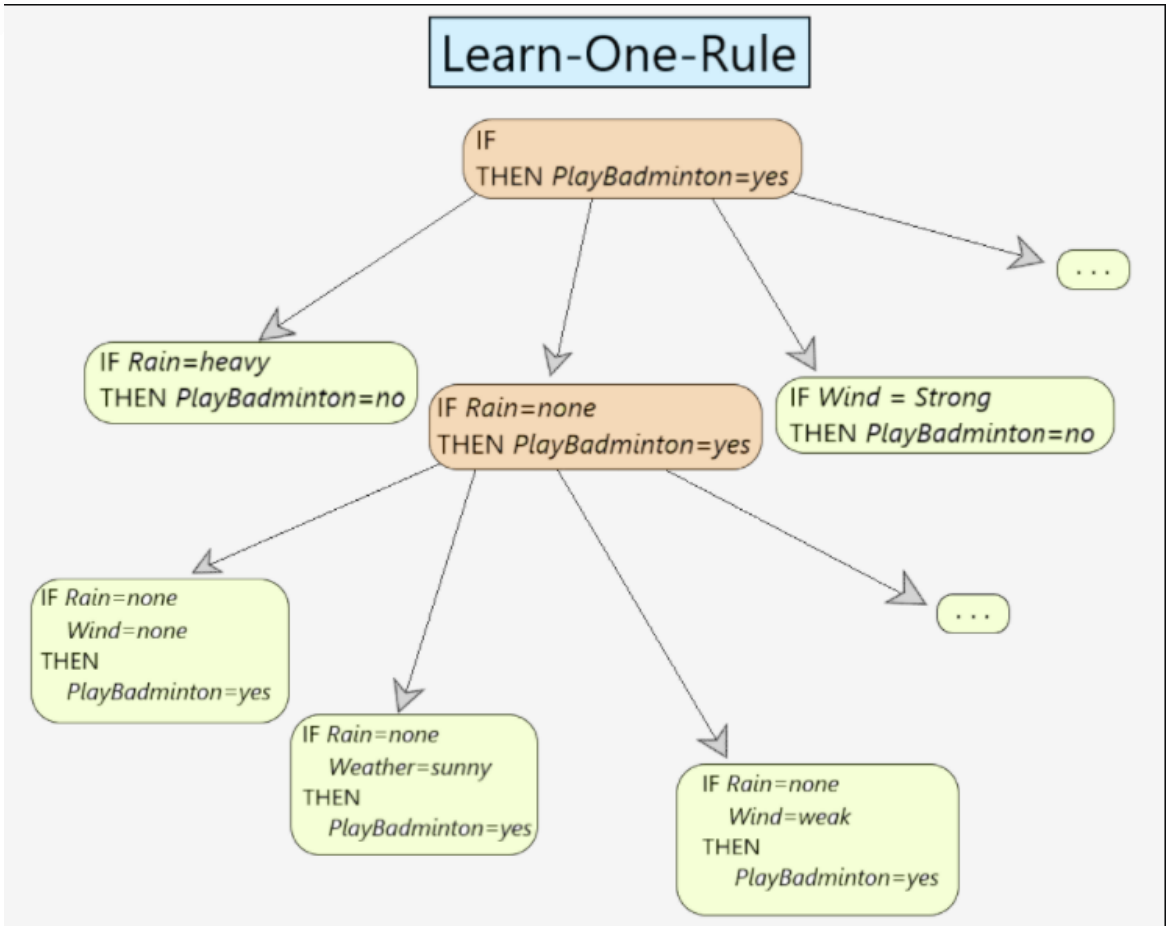
اگر چندین کلاس داشته‌باشیم آن‌ها را به صورت صعودی مرتب خواهیم‌کرد:(یعنی به‌صورت زیر

C1,C2,C3,...,Cn

که در آن **Cn** تکرارهای بیشتری نسبت به سایرین داشته‌است و کلاس پیش‌فرض می‌باشد) سپس کلاس **C1** انتخاب شده و اعضای متعلق را **ve+** و سایر نمونه‌ها را **ve-** می‌نامیم و تلاش می‌کنیم قواعدی را جهت تمییز اعضای این دو گروه تولید نماییم.

برای این کار از الگوریتم پوششی متوالی(**Sequential Covering**) استفاده می‌نماییم که یک روش مبتنی بر قاعده است.

پیش‌نیاز: یک قاعده بیاموز (**Learn-One-Rule**) چیست؟ این روش یک قاعده را انتخاب می‌کند که حداقل تعدادی از نمونه‌ها را شامل شود. این روش، این قابلیت را دارد که ویژگی‌ها را با یکدیگر ترکیب نماید.



این روش به صورت حریصانه، قواعدی را انتخاب می‌نماید که دقت بالایی داشته‌باشند (عدم وجود نمونه از کلاس‌های دیگر) اما ممکن است دادگان کمتری را شامل شود. به همین ترتیب پیش می‌رود که در نهایت به بیش‌ترین دقت ممکن برسد. یعنی قاعده (ویژگی = مقدار) دیگری اضافه می‌کند که تعداد نمونه‌ها از کلاس افزایش یابد.

حال پوشی متوالی چه کاری می‌کند؟ این روش، یک حلقه بی‌نهایت است که براساس یک قاعده بیاموز پیش می‌رود:

۱) در ابتدا یک قاعده بیاموز
۲) تمام نمونه‌هایی که شامل آن قاعده می‌شوند را درنظر بگیر.
۳) تمام نمونه‌های یافت شده را دسته‌بندی کرده و از گروه اصلی حذف کن.
۴) حال گروه اصلی را برابر گروه جدید (اعضایی که داخل گروه حذف شده نیستند.) در نظر بگیر.
۵) اگر گروه اصلی تهی شد یا نمی‌توانستیم قاعده‌ایی را انتخاب کنیم که شمولیت داشته‌باشد: تمام قواعد تولیدی را برگردان و پایان
۶) به ۱ برگرد.

ریپر از این الگوریتم استفاده نموده تا تمام قواعدی که بین **ve+** و **ve-** تمییز دهد را بیابد.سپس به سراغ کلاس بعدی می‌رود. این کار آنقدر تکرار می‌شود که به کلاس پایانی برسیم. یعنی از کلاس کوچکتر به کلاس بزرگ‌تر

یعنی در واقعی دارای توالی‌هایی از قواعد هستیم.

حال که این قواعد زیاد شد چه کار باید کرد؟ هرس

فرض کنید که یک دنباله از قواعد **ABCD** تولیدشد. از راست‌ترین قاعده شروع کرده با حذف آن اگر **(P-N)/(P+N)** مقدار بیشتری داشت حذف می‌شود وگرنه **CD** و **BCD** و **C** و... بررسی می‌شوند. تا در نهایت به بهترین دقت بدون بیش‌برازش برسیم.

https://www.geeksforgeeks.org/ripper-algorithm/

۱۵. یکی از کاربردهای درخت تصمیم در سری‌های زمانی می‌باشد که در ادامه باختصار شرح‌می‌دهیم:

در ابتدا ببینیم که سری زمانی چیست؟ داده (نمونه)هایی هستند که به ترتیب زمانی مرتب شده‌اند. بنابراین مشخص است که نمی‌توان درخت تصمیم را بر روی دادگان خام مدل کرد و باید تغییراتی را برروی دادگان انجام داد.

برای این کار باید یک جدول جدید از دادگان بسازیم.فرض کنید که برای مثال مقدار داده در امروز به دادگان در مثلا طول ۶۰ روز قبل وابسته باشد؛ بنابراین می‌توانیم یک جدول جدید متشکل از ۶۱ ستون بسازیم که ستون ۶۱ام مقدار در امروز و ستون ۶۰ام مقدار در دیروز و... ستون ۱ در ۶۰ روز قبل باشد.(البته برای این کار باید از داده ۶۱ به بعد انجام دهیم) بنابراین یک دادگان با ۶۰ ویژگی داریم.

مسلما این حجم از دادگان پیوسته برای درخت تصمیم مناسب نیست. پس می‌توانیم آماره‌هایی را از آن‌ها استخراج کرده و سپس براساس آن‌ها درخت خود را بسازیم؛ مانند: میانگین، میانه، انحراف معیار و بازه و...

حال این آماره‌ها مربوط به چندروز باشد بهتر است؟ تمام ۶۰ روز؟ مسلما برای دادگان مختلف و فراز و فرودشان متفاوت است. پس باید بررسی نماییم یعنی بعنوان مثال برای ۲، ۵، ۸ و ... و ۶۰ روز را بررسی کنیم و هریک که دقت بیشتری در دادگان آموزش داشت را در نظر بگیریم.

البته باید تاریخ‌های انتهایی را بعنوان آزمون درنظرگرفت.

https://towardsdatascience.com/approaching-time-series-with-a-tree-based-model-۸۷c۶d۱fb۶۶۰۳

۱۶.

الف) در ابتدا دادگان دریافت شد. و سپس با قطعه‌کد زیر به تایم‌استمپ تبدیل گردید.

```
from datetime import datetime
import time

def mdy_to_ymd(d):
#     return datetime.strptime(, ).strftime('%Y-%m-%d')
    return time.mktime(datetime.strptime(d, '%d-%b-%y').timetuple())
```

```
pd.read_csv("dataset.csv").apply(lambda row : mdy_to_ymd(row['Date']), axis = 1)
```

متد **apply** برروی هر ردیف یا ستون اعمال می‌گردد. و تابع مورد نظر تاریخ را در فرمت ورودی دریافت کرده و به تایمپ استمپ تبدیل می‌نماید.

ب) و از قطعه کد زیر نیز برای تقسیم استفاده می‌نماییم:

```
test = dataset[dataset.Date >= time.mktime(datetime.strptime("20-01-02", '%y-%m-%d').timetuple())]
train = dataset[dataset.Date < time.mktime(datetime.strptime("20-01-02", '%y-%m-%d').timetuple())]
```

۱۷. برای این سوال، مدل‌های زیر را استفاده‌خواهیم‌کرد:

۱۷.۱.۱. رگرسیون خطی

در اولین بخش به سراغ رگرسیون خطی می‌رویم:

برای این که مشخص شود بیشترین تاثیر بر قیمت امروز از چندروز قبل می‌باشد؛ جدول به این‌وصورت تغییر کرد که علاوه بر قیمت امروز شامل قیمت ۶۰ روز گذشته نیز می‌باشد.

	t0	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13	t14	t15	t16	t17	t18	t19	t20	t21	t22	t23	t24	t25	t26	t27	t28
0	57807.1	57720.3	53560.8	54841.4	55036.5	54020.5	48963.6	50088.9	51143.6	51729.5	53820.2	56483.2	55646.1	56207.1	60041.9	61379.7	63216.0	62980.4	63540.9	59863.8	59978.7	59748.4	58118.7	58077.4	55948.7	57996.3	58993.4	58199.9	57059.9
1	57720.3	53560.8	54841.4	55036.5	54020.5	48963.6	50088.9	51143.6	51729.5	53820.2	56483.2	55646.1	56207.1	60041.9	61379.7	63216.0	62980.4	63540.9	59863.8	59978.7	59748.4	58118.7	58077.4	55948.7	57996.3	58993.4	58199.9	57059.9	58977.3
2	53560.8	54841.4	55036.5	54020.5	48963.6	50088.9	51143.6	51729.5	53820.2	56483.2	55646.1	56207.1	60041.9	61379.7	63216.0	62980.4	63540.9	59863.8	59978.7	59748.4	58118.7	58077.4	55948.7	57996.3	58993.4	58199.9	57059.9	58977.3	58716.3
3	54841.4	55036.5	54020.5	48963.6	50088.9	51143.6	51729.5	53820.2	56483.2	55646.1	56207.1	60041.9	61379.7	63216.0	62980.4	63540.9	59863.8	59978.7	59748.4	58118.7	58077.4	55948.7	57996.3	58993.4	58199.9	57059.9	58977.3	58718.3	58763.7
4	55036.5	54020.5	48963.6	50088.9	51143.6	51729.5	53820.2	56483.2	55646.1	56207.1	60041.9	61379.7	63216.0	62980.4	63540.9	59863.8	59978.7	59748.4	58118.7	58077.4	55948.7	57996.3	58993.4	58199.9	57059.9	58977.3	58718.3	58763.7	58771.3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3876	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
3877	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
3878	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
3879	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
3880	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1

```
SIZE = 60 #we take 2 past months here for each time-series point
COLUMNS = ['t{}'.format(x) for x in range(SIZE)] + ['target']
df = []
for i in range(SIZE, dataset.shape[0]):
    df.append(dataset.loc[i-SIZE:i, 'Price'].tolist())
df = pd.DataFrame(df, columns=COLUMNS)
df
```

با استفاده از قطعه کد بالا برای هرروز (از روز ۶۰ به بعد) قیمت ۶۰ روز قبل نیز ثبت می‌گردد.

```
1: 47
2: 9
3: 4
4: 1
5: 1
6: 10
7: 1
8: 2
9: 2
11: 9
12: 1
14: 5
15: 5
17: 1
19: 1
21: 1
```

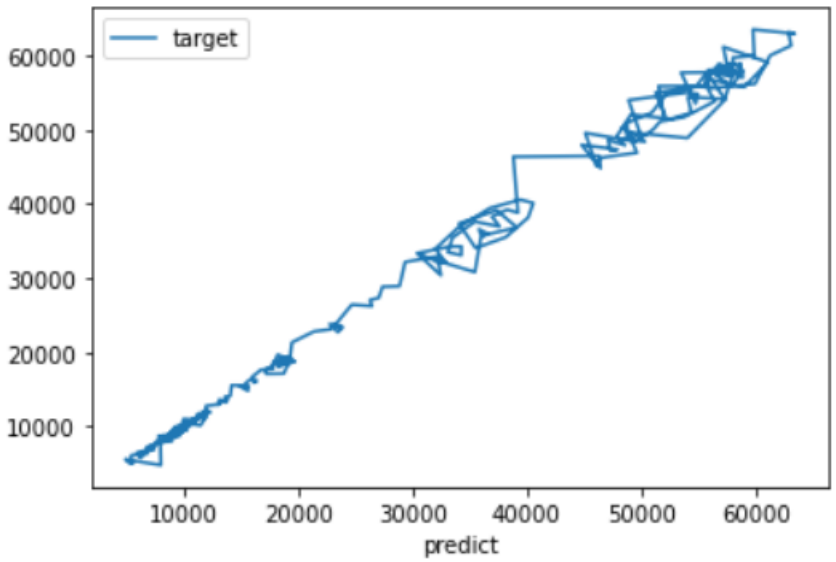
```python
from sklearn.linear_model import LinearRegression
model = LinearRegression()
min_ = 999999
s = []
mins = []
for _ in range(100):
    X_train, X_test, y_train, y_test = train_test_split(df[[f't{i}' for i in range(0,60)]],df.target)
    for i in range(1,60):
        X_tr, X_te = X_train[[f't{i}' for i in range(59 - i + 1,60)]], X_test[[f't{i}' for i in range(59 - i + 1,60)]]
        model.fit(X_tr,y_train)
        if (m:=mean_squared_error(model.predict(X_te),y_test)) < min_:
            min_ = m
            ii = i
    mins.append(min_)
    s.append(ii)
    min_ = 999999
print(s,min_,len(s))

data = dict(zip([i for i in range(60)],[0 for _ in range(60)]))
for i in s:
    data[i]+=1
for i,v in data.items():
    if v != 0:
        print(f"{i}: {v}")
```

با استفاده از قطعه کد بالا، در ۱۰۰ دور چندین بار و هر بار با در نظر گرفتن تعداد روز متفاوت بعنوان ویژگی تلاش کردیم یک ندل بسازیم تا ببینیم که بیشترین تاثیر در اثر چند روز است و مشخص شد اگر تنها یک روز قبل را به عنوان ویژگی داشته‌باشیم؛ در حدود ۵۰ درصد موارد بهترین پیش‌بینی را خواهیم‌داشت.

```
df.plot(x = "predict", y = "target")
```

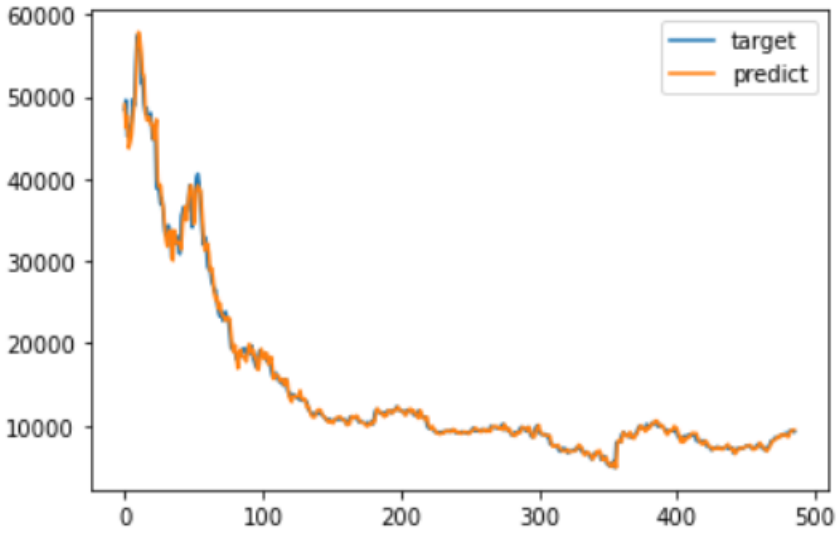
```
<AxesSubplot:xlabel='predict'>
```



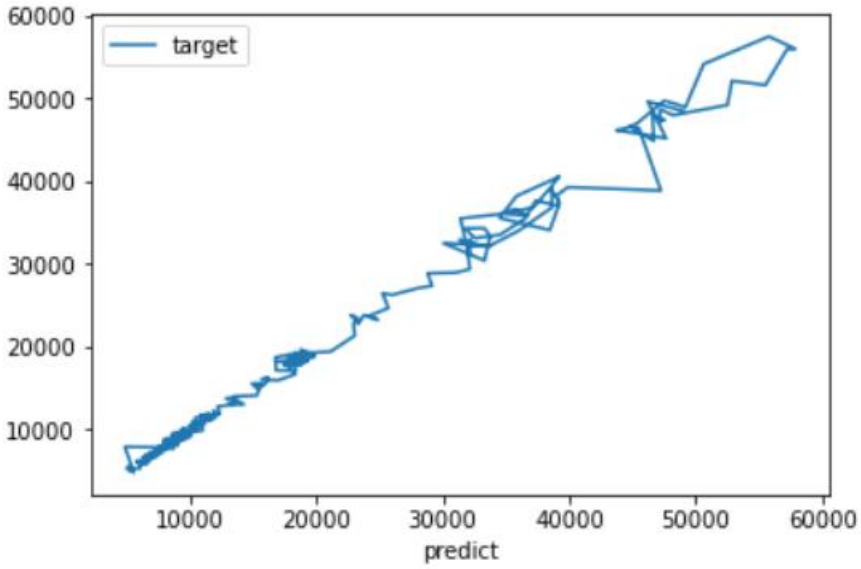
همانطور که از خط پیش‌بینی قابل مشاهده‌است؛ چندان انطباق بالایی وجود ندارد و میانگین خطا ۵۸۲ می‌باشد.

۱۷.۱.۲. رگرسیون لسو

در اینجا نیز همانند قبل ۶۰ روز پیشین را در نظرگرفته و از لسو برای تاثیردهی ویژگی‌ها استفاده می‌نماییم. مشاهده می‌کنید که:

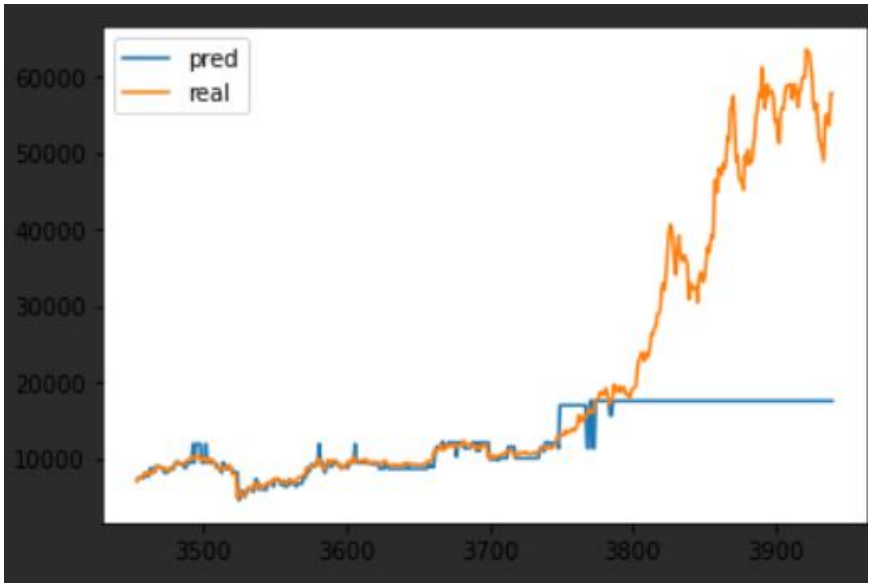


همچنان دارای خطا می‌باشد اما میانگین خطا برابر ۸۷۱ شده‌است که نشان‌دهنده بهبود نسبی است.



۱۷.۱.۳. درخت تصمیم

حال از روش درخت تصمیم برای پیش‌بینی استفاده خواهیم‌کرد. این روش همانند روش نخست، تمام دادگان را نگهداری می‌کند.



حال می‌بینیم که پیش‌بینی تا حدی مناسب است اما زمانی که شدت نوسان زیاد می‌شود دیگر نمی‌تواند به درستی پیش‌بینی نماید که چرایی آن را می‌توان در اورفیت‌شدن آن دانست که هنگامی که قیمت واقعی به شدت نوسانی می‌شود دیگر موثر نیست و مقدار میانگین خطا برابر۲۶۲۷۱۹۳۳۳.۹۴۶۵۰۲۰۶ می‌گردد که بسیار رقم بالایی است (به دلیل مقادیر انتهایی) اما با این‌حال باز هم برخی‌ها با دقت خوبی پیش‌بینی می‌شود.

۱۷.۱.۴. LSTM

۱۷.۱.۵. GRU

۱۷.۱.۶. BiLSTM

حال می‌خواهیم در یک سه‌گانه از روش‌های مبتنی بر شبکه استفاده نماییم. نخست به سراغ LSTM می‌رویم.

این سه روش بسیار به هم مشابهت دارند. هر سه این شبکه‌ها از گروه بازگشتی هستند؛ که این توانایی را دارند که با داده‌های متوالی کار نمایند. اما یکی از مشکلات آن‌ها هنگام کار با داده‌های بسیار زیاد است که برای هر بروزرسانی باید داده‌ها محاسبه‌گردند که ممکن است تاثیرات نامناسبی برروی ضرایب بگذارند. (که به جهت حافظه کوتاه‌مدت آن‌هاست.) بنابراین روش‌های تازه‌تر به‌وجودآمدند که نحوه بهره‌گیری از اطلاعات را مشخص می‌کنند؛ یعنی کدام داده‌ها نگهداری شده و کدام حذف‌گردند. یعنی همانند روش دوم (لسو) مشخص می‌کنند که چه فیچرهایی (محدوده زمانی) اهمیت دارند.

در این کد چند پیش‌پردازش مانند تقیسم‌کردن و... انجام‌شده‌است که همانند قبل بوده و دارای نکته خاصی نیست اما آن‌چه که در اینجا تازه‌است نرمال‌سازی آن می‌باشد که داده‌ها را در مقیاس ۱ و ۱- قرار می‌دهد که برای این‌کار از کتابخانه سایکیت استفاده می‌گردد و ابتدا برروی دادگان آموزش فیت شده و سپس برای تمامی دادگان انجام‌می‌شود.

اما مهم‌ترین گام پیش‌پردازش ساخت دادگان مناسب کراس می‌باشد که داده‌ها را به صورت دسته‌ایی در می‌آورد یعنی مثلا در اینجا ما تصمیم داریم که قیمت فردا را پیش‌بینی نماییم و برای هرروز، قیمت ده روز قبل آن را به عنوان ویژگی در نظر می‌گیریم.

حال به ساخت مدل می‌پردازیم:

مدل‌ها مشابه یک‌دیگر هستند و از لایه مخصوص خود کراس بهره می‌برند که دارای ۶۴ نورون می‌باشد و همچنین چند نورون (مثلا یکی) برای بردار نهایی (یعنی چندبعدی بودن که برای نتیجه عمدتا یک می‌باشد) از دنس و از لایه دراپ‌اوت نیز جهت حذف نورون‌ها برای جلوگیری از اورفیت‌شدن استفاده‌می‌نماییم. در انتها نیز این مدل‌ها را فیت‌کرده و نتایج را بررسی نمودیم.

۱۷.۱.۷. ARIMA

۱۷.۱.۸. SARIMAX

حال به سراغ دو روش آریما و ساریمکس می‌رویم. این دو روش، شباهت بسیاری به یک دیگر دارند و آریما مخفف میانگین متحرک خودهمبسته یکپارچه می‌باشد که براساس این فرض می‌باشد که مقدار فعلی را می‌توان با استفاده از مقادیر از لحظه ۰ تا یک لحظه قبل‌تر به دست آورد. آریما یعنی آر + یی + ما که هریک مفهومی جداگانه دارا هستند.

آریما سه پارامتر جداگانه دریافت می‌کند یکی برای آر و دیگری برای ما . در نهیت پارامتری برای تفاضل.

P for AR, Q for MA & D fir differencing

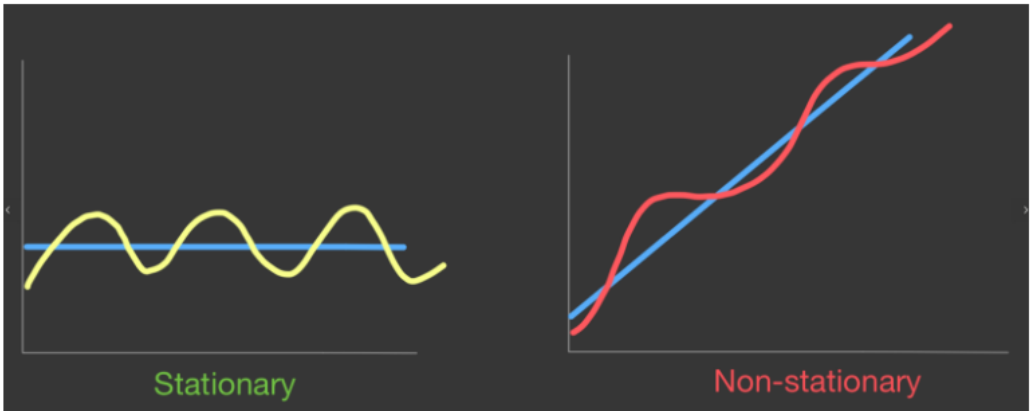
قسمت آر از مقادیر مربوط به لحظات پیشین استفاده نموده و برحسب آن‌ها یک رگرسیون تولید می‌نماید که فرمول آن بسیار شبیه به رگرسیون خطی می‌باشد و پارامتر پی تعیین می‌نماید که تا چند پارامتر قبل باید بررسی شود و رگرسیون به دست آید. یعنی یک ترکیب خطی از پی لحظه قبل.

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \epsilon_t$$

اما قسمت ما چیست؟ این قسمت نیز مشابه یک رگرسیون خطی است اما براساس خطاهای پیش بینی‌های قبلی می‌باشد.

$$x_t = \mu + \sum_{i=1}^{q} \Phi_i \epsilon_{t-1}$$

یعنی خطای پیش‌بینی کیو لحظه قبلی. و پارامتر آخر هم تعیین می‌کند که چندبار اختلاف محاسبه گردد تا بیشتر ثابت شود.



حال باید دید که تفاوت ثابت و غیرثابت چیست؟ همانطور که در تصویر بالا دیده‌می‌شود نمودار مربوط به ثابت با گذر زمان تغییر نمی‌کند اما دومی به مرور افزایش می‌یابد. و اصطلاحا ترند دار و دادگان باید ثابت باشند. برای آن که بدانیم چگونه آریما را پیاده نماییم اول باید مشخص‌کنیم که دادگان ما از کدام نوع می‌باشد که با متد دیف پانداس می‌توان آن را فهمید.

```
dataset["high"].diff().plot()
```

ابتدا باید باکس‌کاکس دادگان را بیابیم که فرمول آن به صورت

$$y = (x^{**}lmbda - ۱) / lmbda, \quad for\ lmbda \neq ۰$$

$$log(x), \quad for\ lmbda = ۰$$

می‌باشد که برای نرمال‌کردن داده‌ها استفاده می‌گردد که بتوانیم اختلاف‌های زمانی را به طور مناسب‌تری (بدون نوسان‌های شدید) بیابیم که در ادامه نوسان ۱۲ روزه و سه روزه و ۱ روزه نیز به دادگان اضافه می‌گردد. در انتها نیز پارامترهای مختلفی را برای آریما درنظر می‌گیریم که بهترین آن‌ها برحسب معیار **aic** انتخاب‌می‌گردد و دادگان فیت می‌شود.

حال ساریمکس چیست؟ باید گفت آریما یک حالت خاص از ساریمکس است که در آن فصلی بودن لحاظ نمی‌شود یعنی پارامتری برای بررسی آن ندارد که در کدها نیز مشخص می‌باشد. یعنی ترتیب فصلی بودن را نیز با پارامترهای دیگری بررسی می‌نماید. که البته باید دقت شود این پارامترها نیز براساس همان قبلی‌ها می‌باشد که تنها سیکل موجود را بررسی می‌نماید.

۱۷.۱.۹. **Naïve**

همانطور که می‌دانیم این روش براساس تقسیم دادگان به چنددسته می‌باشد؛ لذا در مرحله نخست، دادگان را تقسیم می‌نماییم و براساس آن بین‌ها را می‌سازیم سپس بعد از تقسیم به ترین و تست، مشخص می‌نماییم که تا چند روز قبل به عنوان ویژگی باشد که تمامی آن ها در قسمت ترین و تست به عنوان فیچر ذخیره خواهندشد.

در نهایت مدل ساخته‌شده و پیش‌بینی می‌گردد.

۱۷.۱.۱۰. **KNN**

این مدل نیز دقیقاً همانند قبل می‌باشد با این تفاوت که تنها از مدل چندهمسایه برای پیش‌بینی استفاده‌گردید.

۱۸. در نتیجه بهترین مدل‌ها: خطی، لسو و آریما و نیوبیز و چندهمسایه می‌باشد. برای روش وتینگ از خطی و لسو و چندهمسایه استفاده‌نمودیم که نتیجه برابرا دقت ۵۳ درصد و خطای ۳۵۶۰۰۹۱۵ شد و همچنین نتیجه بوستینگ نیز برابر ۶۲.۷۵ و ۲۶۴۸۹۴۴۹۶.۹۲۹۳۷۰۰۲ گردید و برای بگینگ به صورت زیر می‌باشد:

در موارد زیر نام مدل به معنی مدل بیس بوده و مدل‌های دیگر به عنوان سایر تخمین‌گرها می‌باشند.

Lasso()

۸۷.۲۴۲۷۹۸۳۵۳۹۰۹۴۶

۱۰۸۸۵۶۳.۸۹۰۵۷۲۶۳۰۶

LinearRegression()

۸۷.۲۴۲۷۹۸۳۵۳۹۰۹۴۶

۱۰۹۱۴۱۶.۷۲۳۲۰۷۲۹۴

KNeighborsRegressor()

۲۳.۶۶۲۵۵۱۴۴۰۳۲۹۲۱۷

۳۰۳۲۳۳۸۵۹.۶۹۹۵۹۳

۱۹. حال برای آدابوست ۵ نوع مختلف را درنظرمی‌گیریم:

۱) بدون هیچ‌گونه پارامتر:

accuracy: ۵۶.۳۷۸۶۰۰۸۲۳۰۴۵۲۷

mse: ۲۵۱۲۳۲۴۱۴.۸۴۵۸۶۰۸۷

۲) با تخمین‌گر پایه خطی:

accuracy: ۸۹.۹۱۷۶۹۵۴۷۳۲۵۱۰۳

mse: ۱۰۳۸۲۸۴.۹۲۱۷۵۶۴۵۹۹

۳) خطای exponential

accuracy: ۵۶.۵۸۴۳۶۲۱۳۹۹۱۷۷

mse: ۲۶۴۰۷۴۸۲۲.۲۵۶۲۷۱۱۵

که نشان می‌دهد تغییری رخ نمی‌دهد.

۴) ضریب یادگیری‌های مختلفی و با دو حالت بدون تخمین‌گر و تخمین‌گر خطی آزموده شد اما معیارها به طور کلی تغییری نکردند.

۵) تعداد تخمین‌گر

نکته جالب درباره این پارامتر می‌باشد. در صورتی که تخمین‌گری تعیین نگردد بسیار متفاوت می‌باشد و اگر ۱ در نظر بگیریم:

accuracy: ۱۶.۶۶۶۶۶۶۶۶۶۶۶۶۶۴

mse: ۲۹۹۶۱۴۷۶۱.۰۳۱۵۲۱۲

ولی اگر ۱۰ باشد:

accuracy: ۵۰.۸۲۳۰۴۵۲۶۷۴۸۹۷۱

mse: ۲۹۱۰۶۵۶۴۱.۷۸۲۷۰۴۵۳

و در ۱۰۰:

accuracy: ۵۷.۶۱۳۱۶۸۷۲۴۲۷۹۸۴

mse: ۲۵۳۹۱۳۷۳۰.۷۱۵۶۲۰۸۸

اما با تخمین‌گر خطی:

تفاوت دقت برابر ۳ درصد و تفاوت خطا برابر ۱۰۰ می‌باشد. که نشان می‌دهد در خطی به اندازه کافی آموزش انجام‌شده‌است.

۲۰. ۵ نوع مختلف را بررسی می‌نماییم.

۱) بدون پارامتر:

accuracy: ۶۱.۱۱۱۱۱۱۱۱۱۱۱۱۱۴

mse: ۲۶۱۶۲۶۵۲۲.۵۹۷۱۵۶۱۷

۲) تعداد تخمین‌گر

اگر برابر ۱ باشد:

accuracy: ۴۵.۲۶۷۴۸۹۷۱۱۹۳۴۱۵۴

mse: ۳۷۳۲۵۸۷۶۳.۲۴۱۶۴۶۱

اگر برابر ۱۰ باشد:

accuracy: ۵۸.۶۴۱۹۷۵۳۰۸۶۴۱۹۸

mse: ۲۶۲۳۱۵۸۶۶.۱۷۹۹۱۳۶

۳) با تغییر میزان عمق و کاهش آن، آندرفیت شده و دقت آن به طور قابل‌توجهی کاهش می‌یابد؛ به طوری که برای ۲ به دقت ۱۶ و خطای ۳۶۹۵۰۳۷۶۹ می‌رسد.

۴) با تغییر پارامتر **min_samples_split** تاثیر چندانی به دست نیامد.

۵) اما اگر از روش بوت‌سرپ استفاده نکند طبیعی است که نتایج بدتر می‌شود؛ یعنی:

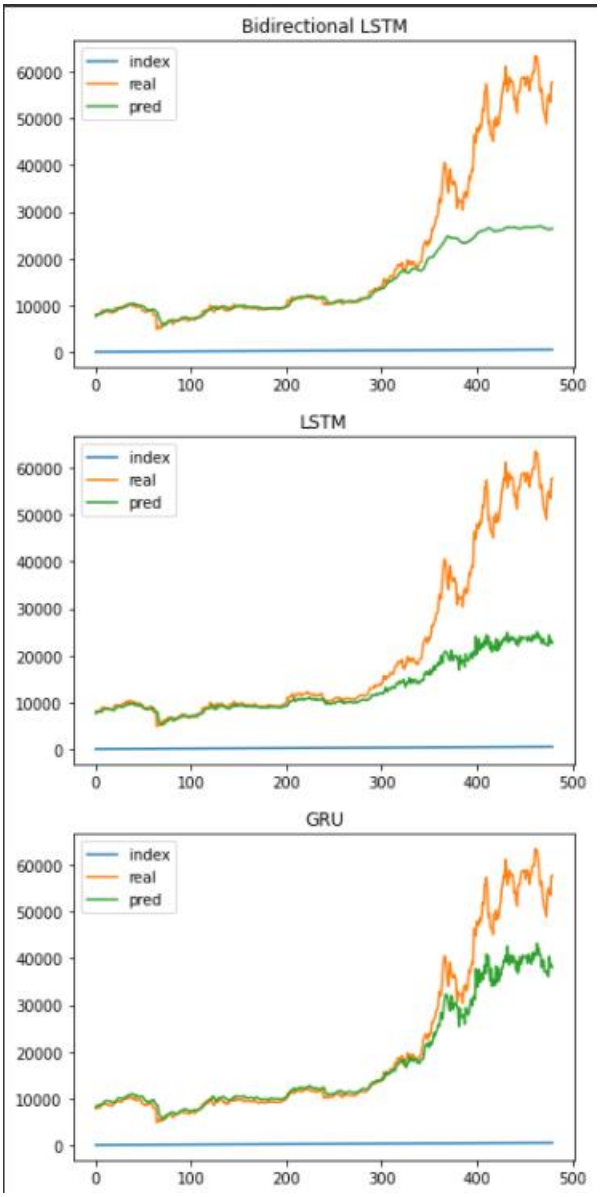
accuracy: ۵۳.۴۹۷۹۴۲۳۸۶۸۳۱۲۸

mse: ۲۶۲۱۵۸۲۵۶.۵۹۹۱۷۵۵۴

۲۱. دقت‌ها به صورت بازه:

- در مدل خطی دقت برابر با ۸۵ درصد می‌باشد.
- اما در مدل لسو دقت برابر ۸۱ درصد می‌باشد که علت این تفاوت را می‌توان نزدیکی بیشتر در روزهای تکرار و بدون نوسان دید؛ هنگامی که نوسانی در دو روز پیاپی وجود نداشته‌باشد؛ مدل خطی بهترین پیش‌بینی را ارائه می‌دهد اما مدل لسو که براساس تمام ۶۰ روز گذشته‌است؛ تلاش می‌کند تا بهترین پیش‌بینی براساس کل دادگان را داشته‌باشد که در نتیجه نوسان دارد و متفاوت از مقدار اصلی‌ست.
- روش درخت پیش‌بینی ابدا برای کار مناسب نیست و نوسان‌های بالا را نمی‌تواند پاسخگو باشد؛ به همین دلیل میزان دقت آن برابر ۵۳.۹ شد که نسبت به موارد قبلی کمتر است.
- سه مدل مبتنی بر شبکه‌های بازگشتی دارای نتایج مختلفی در پارامترهای متفاوتی بودند به طوری که برای یکی از حالات دقت یکی به بالای ۸۰ درصد نیز رسید و...

```
Bidirectional LSTM
Accuracy:  49.166666666666664
Mean Absolute Error: 151117609.2267
LSTM
Accuracy:  28.125
Mean Absolute Error: 197195372.8009
GRU
Accuracy:  32.916666666666664
Mean Absolute Error: 50040093.6757
```



- روش آریما دارای میانگین خطای ۹۹۱۱۲۶.۳۷۲۴۲۱۷۴۲۴ و دقت ۱ بود و ساریمکس نیز توانست مانگین خطای ۹۸۳۷۳۳.۱۴۲۳۱۸۴۷۶ و دقت ۱ داشته‌باشد.
- در این روش، با اتخاذ ۴۵ به عنوان لبه‌ها، میزان خطا برابر ۶۸۸۲۰۸۳.۶۴۲۶۹۴۲۵۷ و دقت نیز ۶۲.۹۵ درصد می‌باشد.
- در این روش، با اتخاذ ۴۵ به عنوان لبه‌ها، میزان خطا برابر ۱۴۸۸۹۶۶۰.۳۴۷۸۳۲۹۲۶ و دقت نیز ۵۲.۲۴ درصد می‌باشد.

۲۲. برای این کار مشابه قبل عمل‌کردیم با این تفاوت که هدف را برابر ستون تغییر (۱ یعنی افزایش و ۱- یعنی غیرافزایش) می‌باشد. و نتیجه به میزان ۵۱ درصد حاصل گردید.

۲۳. در حالت خطی، بهبود به میزان ۱ تا ۲ درصد حاصل شد که بسیار کم و شکننده می‌باشد و درباره لسو نیز این میزان صادق می‌باشد و در حالت چند همسایه نیز چندان تفاوتی ایجاد نشد که چرایی آن را می‌توانست در این دانست که همین میزان تغییر در سایر پارامترها موجود است (به طور ضمنی) و چندان موثر نیست.

۲۴. در این سوال از روش پرافت فیس‌بوک استفاده می‌نماییم. یک توضیح مختصر از این روش به قرار زیر است. در این روش مبنا این است که بدون دانش اولیه به تخمین بپردازیم. یعنی چه؟ یعنی فرض کنید که یک مدل خطی ار داده‌ها ایجاد می‌کنیم و تلاش می‌کنیم با مدل‌های مختلفی آن را غیرخطی نماییم تا به دادگان آموزش فیت شود. همچنین دادگان را به سه قسمت فصلی، ترند و تعطیلات و رویداد تقسیم می‌نماید. یعنی می‌تواند جهش‌های آنی را به عنوان تعطیلات بداند که در نتیجه عیب روش‌های قبلی را رفع کند. در نهایت داریم؛

$$f(t) = g(t) + s(t) + h(t) + e(t)$$

اما ملزوماتی دارد:

الف) باید مشاهدات ترتیبی کامل و به طول حداقل یک و ترجیحا ۱۲ ماه باشد.

ب) فصلی بودن آن قوی باشد.

ج) در نظر گرفتن تعطیلات و رویدادها

د) تعداد داده‌های از دست‌رفته منطقی و محدود باشد.

که در نتیجه‌ی آن جدولی حاصل خواهدشد ک دارای اطلاعات بسیار بالایی می‌باشد اما با روش‌ها و پارامترهای متفاوت اصلا دقت خوبی نداشت و در نهایت در حدود سه درصد بود (تست‌شده در کولب) که البته در مقالات دیگر نیز دیده شد که برای چنین داده‌های پرنوسانی و با توجه به نوع تقسیم مناسب نمی‌باشد.

۲۵. در این سوال ۵ شاخص مختلف را استخراج کردیم که به ترتیب زیر می‌باشد:

```
def RSI(series, period = 5):
    delta = series.diff().dropna()
    u = delta * 0
    d = u.copy()
    u[delta > 0] = delta[delta > 0]
    d[delta < 0] = -delta[delta < 0]
    u[u.index[period-1]] = np.mean( u[:period] ) #first value is sum of avg gains
    u = u.drop(u.index[:(period-1)])
    d[d.index[period-1]] = np.mean( d[:period] ) #first value is sum of avg losses
    d = d.drop(d.index[:(period-1)])
    rs = pd.DataFrame.ewm(u, com=period-1, adjust=False).mean() / \
         pd.DataFrame.ewm(d, com=period-1, adjust=False).mean()
    return 100 - 100 / (1 + rs)
```

```
def SOI(ds,priod=14):
    ds['14-high'] = ds['high'].rolling(priod).max()
    ds['14-low'] = ds['low'].rolling(priod).min()
    ds['K'] = (ds['14-low'])*100/(ds['14-high'] - ds['14-low'])
    ds['D'] = ds['K'].rolling(3).mean()
    return ds
```

```
def SMA(series,w = 3):
    return series.rolling(window=w).mean()
```

```
def CMA(series,mp=4):
    return series.expanding(min_periods=mp).mean()
```

```
def EMA(series,sp=40):
    return series.ewm(span=sp,adjust=False).mean()
```

می‌باشد که برای مثال مانند میانگین متحرک‌ها حاصل روابط بین داده‌های قبلی است و نتیجه نهایی به صورت زیر گردید:

```
res[(res.pred * 0.95 <= res.real) & (res.pred * 1.05 >= res.real)].shape[0] / res.shape[0] * 100
```

```
100.0
```

```
from sklearn.metrics import mean_squared_error
mean_squared_error(res.pred,res.real)
```

می‌بینیم که دقت به صورت فزاینده‌ایی افزایش یافت و از خطا نیز کاسته‌شد؛ که می‌تواند دلیل آن کشف روابط پنهان بین داده‌ها می‌باشد. یعنی برای مثال شاخص قدرت وابسته بسیار به نوسان بالا وابسته است و اگر بسیار بالا باشد؛ نشان می‌دهد که داده‌ها به صورت تصاعدی و افراطی روبه‌رشد هستند یا میانگین‌های متحرک نیز نشان دهنده فراز و فرود در لحظه داده‌ها می‌باشند.

۲۶.