به نام خدا

گزارش تمرین سری دوم – قسمت اول درس داده کاوی

نام و نام خانوادگی: پویا شاعری

شماره دانشجویی: ۴۰۰۴۲۲۱۰۵

لینک Colab:

https://colab.research.google.com/drive/1U-xV5J941VoT0YEMf2k25aIn2XCgE58Y?usp=sharing

چکیده

در این گزارش قصد داریم مرحله به مرحله کارهای انجام شده را توضیح دهیم و تحلیل کنیم. گرچه در داخل نوت‌بوک، به صورت تکست بالای هر سلول توضیح داده شده است ولی ضروری است توضیحات و تحلیل های لازم و مقایسه ها را در اینجا داشته باشیم.

مقدمه

در این سری از تمارین به دلیل ایجاد تفاوت با سری پیش و در دسترس نبودن دیتاست قسمت دوم تصمیم گرفتیم که دیتاست را از گوگل‌درایو، mount کنیم که این کار را انجام دادیم.

در اینجا دو دیتاست train و test جدا از هم هستند که دیتاست train یک ستون اضافه‌تر دارد. این ستون، ستون price_range بوده و در دیتاست test موجود نمی‌باشد. از طرفی دیتاست تست یک ستون id اضافه دارد. پس لازم است وقتی به ابعاد این دو دیتاست از بیرون نگاه می‌کنیم، یکی بودن بعد فیچر ها را مبنی بر یکی بودن فیچر ها ندانیم.

بخش صفر: پیش‌پردازش داده ها

در این بخش باتوجه به راهنمایی صورت سوال ستون price_range را از چهار کلاس ۰ و ۱ و ۲ و ۳ ، به دو کلاس ۰ و ۱ بردیم. به این صورت که از ماسک کردن روی اینستنس هایی که ستون داده آنها متناظر با عدد ۰ و ۱ و سپس با دستور کانکت

```
pd.concat([df_lowPrice0, df_lowPrice1])
```

و برای دو کلاس گران ۲ و ۳ نیز به همین صورت و در نهایت هر چهار کلاس را که درواقع دو کلاس شده به هم کانکت کرده و اینبار داده های train_df را در ستون price_range با دو کلاس ۰ و ۱ داریم.

هندل کردن میسینگ‌ها: خوشبختانه در این دیتاست میسینگ نداشتیم.

همچنین همه داده ها عددی هستند.

یک آوتلایر دیتکشن z-score و نرمالایز (در آینده به هنگام مدل) کار را تمام می‌کند.

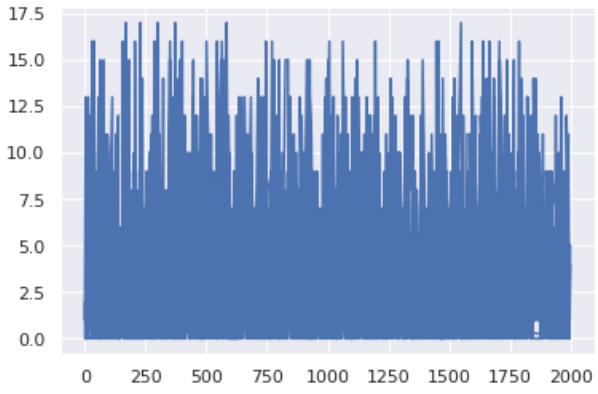
```python
# Remove outlier instances (considering the Numerical features)
# An instance is outlier if it's value is 3*std higher than mean (z-score = 3)
for col in train_df.columns:
  if train_df[col].dtype == 'int64' or train_df[col].dtype == 'float64':
    print('before', col, len(train_df), train_df[col].mean(), train_df[col].std())
    upper_range = train_df[col].mean() + 3 * train_df[col].std()
    lower_range = train_df[col].mean() - 3 * train_df[col].std()

    train_df = train_df[(train_df[col]>= lower_range) & (train_df[col]<= upper_range)]

    print('after', col, len(train_df), train_df[col].mean(), train_df[col].std())

```

```
before battery_power 2000 1238.5185 439.41820608353095
after battery_power 2000 1238.5185 439.41820608353095
before blue 2000 0.495 0.5001000400170029
after blue 2000 0.495 0.5001000400170029
before clock_speed 2000 1.5222500000000014 0.8160042088950716
after clock_speed 2000 1.5222500000000014 0.8160042088950716
before dual_sim 2000 0.5095 0.5000347661750046
after dual_sim 2000 0.5095 0.5000347661750046
before fc 2000 4.3095 4.341443747983878
after fc 1988 4.22635814889336 4.220050567308204
before four_g 1988 0.5206237424547284 0.4997001751072287
after four_g 1988 0.5206237424547284 0.4997001751072287
before int_memory 1988 32.089034205231385 18.128174760682253
```

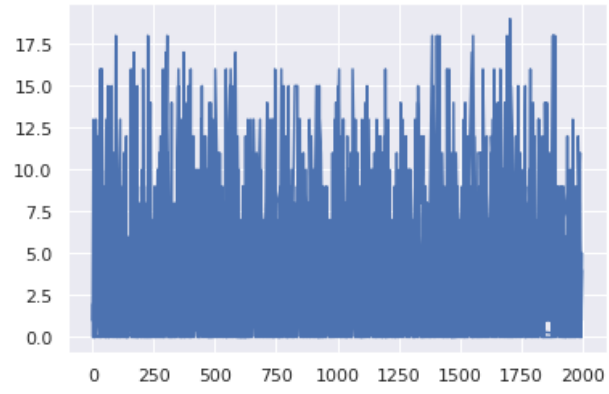
```
after int_memory 1988 32.089034205231385 18.128174760682253
before m_dep 1988 0.502012072434609 0.2884384765614282
after m_dep 1988 0.502012072434609 0.2884384765614282
before mobile_wt 1988 140.15945674044266 35.36107838039017
after mobile_wt 1988 140.15945674044266 35.36107838039017
before n_cores 1988 4.5241448692152915 2.2899795994324066
after n_cores 1988 4.5241448692152915 2.2899795994324066
before pc 1988 9.85814889336016 6.035626405240603
after pc 1988 9.85814889336016 6.035626405240603
before px_height 1988 643.9265593561369 442.9610386595417
after px_height 1988 643.9265593561369 442.9610386595417
before px_width 1988 1251.5357142857142 432.0867720643595
after px_width 1988 1251.5357142857142 432.0867720643595
before ram 1988 2126.5442655935612 1084.1863218216115
after ram 1988 2126.5442655935612 1084.1863218216115
before sc_h 1988 12.308350100603622 4.215626344896277
after sc_h 1988 12.308350100603622 4.215626344896277
before sc_w 1988 5.7716297786720325 4.361399049687776
after sc_w 1988 5.7716297786720325 4.361399049687776
before talk_time 1988 11.014587525150905 5.459397505050636
after talk_time 1988 11.014587525150905 5.459397505050636
before three_g 1988 0.7605633802816901 0.4268470146757874
after three_g 1988 0.7605633802816901 0.4268470146757874
before touch_screen 1988 0.5025150905432596 0.5001194746776364
after touch_screen 1988 0.5025150905432596 0.5001194746776364
before wifi 1988 0.5050301810865191 0.5001004922612214
after wifi 1988 0.5050301810865191 0.5001004922612214
before price_range 1988 0.5005030181086519 0.5001255488987242
after price_range 1988 0.5005030181086519 0.5001255488987242
```

آوتلایر زیادی نداشتیم. فقط ۱۲ رکورد به عنوان آوتلایر در ستون fc بودند. که نمودار توزیع آن ستون قبل و بعد از z-score رسم شده است و همینطورکه از نمودار مشخص است، تغییر ملموسی نکرد پس به دیتاست قبل از z-score بر می‌گردیم.

قبل از z-score



بعد از z-score



شاید این سوال پیش بیاید که شاید داده ها در ستون price_range، از ترتیب برخوردار شده‌اند. چراکه همه ۰ ها بالا و همه ۱ ها پایین هستند. در پاسخ میگوییم می‌توان شافل کرد یا وقتی که ترین و تست را اسپلیت میزنیم، آن را رندوم شافل اسپلیت می‌کند.

بخش اول: روش انتخاب ویژگی Forward Selection

تست و ترین اسپلیت کردیم و داده ها را اسکیل کردیم. ما در این روش انتخاب ویژگی برای این منظور در روش Forward Selection، از مدل لاجیستیک استفاده کرده ایم. میتوانستیم رندوم‌فارست را استفاده کنیم و اورهد کمتری هم داشت ولی از لاجیستیک استفاده کردیم و تابع آن را تعریف کرده ایم. سپس با استفاده از auc در هر مرحله در تابع forward به پیاده‌سازی آن پرداخته ایم.

```python
def LR(X_train, y_train, X_test, y_test, i):
    LR = LogisticRegression()
    LR = LR.fit(X_train, y_train)
    yhat = LR.predict(X_test)
    y_pred = LR.predict_proba(X_test)[:, 1]
    acc = roc_auc_score(y_test, y_pred)
    if isinstance(i, int):
        #Forward, Backward selection we need acc
        if i == 22:
            return acc
        #pca we need acc
        elif 0 < i < 16:
            return acc
        #Inbalance data
        elif i == 0 :
            message = f"Accuracy of sklearn's Logistic Regression Classifier: {acc}"
            return (message, yhat)
        #predict test data we need model
        elif i == 23:
            return LR
    #all features ,ram
    else:
        message = f"Accuracy of sklearn's Logistic Regression Classifier with {i}: {acc}"
        return message, yhat
```

تذکر: کد نوشته شده چند حالته است ولی ما با حالت i=22 فوروارد و بکوارد آن را ران گرفتیم.

```
[85]  1 def forward(X_train, y_train, X_test, y_test, best_cols, all_cols):
      2     init_acc = 0
      3     for col in all_cols:
      4         best_cols.append(col)
      5         X_train_f = pd.DataFrame(data=X_train, columns=best_cols)
      6         X_test_f = pd.DataFrame(data=X_test, columns=best_cols)
      7         acc = LR(X_train_f, y_train, X_test_f, y_test, i=22)
      8         if acc > init_acc:
      9             init_acc = acc
     10         else:
     11             best_cols.pop()
     12     return(best_cols, init_acc)
```

```
1 best_cols, all_cols = [], X.columns.to_list()
2 forward_col = forward(X_train, y_train, X_test, y_test, best_cols, all_cols)
3 print(forward_col)
```

```
(['battery_power', 'dual_sim', 'm_dep', 'n_cores', 'px_height', 'ram', 'sc_w'], 0.9968250000000001)
```

می‌بینیم که ۷ فیچر ماند.

بخش دوم: اعمال روش لاجیتیک روی ستون های بدست آمده

```
1 message, yhat = LR(X_train[forward_col[0]], y_train, X_test[forward_col[0]], y_test, forward_col[0])
2 print(message)
3 print(classification_report(y_test, yhat))
```

```
Accuracy of sklearn's Logistic Regression Classifier with ['battery_power', 'dual_sim', 'm_dep', 'n_cores', 'px_height', 'ram', 'sc_w']: 0.9968250000000001
              precision    recall  f1-score   support

           0       0.95      0.98      0.97       200
           1       0.98      0.95      0.97       200

    accuracy                           0.97       400
   macro avg       0.97      0.97      0.97       400
weighted avg       0.97      0.97      0.97       400
```

بخش امتیازی اول: روش Backward Selection و اعمال روش لاجیتیک روی ستون‌های بدست آمده

در این قسمت ابتدا همه ستون ها را درنظر گرفتیم و برعکس روال قبل را رفتیم. یعنی به حذف ستون ها پرداختیم تا بهتر شدن یا بدتر شدن نتیجه را مقایسه کنیم.

```
message, yhat = LR(X_train[backward_col[0]], y_train, X_test[backward_col[0]], y_test, backward_col[0])
print(message)
print(classification_report(y_test, yhat))
```

```
Accuracy of sklearn's Logistic Regression Classifier with ['wifi', 'battery_power', 'blue', 'int_memory']: 0.555475
              precision    recall  f1-score   support

           0       0.56      0.58      0.57       200
           1       0.56      0.54      0.55       200

    accuracy                           0.56       400
   macro avg       0.56      0.56      0.56       400
weighted avg       0.56      0.56      0.56       400
```

نتیجه خوبی از Backward بدست نیامد و به‌نظر می‌رسد که روش Forward Selection عملکرد بهتری داشت.

سوال سوم و چهارم: اعمال PCA و بررسی لاجیستیک روی داده های تغییریافته

در این قسمت ابتدا بار دیگر از روی داده اورجینال کپی میکنیم و پردازش داده را از پایه پیاده سازی کرده ایم. سپس از PCA استفاده کرده و تعداد component ها را برابر با ۷ قرار می دهیم که شرایط با قسمت قبل برابر باشد. PCA را بر روی داده ها fit کرده و آن ها را تغییر می دهیم. با استفاده از داده های تغییر یافته یک مدل لاجستیک پیاده سازی کرده و نتایج را برای ۴ کلاس به شرح زیر نمایش می دهیم. همانطور که مشاهده می شود با داده های تغییر یافته با PCA نتایج افت می کند.

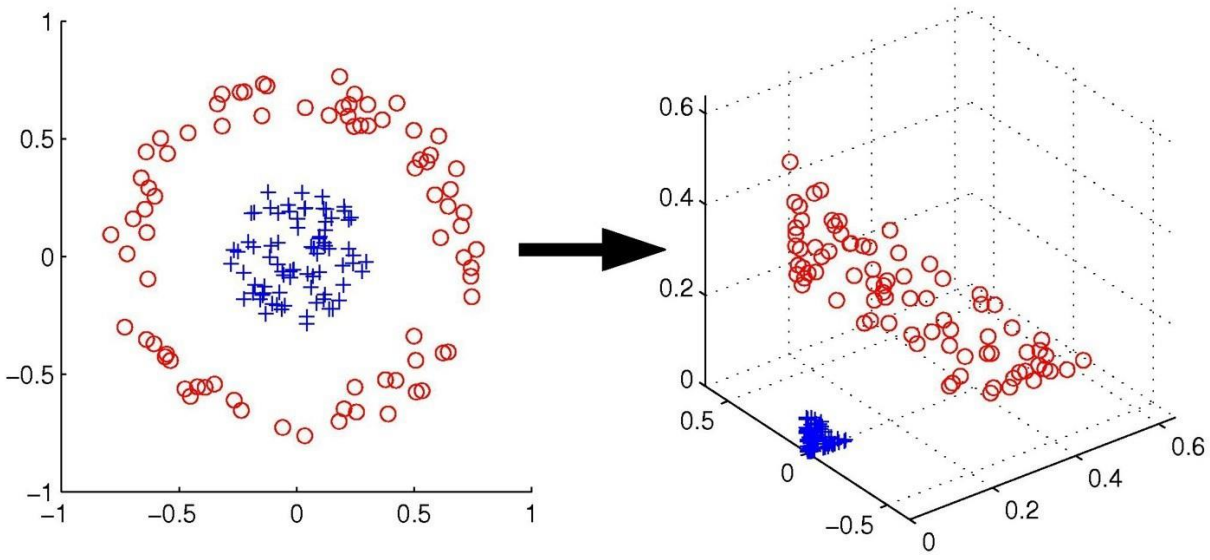
precision: 0.52, 0.30, 0.21, 0.51

recall: 0.58, 0.17, 0.21, 0.70

f1-score: 0.55, 0.22, 0.20, 0.59

سوال پنجم: کرنل های پرکاربرد روش SVM

اگر بتوان داده ها را توسط یک خط جدا کرد، داده ها را خطی‌جدایی پذیر و در غیر این صورت به آنها خطی جدایی ناپذیر می گویند و به کرنل ها نیازمند می شویم. مثال زیر این مساله را در دو بعد نشان می دهد. همانطور که در شکل چپ دیده می شود نمی توان رکورد های قرمز و آبی را با یک خط از هم جدا کرد. در چنین موقعیتی از کرنل استفاده می کنیم.



بدین صورت که اگر نمی توان در بعد فعلی (سمت چپ) کلاس ها را از هم جدا کرد یک بعد به مساله اضافه کرده (سمت راست) و سپس با یک ابرصفحه به جداکردن آن ها پرداخت. البته این کار به راحتی اضافه کردن یک بعد نیست. بلکه نیاز به تبدیل فضا (TRANSFORM) می‌باشد. یک نمونه برای تغییر بعد برای مثال بالا مانند مثال شکل سمت راست است.

این تغییر فضا کرنل نامیده می شود. کرنل ها انواع مختلفی دارند که می توان به موارد زیر اشاره کرد:

Polynomial Kernel, Gaussian Kernel ,Radial Basis Function (RBF), Laplace RBF Kernel, Sigmoid Kernel

نکته مهم در اینجا این است که همیشه نمی توان نظر قطعی و کلی برای استفاده از کرنل داد. شاید بتوانیم از کرنل خطی برای جدا کردن داده های خطی و از کرنل RBF برای تفکیک داده های غیرخطی استفاده کنیم اما به طور کلی برای استفاده از کرنل های پیچیده تر نیاز به بررسی بیشتر مساله داریم. باید توجه کرد که پیچیدگی کرنل RBF به علت داشتن پارامترهای بیشتر و اینکه باید ماتریس کرنل را حفظ کرد همیشه از کرنل خطی (تنها support vector ها کافی هستند) بیشتر بوده و بنابراین باید نهایت دقت رادر انتخاب کرنل مورد نظر کرد. به طور کلی و بر اساس نظریه Occam's Razor پیشنهاد شده است که همیشه از ساده ترین کرنل یعنی کرنل خطی شروع کرده و تنها در صورت نیاز به کرنل های پیچیده تر پناه برد.

سوال ششم: اعمال روش SVM

در این سوال ۸۰ درصد داده را برای ترین و ۲۰ درصد را برای تست انتخاب و عملکرد مدل را باتوجه به معیارهای زیر ارزیابی می‌کنیم.

- Accuracy
- Precision, recall and F1 for each class, and also averaged over all 4 classes

برای پیاده سازی از پکیج sklearn.svm و از تابع SVC استفاده می‌کنیم.

سوال هفتم

برای کلسیفایر svm پارامترهای مختلفی را امتحان کردیم که عبارتند از:

- Kernel: ‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’
- Gamma: if kernel is 'poly', 'rbf', 'sigmoid', then examine two possible values for gamma:
    - Auto, scale

- multi-label classification setting: one-versus-one (ovo), one-vs-rest (ovr)
- C (which is the regularizer): default value 1

که مقادیر مختلف این پارامتر در سوال های بعدی امتحان شده است.

به طور کلی ۱۴ حالت مختلف را بررسی کردیم. نتایج برای multi-label classification setting برای هر دو حالت ovo و ovr یکسان بود. بنابراین نتایج را فقط برای یک حالت گزارش می‌کنیم.

- بهترین عملکرد از نظر accuracy برای حالتی است که از linear kernel استفاده میکنیم و accuracy = 0.9725
- بهترین عملکرد از نظر میانگین ۴ کلاس برای F1 برای حالتی است که از linear kernel استفاده میکنیم و F1 = 0.9721
- بهترین عملکرد از نظر میانگین ۴ کلاس برای precision برای حالتی است که از linear kernel استفاده میکنیم و precision = 0.9724
- بهترین عملکرد از نظر میانگین ۴ کلاس برای recall برای حالتی است که از linear kernel استفاده میکنیم و recall = 0.9719

```
SVM results for kernel linear-- accuracy:  0.9723618090452262
Precision:  0.9723 [0.99019608 0.95192308 0.96703297 0.98019802]
Recall:  0.9717 [0.99019608 0.97058824 0.93617021 0.99      ]
Fscore:  0.9719 [0.99019608 0.96116505 0.95135135 0.98507463]
=========================
SVM results for kernel poly and Kernel coeff auto-- accuracy:  0.7914572864321608
Precision:  0.8033 [0.89583333 0.7029703  0.66086957 0.95348837]
Recall:  0.7919 [0.84313725 0.69607843 0.80851064 0.82      ]
Fscore:  0.7943 [0.86868687 0.69950739 0.72727273 0.88172043]
=========================
SVM results for kernel poly and Kernel coeff scale-- accuracy:  0.7914572864321608
Precision:  0.8033 [0.89583333 0.7029703  0.66086957 0.95348837]
Recall:  0.7919 [0.84313725 0.69607843 0.80851064 0.82      ]
Fscore:  0.7943 [0.86868687 0.69950739 0.72727273 0.88172043]
=========================
SVM results for kernel rbf and Kernel coeff auto-- accuracy:  0.8819095477386935
Precision:  0.8834 [0.92929293 0.80188679 0.83333333 0.96907216]
Recall:  0.8816 [0.90196078 0.83333333 0.85106383 0.94      ]
Fscore:  0.8823 [0.91542289 0.81730769 0.84210526 0.95431472]
=========================
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.8819095477386935
Precision:  0.8834 [0.92929293 0.80188679 0.83333333 0.96907216]
Recall:  0.8816 [0.90196078 0.83333333 0.85106383 0.94      ]
Fscore:  0.8823 [0.91542289 0.81730769 0.84210526 0.95431472]
=========================
SVM results for kernel sigmoid and Kernel coeff auto-- accuracy:  0.9120603015075377
Precision:  0.9141 [0.96938776 0.85981308 0.84848485 0.9787234 ]
Recall:  0.9117 [0.93137255 0.90196078 0.89361702 0.92      ]
Fscore:  0.9123 [0.95       0.88038278 0.87046632 0.94845361]
=========================
SVM results for kernel sigmoid and Kernel coeff scale-- accuracy:  0.9271356783919598
Precision:  0.928 [0.97       0.87850467 0.88421053 0.97916667]
Recall:  0.9265 [0.95098039 0.92156863 0.89361702 0.94      ]
Fscore:  0.927 [0.96039604 0.89952153 0.88888889 0.95918367]
=========================
```

سوال هشتم

پارامتر C مسئول کنترل Soft margin و یا Hard margin بودن کلسیفایر SVM است. هرچه مقدار این پارامتر کمتر باشد به این معنی است که regularization کمتر است و تنبیه کمتری برای missclassification در نظر گرفته می‌شود و در نتیجه به عبارتی softer margin خواهیم داشت و هرچه مقدار C بیشتر باشد harder margin خواهیم داشت. برای پاسخ به این سوال مقادیر مختلفی از C را آزمایش کردیم

{0.0001, 0.001, 0.1, 1, 10, 100, 1e10}

برای اینکه مطمپن باشیمhard ترین marging را درنظر گرفتیم و مدل عملا hard margin است مقدار 1e10 را نیز امتحان کردیم و آن را در نتایج حالت hard margin نامیدیم.

در زیر نتایج soft and hard margin برای کرنل های مختلف را میبینیم و حالت کلسیفیکیشن ovr :

- Linear kernel

```
Soft Margin, C = 0.0001
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWa
  _warn_prf(average, modifier, msg_start, len(result))
SVM results for kernel linear-- accuracy:  0.23618090452261306
Precision:  0.059 [0.        0.        0.2361809 0.       ]
Recall:  0.25 [0. 0. 1. 0.]
Fscore:  0.0955 [0.         0.         0.38211382 0.        ]
Soft Margin, C = 0.001
SVM results for kernel linear-- accuracy:  0.5100502512562815
Precision:  0.5874 [0.70093458 0.24324324 0.40528634 1.        ]
Recall:  0.5181 [0.73529412 0.08823529 0.9787234  0.27      ]
Fscore:  0.4614 [0.71770335 0.1294964  0.57320872 0.42519685]
Soft Margin, C = 0.1
SVM results for kernel linear-- accuracy:  0.9472361809045227
Precision:  0.9479 [0.98979592 0.90740741 0.93406593 0.96039604]
Recall:  0.9465 [0.95098039 0.96078431 0.90425532 0.97      ]
Fscore:  0.9469 [0.97       0.93333333 0.91891892 0.96517413]
Soft Margin, C = 1
SVM results for kernel linear-- accuracy:  0.9723618090452262
Precision:  0.9723 [0.99019608 0.95192308 0.96703297 0.98019802]
Recall:  0.9717 [0.99019608 0.97058824 0.93617021 0.99      ]
Fscore:  0.9719 [0.99019608 0.96116505 0.95135135 0.98507463]
Soft Margin, C = 10
SVM results for kernel linear-- accuracy:  0.964824120603015
Precision:  0.9651 [0.99       0.93396226 0.94680851 0.98979592]
Recall:  0.9645 [0.97058824 0.97058824 0.94680851 0.97      ]
Fscore:  0.9647 [0.98019802 0.95192308 0.94680851 0.97979798]
Soft Margin, C = 100
SVM results for kernel linear-- accuracy:  0.9698492462311558
Precision:  0.9708 [0.99009901 0.92592593 0.96703297 1.        ]
Recall:  0.9692 [0.98039216 0.98039216 0.93617021 0.98      ]
Fscore:  0.9697 [0.98522167 0.95238095 0.95135135 0.98989899]
Hard Margin
SVM results for kernel linear-- accuracy:  0.964824120603015
Precision:  0.9655 [0.98058252 0.92523364 0.95604396 1.        ]
Recall:  0.9641 [0.99019608 0.97058824 0.92553191 0.97      ]
Fscore:  0.9645 [0.98536585 0.94736842 0.94054054 0.98477157]
=========================
```

  - همانطور که میبینیم اگر مدل خیلی soft margin باشد (C=0.0001) نتایج بد است ولی اگر کاملا هم hard margin (C=1e10) باشد باز نتایج خوب نیست. بهترین حالت زمانی است که بین این دو حالت است کمی soft و کمی هم hard باشد C = 1 که دقت برابر با ۰.۹۷۲۳ است F1 نیز بیشترین مقدار را در این حالت دارد.

- Poly Kernel

```
Soft Margin, C = 0.0001
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Undefinec
  _warn_prf(average, modifier, msg_start, len(result))
SVM results for kernel poly and Kernel coeff scale-- accuracy:  0.23618090452261306
Precision:  0.059 [0.         0.         0.2361809 0.        ]
Recall:  0.25 [0. 0. 1. 0.]
Fscore:  0.0955 [0.         0.         0.38211382 0.        ]
Soft Margin, C = 0.001
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Undefinec
  _warn_prf(average, modifier, msg_start, len(result))
SVM results for kernel poly and Kernel coeff scale-- accuracy:  0.23618090452261306
Precision:  0.059 [0.         0.         0.2361809 0.        ]
Recall:  0.25 [0. 0. 1. 0.]
Fscore:  0.0955 [0.         0.         0.38211382 0.        ]
Soft Margin, C = 0.1
SVM results for kernel poly and Kernel coeff scale-- accuracy:  0.5301507537688442
Precision:  0.6602 [0.86666667 0.40740741 0.36653386 1.        ]
Recall:  0.5378 [0.76470588 0.10784314 0.9787234  0.3       ]
Fscore:  0.4945 [0.8125     0.17054264 0.53333333 0.46153846]
Soft Margin, C = 1
SVM results for kernel poly and Kernel coeff scale-- accuracy:  0.7914572864321608
Precision:  0.8033 [0.89583333 0.7029703  0.66086957 0.95348837]
Recall:  0.7919 [0.84313725 0.69607843 0.80851064 0.82      ]
Fscore:  0.7943 [0.86868687 0.69950739 0.72727273 0.88172043]
Soft Margin, C = 10
SVM results for kernel poly and Kernel coeff scale-- accuracy:  0.7738693467336684
Precision:  0.7812 [0.85148515 0.66346154 0.66666667 0.94318182]
Recall:  0.7736 [0.84313725 0.67647059 0.74468085 0.83      ]
Fscore:  0.7759 [0.84729064 0.66990291 0.70351759 0.88297872]
Soft Margin, C = 100
SVM results for kernel poly and Kernel coeff scale-- accuracy:  0.7713567839195979
Precision:  0.7806 [0.84158416 0.65420561 0.67307692 0.95348837]
Recall:  0.7711 [0.83333333 0.68627451 0.74468085 0.82      ]
Fscore:  0.774 [0.83743842 0.66985646 0.70707071 0.88172043]
Hard Margin
SVM results for kernel poly and Kernel coeff scale-- accuracy:  0.7713567839195979
Precision:  0.7806 [0.84158416 0.65420561 0.67307692 0.95348837]
Recall:  0.7711 [0.83333333 0.68627451 0.74468085 0.82      ]
Fscore:  0.774 [0.83743842 0.66985646 0.70707071 0.88172043]
```

- همانطور که میبینیم اگر مدل خیلی soft margin باشد (C=0.0001)نتایج بد است ولی اگر کاملا هم hard margin (C=1e10) باشد باز نتایج خوب نیست. بهترین حالت زمانی است که بین این دو حالت است کمی soft و کمی هم hard باشد C = 1 که دقت برابر با ۰.۷۹۱۴ است F1 نیز بیشترین مقدار را در این حالت دارد.

- RBF Kernel

```
========================
Soft Margin, C = 0.0001
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning:
  _warn_prf(average, modifier, msg_start, len(result))
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.2375
Precision:  0.0594 [0.     0.     0.2375 0.    ]
Recall:  0.25 [0. 0. 1. 0.]
Fscore:  0.096 [0.         0.         0.38383838 0.        ]
Soft Margin, C = 0.001
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning:
  _warn_prf(average, modifier, msg_start, len(result))
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.2375
Precision:  0.0594 [0.     0.     0.2375 0.    ]
Recall:  0.25 [0. 0. 1. 0.]
Fscore:  0.096 [0.         0.         0.38383838 0.        ]
Soft Margin, C = 0.1
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.74
Precision:  0.7741 [0.93975904 0.60714286 0.58730159 0.96202532]
Recall:  0.7408 [0.75       0.66666667 0.77894737 0.76767677]
Fscore:  0.7483 [0.8342246  0.63551402 0.66968326 0.85393258]
Soft Margin, C = 1
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.8875
Precision:  0.891 [0.96808511 0.81081081 0.83673469 0.94845361]
Recall:  0.8875 [0.875      0.88235294 0.86315789 0.92929293]
Fscore:  0.8882 [0.91919192 0.84507042 0.84974093 0.93877551]
Soft Margin, C = 10
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.885
Precision:  0.885 [0.93069307 0.82857143 0.875      0.90566038]
Recall:  0.8843 [0.90384615 0.85294118 0.81052632 0.96969697]
Fscore:  0.8839 [0.91707317 0.84057971 0.84153005 0.93658537]
Soft Margin, C = 100
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.8925
Precision:  0.8938 [0.94897959 0.82568807 0.88636364 0.91428571]
Recall:  0.8918 [0.89423077 0.88235294 0.82105263 0.96969697]
Fscore:  0.8919 [0.92079208 0.85308057 0.85245902 0.94117647]
Hard Margin
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.8925
Precision:  0.8938 [0.94897959 0.82568807 0.88636364 0.91428571]
Recall:  0.8918 [0.89423077 0.88235294 0.82105263 0.96969697]
Fscore:  0.8919 [0.92079208 0.85308057 0.85245902 0.94117647]
========================
```

- همانطور که میبینیم اگر مدل خیلی soft margin باشد (C=0.0001)نتایج بد است ولی اگر کاملا هم hard margin (C=1e10) باشد باز نتایج خوب نیست. بهترین حالت زمانی است که بین این دو حالت است کمی soft و کمی هم hard باشد C=100 که دقت برابر با ۰.۸۹۲۵ است F1 .نیز بیشترین مقدار را در این حالت دارد. در این کرنل مدل کمی hard marginتر از کرنلهای قبلی است تا نتایج بهتری بگیریم.

- Sigmoid kernel

```
Soft Margin, C = 0.0001
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarni
  _warn_prf(average, modifier, msg_start, len(result))
SVM results for kernel sigmoid and Kernel coeff scale-- accuracy:  0.2375
Precision:  0.0594 [0.     0.     0.2375 0.    ]
Recall:  0.25 [0. 0. 1. 0.]
Fscore:  0.096 [0.         0.         0.38383838 0.        ]
Soft Margin, C = 0.001
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarni
  _warn_prf(average, modifier, msg_start, len(result))
SVM results for kernel sigmoid and Kernel coeff scale-- accuracy:  0.2375
Precision:  0.0594 [0.     0.     0.2375 0.    ]
Recall:  0.25 [0. 0. 1. 0.]
Fscore:  0.096 [0.         0.         0.38383838 0.        ]
Soft Margin, C = 0.1
SVM results for kernel sigmoid and Kernel coeff scale-- accuracy:  0.8725
Precision:  0.883 [0.96907216 0.79130435 0.77142857 1.        ]
Recall:  0.8718 [0.90384615 0.89215686 0.85263158 0.83838384]
Fscore:  0.874 [0.93532338 0.83870968 0.81       0.91208791]
Soft Margin, C = 1
SVM results for kernel sigmoid and Kernel coeff scale-- accuracy:  0.905
Precision:  0.906 [0.96116505 0.84684685 0.85869565 0.95744681]
Recall:  0.9035 [0.95192308 0.92156863 0.83157895 0.90909091]
Fscore:  0.9042 [0.95652174 0.88262911 0.84491979 0.93264249]
Soft Margin, C = 10
SVM results for kernel sigmoid and Kernel coeff scale-- accuracy:  0.845
Precision:  0.8458 [0.9047619  0.78899083 0.78723404 0.90217391]
Recall:  0.8435 [0.91346154 0.84313725 0.77894737 0.83838384]
Fscore:  0.8441 [0.90909091 0.81516588 0.78306878 0.86910995]
Soft Margin, C = 100
SVM results for kernel sigmoid and Kernel coeff scale-- accuracy:  0.8375
Precision:  0.839 [0.89719626 0.76146789 0.76344086 0.93406593]
Recall:  0.8357 [0.92307692 0.81372549 0.74736842 0.85858586]
Fscore:  0.8367 [0.90995261 0.78672986 0.75531915 0.89473684]
Hard Margin
SVM results for kernel sigmoid and Kernel coeff scale-- accuracy:  0.815
Precision:  0.8195 [0.88349515 0.71304348 0.75824176 0.92307692]
Recall:  0.8134 [0.875      0.80392157 0.72631579 0.84848485]
Fscore:  0.8153 [0.87922705 0.75576037 0.74193548 0.88421053]
=========================
```

- همانطور که میبینیم اگر مدل خیلیsoft margin باشد (C=0.0001)نتایج بد است ولی اگر کاملا هم hard margin (C=1e10) باشد باز نتایج خوب نیست. بهترین حالت زمانی است که بین این دو حالت است کمی soft و کمی هم hard باشد C =1 که دقت برابر با ۰.۹۰۵ است F1 .نیز بیشترین مقدار را در این حالت دارد.

سوال نهم و دهم

برای تمامی حالت ها در این دو سوال از کرنل RBF و با گاما scale استفاده کردیم و مدل svm را ترین کردیم. قابل ذکر است که عملکرد مدل با این پارامترها و با تمامی فیچرها به طور معمولی به شرح زیر است:

```
=========================
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.8875
Precision:  0.891 [0.96808511 0.81081081 0.83673469 0.94845361]
Recall:  0.8875 [0.875      0.88235294 0.86315789 0.92929293]
Fscore:  0.8882 [0.91919192 0.84507042 0.84974093 0.93877551]
```

آ) Binning ‘battery power’ feature

سه حالت مختلف برای سایز بین ها در نظر گرفتیم. ۵ بین , ۱۰ بین , و۵ بین با سایز مختلف:

- 5 equal bins
- 10 equal bins
- 5 unequal bins:

(-inf, -1.67939025], (-1.67939025, -1.2], (-1.2 -0.3), (-0.3 0], (0, 1.6], (1.6, 1.73016828]

نتایج را برای هر سه حالت در زیر میبینیم. دقت کنین که در این حالت باقی فیچر ها به شکل قبل هستند و تنها فیچر battery power را پردازش کردیم(در کل به مدل تمامی فیچرها داده شده است به جز فیچر battery power که نسخه پردازش شده داده شده است)

```
========== 5 equal bins ============
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.9025
Precision:  0.9034 [0.94059406 0.85294118 0.85148515 0.96875   ]
Recall:  0.9028 [0.91346154 0.85294118 0.90526316 0.93939394]
Fscore:  0.9028 [0.92682927 0.85294118 0.87755102 0.95384615]
========================
========== 10 equal bins ============
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.895
Precision:  0.8982 [0.96842105 0.85046729 0.82692308 0.94680851]
Recall:  0.8953 [0.88461538 0.89215686 0.90526316 0.8989899 ]
Fscore:  0.8955 [0.92462312 0.8708134  0.86432161 0.92227979]
========================
========== 5 unequal bins ============
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.8825
Precision:  0.8835 [0.92156863 0.81730769 0.83673469 0.95833333]
Recall:  0.8824 [0.90384615 0.83333333 0.86315789 0.92929293]
Fscore:  0.8828 [0.91262136 0.82524272 0.84974093 0.94358974]
========================
```

همانطور که میبینیم بهترین عملکرد برای حالتی است که ۵ بین با سایز یکسان داشته باشیم و عملکرد نسبت معمولی که هیچ Binning نداریم بهتر است (دقت ۰.۹۰۲۵ در مقایسه با ۰.۸۸۷۵)

ب) One hot encoding for categorical features

بسیاری از الگوریتم های یادگیری ماشین نیاز به فیچرهای با مقدار عددی (numeric) دارند. این در حالیست که فیچرهای کتگوریکال مقادیر عددی نداشته و به جای آن برچسبی (label) یا کتگوری (category) هستند (به عنوان نمونه فیچر رنگ دارای مقادیر سفید سیاه قرمز و ... می باشد). بنابراین نیاز به روش های کدگذاری نظیر integer و one-hot داریم که بتوان این فیچرها را برای استفاده در الگوریتم های مختلف آماده کرد. در روش کدگذاری one-hot از متغیرهای باینری برای تبدیل فیچر استفاده می کنیم. به عنوان نمونه در مثال رنگ می توان از کدگذاری زیر استفاده کرد:

سفید	سیاه	قرمز
۱	۰	۰
۰	۱	۰
۰	۰	۱

در این قسمت فیچرهای categorical را با روش One Hot encoding پردازش کردیم. در مجموع ۶ فیچر categorical داشتیم:

categorical_features = ['blue', 'dual_sim', 'four_g', 'three_g', 'touch_screen', 'wifi']

نتایج را در زیر برای حالتی که این فیچرهای جدید را استفاده کردیم به شرح زیر است: همانطور که میبینیم عملکرد در این حالت نسبت به عملکرد با حالت معمولی که هیچ one hot encoding نداریم کمی بدتر شده است.

```
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.8825
Precision:  0.8843 [0.93877551 0.80555556 0.84375    0.94897959]
Recall:  0.8824 [0.88461538 0.85294118 0.85263158 0.93939394]
Fscore:  0.8829 [0.91089109 0.82857143 0.84816754 0.94416244]
```

ج) Log and Exponential Transformation

برای این سوال transformation های مختلفی را برای هر فیچر جداگانه بررسی کردیم:

- Log transformation
- Exponential transformation with base ‘e’
- Power 2 transformation
- Square root transformation

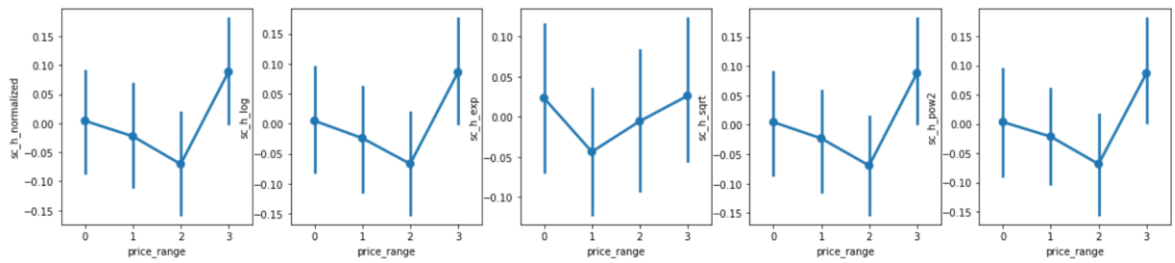
برای هر کدام از این تغییرات ,تابع مورد نظر را روی فیچر انتخابی اعمال کرده و عملکرد مدل svm را با فیچر جدید (در حال یکه باقی فیچرها مقدار طبیعی خود را دارند) مقایسه کردیم. همچنین نموداری از رنج مقادیر این فیچر تبدیل شده برای هر کلاس رسم کردیم. هرچه مقادیر فیچر برای هر کلاس از کلاس دیگر قابل تمیز تر باشد، می‌توان گفت آن تبدیل مناسب بوده است چراکه به جدا سازی داده های کلاسهای مختلف کمک کرده است. اگر مقدار جداسازی آن فیچر نسبت به کلاس های مختلف در قبل از تبدیل با بعد از تبدیل تفاوتی نکند و همچنین عملکرد مدل در classification بهبود قابل توجهی نیابد، می‌توان گفت که تبدیل تاثیر خاصی نداشته است. لزوما تبدیل ها عملکرد مدل را بهبود نمیدهند. بنابراین این مساله برای هر فیچر باید بررسی شود. گاهی تبدیل ها جداسازی داده ها را بدتر هم میکنن چرا که قدرت جداسازی فیچر در فضای جدید بعد از تبدیل کمتر میشود. اما در برخی موارد تبدیل ها باعث میشودند که فیچرها به فضای جدید بروند که جداسازی داده های کلاس های مختلف در ان فضا بهتر باشد و در نتیجه دقت و عملکرد مدل بهبود یابد. به طور مثال فیچر n_cores بعضی تبدیل ها مانند تبدیل اکسپوننشیال باعث میشوند که جدساازی کلاسها با توجه به مقدار این فیچر بهتر و راحتتر شود و در نتیجه عملکرد و دقت مدل بهبود یابد. برای مثال این فیچر تعداد کورهای موبایل را نشان میدهد و هرچه بیشتر باشد قاعدتا قیمت موبایل باید بالاتر باشد. تبدیل اکسپوننشیال باعث میشود که کورهای بالاتر مقدار بالاتری برای این فیچر داشته باشند و بهتر از داده های با کور پایین تر قابل جداسازی باشند. بنابراین این تبدیل باعث میشود که کلاس قیمت بالاتر جداسازی بهتری داشته باشد.

برای سنجیدن عملکرد مدل از svm with RBF Kernel and scale gamma استفاده کردیم و عملکرد مدل را با accuracy گزارش دادیم. در زیر نتایج تبدیل برای هر یک از فیچرها را میبینیم. دقت کنین که بدون هیچ گونه تبدیلی روی فیچر ها دقت عملکرد مدل ۰.۸۸۷۵ می‌باشد.

Sc_h Feature:

دقت مدل برای هر کدام از تبدیل ها از چپ به راست (از بالا به پایین) نوشته شده است. به این معنی که اگر تبدیلی لگاریتمی بکنیم عملکرد ۰.۸۹۲۵و اگر تبدیل power 2 را اعمال کنیم دقت ۰.۸۸۷۵ خواهد بود.

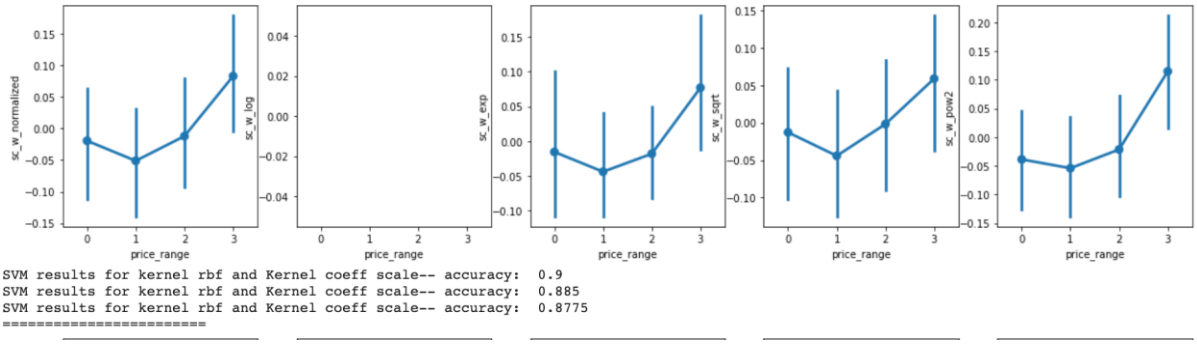
هر یک از نمودار ها رنج مقدار فیچر را برای هر یک از کلاس ها را نشان میدهد. هر نمودار مربوط به یک تابع تبدیل است. چپ ترین نمودار هیچ تبدیلی نداشته است و مقدار فیچر تنها normalized شده است.همانطور که مشخص است تبدیل وسطی تاثیری در بهتر جداسازی فیچر در کلاسهای مختلف نداشته است و مقادیر فیچر در کلاس های ۰و۲و۳ همپوشانی زیادی دارند ولی نمودار سمت چپ کلاس ۳م مقادیر متفاوت تری از باقی کلاس ها دارد هم چنین کلاس ۲ مقادیر متفاوتی از سه کلاس دیگر دارد. در کل ۴ تابع تبدیل تاثیر چندان و چشم گیری در بهتر جداسازی کلاس ها با توجه به این فیچر نداشته اند. هم چنین اگر به عملکرد مدل با توجه به هرکدام از این تبدیل ها نگاه کنیم , دقت مدل تغییر چشم گیری نداشته است و دقت تنها ۰.۰۰۵ افزایش داشته است (۰.۸۸۷۵ بدون تابع تبدیل به نسبت ۰.۸۹۲۵ با تابع تبدیل لگاریتم و یا توانی اکسپوننشیال)




```
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.8925
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.8925
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.89
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.8875
========================
```

sc_w Feature:

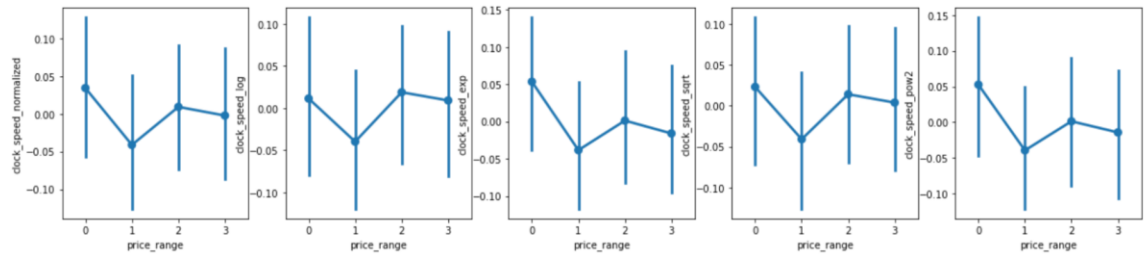
همانطور که مشخص است تبدیل اکسپوننشیال دقت مدل را نسبت به باقی تبدیل ها بیشتر بهبود داده است (۰.۹با تبدیل اکسپونشیان به نسبت ۰.۸۸۷۵ بدون تبدیل) . هم چنین اگر مقدار رنج فیچرها با این تبدیل را با حالت های دیگر مقایسه کنیم مشاهده میکنیم که داده های کلاس صفر اندکی بهتر از حالت های دیگر قابل تمیز از داده های کلاسهای ۲ و ۳ هستند (در نمودارهای دیگر میبینیم این سه کلاس هم پوشانی بیشتری دارند). بعد از تبدیل داده های کلاس صفر تا رنج ۰.۱ قرار گرفتند و در حالت بدون تبدیل (چپ ترین نمودار) تا رنج کمتر از ۰.۰۵ بودند که با داده های کلاس ۲و ۳ هم پوشانی بیشتری داشت.




```
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.9
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.885
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.8775
========================
```

Clock speed Feature:

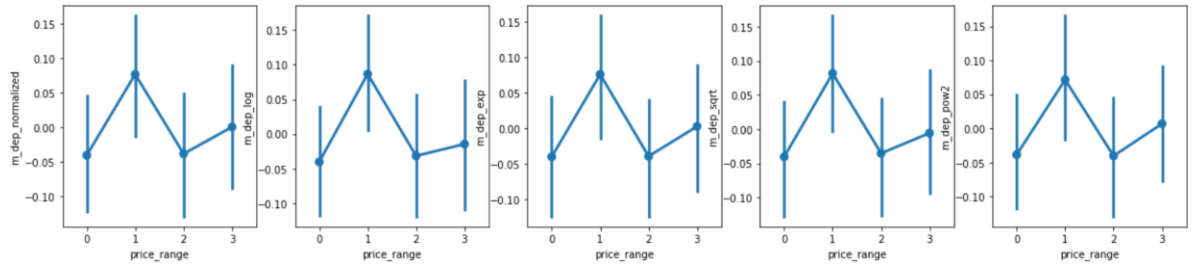
همانطور که مشخص است تبدیل اکسپوننشیال دقت مدل را نسبت به باقی تبدیل ها بیشتر بهبود داده است (۰.۹با تبدیل اکسپونشیان به نسبت ۰.۸۸۷۵ بدون تبدیل) . هم چنین اگر مقدار رنج فیچرها با این تبدیل را با حالت های دیگر مقایسه کنیم مشاهده میکنیم که داده های کلاس صفر اندکی بهتر از حالت های دیگر قابل تمیز از داده های کلاسهای ۲ و ۳ هستند (در نمودرهای دیگر میبینیم این سه کلاس هم پوشانی بیشتری دارند). بعد از تبدیل داده های کلاس صفر در رنج [۰.۱ ,۰.۱۵] قرار گرفتند و در حالت بدون تبدیل (چپ ترین نمودار) در رنج کمتر از ۰.۱ بودند که با داده های کلاس ۲و ۳ هم پوشانی بیشتری داشت.




```
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.8875
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.9
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.885
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.8975
========================
```

m_dep Feature:

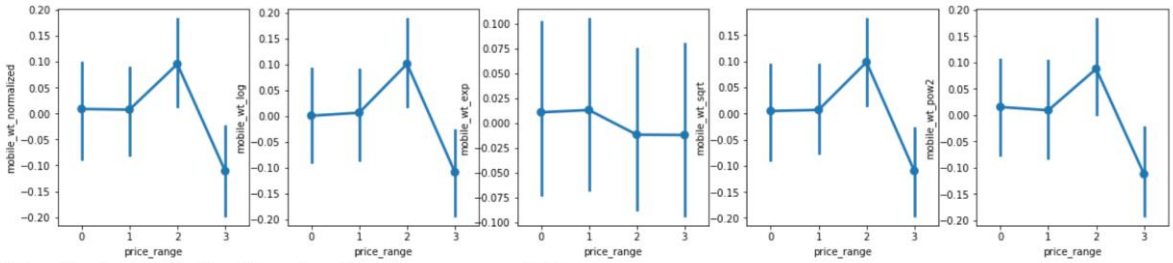
با توجه به ۵ نمودار زیر میتوان گفت که رنج مقداری فیچرها بعد از هیچ کدام از تبدیل ها فرق چشم گیری با قبل تبدیل نداشته است و قدرت تمیز دادن داده های کلاس های مختلف تغییری نکرده است. هم چنین دقت مدل بهبود چشمگیری نداشته است (۰.۸۹۲۵ در بهترین حالت تبدیل ها به نسبت ۰.۸۸۷۵ بدون هیچ گونه تبدیلی). حتی میتوان گفت که تبدیل توان دوم اثر منفی داشته است چرا که دقت عملکرد مدل به ۰.۸۸۵ کاهش یافته است.




```
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.8925
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.8875
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.8925
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.885
======================
```

mobile_wt Feature:

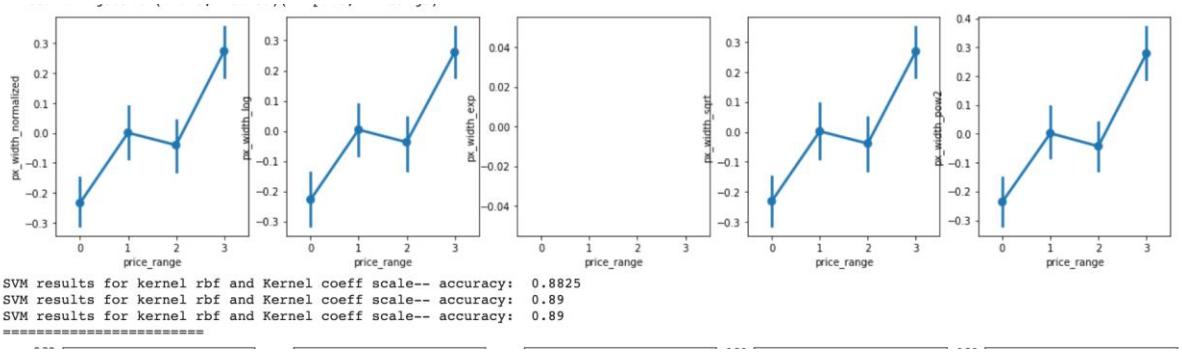
با توجه به ۵ نمودار زیر میتوان گفت که رنج مقداری فیچرها بعد از هیچ کدام از تبدیل ها فرق چشم گیری با قبل تبدیل نداشته است و قدرت تمیز دادن داده های کلاس های مختلف تغییری نکرده است. هم چنین دقت مدل بهبود چشمگیری نداشته است (۰.۸۹ در بهترین حالت تبدیل ها به نسبت ۰.۸۸۷۵ بدون هیچ گونه تبدیلی). حتی میتوان گفت که تبدیل اکسپوننشیال اثر منفی داشته است چرا که دقت عملکرد مدل به ۰.۸۷۵ کاهش یافته است و جداسازی کلاس ها با این فیچر بدتر شده است چرا که رنج مقدار این فیچر در کلاسهای مختلف اشتراک زیادی داشته است!(نمودار سوم وسطی)




```
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.885
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.875
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.885
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.89
======================
```

px_width Feature:

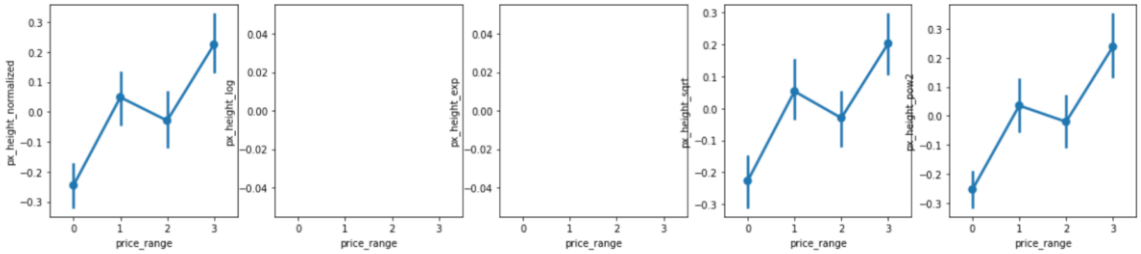
با توجه به ۵ نمودار زیر میتوان گفت که رنج مقداری فیچرها بعد از هیچ کدام از تبدیل ها فرق چشم گیری با قبل تبدیل نداشته است و قدرت تمیز دادن داده های کلاس های مختلف تغییری نکرده است. هم چنین دقت مدل بهبود چشمگیری نداشته است (۰.۸۹ در بهترین حالت تبدیل ها به نسبت ۰.۸۸۷۵ بدون هیچ گونه تبدیلی). حتی میتوان گفت که تبدیل لگاریتمی اثر منفی داشته است چرا که دقت عملکرد مدل به ۰.۸۸۲۵ کاهش یافته است. تبدیل نمایی برای این فیچر باعث شده است که اکثر داده ها مقدار بسیار بسیار زیادی بگیرند و overflow کرده و مقدار آنها در متغیر float جا نشود بنابراین نمودار مربوط به این تغییر خالی است و هم چنین دقت مدل با این تبدیل گزارش نشده است.




```
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.8825
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.89
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.89
========================
```

px_height Feature:

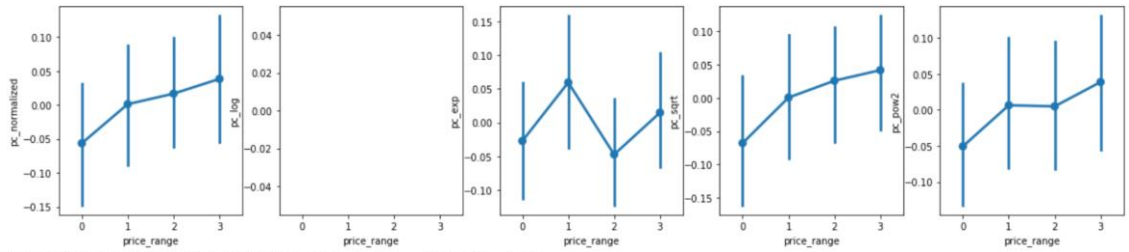
با توجه به ۵ نمودار زیر میتوان گفت که رنج مقداری فیچرها بعد از هیچ کدام از تبدیل ها فرق چشم گیری با قبل تبدیل نداشته است و قدرت تمیز دادن داده های کلاس های مختلف تغییری نکرده است. هم چنین دقت مدل بهبود چشمگیری نداشته است (۰.۸۹ در بهترین حالت تبدیل ها به نسبت ۰.۸۸۷۵ بدون هیچ گونه تبدیلی). تبدیل نمایی برای این فیچر باعث شده است که اکثر داده ها مقدار بسیار بسیار زیادی بگیرند و overflow کرده و مقدار آنها در متغیر float جا نشود بنابراین نمودار مربوط به این تغییر خالی است و هم چنین دقت مدل با این تبدیل گزارش نشده است. به طور مشابهی تبدیل لگاریتمی نیز باعث شده است مقادیر این فیچر برای اکثر داده ها nanشود و دقت مدل گزارش نشده است.




```
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.89
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.89
=======================
```

pc Feature:

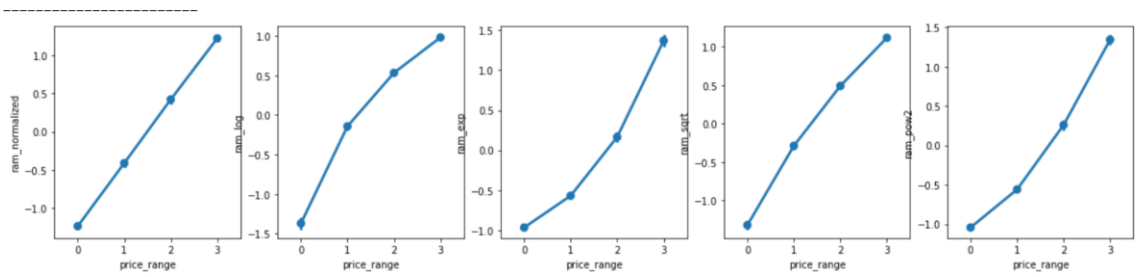
با توجه به ۵ نمودار زیر میتوان گفت که رنج مقداری فیچرها بعد از هیچ کدام از تبدیل ها فرق چشم گیری با قبل تبدیل نداشته است و قدرت تمیز دادن داده های کلاس های مختلف تغییری نکرده است. هم چنین دقت مدل بهبود چشمگیری نداشته است (۰.۸۹۷۵ در بهترین حالت تبدیل ها به نسبت ۰.۸۸۷۵ بدون هیچ گونه تبدیلی). در تبدیل جذری(نمودار راست) جداسازی داده های کلاس ۱ و ۲ نسبت به حالت بدون تبدیل کمی بهبود داشته است و دقت مدل هم کلی بالاتر رفته و به ۰.۸۹۷۵ رسیده است. همچنین میتوان گفت که تبدیل نمایی اکسپوننشیال اثر منفی داشته است چرا که دقت عملکرد مدل به ۰.۸۶ کاهش یافته است و جداسازی کلاس ها با این تبدیل فیچر بدتر شده است چرا که در تبدیل های دیگر و بدون تبدیل هر چه مقدار این فیچر بالاتر بود شانس بیشتری داشت که به کلاسهای با شماره بالاتر تعلق داشته باشد ولی با تبدیل اکسپوننشیال رنج مقدار این فیچر در کلاسهای مختلف اشتراک زیادی داشته است!(نمودار سوم وسطی با تبدیل اکسپوننشیال به نسبت بدون تبدیل در نمودار سمت چپ)




```
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.86
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.8875
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.8975
=======================
```

RAM Feature:

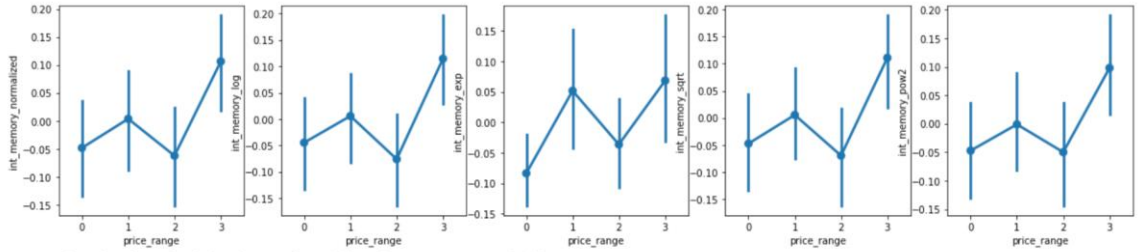
با توجه به ۵ نمودار زیر میتوان گفت که رنج مقداری فیچرها بعد از هیچ کدام از تبدیل ها فرق چشم گیری با قبل تبدیل نداشته است و قدرت تمیز دادن داده های کلاس های مختلف تغییری نکرده است. حتی قدرت مدل کمتر هم شده است چرا که با تبدیل اکسپوننشیال قدرت به ۰.۸۳۷ هم کاهش پیدا کرده است. تبدیل توان دوم بین ۴ تبدیل بهتر بوده است و دقت ۰.۸۶۲۵ است ولی باز به نسبت حالتی که تبدیلی نباشد دقت کمتر است!




```
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.855
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.8375
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.88
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.8625
```

int_memory Feature:

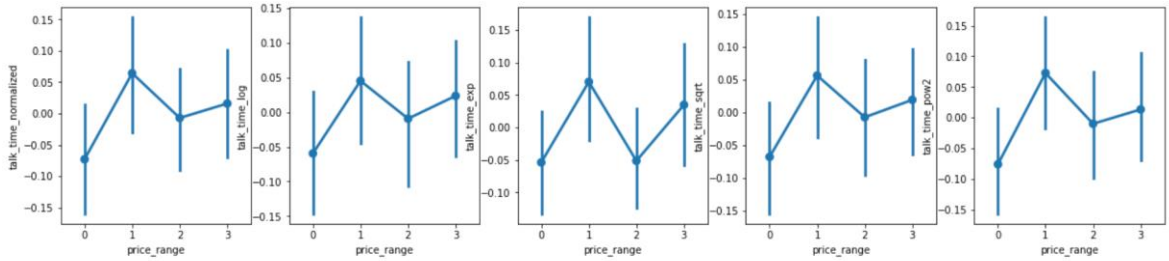
با توجه به ۵ نمودار زیر میتوان گفت که به طور کلی رنج مقداری فیچرها بعد از هیچ کدام از تبدیل ها فرق چشم گیری با قبل تبدیل نداشته است و قدرت تمیز دادن داده های کلاس های مختلف تغییری نکرده است. فقط دقت و عملکرد مدل بعد از تبدیل توان دوم کمی بهبود یافته است (۰.۹ به نسبت ۰.۸۸۷۵ بدون تبدیل). هم چنین به جز این ۴ تبدیل برای این فیچر ما هم مموری را بر ۱۰۲۴ نیز تقسیم کردیم تا به صورت گیگابایت شود ولی عملکرد مدل تغییری نداشت و همان ۰.۸۸۷۵ بود.




```
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.89
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.885
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.885
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.9
```

Talk_Time Feature:

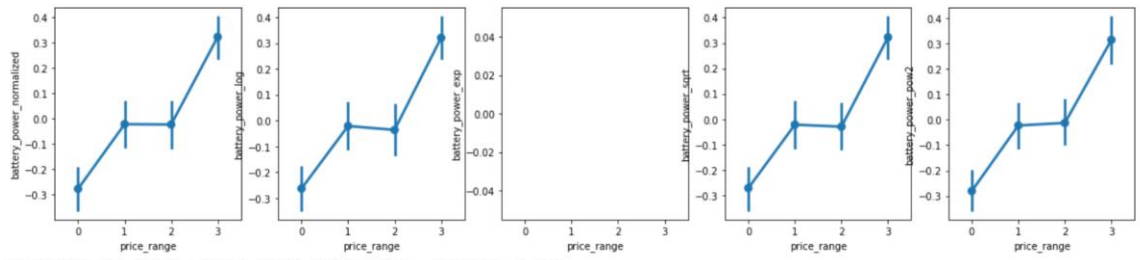
با توجه به ۵ نمودار زیر میتوان گفت که به طور کلی رنج مقداری فیچرها بعد از هیچ کدام از تبدیل ها فرق چشم گیری با قبل تبدیل نداشته است و قدرت تمیز دادن داده های کلاس های مختلف تغییری نکرده است.در بهترین حالت عملکرد مدل ۰.۸۹ با تبدیل لگاریتمی است که بهبود چندانی نسبت به حالت بدون تبدیل که دقت ۰.۸۸۷۵ است نداردو. برای تبدیل اکسپوننشیال و توانی دقت کمی هم کاهش پیدا کرده است و به ۰.۸۸۵ رسیده است.




```
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.89
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.885
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.8875
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.885
------------------------
```

Battery_power Feature:

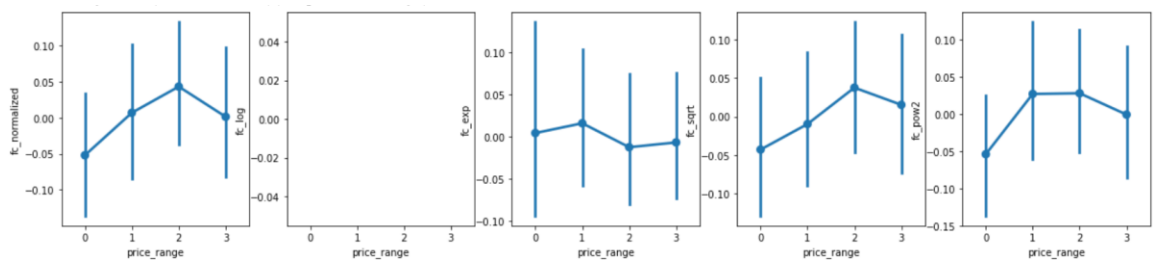
به طور کلی رنج مقداری فیچرها با این تبدیل تغییری نکرده است. دقت مدل پس از تبدیل جذری کمی بهبود یافته است (دقت ۰.۹ به نسبت ۰.۸۸۷۵)




```
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.8975
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.9
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.8825
========================
```

fc Feature:

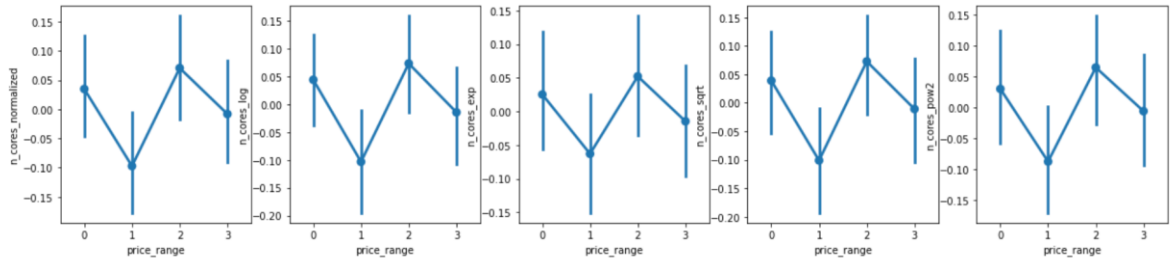
در این حالت تنها تبدیل جذری توانسته است قدرت مدل را کمی بهبود دهد و به ۰.۹ برساند. اگر به نمودار ها هم نگاه کنیم این تبدیل باعث شده است که کلاس ۰ و ۱ قابلیت جداسازی کمی بهتری داشته باشند.




```
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.8975
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.9
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.8925
======================
```

n_cores Feature:

همانطور که مشخص است تبدیل اکسپوننشیال دقت مدل را نسبت به باقی تبدیل ها بیشتر بهبود داده است (۰.۹با تبدیل اکسپوننشیان به نسبت ۰.۸۸۷۵ بدون تبدیل) . دلیل اثر مثبت این تبدیل این است که این فیچر تعداد کورهای موبایل را نشان میدهد و هرچه بیشتر باشد قاعدتا قیمت موبایل باید بالاتر باشد. تبدیل اکسپوننشیال باعث میشود که کورهای بالاتر مقدار بالاتری برای این فیچر داشته باشند و بهتر از داده های با کور پایین تر قابل جداسازی باشند. بنابراین این تبدیل باعث میشود که کلاس قیمت بالاتر جداسازی بهتری داشته باشد.




```
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.8825
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.9
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.885
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.89
```

د) New Feature: Area w.r.t to Pixel and phone size

برای این سوال دو نوع فیچر را تست کردیم. یکی فیجر مساحت با توجه به پیکسلهای گوشی و دیگری مساحت با توجه به اندازه خود گوشی. در حالت اول دقت بدتر و در حالت دوم دقت بهبود یافت:

- یک فیچر جدید به نام مساحت صفحه گوشی با توجه به پیکسل ساختیم که معادل ضرب طول و عرض پیکسل هاست. دقت در زیر گزارش شده است. دقت کنین به منظور ارزیابی این فیچر جدید, دو فیچر مرتبط قبلی px_heigh and px_width را حذف کردیم چون اگر حذف نمیکردیم نگه داشتن فیچر تکراری باعث بدتر شدن عملکرد مدل میشود!

```
X['area_pixel'] = X['px_height'] * X['px_width']
```

```
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.7925
Precision:  0.7964 [0.89473684 0.69298246 0.72043011 0.87755102]
Recall:  0.7914 [0.81730769 0.7745098  0.70526316 0.86868687]
Fscore:  0.7929 [0.85427136 0.73148148 0.71276596 0.87309645]
========================
```

همانطور که میبینیم دقت مدل از ۰.۸۸۷۵ به ۰.۷۹۲۵ کاهش یافته است. این نشان میدهد ترکیب کردن این دو فیچر تاثیر مثبتی نداشته است و همانطور که قبلا جدا بودند مدل عملکرد بهتری داشت.

- یک فیچر جدید به نام مساحت صفحه گوشی با توجه به اندازه گوشی ساختیم که معادل ضرب طول و عرض موبایل هاست. دقت در زیر گزارش شده است. دقت کنین به منظور ارزیابی این فیچر جدید, دو فیچر مرتبط قبلی را حذف کردیم چون اگر حذف نمیکردیم نگه داشتن فیچر تکراری باعث بدتر شدن عملکرد مدل میشود!

```
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.8975
Precision:  0.8999 [0.95918367 0.81981982 0.86170213 0.95876289]
Recall:  0.897 [0.90384615 0.89215686 0.85263158 0.93939394]
Fscore:  0.8978 [0.93069307 0.85446009 0.85714286 0.94897959]
========================
```

همانطور که میبینیم دقت مدل از ۰.۸۸۷۵ به ۰.۸۹۷۵ افزایش یافته است. این نشان میدهد ترکیب کردن این دو فیچر بر خلاف فیچرهای مرتبط به پیکسل تأثیر مثبتی داشته است.

سوال ۱۰) برای هریک از حالت های سوال قبلی یک مدل SVM بسازید و بررسی کنید یکبار هم هر ۵ حالت را باهم اعمال کنید و مدل SVM روی آنها اجرا کنید . حاصل این مدل ها را گزارش کنید.

One hot encoding + 5 equal binning + Transformation

در این حالت ترکیب حالتهای قبلی را بر روی فیچرها اعمال کردیم. به این ترتیب که فیچرهای کتگوریکال را one hot encoding کردیم. برای فیچر battery power عمل binning را انجام دادیم و ۵ گروه مساوی در نظر گرفتیم. فیچر مساحت خود گوشی را اضافه کردیم (مساحت بر اساس پیکسل را اضافه نکردیم چرا که باعث کاهش دقت شده بود). از بین تابع های تبدیل روی فیچرهای مختلف بهترین ها را که باعث افزایش عملکرد شده بودند را انتخاب کردیم. تابع تبدیل اکسپوننشیال روی clock speed و تابع تبدیل اکسپوننشیال روی n_cores و تابع تبدیل جذر روی فیچر fc

```
X_w_dummies_all = X_w_dummies.copy(deep = True)
X_w_dummies_all['batter_power_5_bins'], bins_5 = pd.cut(X_w_dummies_all['battery_power'], 5, retbins = True, labels=range(5))#labels = range(10)
X_w_dummies_all['area'] = X_w_dummies_all['sc_h'] * X_w_dummies_all['sc_w']

#exp transformation on clock_speed
f_val = train_df_org['clock_speed']
z1 = np.exp2(f_val)
X_w_dummies_all.loc[:,'clock_speed'+'_exp'] = ((z1 - z1.mean())/z1.std())

#exp transformation on n_cores
f_val = train_df_org['n_cores']
z1 = np.exp2(f_val)
X_w_dummies_all.loc[:,'n_cores'+'_exp'] = ((z1 - z1.mean())/z1.std())

#sqrt transformation on fc
f_val = train_df_org['fc']
z1 = np.sqrt(f_val)
X_w_dummies_all.loc[:,'fc'+'_sqrt'] = ((z1 - z1.mean())/z1.std())
```

عملکرد مدل بعد از تمام این تبدیل ها کاهش پیدا کرد!!! مدل با کرنل RBF درنظر گرفتیم. دقت از حالت معمولی ۰.۸۸۷۵ به ۰.۷۹۲۵ رسید و همچنین precision recall f1 نیز کاهش پیدا کردند. دلیل این اتفاق این است که این فیچرها همگی با هم باعث overfitting و افزایش واریانس مدل شده اند. مدل بر روی داده اموزشی خوب عمل میکند ولی بر روی تست بد می‌شود.

```
SVM results for kernel rbf and Kernel coeff scale-- accuracy:  0.7925
Precision:  0.7951 [0.89       0.6952381  0.69       0.90526316]
Recall:  0.7916 [0.85576923 0.71568627 0.72631579 0.86868687]
Fscore:  0.793 [0.87254902 0.70531401 0.70769231 0.88659794]
========================
```

سوال یازدهم

به طور کلی الگوریتم های ساخت درخت تصمیم در موارد زیر با یکدیگر متفاوت هستند.

- معیار تفکیک :(splitting criterion) در فرآیند ساخت درخت تصمیم برای تفکیک داده ها در هر راس (node) به یک معیار نیاز داریم. این معیار در درخت های کلاسه بندی (داده های گسسته) و رگرسیون (داده های پیوسته) متفاوت است. در مساله کلاسه بندی هدف تقسیم داده ها به گروه های کوچکتر و همگون تر (homogeneous) و خالص تر (pure) می باشد به گونه ای که تا حتی الامکان داده ها در هر راس از یک کلاس باشند. در نتیجه می توان از معیارهایی نظیر Information Gain و Information Gain استفاده کرد. در مقابل برای مساله رگرسیون می بایست از معیارهایی نظیر مجموع مربعات خطا SSE و یا کاهش در واریانس استفاده کرد به گونه ای که با افزایش اندازه درخت) به صورت بازگشتی یا (recursive این مقدار کم و کمتر شود .
- روش های کاهش overfitting
- توانایی حل مساله با داده های غیر کامل (incomplete)

به عنوان نمونه در روش ID3 درخت تصمیم کلاسه بندی به صورت حریصانه و بالا به پایین ساخته می شود به گونه ای که در هر مرحله فیچری انتخاب می شود که به ترتیب به کمترین و بیشترین مقدار Entropy Information Gain منجر شود. نسل بعدی این روش الگوریتم C4.5 است که می تواند با فیچرهای گسسته و پیوسته کار کند. همچنین روش CART که معروف به درخت کلاسه بندی و رگرسیون است که مانند روش C4.5 عمل می کند اما بر خلاف آن به جای استفاده از مجموعه قوانین (rule sets) از تفکیک عددی بازگشتی (recursive numerical splitting) استفاده می کند.

سوال دوازدهم و سیزدهم

ابتدا یک درخت با پارامترهای دیفالت پکیج میسازیم. در پارامترهای پیش فرض معیار gini است. هم چنین به شکل پیش فرض مینیم داده های مورد نیاز در هر گره ۱ و مینیمم داده های مورد نیاز برای هر node برابر با ۲ است. هم چنین هیچ محدودیتی روی عمق درختها وجود ندارد.نتایج درخت با حالت پیش فرض در زیر گزارش شده است:

```
Decision Tree Results with Default Parameters
Accuracy:  0.8075
Precision:  0.8088 [0.89320388 0.69444444 0.72043011 0.92708333]
Recall:  0.806 [0.88461538 0.73529412 0.70526316 0.8989899 ]
Fscore:  0.8072 [0.88888889 0.71428571 0.71276596 0.91282051]
```

همچنین با استفاده از روش grid search پارامترهای مختلف و مقادیر مختلف پارامترها را ازمایش کردیم که در زیر گزارش شده اند:

- Criterion: Gini, Entropy
- Max depth: maximum depth of the tree: 3, 5, 7, 10
- Min samples split: minimum sample required to split a node: 1,2, 3, 4, 5, 6, 7, 8, 9
- Min samples leaf: minimum sample required for a leaf: 1,2, 3, 4,5, 6, 7,8,9

در grid search از روش cross validation با ۵ فولد استفاده میشود و میانگین عملکرد در فولدها در نظر گرفته میشود .

بهترین درخت برای پارامترهای زیر است که از روش entropy استفاده شود و ماکسیمم عمق درخت ۱۰ باشد و مینیمم نمونه مورد نیاز برای هر گره ۳ و مینیمم نمونه برای برگها ۵ باشد. عملکرد نسبت به درخت ساخته شده با پارامترهای پیش فرض بسیار بهتر است و دقت از ۰.۸۰۷۵ به ۰.۸۸ بهبود یافته است.

همانطور که از نتایج مشخص است افزایش عمق درخت ها به بهبود عملکرد تاثیر مبثتی دارد. هرچه تعداد نمونه های برگ و گره ها بیشتر باشد عملکرد بهتر میشود ولی از یک حدی اگر بیشتر شود عملکرد کاهش پیدا میکند بنابراین بهترین مقدار برای نمونه های برگ و گره حداقل ۵ و ۳ است.

```
Fitting 5 folds for each of 512 candidates, totalling 2560 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done 956 tasks      | elapsed:    5.0s
Grid Search best parameters
{'criterion': 'entropy', 'max_depth': 10, 'min_samples_leaf': 5, 'min_samples_split': 3}
Decision Tree Results with Best Estimator
Accuracy:  0.88
Precision:  0.8788 [0.92307692 0.83333333 0.84615385 0.91262136]
Recall:  0.8791 [0.92307692 0.83333333 0.81052632 0.94949495]
Fscore:  0.8788 [0.92307692 0.83333333 0.82795699 0.93069307]
[Parallel(n_jobs=-1)]: Done 2560 out of 2560 | elapsed:   17.2s finished
```

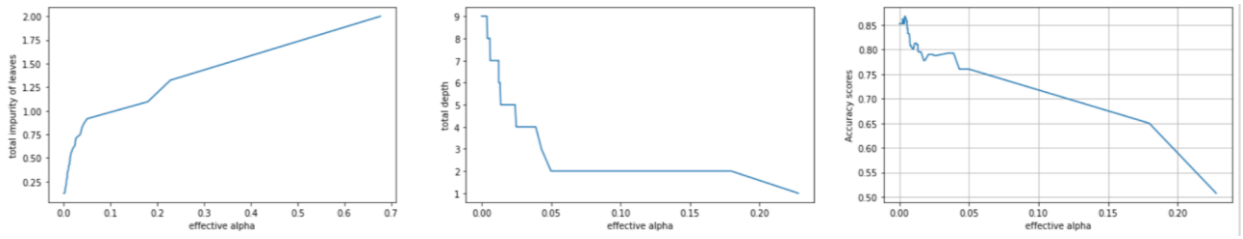
سوال چهاردهم

در بسیاری از موارد درخت های تصمیم overfit می کنند به این ترتیب که عملا داده های مجموعه آموزش (train) را به خاطر سپرده اند .(memorize) دلیل این مشکل اینست که درخت های تصمیم متمایل به افزایش اندازه دارند و در بدترین حالت ممکن است برای هر داده یک برگ (leaf) اختصاص دهند. در چنین حالتی خطای آموزش صفر خواهد شد و درخت دچار ovefitمی شود. یک روش برای حل این مشکل استفاده از مفهوم حرس یا pruning می باشد. در این روش برخی از شاخه های درخت و همچنین راس های تصمیم (decision nodes) حذف می شوند (با شروع از برگ ها) تا از افزایش بی رویه درخت جلوگیری شود. برای این کار داده های آموزشی به دو مجموعه آموزشی و اعتباری (validation) تقسیم شده و درخت با استفاده از داده های آموزشی ساخته و با استفاده از مجموعه اعتباری حرس می شود.

سوال چهارم امتیازی

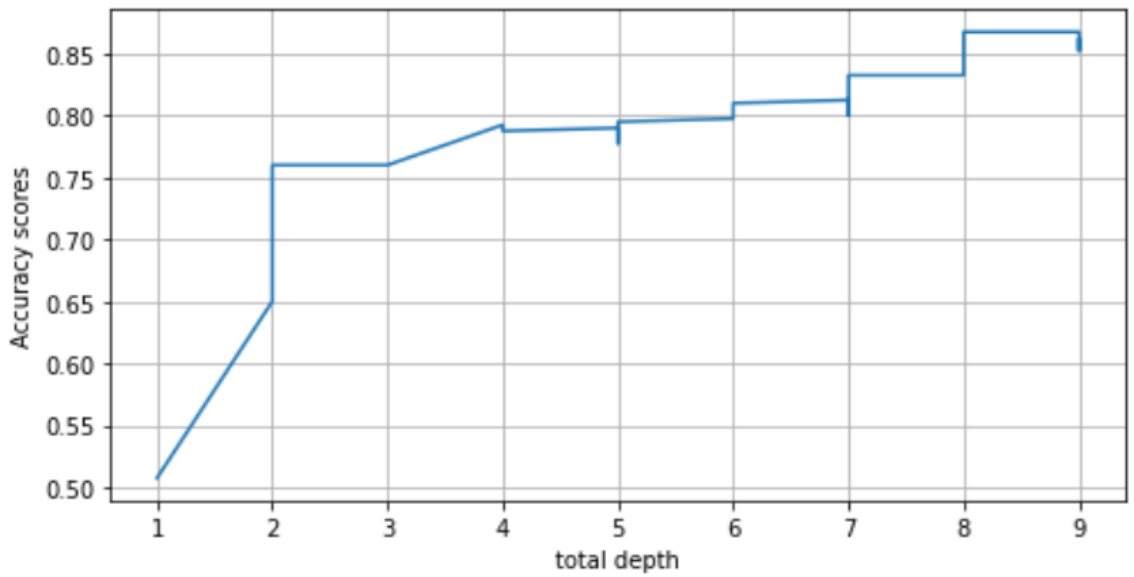
در سوال ۱۲و ۱۳ با بررسی پارامترهایی مثل عمق درخت و مینیم نمونه های مورد نیاز برای گره و نود مساله pre prunning درخت را بررسی کردیم و ارزیابی کردیم که برای این داده چه پارامترهایی بهترین هستند. بنابراین برای این سوال ابتدا درخت با بهترین پارامترها را که توسط

سوال ۱۲ و ۱۳ پیدا کردیم میسازیم. برای post prunning از تابع cost_complexity_pruning_pathاستفاده کردیم .با توجه به Post prunning cost complexity pruningاست و هدف این است که مقدار بهینه پارامتر آلفا را پیدا کنیم. بنابراین اطلاعات مربوط به تمام درخت های ممکن با توجه به پارامترهای مختلف آلفا را ذخیره میکنیم و رابطه بین آلفا و موارد زیر را میسنجیم: عمق درخت , دقت مدل و impurity برگ ها. نمودارهای زیر رابطه مقادیر مختلف آلفا با هر کدام از این موارد را نشان می‌دهد:



همانطور که مشاهده میکنیم با افزایش آلفا مقدار عمق درخت کاهش پیدا میکند. هم چنین با افزایش آلفا مقدار Impurity برگ ها بیشتر میشود. قابل ذکر است که هرچقدر مقدار impurityبیشتر شود عملکرد مدل میتواند بدتر شود چرا که ما به دنبال برگهایی هستیم که تا جای ممکن pure باشند. هم چنین قابل ذکر است که اگر اندازه نمونه های برای هر برگ خیلی کوچک باشد باعث ovrfitting مدل میشود. نمودار سمت راست که رابطه بین دقت و آلفا است نشان میدهد که با افزایش آلفا در ابتدا دقت افزایش میابد ولی از یک نقطه ای به بعد(وقتی آلف بیشتر از ۰.۰۰۳ میشود) دقت شروع به کاهش میکند. این نشان میدهد که افزایش آلفا تا حدی نه تنها باعث بهبود عملکرد میشود، بلکه باعث کمتر شدن عمق درخت میشود و همچنین از ovrfit شدن مدل جلوگیری می‌کند.

همچنین رابطه بین دقت مدل و عمق درخت را در نمودار زیر نمایش دادیم که نشان میدهد با افزایش عمق عملکرد بهتر میشود ولی هرچه عمق بالاتر میرود , سرعت افزایش دقت مدل کمتر میشود. علی الخصوص افزایش عمق ۸ به ۹ حتی کمی باعث کم شدن دقت می‌شود.



نهایتا با توجه به این post prunning متوجه میشویم که بهترین مقدار آلفا ۰.۰۰۳۷ میباشد با عمق درخت ۹ و هم چنین دقت مدل ۰.۸۶۷۵ میباشد. عملکرد مدل با توجه به معیارهای precision recall F1 نیز در زیر نمایش داده شده است:

```
Precision:  0.8647 [0.90909091 0.8        0.81818182 0.93137255]
Recall:  0.8641 [0.88235294 0.80769231 0.83505155 0.93137255]
Fscore:  0.8643 [0.89552239 0.80382775 0.82653061 0.93137255]
```

سوال پانزدهم

هر دو روش های resampling هستند ولی cross validation سمپل بدون جایگزین است اما bootstrap سمپل با جایگزین است.

Cross Validation: برای ایجاد چندین مجموعه ، داده موجود را تقسیم می کند و به منظور validation و اعتبار سنجی استفاده میشود. اندازه مجموعه داده تولید شده کوچکتر از سایز اصلی دیتاست اموزشی میباشد (چون نمونه گیری بدون جایگزین است)

Bootstrapping: پس از نمونه برداری مجدد با جایگزینی از مجموعه داده اصلی برای ایجاد مجموعه داده‌های متعدد استفاده می کند. در نتیجه اندازه مجموعه داده تولید شده برابر با سایز اصلی دیتاست است چرا که نمونه گیری با جایگزین است. بوت استرپینگ به اندازه اعتبارسنجی قوی نیست و بیشتر در مورد ساخت مدل های ensemble یا فقط تخمین پارامترها است.

سوال شانزدهم

این روش در واقع ۵ بار ۲-fold cross validation را تکرار میکند. در هر تکرار دو فولد در نظر میگیرد و داده را به طور رندوم به دو فولد تقسیم میکند. یک فولد برای تست و یک فولد برای train و عملکرد مدل را میسنجد و اینکار را ۵بار تکرار میکند. این روش زمانی استفاده میشود که بخواهیم عملکرد مدلهای مختلف ماشین لرنینگ را با هم مقایسه کنیم. استفاده از روش ۱۰-فولد باعث میشود که به این نتیجه برسیم که مدلها significantly different هستند در حالی که نیستند .بنابراین برای مقایسه عملکرد مدلهای مختلف از این روش ۵x2 cross validation استفاده میکنیم. با فرض داشتن دو مدل A , B اعداد زیر را خواهیم داشت:

Partition data into sets S_1 and S_2. Error estimates are:
$p_A^1, p_A^2, p_B^1, p_B^2$. $p^1 = p_A^1 - p_B^1$ and $p^2 = p_A^2 - p_B^2$,
$\overline{p} = \frac{p^1 + p^2}{2}$, variance from i^{th} replication is
$s_i^2 = (p_i^1 - \overline{p_i})^2 + (p_i^2 - \overline{p_i})^2$ use statistic
$\tilde{t} = \frac{p_1^1}{\sqrt{\frac{1}{5}\sum_{i=1}^{5} s_i^2}}$

مقدار t-value با توجه به عملکرد دو مدل بر روی فولدهای مختلف محاسبه میشود. سپس با داشتن مقدار t-value میتوانیم نتیجه بگیریم که آیا مدلها متفادت هستند یا نه. در واقع فرض نال این میشود که مدلها متفاوت نیستن (not statistically significantly different). اگر مقدار t-value با degree of freedom = 5 در بازه confidence interval قرار بگیرد میتوان گفت که مدلها متفاوت نیستند. مقدار بازه confidence interval با مقدار thresholdهای مختلف

را می‌توان از این صفحه دید(https://www.medcalc.org/manual/t-distribution-table.php) . نکته این که اگر از روش ۱۰- فولد استفاده کنیم مدلها با احتمال خیلی زیاد statistically different خواهند شد که لزوما درست نیست ولی این روش ۲x۵ کمک میکند که واریانس مربوط به model instability را اندازه بگیریم.

سوال ۱۷ - استفاده از روش مشابه elbow برای تعیین مرتبه مدل

در این روش ایده اصلی این است که مقادیر مختلف مرتبه مدل را امتحان کنیم و مدلی با آن مرتبه بسازیم و ببینیم هربار بایاس و واریانس مدل چه تغییری میکند. نهایتا مرتبه ای که مقدار بایاس و واریانس در بهیته ترین حالت خود قرار میکند را انتخاب میکنیم. بهینه ترین حالت در واقع میتوان گفت زمانی است که دیگر از آنجا به بعد یکی از بایاس و واریانس و یا هردو دیگر تغییر چشم گیری نمیکنند و یا بدتر میشودند! برای مثال در مورد نموداری که در صورت سوال است در مرتبه ۳ بایاس و واریانس هر دو کمترین مقدار خود هستند و در مراتب بعدی اگرچه که بایاس کمتر میشود ولی واریاسن شروع به زیاد شدن میکند که نشان میدهد در مرتبه بالاتر از ۳ مدل دچار over fitting میشود. از این روش برای پیدا کردن مرتبه قابل قبول میتوان استفاده کرد و در جواب میدهد اما در حالتی که دیتا از مدل polynomial نباشد این روش جواب نمیدهد.

در واقع به طور مثال اگر رابطه فیچرها و لیبل هر رابطه ای به جز رابطه polynomial باشد این روش جواب نمیدهد. مثلا اگر رابطه سینوسی کسینوسی و یا مثلا لگاریتمی و یا ترکیب تابع های این چنینی باشد این روش اصلا جواب نمیدهد. اگرچه که با مرتبه های مختلف و با فرض polynomialبودن رابطه ممکن است به مرتبه ای برسیم که مدل بایاس و واریانس بهینه و کم داشته باشد اما به هر حال مدل ساخته شده true functional form را نشان نمیدهد و captureنمیکند. اگر مثلا true functional form به شکل تابع سینسوسی باشد , صرف نظر از اینکه بایاس چقدر کم میشود و مدل polynomial ساخته شده (با بهترین مرتبه) به دیتا فیت میشود, این مدل true functional form و رابطه سیسنوسی دیتا راcapture نکرده است.

بنابراین این روش اگرچه در تیوری و تحلیل ممکن است جواب بدهد اما در دنیای واقعی که رابطه ها و شکل دیتا ممکن است چیزی به جز polynomial باشد این روش جواب نمیدهد . چالش این است که خیلی مواقع ما true functional form را نمی‌دانیم و نمی‌دانیم که مثلا دیتا رابطه سیسنوسی دارد و مجبوریم با تابع های polynomial مدل را فیت کنیم.

سوال ۲ امتیازی

statistical significance tests

فرض یک آزمون آماری، فرض صفر نامیده می‌شود و ما می توانیم اندازه گیری های آماری را محاسبه کرده و آنها را تفسیر کنیم تا در مورد پذیرش یا رد فرض صفر تصمیم بگیریم. در مورد انتخاب مدل ها بر اساس مهارت تخمینی آنها ، ما علاقه مندیم بدانیم که آیا تفاوت واقعی یا آماری معنی داری بین این دو مدل وجود دارد یا خیر.

اگر نتیجه آزمون نشان دهنده این باشد که شواهد کافی برای رد فرض صفر وجود ندارد، پس هرگونه تفاوت مشاهده شده در مهارت مدل، احتمالاً به دلیل شانس آماری است. اگر نتیجه آزمون نشان دهد که شواهد کافی برای رد فرض صفر وجود دارد ، بنابراین هرگونه تفاوت مشاهده شده در مهارت مدل به دلیل تفاوت در مدل ها است. نتایج آزمون احتمالاتی است یعنی می توان نتیجه را به درستی تفسیر کرد و نتیجه با خطای نوع ۱ یا ۲ اشتباه است. به طور خلاصه ، یک یافته مثبت کاذب یا منفی کاذب است .

مقایسه مدل‌های یادگیری ماشین از طریق آزمون‌های آماری معنادار ، انتظاراتی را تحمیل می‌کند که به نوع خود بر انواع آزمون‌های آماری قابل استفاده، تاثیر می گذارد. به عنوان مثال برآورد مهارت، معیار مشخصی برای مهارت مدل باید انتخاب شود. این می‌تواند دقت طبقه بندی یا خطای مطلق باشد که نوع تست‌های قابل استفاده را محدود می‌کند.

آموزش و آزمایش مکرر یک مدل معین بر روی داده های مشابه یا متفاوت ، بر نوع آزمایشی که می توان استفاده کرد تاثیر می گذارد. توزیع تخمین‌های نمونه برآورد نمره مهارت، شاید گوسی باشد یا نه. با این کار مشخص خواهد شد که آیا می توان از آزمون‌های پارامتریک یا غیر پارامتری استفاده کرد. گرایش مرکزی مهارت مدل، اغلب بسته به توزیع نمرات مهارت با استفاده از یک آمار خلاصه مانند میانگین یا متوسط توصیف و مقایسه می‌شود. آزمون ممکن است این را مستقیماً در نظر بگیرد.

★ سوال ۳ امتیازی

معیار Matthews Correlation Coefficient

معیاری است که برای ارزیابی کارایی الگوریتم های یادگیری ماشین از آن استفاده می شود. این پارامتر بیانگر کیفیت کلاس بندی برای یک مجموعه باینری می باشد. این معیار سنجه‌ای است که بیانگر وابستگی ما بین مقادیر مشاهده شده از کلاس باینری و مقادیر پیش بینی شده از آن می باشد. مقادیر مورد انتظار برای این کمیت در بازه ۱- و ۱ متغیر می باشند. مقدار ۱+ نشان دهنده پیش بینی دقیق و بدون خطای الگوریتم یادگیر از کلاس باینری می باشد. مقدار ۰ نشان دهنده پیش بینی تصادفی الگوریتم یادگیر از کلاس باینری می باشد. مقدار ۱- نشان دهنده عدم تطابق کامل ما بین موارد پیش بینی شده از کلاس باینری و موارد مشاهده شده از آن می باشد. مقدار این پارامتر را به طور صریح، با توجه به مقادیر ماتریس آشفتگی به شرح زیر، می توان محاسبه نمود:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$