محمد عبدالهی 400422133

مقدمه:

در این تمرین به بررسی دیتاست mobile price classification پرداختیم. این دیتاست شامل مشخصات گوشیهای همراه و رنج قیمتی آنهاست به صورتی که براساس ویژگی هایی همچون ram سایز صفحه , قابلیت 4g و . . . میخواهیم رنج قیمتی گوشی ها را پیش بینی کنیم. رنج قیمتی گوشی ها دارای چهار دسته 0, 1, 2, 3 می باشند که ما برای سادگی مسئله دو دسته با قیمت پایینتر(0,1) را به عنوان دسته 0 و دو دسته با قیمت بالاتر (2,3) را به عنوان دسته 1 در نظر گرفتیم.

سوال ها:

1. روش انتخاب پیشرو را بر روی دیتاست خود اعمال کرده و در ابتدا مجموعه فیچرهای ما تهی بوده و در هر مرحله فیچری که بیشترین حد ممکن به بهبود auc کمک می کرد را به مجموعه فیچرهای منتخب اضافه میکردیم. و در نهایت مجموعه بهترین فیچرها شامل:

```
['ram',
 'battery_power',
 'four_g',
 'm_dep',
 'mobile_wt',
 'pc',
 'px_width',
 'wifi']
```

شد.

2. به کمک مجموعه ویژگی های بدست آمده یک مدل لجستیک بر روی داده ها ترین کردیم و نتایج زیر حاصل شد.

```
f1_score: 0.9319899244332494
precision: 0.9390862944162437
recall: 0.925
```

که بهبود قابل توجهی نسبت به حالتی که تمام فیچرها را استفاده می کردیم داشت.

```
f1_score: 0.8759493670886076
precision: 0.8871794871794871
recall: 0.865
```

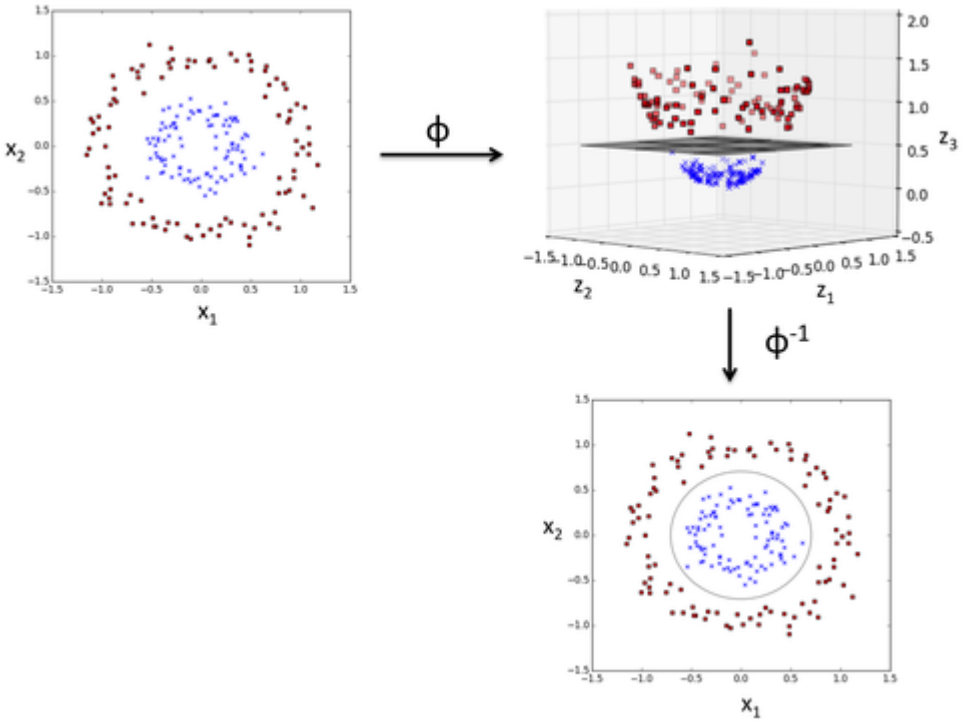
3. به کمک کتابخانه sklearn متد pca را بر روی داده های خود اعمال کرده و از انجا که به کمک روش انتخاب پیشرو ۸ فیچر انتخاب شد لذا در این مرحله هم عدد ۸ را به عنوان ورودی به الگوریتم pca دادیم.
4. پس از انجام دسته بندی بر روی داده های حاصل از الگوریتم pca به نتایج زیر رسیدیم که بهبود قابل توجهی را نشان میدهند.

```
f1_score: 0.9670886075949366
precision: 0.9794871794871794
recall: 0.955
```

5. کرنل های پرکاربرد svm عبارتند از: linear, polynomial, rbf

توابع کرنل به نوعی بر روی شکل و فضای داده ها عمل کرده و به طور معمول با انجام ضرب داخلی فضای داده ها را دچار تغییر میکنند. به عنوان مثال در کرنل چند جمله ای به نوعی فضای داده ها را به ابعاد بالاتر می بریم و فیچرهای جدیدی را در نظر میگیریم مثلا حاصل ضرب دو فیچر و این کار را بدون اضافه کردن فیچر جدید به دیتاست انجام میدهیم. به طور کلی در مسائل خطی میتوان از کرنل خطی و در مسایل غیر خطی میتوان از کرنل هایی همچون polynomial و rbf استفاده نمود.

به عنوان مثال در کرنل RBF در واقع ترکیبات غیر خطی از ویژگی های شما ایجاد می کند تا نمونه های شما را در فضای ویژگی با ابعاد بالاتر تبدیل کند سپس می توانید از یک مرز تصمیم گیری برای جداسازی کلاس های خود استفاده کنید.



6. روش svm بر روی دیتاست انجام شده و نتایج حاصل به فرم زیر میباشد:

```
f1_score: 0.9796954314720813
precision: 0.9948453608247423
recall: 0.965
```

7. بر روی دیتاست خود الگوریتم svm را به کمک کرنل های مختلف و پارامتر c (که برای regularize کردن به کار میرود) تست کردیم و نتایج به این صورت بود که با استفاده از کرنل خطی و پارامتر c برابر یک بهترین نتیجه حاصل میشد چرا که دیتای ما به طور خطی قابل جداسازی بوده و همچنین پارامتر c هر چه کمتر باشد در ازای دسته بندی اشتباه مدل خطای کمتری پیدا کرده و به نوعی soft margin خواهد بود که همین امر به generalize کردن مدل و پیدا کردن نتایج بهتر کمک میکند.

```
kernel: linear c:1
f1_score: 0.9874055415617129  precision: 0.9949238578680203  recall: 0.98
kernel: linear c:10
f1_score: 0.9849246231155778  precision: 0.98989898989899  recall: 0.98
kernel: linear c:100
f1_score: 0.9849246231155778  precision: 0.98989898989899  recall: 0.98
kernel: poly c:1
f1_score: 0.9848484848484485  precision: 0.9948979591836735  recall: 0.975
kernel: poly c:10
f1_score: 0.9874686716791979  precision: 0.9899497487437185  recall: 0.985
kernel: poly c:100
f1_score: 0.9874686716791979  precision: 0.9899497487437185  recall: 0.985
kernel: rbf c:1
f1_score: 0.9796954314720813  precision: 0.9948453608247423  recall: 0.965
kernel: rbf c:10
f1_score: 0.9874686716791979  precision: 0.9899497487437185  recall: 0.985
kernel: rbf c:100
f1_score: 0.9874686716791979  precision: 0.9899497487437185  recall: 0.985
kernel: sigmoid c:1
f1_score: 0.43373493975903615  precision: 0.4186046511627907  recall: 0.45
kernel: sigmoid c:10
f1_score: 0.4213075060532688  precision: 0.4084507042253521  recall: 0.435
kernel: sigmoid c:100
f1_score: 0.4213075060532688  precision: 0.4084507042253521  recall: 0.435
```

8. در مبحث soft margin و hard margin الگوریتم ما دارای پارامتری به نام c میباشد که هر چه این مقدار کمتر باشد مقدار مجازات مدل برای نمونه های اشتباه کمتر بوده و به نوعی حاشیه نرم تر است. و با اعمال پارامتر c برابر یک و حالت soft margin نتیجه به صورت :

```
f1_score: 0.9874055415617129
precision: 0.9949238578680203
recall: 0.98
```

و در حالتی که c برابر ۱۰۰ و حالت hard margin:

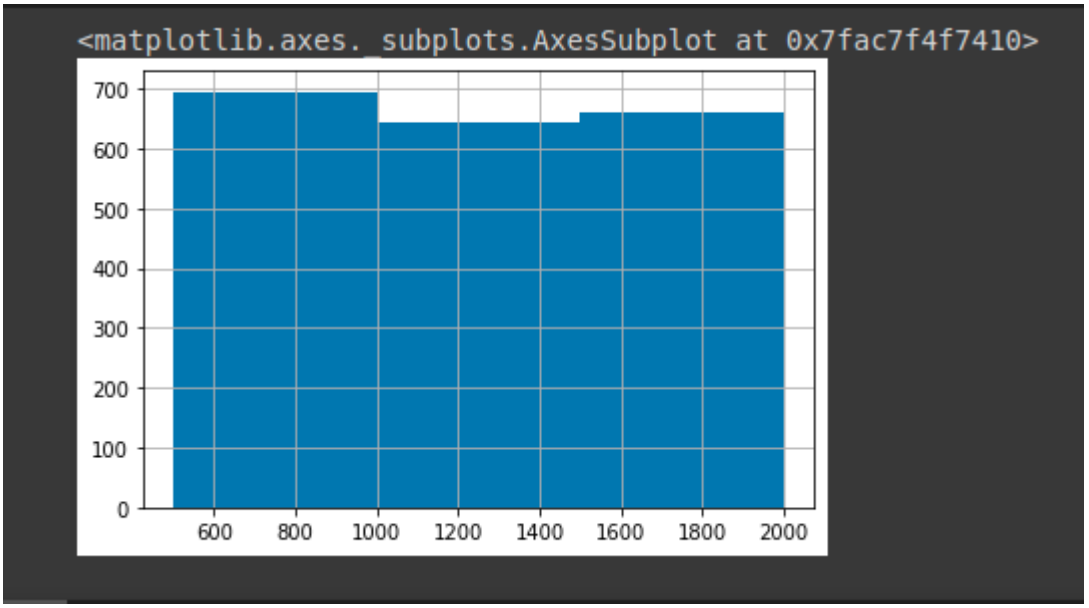
```
f1_score: 0.9849246231155778
precision: 0.98989898989899
recall: 0.98
```

خواهد بود که مشاهده می شود در حالت hard margin نتایج به مراتب ضعیف تر است.

9.

الف)با توجه به نمودار battery power سطوح ۱۰۰۰ و ۱۵۰۰ را برای دسته بندی این فیچر به سه دسته در نظر گرفتیم. کد های مربوط به این قسمت در نوت بوک موجود

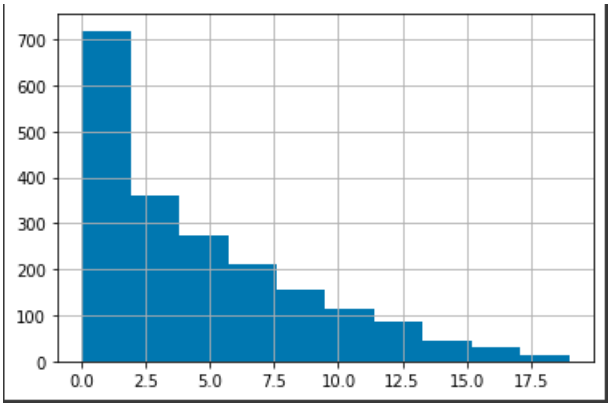
است.



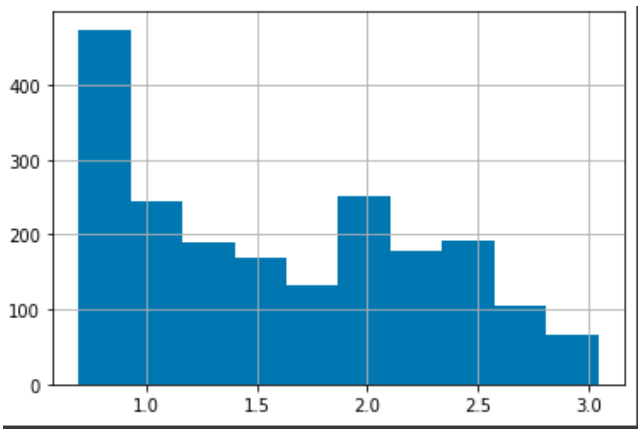
ب)کد های مربوطه ضمیمه شده.

به طور کلی تعداد زیادی از الگوریتم های یادگیری ماشین توانایی عملکرد بر روی دادگان کتگوریکال را نداشته و لازم است که این اطلاعات را به صورت عددی تبدیل کرده و وارد الگوریتم کنیم. و لذا با این تبدیل هر مقدار از متغیر کتگوریکال را به عنوان یک متغیر جدید صفر یا یکی دیده و الگوریتم را بر روی آنها اعمال میکنیم.

ج) به طور کلی از log transform زمانی که دیتای ما دچار skewness زیادی باشد استفاده میکنیم و در این مثال متغیر fc دارای چنین شرایطی بود که با تبدیل سعی در بهبود آن داشتیم. توزیع متغیر fc قبل از اعمال تبدیل:



توزیع متغیر fc بعد از اعمال تبدیل:



د) کدهای مربوط به این قسمت ضمیمه شده.

10. در حالت اول و پس از اعمال binning بر روی فیچر battery power اعمال کرده و نتایج زیر حاصل میشود:

```
f1_score: 0.9329896907216496
precision: 0.9627659574468085
recall: 0.905
```

در حالت دوم بر روی فیچرهای کتگوریکال one-hot encoding اعمال کرده و نتایج زیر حاصل میشود:

```
f1_score: 0.9796954314720813
precision: 0.9948453608247423
recall: 0.965
```

در حالت سوم بر روی فیچر fc عملیات log transformation را اعمال کرده و به نتیجه زیر میرسیم:

```
f1_score: 0.9796954314720813
precision: 0.9948453608247423
recall: 0.965
```

در حالت چهار هم فیچر area را ساخته و با اضافه کردن ان به دیتاست به دسته بندی می پردازیم که متاسفانه این فیچر باعث ضعیف تر شدن نتایج ما شده و فیچر مناسبی نمی

باشد:

```
f1_score: 0.45029239766081874
precision: 0.5422535211267606
recall: 0.385
```

در حالت پنجم هم تمام عملیاتهای بالا را انجام داده و به دسته بندی می پردازیم:

```
f1_score: 0.45029239766081874
precision: 0.5422535211267606
recall: 0.385
```

و یک بار هم بدون فیچر area دسته بندی را انجام میدهیم:

```
f1_score: 0.9329896907216496
precision: 0.9627659574468085
recall: 0.905
```

11. تفاوت عمده الگوریتم های مختلف ساخت درخت در معیار انتخاب برای انتخاب فیچری است که در هر سطح از درخت بیشترین تاثیر را ایجاد کرده و به بهترین شکل دسته بندی را انجام میدهد مثلا در الگوریتم CART از متد gini استفاده میشود در حالیکه در الگوریتم ID3 از متد information gain استفاده میشود.

12. به کمک کتابخانه sklearn اینکار انجام شد و کدها ضمیمه شده.

```
f1_score: 0.9393939393939393
precision: 0.9489795918367347
recall: 0.93
```

13. پس از اجرا کردن الگوریتم به ازای مقادیر مختلف برای عمق و تعداد نمونه های موجود در هر گره مشاهده کردیم که افزایش عمق درخت تا جایی میتواند با بهبود عملکرد شده اما افزایش بیش از حد عمق درخت باعث افزایش واریانس شده(overfit) و نتیجه معکوس خواهد داشت همچنین به برای تعداد نمونه های موجود در هر گره تعداد بیشتر باعث

افزایش بایاس شده و تعداد کمتر باعث افزایش واریانس مدل میشود.

14. به طور کلی این الگوریتم ها به دو دسته pre-pruning و post-pruning تقسیم میشوند.

Pre-pruning : در این دسته الگوریتم ها که به early stopping هم معروف هستند روشی است که در آن ساخت زیر درخت در یک گره خاص پس از ارزیابی یک measure متوقف می شود. این measure می تواند gini impurity یا information gain باشد. در pre-pruning شرایط هرس را بر اساس measure فوق در هر گره ارزیابی کرده و در صورت مساعد بودن شرایط اقدام به هرس کردن میکنیم.

Post-pruning : در این دسته از الگوریتم ها پس از ساخت درخت اقدام به هرس کردن ان می نماییم مثل الگوریتم reduced error pruning یا cost complexity pruning در الگوریتم reduced error pruning هرس با کمک یک validation set انجام می شود. در REP، همه گره ها برای هرس به صورت پایین به بالا ارزیابی می شوند. اگر درخت هرس شده بدتر از درخت اصلی در validation set نباشد، یک گره هرس می شود. درخت فرعی در گره با یک گره برگ جایگزین می شود که رایج ترین کلاس را به خود اختصاص می دهد.

هرس کردن درخت به ما کمک میکند تا بخش های اضافی و غیر مهم را کمک کرده و سایز درخت را کاهش دهیم و لذا باعث کاهش variance و overfitting شده و باعث بهبود عملکرد مدل ما میشود.

15. در تکنیک bootstrapping به resample کردن دیتا با جایگذاری می پردازیم و ممکن است داده های تکراری در دیتاست جدید موجود باشد. در حالیکه در cross validation به resample کردن دیتا بدون جایگذاری میپردازیم و دیتاست اصلی را به k قسمت تبدیل

کرده و اموزش را بر روی k-1 قسمت انجام داده و بر روی یک قسمت باقیمانده عملیات تست را انجام میدهیم.

تکنیک bootstrapping برای تخمین توزیع نمونه‌گیری تقریباً هر آماره ای را با استفاده از روش‌های نمونه‌گیری تصادفی فراهم می‌کند و باعث کاهش واریانس و افزایش بایاس میشود.

16. این متد برای مقایسه classifier ها به کار میرود به صورتی که ۵ بار یک 2fold cv اجرا شده و در نهایت میانگین گیری میشود و بهترین classifier از بین مدلهای موجود انتخاب میگردد. با توجه به اینکه در cross validation واریانس بالایی داریم با این کار تلاش میشود تا بر این نکته غلبه کرده و به نوعی مقایسه عادلانه تری بین مدلها داشته باشیم.

با توجه به مطالب بیان شده به نظر میرسد در مواقعی که دیتاست ما پراکندگی بالایی دارد و دیتا تنوع بالایی دارد بهتر است از این متد برای مقایسه مدلهای مختلف استفاده گردد.

17. با توجه به شکل و با توجه به فرمول های :

$$\mathbf{E}\left[\left(y-\hat{f}\left(x\right)\right)^2\right]=\mathbf{Bias}\left[\hat{f}\left(x\right)\right]^2+\mathbf{Var}\left[\hat{f}\left(x\right)\right]+\sigma^2$$

Where:

$$\mathbf{Bias}\left[\hat{f}\left(x\right)\right]=\mathbf{E}\left[\hat{f}\left(x\right)-f(x)\right]$$

and

$$\mathbf{Var}\left[\hat{f}\left(x\right)\right]=\mathbf{E}[\hat{f}\left(x\right)^2]-\mathbf{E}[\hat{f}\left(x\right)]^2$$

طبیعتا با کاهش بایاس انتظار داریم که خطای مدل هم کاهش پیدا کرده و به سطح مطلوبی برسد اما نکته حائز اهمیت در این مسئله بحث واریانس مدل است که برای اینکه به جواب مطلوب و سطح خطای کمی برسیم باید در حد قابل قبولی باشد(نه خیلی زیاد که باعث

overfit شود) لذا با استفاده از متد elbow و صرفا به واسطه بایاس نمیتوان بهترین مرتبه پیچیدگی مدل را بدست اورد.