گزارش تمرین سری دوم
داده کاوی
علی صالح ۹۷۲۲۲۰۵۳

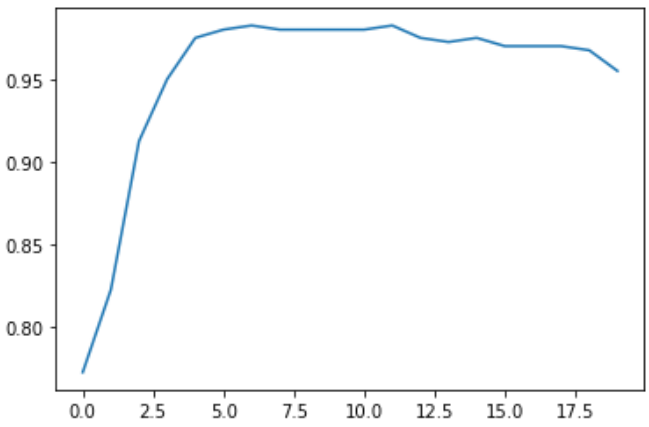
بخش ۱.۱.۱

۱-

روش forward selection را پیاده سازی می‌کنیم.
در خروجی یک لیست از فیچر ها که به ترتیب بهترین فیچر تا بدترین فیچر هستند را می‌دهیم همچنین score معادل آن‌ها و همچنین یک لیست با عنوان best_features که زیر آرایه لیست قبلی است و بهینه ترین فیچر هارا به ما برمی‌گرداند

```python
from sklearn.metrics import auc

def forward_selection(X, y):
  features = []
  final_features = {'features': [], 'scores': []}
  rem_features = X.columns
  for i in range(len(X.columns)):
    max_score = 0
    best_feature = ""
    best_score = 0
    for feature in rem_features:
      new_features = features + [feature]
      new_X = X[features + [feature]]
      X_train, X_test, y_train, y_test = train_test_split(new_X, y, test_size=0.2, random_state=0)
      logisticRegr = LogisticRegression()
      logisticRegr.fit(X_train, y_train)
      score = logisticRegr.score(X_test, y_test)
      if score > max_score :
        max_score = score
        best_feature = feature
        best_score = score
    rem_features = rem_features.drop(best_feature)
    features.append(best_feature)

    final_features['features'].append(best_feature)
    final_features['scores'].append(best_score)
  final_features['features_rank'] = range(len(X.columns))

  best_index = 0
  mx_feature = 0
  for i in range(len(final_features['scores'])):
    if final_features['scores'][i] > mx_feature:
      mx_feature = final_features['scores'][i]
      best_index = i

  final_features['best_features'] = final_features['features'][:best_index]
  return final_features
```

برای دیتاست ما اگر نمودار فیچر ها و دقت را بکشیم.
یعنی هر قدم که در forward selection جلو برویم به چه دقتی می‌رسیم. به نمودار روبرو می‌رسیم .



محور x نمودار تعداد فیچر های استفاده شده است.
همچنین best feature که تابع ما به ما برمی‌گرداند شامل شش فیچر

'ram', 'battery_power', 'px_height', 'px_width', 'mobile_wt', 'dual_sim'

است

-۲

این شش فیچر را انتخاب کرده و روی آن Logistic Regression می‌زنیم.

از Logistic Regression در کتابخانه ی sklearn استفاده می‌کنیم.

مدل ترین شده:

برای نمونه ده دیتای اول تست را به مدل ترین شده می‌دهیم.

y: 3, 0, 2, 2, 2, 0, 0, 3, 3, 1

pred: 3, 0, 2, 2, 2, 0, 0, 3, 3, 1

همانطور که مشخص است مدل ما همه ۱۰ نمونه تصادفی را درست پیشبینی کرد.

دقت این مدل: ۹۸ درصد

	Class 0	Class 1	Class 2	Class 3
percision	0.97894737	0.94736842.	0.98969072	1
recall	0.97894737	0.97826087	0.96969697	0.99122807
f1score	0.97894737	0.96256684	0.97959184	0.99559471

-۳

چون با forward selection به ۶ فیچر رسیده بودیم به pca هم مقدار ۶ را می‌دهیم

-۴

از Logistic Regression در کتابخانه ی sklearn استفاده می‌کنیم.

مدل ترین شده:

برای نمونه ده دیتای اول تست را به مدل ترین شده می‌دهیم.

y: 3, 0, 2, 2, 2, 0, 0, 3, 3, 1

pred: 3, 0, 2, 2, 2, 0, 0, 3, 3, 1

همانطور که مشخص است مدل ما همه ۱۰ نمونه تصادفی را درست پیشبینی کرد.

دقت این مدل: ۹۸ درصد

	Class 0	Class 1	Class 2	Class 3
percision	0.98958333	0.97849462	0.96938776	0.98230088
recall	1	0.98913043	0.95959596	0.97368421
f1score	0.9947644	0.98378378	0.96446701	0.97797357

-۵

به طور کلی ایده کرنل ها برای این مطرح شد زیرا در خیلی از اوقات ما نمیتوانیم با یک هایپر پلین دیتا ها کلاس ها را جدا کنیم و هر جایی که هایپر پلین را می‌گذاریم با خطای بسیار زیادی مواجه می‌شویم. در این موارد به کمک کرنل های مختلف ما دیتا هایمان را به بعد های بالاتر می‌بریم و در بعد های بالا تر سعی می‌کنیم کلاس ها را با یک هایپر پلین جدا کنیم.

کمکی بزرگی که کرنل ها به ما می‌کنند این است که بدون اینکه همه ی دیتا ها را به n بعد بالاتر ببریم میتوانیم به کمک فرمول کرنل ارتباط دیتا ها در بعد n ام را محاسبه کنیم و مقدار زیادی از حجم محاسبات ما کم می‌شود.

ما نمی‌توانیم یک حکم کلی در مورد استفاده از کرنل های مختلف بدهیم. ولی به طور مثال روش rbf کرنل پرکاربردی است زیرا دیتای ما را به بعد بی نهایت میبرد و جدا کردن دیتا ها با یک هایپر پلین در بعد بینهایت کار دشواری نیست.

-۶

همانطور که در نوت بوک ضمیمه قابل مشاهده است ما روش svm را از طریق پکیج sklearn روی دیتا هایمان استفاده میکنیم و به دقت ۸۹ درصد میرسیم

-۷

۵ آزمایش مختلف انجام میدهیم

- کرنل چند جمله ای درجه ۲

  دقت: ۴۷ درصد

- کرنل چند جمله ای درجه ۱۰

  دقت: ۲۵ درصد

- کرنل چند جمله ای دیفالت (بهینه ترین حالت)

  دقت: ۸۰ درصد درجه کرنل به دست آمده : ۳

- کرنل rbf

  دقت: ۸۹ درصد

- کرنل سیگموئید

  دقت: ۹۱ درصد

-۸

ما وقتی از هارد مارجین استفاده میکنیم اجازه میس کلسیفیکیشن را نمی‌دهیم برای همین دقت ترین در این روش بالاتر است اما باعث میشد به نوعی دیتا ها اورفیت شود و از جنرال بودن مدل کم میکند برای همین دقت تست به نسبت سافت مارجین پایین تر است

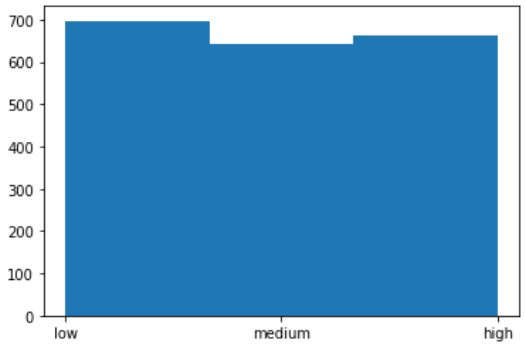
-۹

آ) سه اندازه مختلف bin های مختلف را در نظر میگیریم

- سه قسمت مساوی در دامنه اعداد:

(501-1000), (1000-1499), (1499-1998)

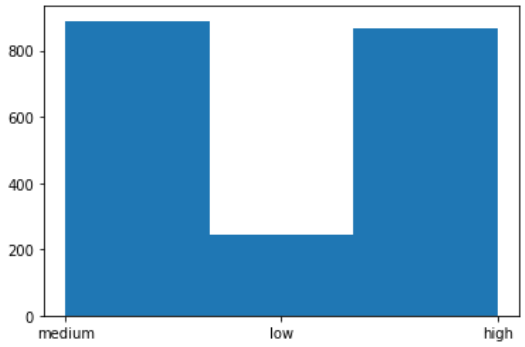
توزیع:



- دومین بین ها:

(0-666), (666-1333), (1333-2000)

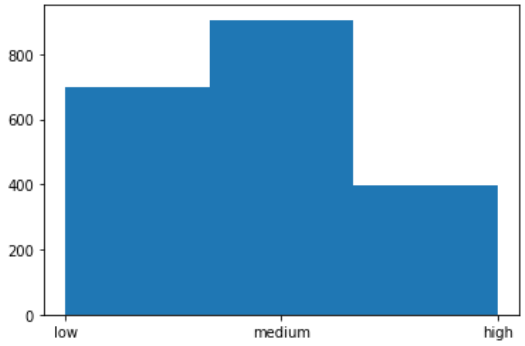
توزیع:



- سومین آزمایش:

(0-1000), (1000-1700), (1700-2000)

توزیع:



ب) این دیتاست فیچر کتگوریکال ندارد.

اما به طور کلی ما باید دیتا هایی که کتگوریکال هستند را به شکلی انکود کنیم تا بتوانیم به مدل های ماشین لرنینگی ورودی بدهیم.

یکی از این روش ها روش وان هات است.

در روش وان هات هر کدام از کتگوری ها یک ستون در نظر گرفته می شوند و اینکه هر دیتا مقدار این فیچرش کدام کتگوری است به این شکل نشان داده میشود که ستون بقیه ی کتگوری ها ۰ و ستون کتگوری آن دیتا مقدار ۱ دارد.

مزیت اصلی این انکود این است که باعث نمیشود دیتا های ما به سمت یک کتگوری خاص بایاس شوند.

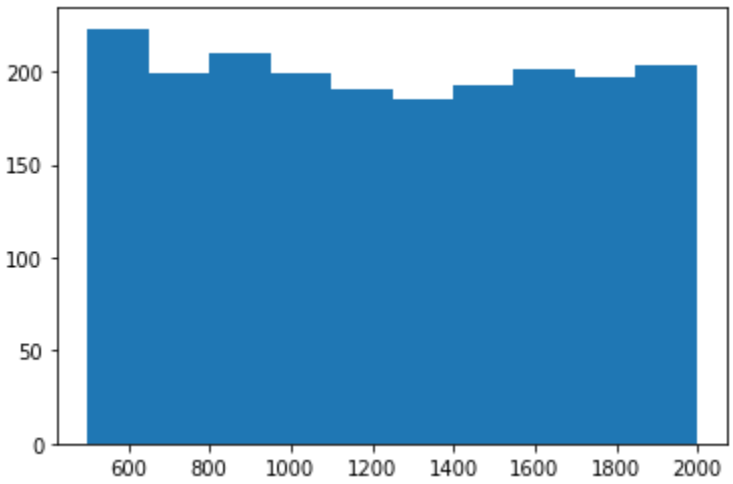
از آنجایی که ما دیتای کتگوریکال نداریم با روش باینینگ با بین های شماره ۱ فیچر های کتگوریکال درست میکنیم و آن را با روش one hot انکود میکنیم.

روی آن با روش svm فرایند کلاس بندی را انجام می دهیم و به دقت ۸۳ درصد میرسیم.

ج) معمولا استفاده از تبدیل ها برای این است که دیتا ها را با توزیع بهتری داشته باشیم

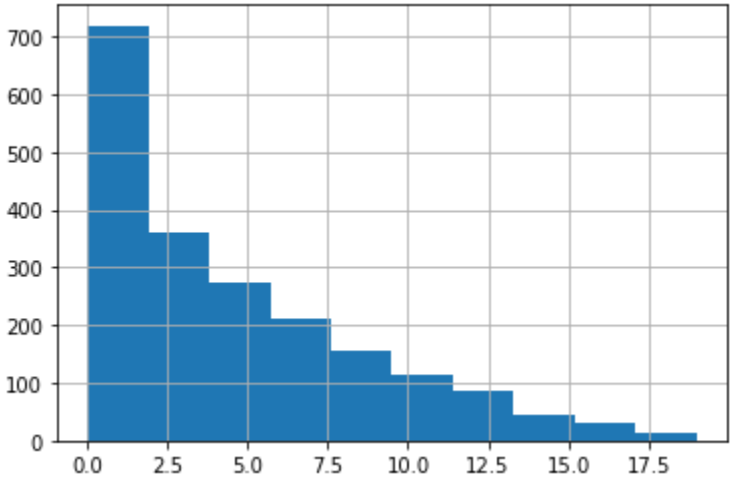
و تبدیل log transform برای دیتا هایی به کار میرود که توزیع نمایی دارند و با نگه داشتن log آن به جای مقدار اصلی میتوانیم ورودی بهتری به مدل ها بدهیم.

در اینجا توزیع فیچر battery power به صورت زیر است.

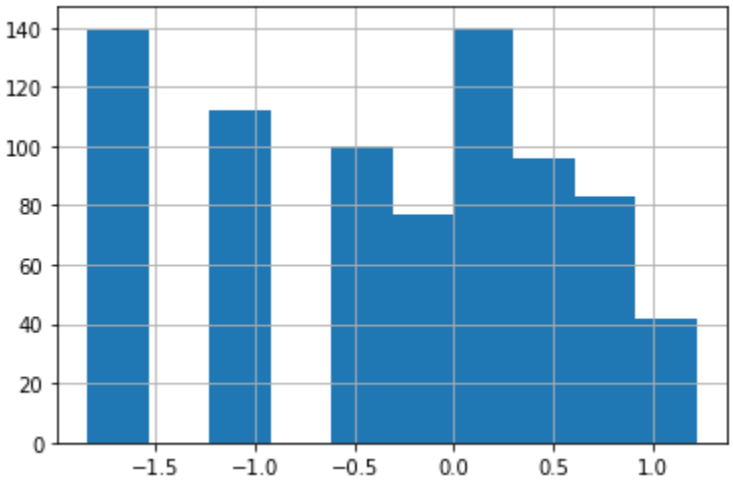


همانطور که مشخص است از یک توزیع نسبتا خطی برخوردار است و نیازی به تبدیل log transform نیست.

اما به طور مثال فیچر fc توزیعی به شکل زیر دارد.



که بعد از اسکیل کردن و انجام log transform به توزیع زیر میرسیم



که توزیع نرمال تری است.

د) فیچر مساحت را می‌سازیم و روی آن SVM میزنیم.

۱۰-در هر قسمت در صورت امکان مدل svm روی فیچرهای خروجی اجرا شده

یک بار برای همه ی پروسس ها با هم svm را اجرا میکنیم.

-۱۱

ما در درخت تصمیم در هر مرحله یک فیچر را انتخاب کرده و بنا بر میانگین و واریانس آن فیچر یک مرزی را مشخص کرده و طبق آن تصمیم گیری میکنیم

تفاوت های الگوریتم های مختلف در روش انتخاب هر فیچر برای هر مرز و انتخاب مرز و همچنین الگوریتم ها در مواجهه با فیچر ها کتگوریکال و میسینگ ولیو ها متفاوت اند.

Features	*ID3*	*C4.5*	*CART*
Type of data	Categorical	Continuous and Categorical	continuous and nominal attributes data
Speed	Low	Faster than ID3	Average
Boosting	Not supported	Not supported	Supported
Pruning	No	Pre-pruning	Post pruning
Missing Values	Can’t deal with	Can’t deal with	Can deal with
Formula	Use information entropy and information Gain	Use split info and gain ratio	Use Gini diversity index

-۱۲

با پکیج **sklearn** و مدل **tree.DecisionTreeClassifier** دیتاست را کلاسبندی میکنیم و به دقت ۸۳ درصد میرسیم.

-۱۳

سه پارامتر را عوض میکنیم

- عمق درخت: ماکسیمم عمق درخت را ۳ می‌گذاریم و دقت ۷۵ درصد میگیریم.
- تعداد فیچر ها: ماکسیمم تعداد فیچر ها را ۱۰ میگذاریم و دقت ۸۱ درصد میگیریم.
- تعداد نمونه های هر گره: این پارامتر را ۵ میگذاریم و دقت ۷۳ درصد میگیریم.

-۱۴

هرس کردن یا pruning به این معنا است که ما قسمتی از درخت تصمیم گیریمان را حذف کنیم.

مثلا فرض کنید ما در لایه n ام درخت تصمیم گیری در مورد فیچر a تصمیم میگیریم. و در صورتی که کمتر از x باشد کلاس c1 را برمیگردانیم و در غیر این صورت کلاس c2.

اما اگر درخت را هرس کنیم باید بگوییم بدون توجه به فیچر a یکی از کلاس c1 یا c2 را برگردانیم.

این کار به وضوح از پیچیدگی مدل ما کم میکند و باعث جلوگیری از اورفیت می‌شود.

-۱۵

بوتسترپینگ یک روش انتخاب نمونه است که در آن همانند کراس ولیدیشن عمل می‌کنیم و هدف ما این است که مقدار بایاس را کم کنیم. با این تفاوت که در کراس ولیدیشن فولد های ما مرز های مشخصی داشتند و اشتراک شان صفر بود اما در بوتسترپینگ به صورت رندوم نمونه ها را مشخص می‌کنیم.

-۱۶

5x2 cross validation به این معنی است که ما ۵ بار از روش 2 fold استفاده می‌کنیم. در حالت عادی 2-fold نمیتواند به خوبی مقدار بایاس را کاهش دهد اما وقتی ما ۵ بار به صورت رندوم استفاده کنیم میتوانیم به خوبی بایاس را کم کنیم.

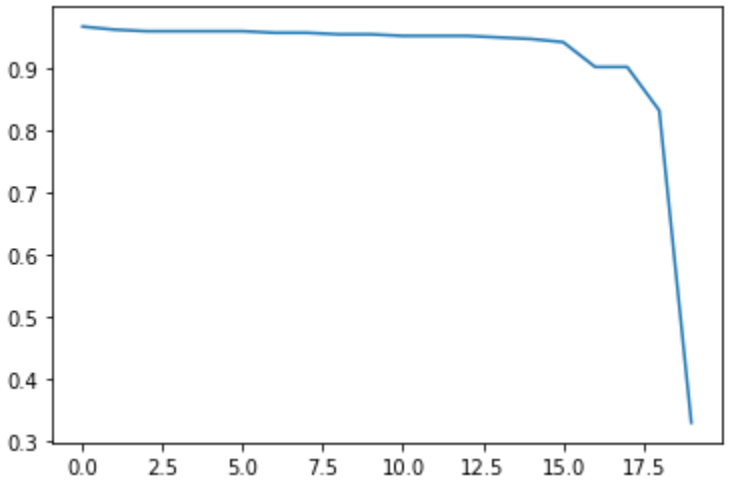
-۱۷

بخش ۱.۱.۲

۱-
تابع backward_selection
را پیاده سازی می‌کنیم.

```python
def backward_selection(X, y):
  features = X.columns
  final_features = {'features': [], 'scores': []}
  rem_features = X.columns
  for i in range(len(X.columns)):
    max_score = 0
    worst_feature = ""
    worst_score = 0
    for feature in rem_features:
      new_X = X[features.drop(feature)]
      X_train, X_test, y_train, y_test = train_test_split(new_X, y, test_size=0.2, random_state=0)
      logisticRegr = LogisticRegression()
      logisticRegr.fit(X_train, y_train)
      score = logisticRegr.score(X_test, y_test)
      if score > max_score :
        max_score = score
        worst_feature = feature
        worst_score = score
    rem_features = rem_features.drop(worst_feature)
    features.drop(worst_feature)

    final_features['features'].append(worst_feature)
    final_features['scores'].append(worst_score)
  final_features['features_rank'] = range(len(X.columns))


  return final_features
```

نمودار فیچر به دقت



تابع backward به ما ۸ فیچر را به عنوان بهترین فیچر می‌دهد.
این ۸ فیچر را به لاجستیک رگرشن می‌دهیم
از Logistic Regression در کتابخانه ی sklearn
استفاده می‌کنیم.
مدل ترین شده:
برای نمونه ده دیتای اول تست را به مدل ترین شده می‌دهیم.

y: 3, 0, 2, 2, 2, 0, 0, 3, 3, 1

pred: 3, 0, 2, 2, 2, 0, 0, 3, 3, 1

همانطور که مشخص است مدل ما همه ۱۰ نمونه تصادفی را درست پیشبینی کرد.
دقت این مدل: ۹۷.۷۵ درصد

	Class 0	Class 1	Class 2	Class 3
percision	0.97916667	0.97752809	0.96039604	0.99122807
recall	0.98947368	0.94565217	0.97979798	0.99122807
f1score	0.98429319	0.96132597	0.97	0.99122807

-۲

ما با تست های آماری امکان این را داریم که بفهمیم که یک فرضیه از نظر آماری درست است یا غلط.
ابتدا یک فرض مطرح می‌شود:
فرض می‌کنیم مدل ۱ به طور میانگین بهتر از مدل ۲ جواب می‌دهد.
سپس به وسیله ی تست های آماری می‌توانیم این فرضیه را با توجه به داده های موجود رد کنیم یا تایید کنیم.

-۳

این به ما ضریبی می‌دهد که نشان می‌دهد که دو فیچر با هم کورلیشن مثبت یا منفی دارند یا کلا ندارند.
این سه حالت را با سه مقدار ۱ ۰ و -۱ نشان می‌دهد.

-۴

بخش ۱.۲.۱

-۱

در این روش ما با توجه به قضیه بیز که در شکل زیر مشخص است احتمال اینکه هر داده برای چه کلاسی باشد را به دست می آوریم
برای این کار نیاز داریم که در نظر بگیریم که هر کلاس از چه توزیعی پیروی می‌کند.

$$P(A|B) = \frac{P(B|A)\ P(A)}{P(B)}$$

به طور مثال در Gaussian naive bayes ما فرض می‌کنیم که کلاس های ما از توزیع گاوسی پیروی می‌کنند.
سپس برای هر داده در نظر می‌گیریم که عضو هر کدام از کلاس ها است و احتمالش را با توجه به فرمول توزیعی که در نظر گرفتیم و قاعده ی بیز به دست می‌آوریم

و بالاترین احتمال را کلاس آن داده در نظر می‌گیریم.
هر کدام از عنوان توزیع های naive bayes برای دیتا هایی با توزیع های مشابه کاربرد دارد.

-۲

کلسیفایر GAUSSIAN NAIVE BAYES را پیاده سازی می‌کنیم و برای هر لیبل تارگت مقدار واریانس و میانگین را حساب می‌کنیم.

-۳

OBJ

```
[ ] gn.test_score(X_test, y_test)

    0.7049180327868853

    from sklearn.metrics import classification_report

    y_pred = gn.predict(X_test)

    print(classification_report(y_test, y_pred))

                  precision    recall  f1-score   support

               0       0.82      0.48      0.61        29
               1       0.66      0.91      0.76        32

        accuracy                           0.70        61
       macro avg       0.74      0.69      0.69        61
    weighted avg       0.74      0.70      0.69        61
```

۴-

```
[ ]  clf.score(X_test, y_test)

     0.7540983606557377

     from sklearn.metrics import classification_report

     y_pred = clf.predict(X_test)

     print(classification_report(y_test, y_pred))
```

```
                  precision    recall  f1-score   support

               0       0.77      0.69      0.73        29
               1       0.74      0.81      0.78        32

        accuracy                           0.75        61
       macro avg       0.76      0.75      0.75        61
    weighted avg       0.76      0.75      0.75        61
```

۵-

به وضوح مدل پکیج در همه معیار ها عملکرد بهتری داشته است که به این دلیل است که ممکن است مدل پیاده سازی شده در SKLEARN پیچیدگی های بیشتری داشته و الگوریتم های بهینه تری پیاده سازی کرده باشند و به این دلیل عملکرد بهتری ارائه می‌دهد.