گزارش قسمت ۳

Mobile Price Classification

علی صالح ۹۷۲۲۲۰۵۳

Data preprocess

- ایتدا چک میکنیم که ستون ها مقدار نال یا غیر عددی نداشته باشد
- دیتاست را اسکیل می کنیم. برای اسکیل کردن از standard scaler استفاده می‌کنیم.
- ۲۰ درصد دیتا را به عنوان دیتای تست جدا میکنیم

بخش ۱

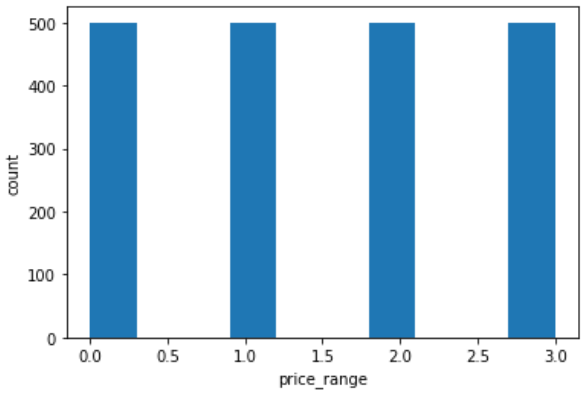
از Logistic Regression در کتابخانه ی sklearn استفاده می‌کنیم.
مدل ترین شده:
برای نمونه ده دیتای اول تست را به مدل ترین شده می‌دهیم.

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

همانطور که مشخص است فقط مدل ما از این ۱۰ نمونه فقط یکی را به اشتباه تشخیص داد.
دقت این مدل: ۹۵.۵ درصد

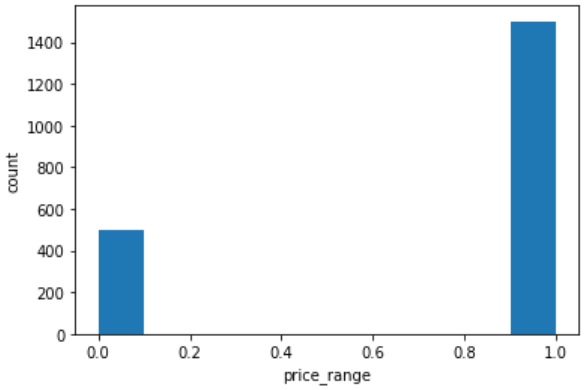
	Class 0	Class 1	Class 2	Class 3
percision	0.97894737	0.94444444	0.93814433	0.95762712
recall	0.97894737	0.92391304	0.91919192	0.99122807
f1score	0.97894737	0.93406593	0.92857143	0.97413793

بخش ۲



همانطور که از نمودار روبرو مشخص است هر چهار کلاس تعداد برابر نمونه دارند

بخش ۳



پس از انجام بخش ۳ به توزیع مقابل می‌رسیم

بخش ۴

با دیتای بالا مدل Logistic Regression کتابخانه sklearn را ترین می‌کنیم.
مدل ترین شده:
برای نمونه ده دیتای اول تست را به مدل ترین شده می‌دهیم.

y: 1, 0, 1, 1, 1, 0, 0, 1, 1, 1

pred: 1, 0, 1, 1, 1, 0, 0, 1, 1, 1

همانطور که مشخص است مدل ما همه ی ۱۰ نمونه ی تصادفی مارا درست پیشبینی کرد.
دقت این مدل: ۹۹ درصد

	Class 0	Class 1
percision	0.97894737	0.99344262
recall	0.97894737	0.99344262
f1score	0.97894737	0.99344262

بخش ۵

مشکلات بالانس نبودن دیتا:

۱- مشکل اول این است که مدل های ماشین لرنینگی برای دیتا های بالانس ساخته شده اند و درست برای دیتاهایی که بالانس نیستند کار نمیکنند

۲- اگر به طور مثال یک کلاس از ۴ کلاس شامل ۷۰ درصد دیتا ها باشد ما می توانیم مدلی ارائه دهیم که فقط خروجی آن کلاس خاص را می‌دهیم. به وضوح این مدل ما مدل درستی نیست ولی به دلیل بالانس نبودن دیتا به ما درصد غیر واقعی و خوب ۷۰ را می‌دهد

راه حل:

۱- استفاده از f1score و recall به جای دقت.

این متریک ها تا حدی با بالانس نبودن دیتا مقابله می‌کنند و آن مشکلی که ممکن بود بالانس نبودن دیتا برای دقت خروجی به وجود بیاورد را ندارند. البته این راه بهترین راه حل نیست.

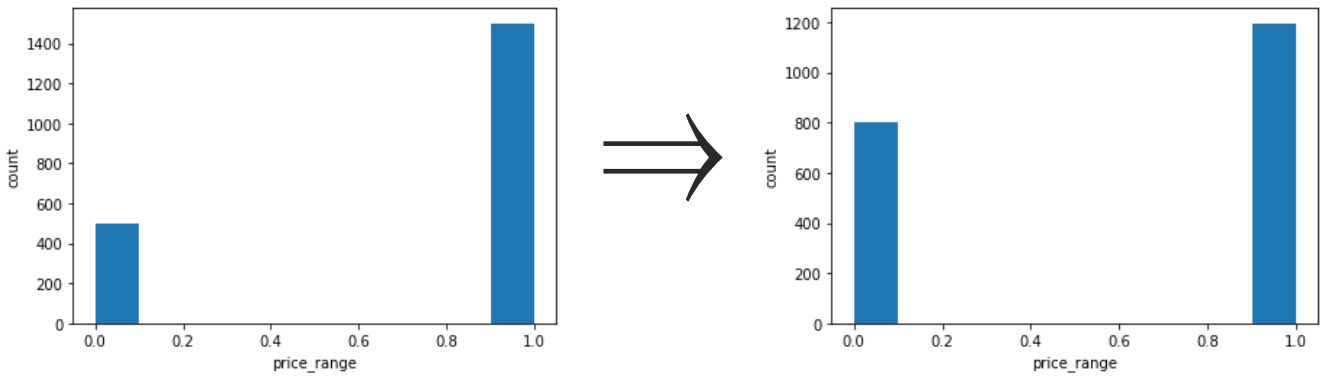
۲- کپی قسمتی از کلاس های با جمعیت کمتر را به آن کلاس اضافه کنیم و به این شکل جمعیت این کلاس ها را بیشتر کنیم یا قسمتی از دیتا های کلاس با جمعیت بیشتر را کم کنیم و کلاس هایمان را بالانس کنیم.

۳-استفاده از الگوریتم Decision trees

این الگوریتم به بالانس نبودن دیتا حساس نیست.

از روش آپ سمپلینگ استفاده میکنیم و به کپی کلاس صفر ها را به خودش اضافه می‌کنیم.

نمودار توزیع را پس از آپ سمپلینگ می‌بینید واضح است که دیتا ها به مراتب بالانس تر شده اند.



با دیتای بالا مدل Logistic Regression کتابخانه sklearn را ترین می‌کنیم.

مدل ترین شده:

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

y: 1, 0, 1, 1, 1, 0, 0, 1, 1, 1

pred: 1, 0, 1, 1, 1, 0, 0, 1, 1, 1

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

دقت این مدل: ۹۰ درصد

جدول percision recall و f1score در صفحه بعد.

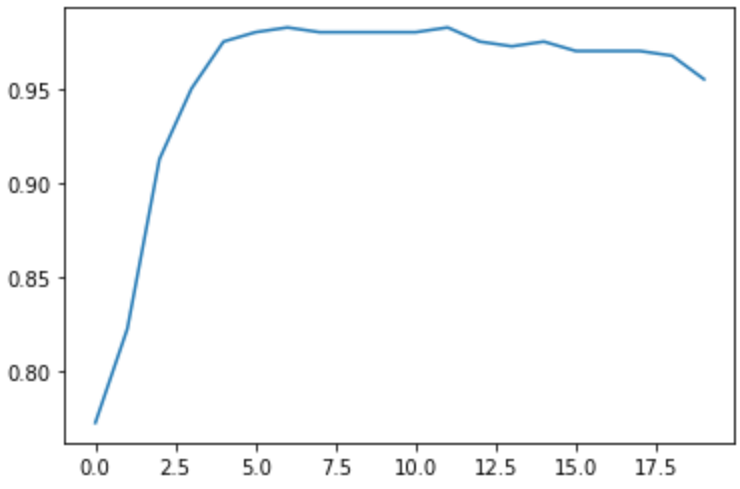
	Class 0	Class 1
percision	0.7037037	1
recall	1	0.86885246
f1score	0.82608696	0.92982456

بخش ۶

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

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

بخش ۷

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

بخش ۸

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

بخش ۹

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

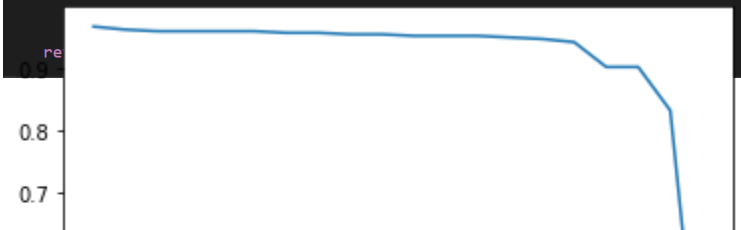
	Class 0	Class 1	Class 2	Class 3
percision	0.98958333	0.97849462	0.96938776	0.98230088
recall	1	0.98913043	0.95959596	0.97368421
f1score	0.9947644	0.98378378	0.96446701	0.97797357

بخش ۱۰

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))
```

نمودار فیچر به دقت



تایع 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

بخش ۱۱

از تابع cross_val_score کتابخانه sklearn استفاده می‌کنیم

5-fold

([0.96 , 0.955 , 0.9675, 0.9625, 0.9675])

10-fold

[0.96, 0.97 ,0.955, 0.965 , 0.95 , 0.975, 0.96 , 0.95 , 0.98 ,0.955]