به نام خدا

دانشگاه تهران

دانشکده مهندسی مکانیک

هوش مصنوعی

تمرین ۲

نام و نام خانوادگی	محمد مشرقی
شماره دانشجویی	
تاریخ ارسال گزارش	

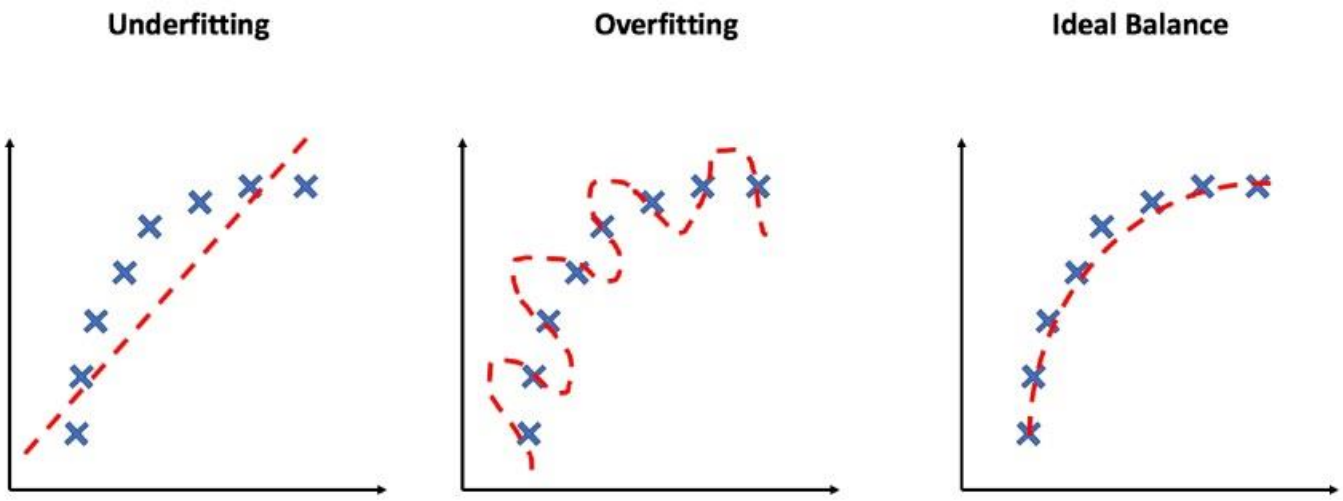
Contents

۱-Regularization

تعریف : در ماشین لرنینگ و دیپ لرنینگ برای اینکه از

۱. پیچیده تر شدن مدل(Overfitting).
۲. و نتایج بسیار نزدیک به داده های تمرینی نزدیک شود که در نهایت ممکن است باعث شود در داده های استفاده نشده (جدید) خطا زیاد شود.

جلوگیری شود از Regularization استفاده می شود.



انواع آن :

- **L1 Regularization (Lasso Regression):** (Least Absolute Shrinkage and Selection Operator)

در اینجا با انتخاب مناسب lambda می توانیم برای هر ضریب به مقداری مناسب برسیم و آن را به کم کنیم(و نهایتا وزن به صفر متمایل کنیم)

این ویژگی وقتی ضریب های زیادی داشته باشیم و بخواهیم وزن ضریب های کم مهم صفر شوند بدرد می خورد.

$$\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

- **L2 Regularization(Ridge regression):**

در اینجا هم مانند L1 عمل می کند اما در انتهای تمرین نتایج فقط به صفر میل می کنن و صفر نمی شوند.

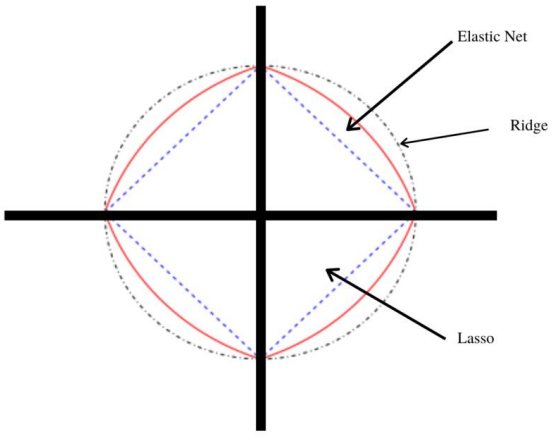
وقتی استفاده می کنیم که ویژگی های هم خطی و همبستگی داشته باشیم.

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Comparison of L1 and L2 regularization	
L1 regularization	***L2 regularization***
Sum of absolute value of weights	Sum of square of weights
Sparse solution	Non-sparse solution
Multiple solutions	One solution
Built-in feature selection	No feature selection
Robust to outliers	Not robust to outliers (due to the square term)

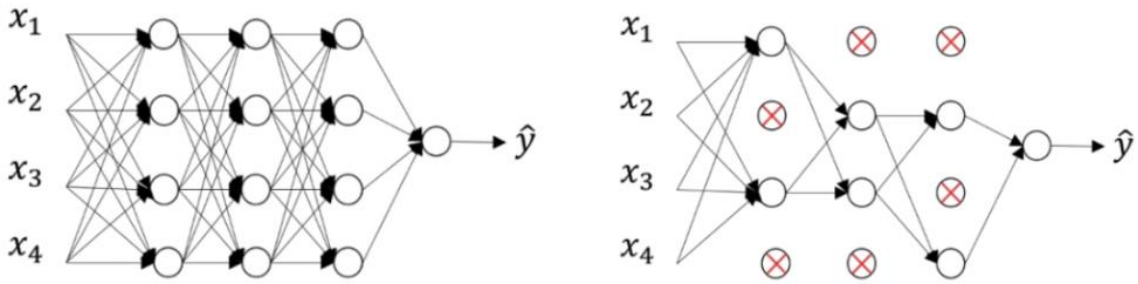
- **Elastic net**

این روش ترکیبی بین L1و L2 هستش.



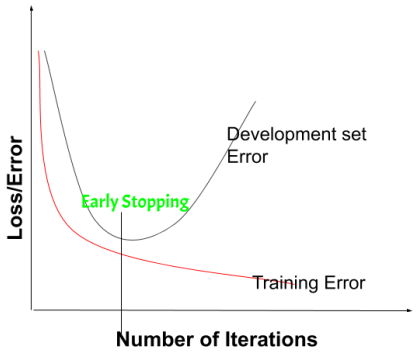
- Dropout

در دیپ لرنینگ گاهی وقت ها با حذف بعضی نود ها از overfitting جلوگیری می کنیم.



- Early Stopping

برای روش هایی در تعیین تعداد دسته استفاده می شود.



۲-

در این قسمت ابتدا داده ها و سپس برای تمرین و تست تقسیم می کنیم.

```
X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size = 0.25, random_state = 0)
```

سپس به درجه ۷ می بریم.

```
poly = PolynomialFeatures(degree = 7)
X_poly_train = poly.fit_transform(X_train)
```

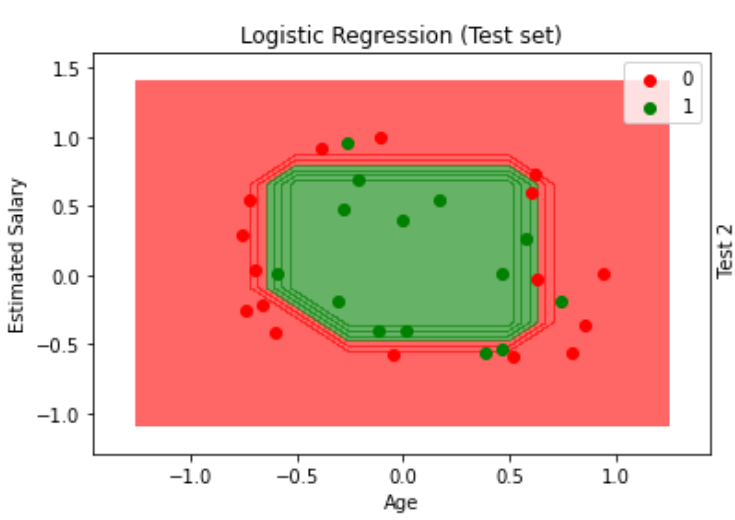
حال به سراغ اموزش می رویم و regularization رو l2 تنظیم می کنیم c=0.01

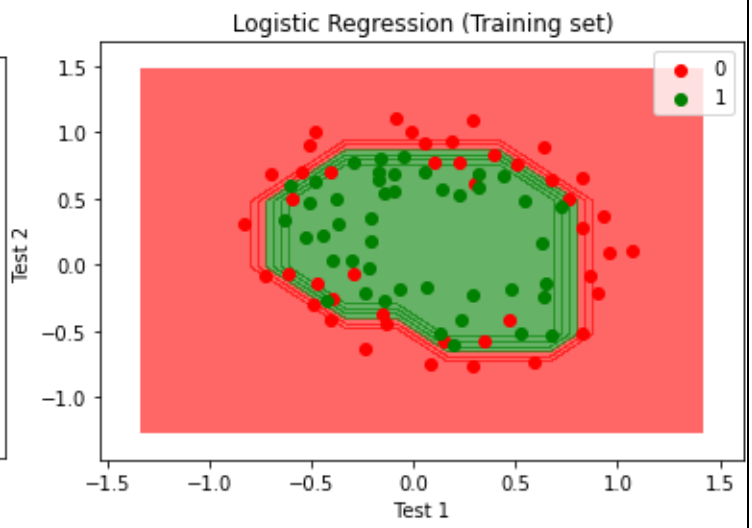
```
classifier = LogisticRegression
(C=0.01, penalty='l2',random_state =43)
classifier.fit(X_poly_train, y_train)
```

نتایج:

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[14  2]
 [ 2 12]]
0.8666666666666667
```





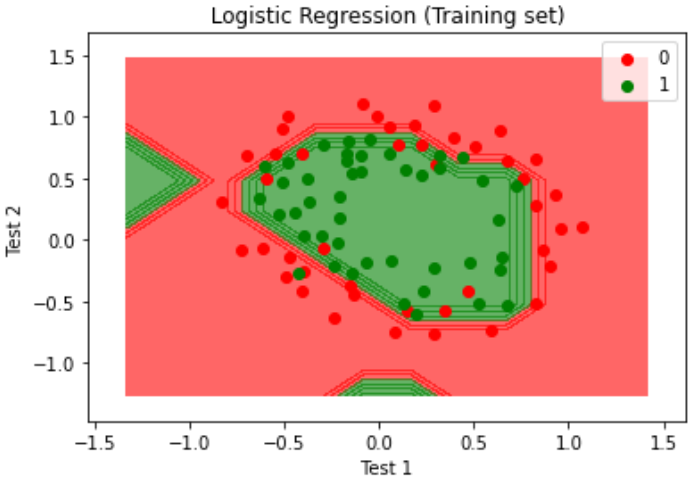
حال اگر c=1داریم:

```
classifier = LogisticRegression(C=1, penalty='l2',random_state =43)
classifier.fit(X_poly_train, y_train)
```

نتایج:

```
[107] from sklearn.metrics import confusion_matrix, accuracy_score
      cm = confusion_matrix(y_test, y_pred)
      print(cm)
      accuracy_score(y_test, y_pred)

      [[14  2]
       [ 3 11]]
      0.8333333333333334
```



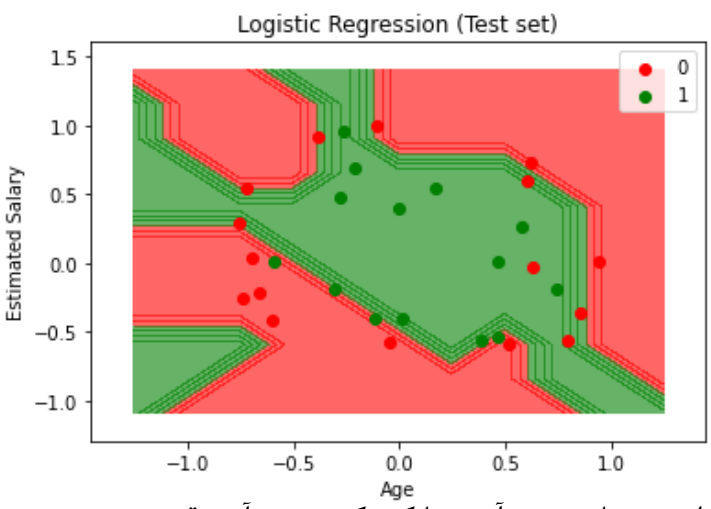
حال اگر c=1000 داریم:

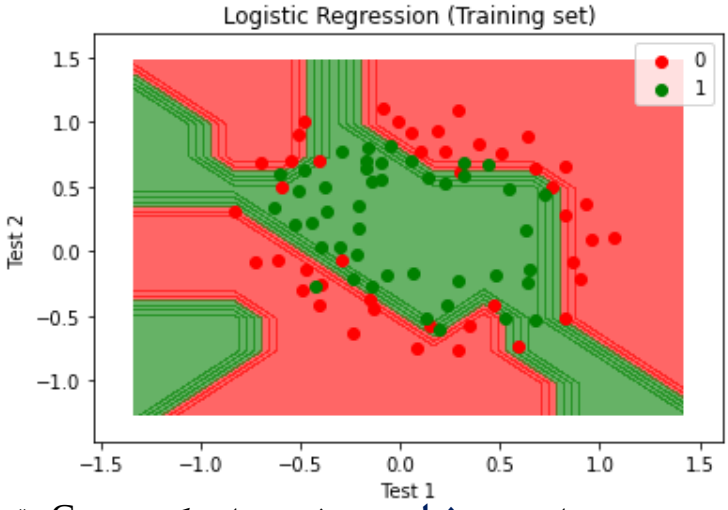
```
classifier = LogisticRegression
(C=10000, penalty='l2',random_state =43)
classifier.fit(X_poly_train, y_train)
```

نتایج:

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[10  6]
 [ 4 10]]
0.6666666666666666
```





با توجه به **نتایج** می فهمیم با بزرگتر شدن C دقت اموزش پایین می آید و با کوچکتر شدن آن دقت بیشتر می شود. $C = \frac{1}{\lambda}$ و $\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda\sum_{j=1}^{p}\beta_j^2$

با بزرگ شدن c ،جریمه کوچکتر می شود و ممکن است overfit رخ دهد یا نتایج غیر قابل قبول باشند و اینکه با توجه به سوال یک مدل بیش از حد در مورد ویژگی های داده های اموزشی یاد می گیرند و نمی توانند با داده های جدید تعمیم داد.

با کوچک شدن c، جریمه بزرگتر می شود و در نهایت overfit رخ نمی دهد و دقت آن نیز بیشتر است اما از یه حدی به بعد مدل بسیار ساده خواهد بود و مدل ممکن است با عدم تناسب داده مواجهه شود و نمی تواند به خوبی یاد بگیرد .

مقدار ایده آل C برای هر مدل بستگی به داده های آن دارد و نمی توان همیشه یه مقدار ثابت گرفت.
حال در این سوال ما c=0.01 فرض می کنیم.

۳-

اعتبار سنجی متقابل یک روش برای ارزیابی یک مدل در ماشین لرنینگ و همچنین آزمایش نحوه عملکرد آن است. از CV اصولا در فعالیت های کاربردی یادگیری ماشین استفاده می شود. این کار درمقایسه و انتخاب یک مدل مناسب برای مسئله مدل‌سازی پیش‌بینی‌کننده خاص کمک می‌کند.

حال تست می کنیم و مقدار k را برابر ۱۰ قرار می دهیم.

اگر درجه ۷ باشد نتایج داریم:

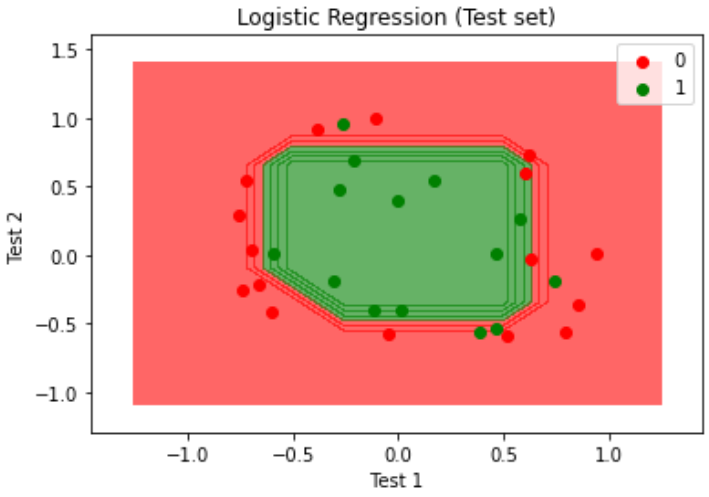
```
[15] from sklearn.metrics import confusion_matrix, accuracy_score
     cm = confusion_matrix(y_test, y_pred)
     print(cm)
     accuracy_score(y_test, y_pred)
```

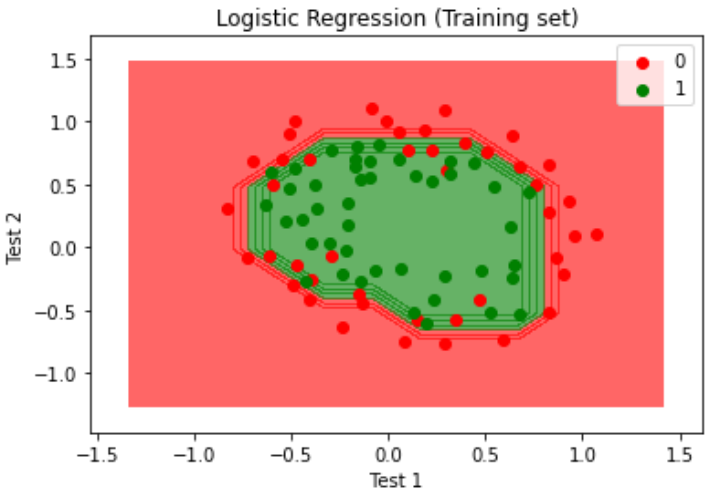
```
[[14  2]
 [ 2 12]]
0.8666666666666667
```

K_fold cv

```
[29] from sklearn.model_selection import KFold
     from sklearn.model_selection import cross_val_score
     from numpy import mean
     from numpy import std
     cv = KFold(n_splits=10, random_state=1, shuffle=True)
     # create model
     model = LogisticRegression()
     # evaluate model
     scores = cross_val_score(model, poly.fit_transform(X), y, scoring='accuracy', cv=cv, n_jobs=-1)
     # report performance
     print('Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

```
Accuracy: 0.805 (0.092)
```





اگر درجه ۳ باشد نتایج داریم:

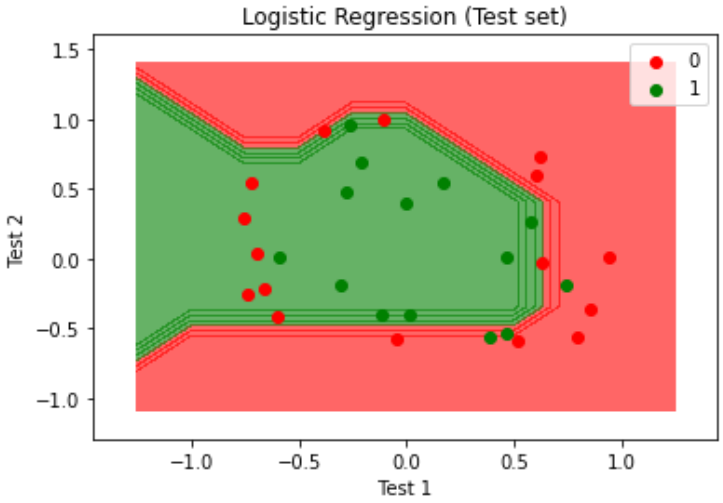
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

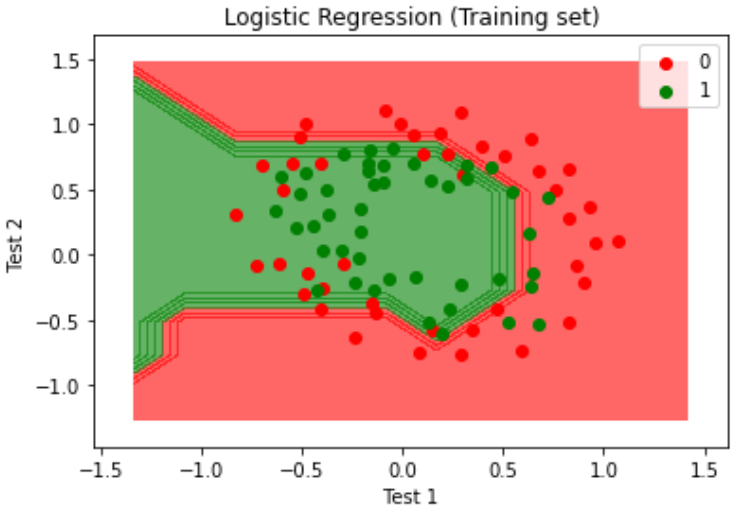
```
[[11  5]
 [ 4 10]]
0.7
```

_fold cv

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from numpy import mean
from numpy import std
cv = KFold(n_splits=10, random_state=1, shuffle=True)
# create model
model = LogisticRegression()
# evaluate model
scores = cross_val_score(model, poly.fit_transform(X), y, scoring='accuracy', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

```
Accuracy: 0.779 (0.107)
```





درجه ۱۰ باشد داریم :

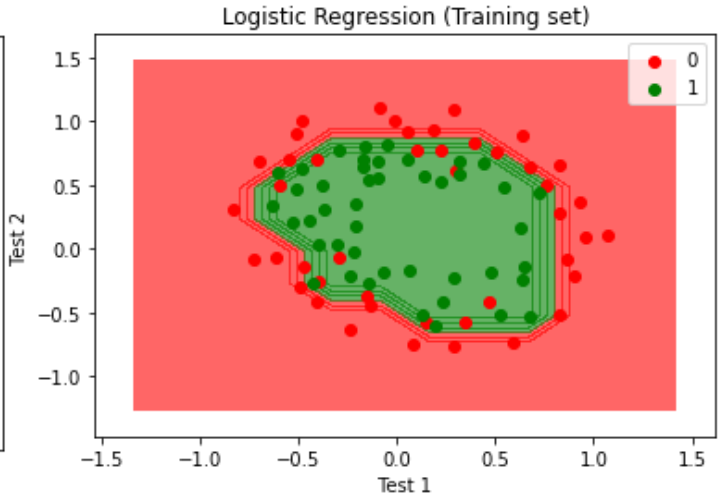
```
[44] from sklearn.metrics import confusion_matrix, accuracy_score
     cm = confusion_matrix(y_test, y_pred)
     print(cm)
     accuracy_score(y_test, y_pred)
```

```
[[13  3]
 [ 2 12]]
0.8333333333333334
```

K_fold cv

```
[45] from sklearn.model_selection import KFold
     from sklearn.model_selection import cross_val_score
     from numpy import mean
     from numpy import std
     cv = KFold(n_splits=10, random_state=1, shuffle=True)
     # create model
     model = LogisticRegression()
     # evaluate model
     scores = cross_val_score(model, poly.fit_transform(X), y, scoring='accuracy', cv=cv, n_jobs=-1)
     # report performance
     print('Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

```
Accuracy: 0.796 (0.086)
```



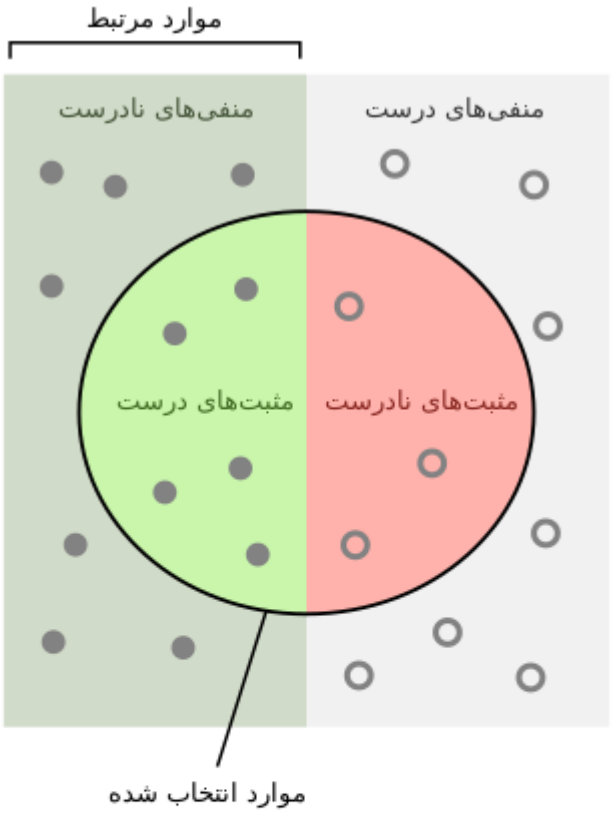
نتیجه :

با توجه به تست k fold cv ، mean(score) هر چقدر به یک نزدیک تر شود اون درجه بهتر است که با توجه به نتایج درجه برابر ۷ بهترین حالت است.

۴-

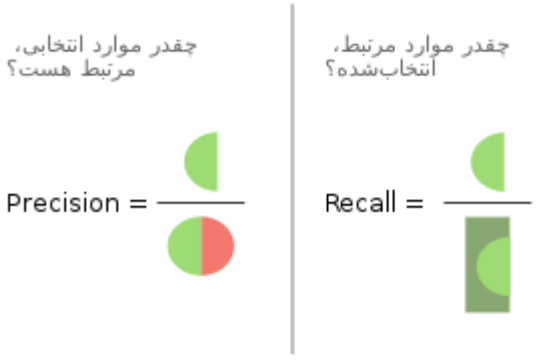
تعاریف :

Recallبرابر است با تقسیم تعداد مواردی که توسط مدل درست تشخیص داده‌اند شده بر تعداد کل مواردی که توسط مدل ایجاد شده‌اند و Precision برابر است با تقسیم تعداد مواردی که توسط مدل درست تشخیص داده شده‌است بر تعداد مواردی که واقعاً درست هستند، درست تشخیص داده شده‌اند.



$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$

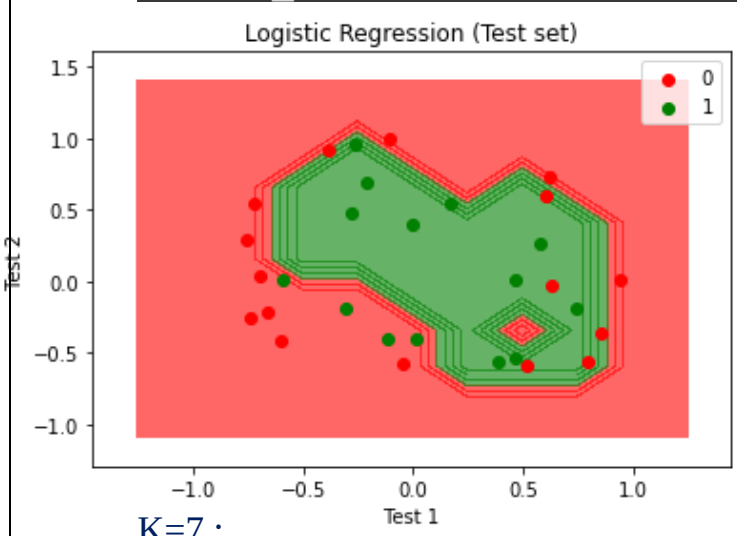


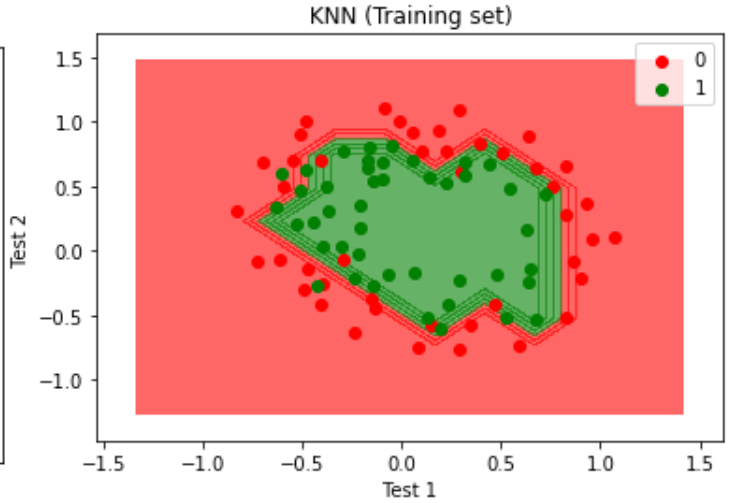
K=1 :

```
classifier = KNeighborsClassifier(n_neighbors = 1 , p = 2)
```

نتایج:

```
accuracy_score =  0.8
precision_score =  0.7857142857142857
recall_score =  0.7857142857142857
```



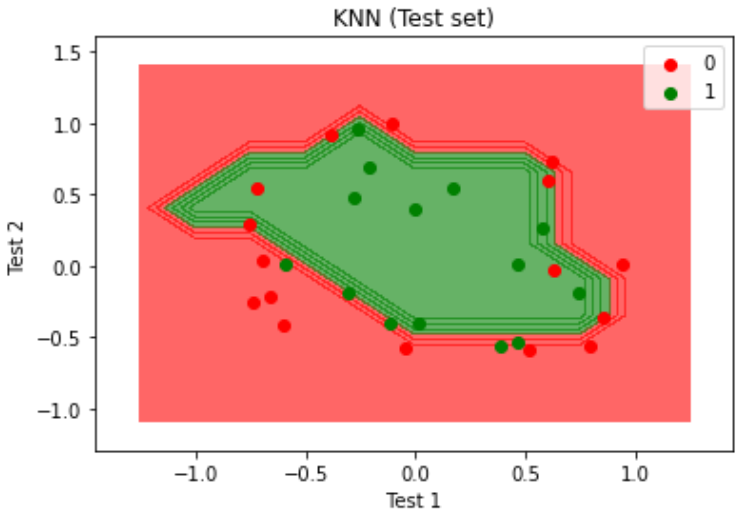


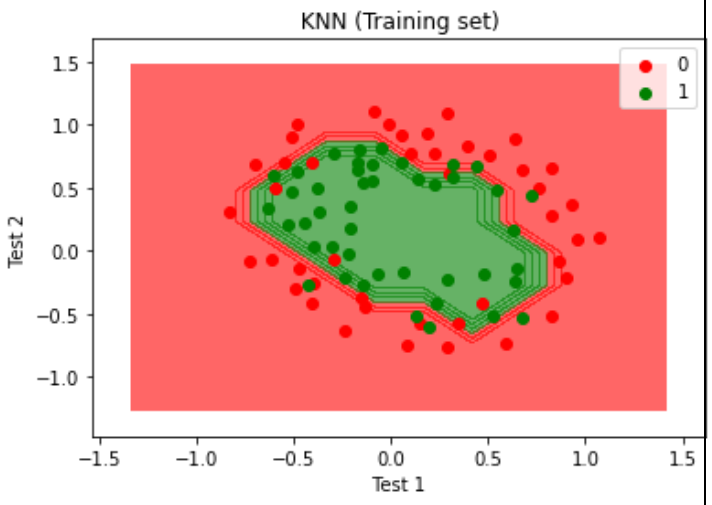
K=7 :

```
classifier = KNeighborsClassifier(n_neighbors = 7 , p = 2)
```

نتایج:

```
accuracy_score =  0.8
precision_score =  0.75
recall_score =  0.8571428571428571
```



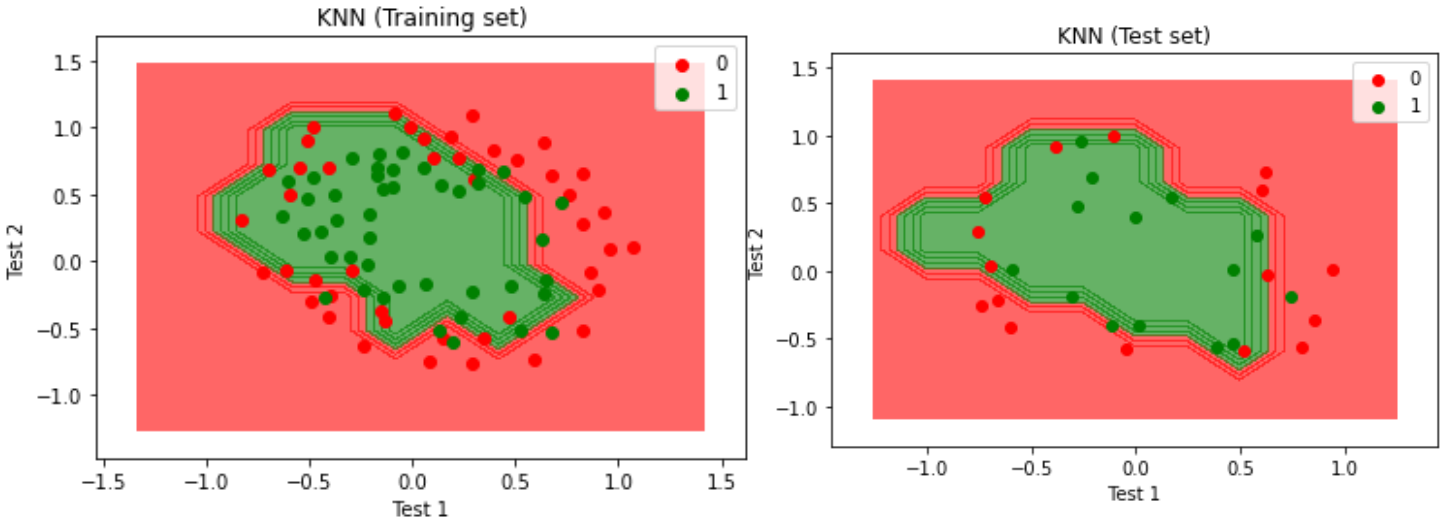


K=13 :

```
classifier = KNeighborsClassifier(n_neighbors = 13 , p = 2)
```

نتایج :

```
accuracy_score =  0.7
precision_score =  0.631578947368421
recall_score =  0.8571428571428571
```

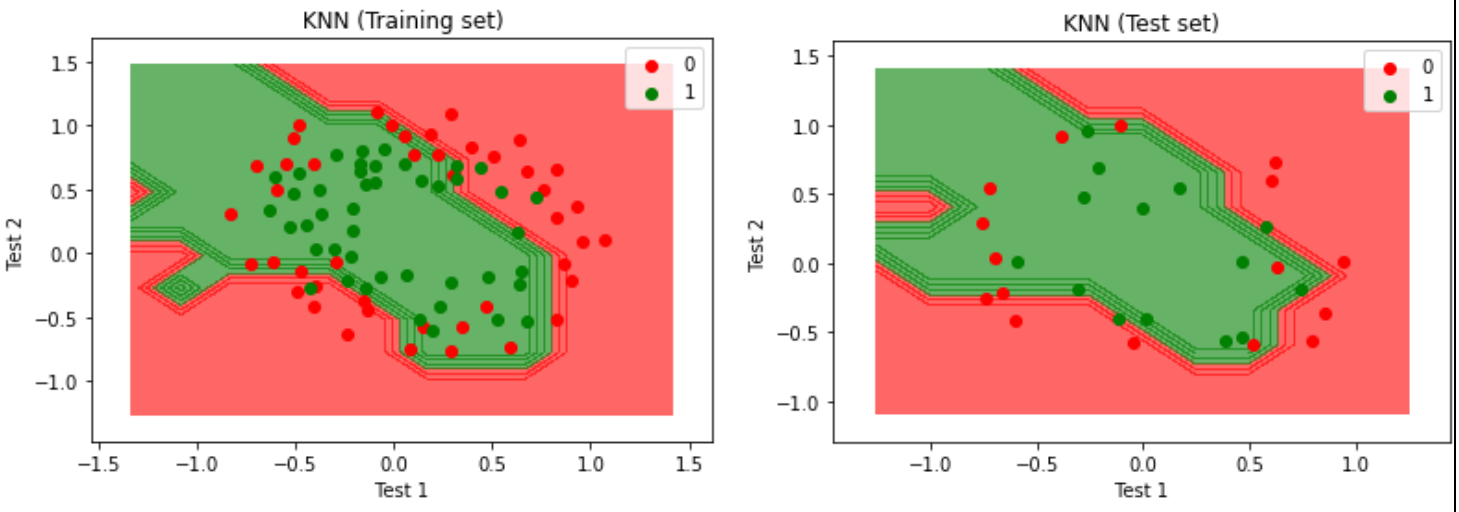


K=19 :

```
classifier = KNeighborsClassifier(n_neighbors = 19 , p = 2)
```

نتایج :

```
accuracy_score =  0.6666666666666666
precision_score =  0.6111111111111112
recall_score =  0.7857142857142857
```



نتیجه:

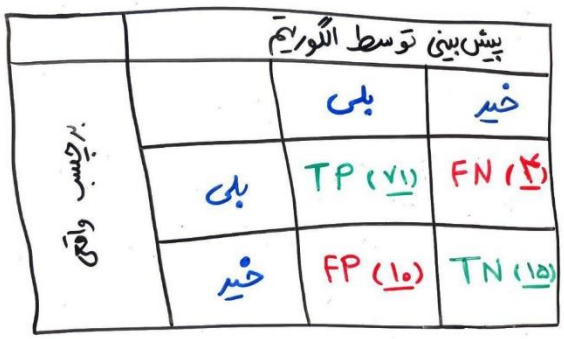
می دانیم که در معیار Accuracy نمی‌تواند تفاوتی بین خطای False Negative و خطای False Positiveداشته باشد.

برای همین از پوشش (recall) و صحت (Precision) استفاده می کنیم.

اگر با دقت به فرمول نگاه کرده باشید، متوجه می‌شوید که تمرکز اصلی این معیار، بر روی درستیِ تشخیص‌های «بلی» توسط الگوریتم است. در واقع معیار صحت (Precision) معیاری است که به ما می‌گوید الگوریتم چند درصدِ «بلی»هایش درست بوده.

همان‌طور که مشاهده می‌کنید، تمرکز اصلی معیار پوشش (Recall) بر خلاف معیار صحت (Precision)بر روی داده‌هایی است که واقعاً «بلی» بوده‌اند.

مثال یکی از سایت ها :



$$\text{Recall (پوشش)} = \frac{TP}{TP + FN} \quad \begin{matrix} (۷۱) \\ (۷۱) \quad (۴) \end{matrix}$$

$$\text{Precision (صحت)} = \frac{TP}{TP + FP} \quad \begin{matrix} (۷۱) \\ (۷۱) \quad (۱۰) \end{matrix}$$

حال با توجه به نتایج و نیازمان از تحلیل دنبال معیار مورد نظر می گردیم و طبق اون دنبال k مورد نظر می رویم.

حال اگر فرض کنیم که قطعه مورد نظر اگر خراب باشد و داخل دستگاه قرار گیرد باعث می شود بقیه قطعات دستگاه نیز خراب شوند پس اینجا باید سعی کنیم تا می توانیم از ورود دستگاه خراب جلوگیری

کنیم پس از روش پوشش Recall را مهمتر از روش صحت فرض می کنیم که طبق این قاعده K = 7 انتخاب می شود.

۵-

برای بدست اوردن بهترین مقدار K با استفاده از روش فاصله منهتن باید جای p =2 ، باید **یک** بگذاریم داریم:

$$d = \sum_{i=1}^{n} |x_i - y_i|$$

فرمول فاصله منهتن

```
classifier = KNeighborsClassifier(n_neighbors = 1 , p = 1)
```

حال این بار از روشی دیگر استفاده می کنیم و با یک for کار را تموم می کنیم:

```
for i in range(1 ,20):
  classifier = KNeighborsClassifier(n_neighbors = i , p = 1)
  classifier.fit(X_train, y_train)
  y_pred = classifier.predict(X_test)
  print( "k = ",i, ' accuracy_score =  %.3f    precision_score =
  %.3f   recall_score =   %.3f '%(accuracy_score(y_test, y_pred),prec
ision_score(y_test, y_pred),recall_score(y_test, y_pred)))
```

نتیجه :

```
k =  1  accuracy_score =  0.800    precision_score =  0.833
recall_score =   0.714

k =  2  accuracy_score =  0.767    precision_score =  0.889
recall_score =   0.571

k =  3  accuracy_score =  0.633    precision_score =  0.615
recall_score =   0.571

k =  4  accuracy_score =  0.700    precision_score =  0.857
recall_score =   0.429

k =  5  accuracy_score =  0.767    precision_score =  0.769
recall_score =   0.714

k =  6  accuracy_score =  0.733    precision_score =  0.875
recall_score =   0.500

k =  7  accuracy_score =  0.767    precision_score =  0.769
recall_score =   0.714
```

```
k =  8  accuracy_score =  0.833    precision_score =  0.909
recall_score =   0.714

k =  9  accuracy_score =  0.767    precision_score =  0.684
recall_score =   0.929

k =  10  accuracy_score =  0.767    precision_score =  0.733
recall_score =   0.786

k =  11  accuracy_score =  0.800    precision_score =  0.722
recall_score =   0.929

k =  12  accuracy_score =  0.833    precision_score =  0.846
recall_score =   0.786

k =  13  accuracy_score =  0.767    precision_score =  0.684
recall_score =   0.929

k =  14  accuracy_score =  0.767    precision_score =  0.733
recall_score =   0.786

k =  15  accuracy_score =  0.700    precision_score =  0.609
recall_score =   1.000

k =  16  accuracy_score =  0.567    precision_score =  0.533
recall_score =   0.571

k =  17  accuracy_score =  0.667    precision_score =  0.600
recall_score =   0.857

k =  18  accuracy_score =  0.600    precision_score =  0.600
recall_score =   0.429

k =  19  accuracy_score =  0.733    precision_score =  0.667
recall_score =   0.857
```

اگر بخواهیم مثل قبل عمل کنیم k=15 بهترین دسته بندی است چون معیار پوشش آن صد درصد است.

اگر از بین اون چهار تا بخواهیم انتخاب کنیم k=13