HW11

INSIK CHO
2021321148

0. Data Generation

```
#data generation

def ygeneration(X):
  fx = (X['X1']*np.sqrt(np.abs(X['X4'])+1)+ np.sin(X['X3'])
   - np.abs(X['X2'])/(np.sqrt(np.abs(X['X5'])+1)) +1.5*np.abs(X['X6']) +X['X7'])
  p = 1/ (1+np.exp(-fx))
  y=[]
  for i in range(len(p)):
    y.append(np.random.binomial(n=1, p=p[i], size=1 ))
  return np.ravel(y)

Xtrain = {}
for i in range(7):
  if i <=2:
    Xtrain[f'X{i+1}'] = np.random.uniform(-10,10,1000)
  else:
    Xtrain[f'X{i+1}'] = np.random.normal(loc=0, scale = np.sqrt(10),size = 1000)
Xtrain = pd.DataFrame(Xtrain)


Xtest = {}
for i in range(7):
  if i <=2:
    Xtest[f'X{i+1}'] = np.random.uniform(-10,10,1000)
  else:
    Xtest[f'X{i+1}'] = np.random.normal(loc=0, scale = np.sqrt(10),size = 1000)
Xtest = pd.DataFrame(Xtest)

ytrain = ygeneration(Xtrain)
ytest = ygeneration(Xtest)
```

위와 같은 코드로 문제에서 요구하는 샘플링을 진행하여 test, train 데이터를 생성했습니다.

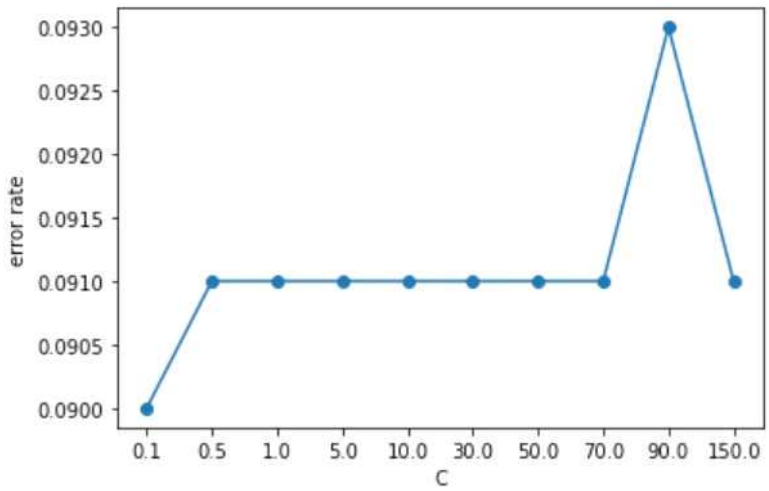
1~3. Linear SVC

```
#1. fitting using linear SVM
C = np.array([ 0.1, 0.5, 1, 5, 10, 30, 50, 70, 90, 150])

linearclf = {}
errorrate = []
for i in range(len(C)):
  linearclf[f'SVC{i}'] = SVC(kernel='linear', random_state=0, C=C[i])
  linearclf[f'SVC{i}'].fit(Xtrain, ytrain)
  errorrate.append(1-accuracy_score(ytest, linearclf[f'SVC{i}'].predict(Xtest)))

#2. drawing graphs
plt.plot(errorrate ,marker="o")
plt.xticks(range(len(C)), C)
plt.xlabel('C')
plt.ylabel('error rate')
plt.show()

#3. the lowest error rate one
linearC = C[np.argmin(errorrate)]
print('the best C :', linearC)
```




```
the best C : 0.1
```

위와 같은 코드로 Linear SVC 이후, 오분류율 그래프를 그렸습니다.

Linear kernel의 경우 C=0.1 일 때 가장 효율적임을 파악할 수 있었습니다.

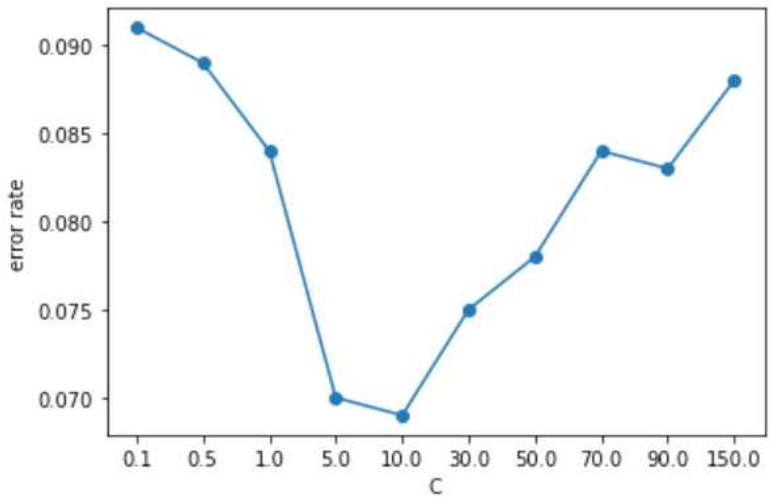
4~6. rbf SVC

```
#4. fitting using non-linear SVM
C = np.array([ 0.1, 0.5, 1, 5, 10, 30, 50, 70, 90, 150])

rbfclf = {}
errorrate = []
for i in range(len(C)):
  rbfclf[f'SVC{i}'] = SVC(kernel='rbf', random_state=0, C=C[i])
  rbfclf[f'SVC{i}'].fit(Xtrain, ytrain)
  errorrate.append(1-accuracy_score(ytest, rbfclf[f'SVC{i}'].predict(Xtest)))

#5. drawing graphs
plt.plot(errorrate, marker = 'o')
plt.xticks(range(len(C)), C)
plt.xlabel('C')
plt.ylabel('error rate')
plt.show()

#6. the lowest error rate one
rbfC = C[np.argmin(errorrate)]
print('the best C :', rbfC)
```




```
the best C : 10.0
```

Linear과 동일한 방식으로, kernel만 바꿔서 진행했으며, C=10.0에서 오분류율이 최소가 됨을 확인했습니다.

7. 결과

```
Linear Kernel SVM(C= 0.1 )
Confusion Matrix( SVM-linear)
-----------------------------------
          predicted class
Actual 1  [399  50]
class  2  [ 40 511]
model summary
---------------------------------
Overall accuracy =  0.91

RBF Kernel SVM(C= 10.0 )
Confusion Matrix( SVM-rbf)
-----------------------------------
          predicted class
Actual 1  [405  44]
class  2  [ 25 526]
model summary
---------------------------------
Overall accuracy =  0.931
```

Linear은 C=0.1, rbf는 C=10.0으로 예측을 한 결과, 0.931 > 0.91로 rbf가 미세하게 더 우수한 것을 확인했습니다. 해당 모델은 non-linear hyperplane이 적합하다고 판단됩니다.