**Report**



**2021 Classification 경진대회**

|  |  |
| --- | --- |
| 작성자 | 소프트웨어학과 김채은 |
| 작성일 | 2021. 10. 25. |
| 담당교수 | 오세종 교수님 |

**■ 개요**

**1. 데이터 셋 준비**

* 훈련 셋을 x, y로 나눈다.
* 훈련 데이터를 표준화한다.

**2. 훈련 모델 선택**

* Logistic Regression, K Nearest Neighbor, Decision Tree, Random Forest, Support Vector Machine 중 가장 성능이 좋은 알고리즘을 선택한다.

**3. Feature 선택**

* 모델을 학습할 때 사용할 feature를 선택한다.
* 모델의 성능을 높일 수 있는 feature를 선별해야 한다.

**4. 하이퍼 파라미터 튜닝**

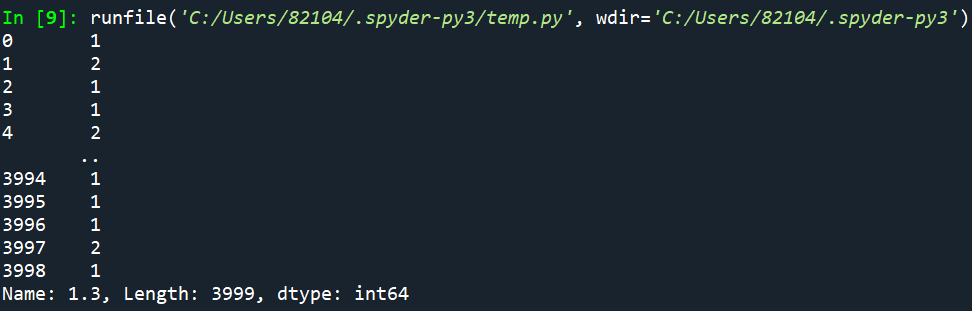
* 하이퍼 파라미터 값에 따라 모델의 성능이 달라지므로 최적의 하이퍼 파라미터 값을 찾아준다.
* 많은 시간이 소요될 수 있으므로 적절한 횟수를 테스트하여 하이퍼 파라미터 값을 찾는다.

**5. 모델 평가**

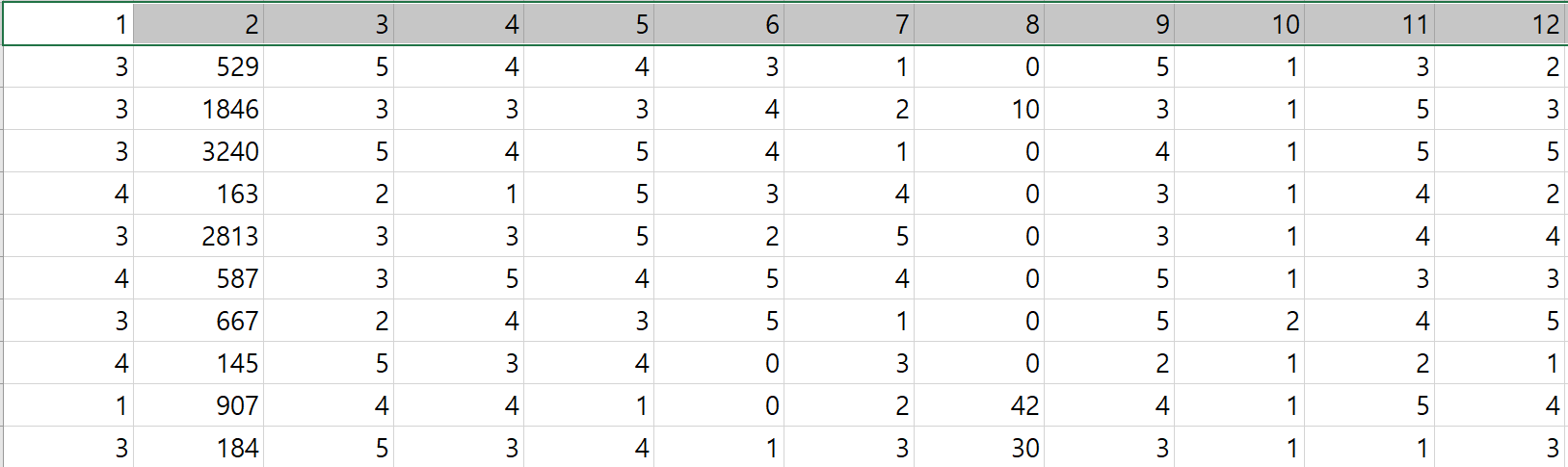
* 모델을 학습시키고 성능을 평가한다.

**■ 개발 과정**

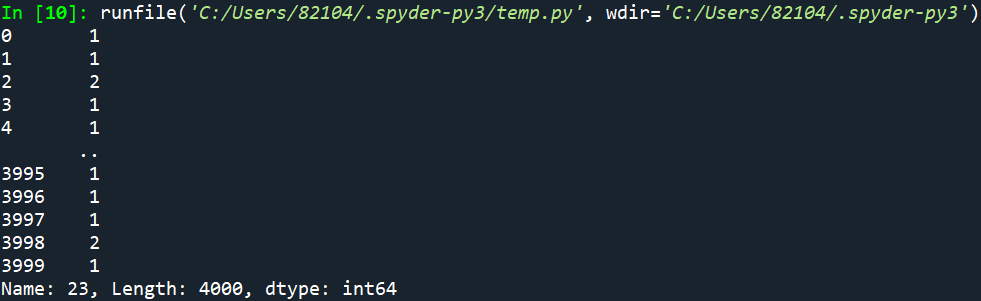
**1. 데이터 셋 준비**



train\_open.csv 파일이 2행부터 읽혀서 1행에 다음과 같이 인덱스를 주었다.

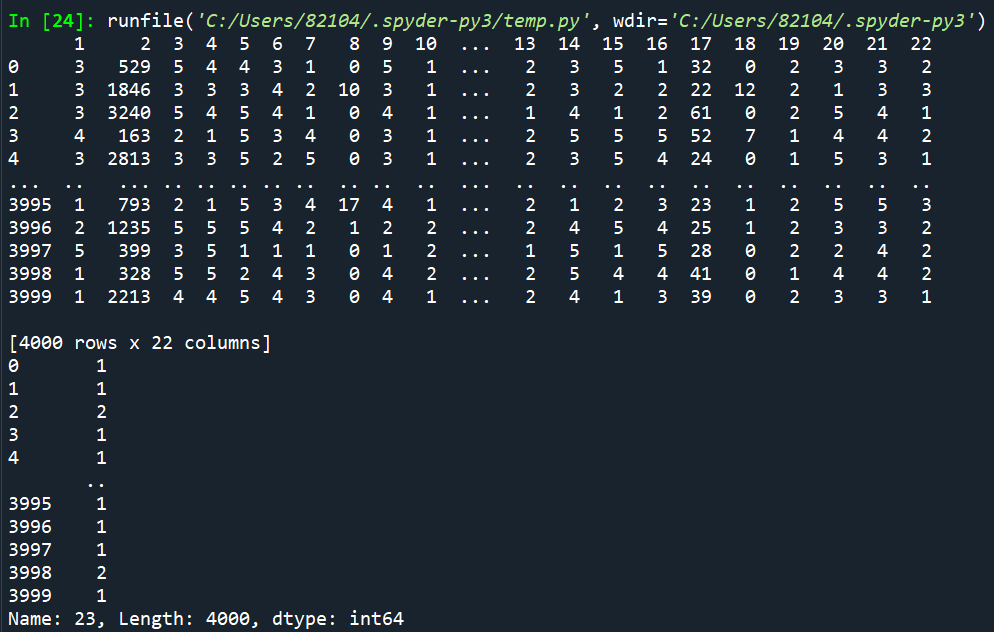


총 4000행의 데이터가 읽힌다.



불러온 trainset을 train\_x와 train\_y로 나누었다.

|  |
| --- |
| import pandas as pd  import numpy as np  trainset = pd.read\_csv('C:/Users/82104/Desktop/deeplearning/competition/train\_open.csv')  train\_x = trainset.iloc[:, 0:22]  train\_y = trainset.iloc[:, -1]  print(train\_x)  print(train\_y) |

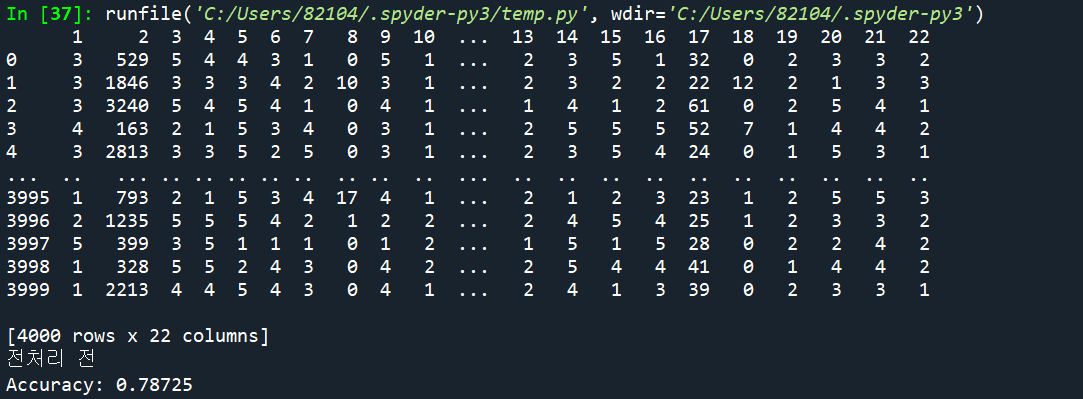


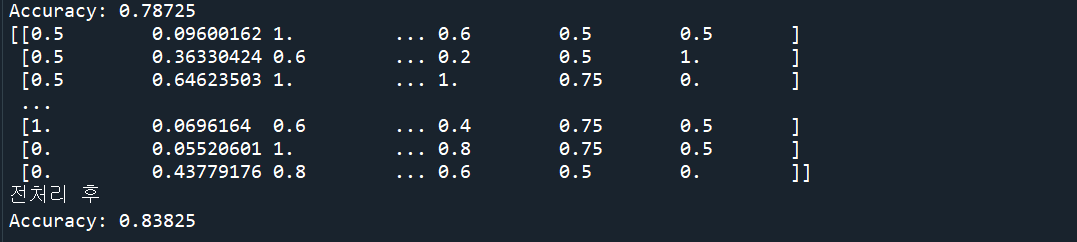
특성마다 범위와 편차의 차이가 있다. 이런 경우를 두고 스케일이 다르다고 한다. 따라서 scaler를 통해 전처리 과정을 거쳐줄 것이다.

처음에는 Standard Scaler를 사용했는데, scaler는 -1부터 1까지의 값으로 표준화해준다. 이후 코드를 실행하는데 마이너스 값을 읽어오는데 문제가 발생했다. 따라서 MinMax Scaler를 사용하여 0부터 1까지의 값으로 데이터를 표준화해주었다.

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.model\_selection import cross\_val\_score  from sklearn.pipeline import make\_pipeline  from sklearn.preprocessing import MinMaxScaler  from sklearn.linear\_model import LogisticRegression  from sklearn.feature\_selection import SelectKBest  from sklearn.feature\_selection import chi2  trainset = pd.read\_csv('C:/Users/82104/Desktop/deeplearning/competition/train\_open.csv')  train\_x = trainset.iloc[:, 0:22]  train\_y = trainset.iloc[:, -1]  scaler = MinMaxScaler()  model = LogisticRegression(solver='saga', max\_iter=10000)  scores = cross\_val\_score(model, train\_x, train\_y)  print(train\_x)  print("전처리 전")  print("Accuracy: "+str(scores.mean()))  train\_x = scaler.fit\_transform(train\_x)  scores = cross\_val\_score(model, train\_x, train\_y)  print(train\_x)  print("전처리 후")  print("Accuracy: "+str(scores.mean())) |

표준화 전처리 전과 후의 accuracy 결과이다. 전처리 후 accuracy가 약 0.05 향상되었다.

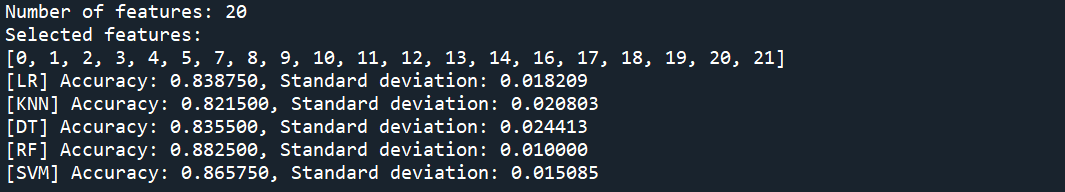




**2. 훈련 모델 선택**

Logistic Regression, K Nearest Neighbor, Decision Tree, Random Forest, Support Vecotr Machine 중 가장 성능이 좋은 알고리즘을 선택할 것이다. KFold로 데이터를 분할한 후 cross validation을 시행한다.

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.model\_selection import cross\_val\_score  from sklearn.pipeline import make\_pipeline  from sklearn.preprocessing import MinMaxScaler  from sklearn.linear\_model import LogisticRegression  from sklearn.tree import DecisionTreeClassifier  from sklearn.neighbors import KNeighborsClassifier  from sklearn.ensemble import RandomForestClassifier  from sklearn.svm import SVC  from sklearn import model\_selection  from sklearn.feature\_selection import RFE  trainset = pd.read\_csv('C:/Users/82104/Desktop/deeplearning/competition/train\_open.csv')  train\_x = trainset.iloc[:, 0:22]  train\_y = trainset.iloc[:, -1]  scaler = MinMaxScaler()  model = LogisticRegression(solver='saga', max\_iter=10000)  train\_x = scaler.fit\_transform(train\_x)  rfe = RFE(model, n\_features\_to\_select = 20)  fit = rfe.fit(train\_x, train\_y)  print("Number of features: "+str(fit.n\_features\_))  temp = fit.support\_.tolist()  fs = [idx for idx, x in enumerate(temp) if x==True]  print("Selected features:")  print(fs)  train\_x = train\_x[:, fs]  models = []  models.append(('LR', LogisticRegression(solver='saga', max\_iter=10000)))  models.append(('KNN', KNeighborsClassifier()))  models.append(('DT', DecisionTreeClassifier()))  models.append(('RF', RandomForestClassifier()))  models.append(('SVM', SVC()))  results = []  names = []  for name, model in models:  kfold = model\_selection.KFold(n\_splits=10, random\_state=42, shuffle=True)  cv\_results = model\_selection.cross\_val\_score(model, train\_x, train\_y, cv=kfold, scoring='accuracy')  results.append(cv\_results)  names.append(name)  msg = "[%s] Accuracy: %f, Standard deviation: %f" % (name, cv\_results.mean(), cv\_results.std())  print(msg) |



Random Forest 모델의 정확도가 가장 높으므로 이 알고리즘을 선택할 것이다.

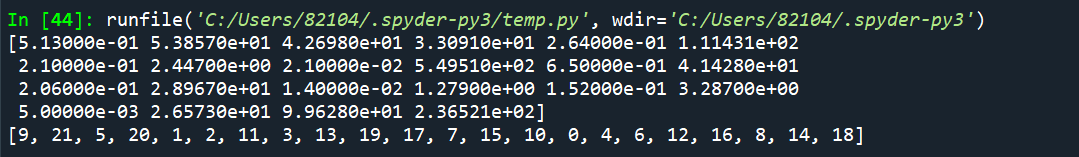
**3. Feature 선택**

처음에 Logistic Regression 모델로 Feature를 선택하고 Random Forest에 그대로 적용했는데, 문득 모델 알고리즘이 달라지면 feature selection에 따른 결과도 달라지지 않을까 하는 생각이 들어 Random Forest 모델으로 다시 feature selection을 진행하였다. 예상대로 결과가 달라졌다.

feature selection은 Filter method, Backward elimination, 두 가지 평가방식을 사용하여 진행했다. Logistic Regression 모델에서 feature selection을 진행하며 Filter method보다 Backward elimination을 사용했을 때 feature selection의 정확도가 높다는 것을 확인하고, Random Forest 모델에서는 진행할 땐 Backward elimination을 통한 feature selection만을 진행했다.

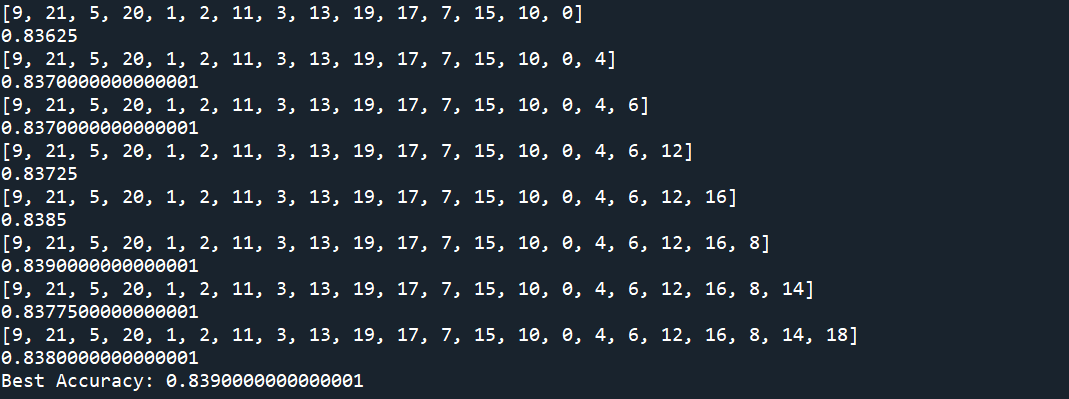
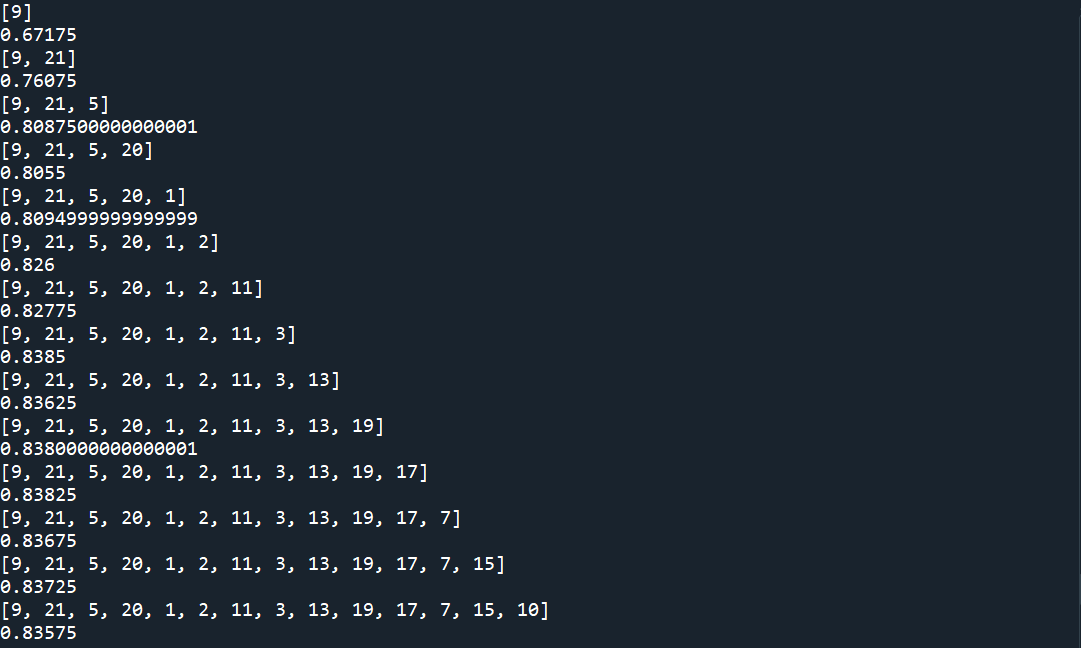
Filter method는 각 feature가 독립적이라는 가정 하에서 feature의 중요도를 평가하는 방법이다. SelectKBest 모듈을 통해 각 feature의 평가점수를 얻을 수 있다.

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.model\_selection import cross\_val\_score  from sklearn.pipeline import make\_pipeline  from sklearn.preprocessing import MinMaxScaler  from sklearn.linear\_model import LogisticRegression  from sklearn.feature\_selection import SelectKBest  from sklearn.feature\_selection import chi2  trainset = pd.read\_csv('C:/Users/82104/Desktop/deeplearning/competition/train\_open.csv')  train\_x = trainset.iloc[:, 0:22]  train\_y = trainset.iloc[:, -1]  scaler = MinMaxScaler()  train\_x = scaler.fit\_transform(train\_x)  test = SelectKBest(score\_func=chi2, k=train\_x.shape[1])  fit = test.fit(train\_x, train\_y)  print(np.round(fit.scores\_, 3))  f\_order = np.argsort(-fit.scores\_)  print(f\_order.tolist()) |



중요도에 따라 feature 조합을 만들어 모델의 성능을 테스트하였다.

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.model\_selection import cross\_val\_score  from sklearn.pipeline import make\_pipeline  from sklearn.preprocessing import MinMaxScaler  from sklearn.linear\_model import LogisticRegression  from sklearn.feature\_selection import SelectKBest  from sklearn.feature\_selection import chi2  trainset = pd.read\_csv('C:/Users/82104/Desktop/deeplearning/competition/train\_open.csv')  train\_x = trainset.iloc[:, 0:22]  train\_y = trainset.iloc[:, -1]  scaler = MinMaxScaler()  model = LogisticRegression(solver='saga', max\_iter=10000)  train\_x = scaler.fit\_transform(train\_x)  test = SelectKBest(score\_func=chi2, k=train\_x.shape[1])  fit = test.fit(train\_x, train\_y)  print(np.round(fit.scores\_, 5))  f\_order = np.argsort(-fit.scores\_)  print(f\_order.tolist())  acc\_selected = np.zeros(train\_x.shape[1] + 1)  for i in range(1, train\_x.shape[1] + 1):  fs = f\_order[0:i].tolist()  print(fs)  train\_x\_selected = train\_x[:, fs]  scores = cross\_val\_score(model, train\_x\_selected, train\_y, cv = 10)  print(scores.mean())  acc\_selected[i] = scores.mean()  print("Best Accuracy: "+str(acc\_selected.max())) |

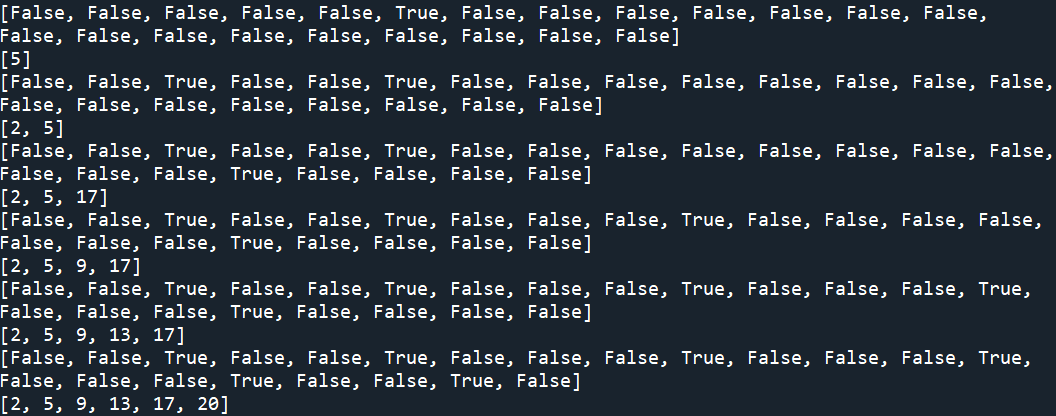


최고 accuracy는 약 0.84이다. 14, 18 feature가 들어가니 accuracy가 하락하는 것을 볼 수 있다.

Backward elimination을 사용한 Feature selection을 진행했다.

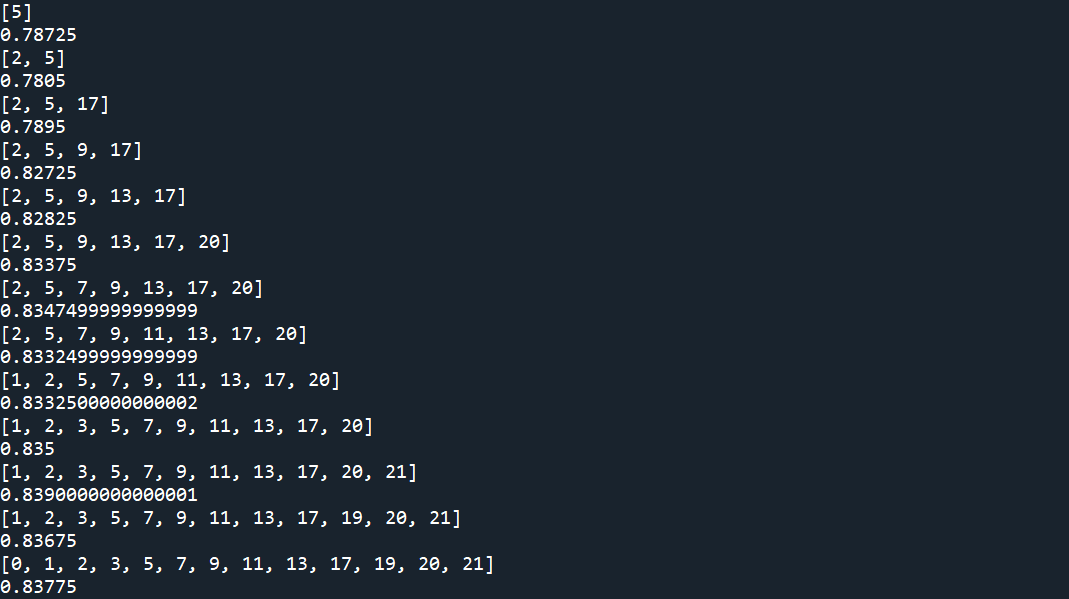
|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.model\_selection import cross\_val\_score  from sklearn.pipeline import make\_pipeline  from sklearn.preprocessing import MinMaxScaler  from sklearn.linear\_model import LogisticRegression  from sklearn.feature\_selection import SelectKBest  from sklearn.feature\_selection import chi2  from sklearn.feature\_selection import RFE  trainset = pd.read\_csv('C:/Users/82104/Desktop/deeplearning/competition/train\_open.csv')  train\_x = trainset.iloc[:, 0:22]  train\_y = trainset.iloc[:, -1]  scaler = MinMaxScaler()  model = LogisticRegression(solver='saga', max\_iter=10000)  train\_x = scaler.fit\_transform(train\_x)  acc\_selected = np.zeros(train\_x.shape[1] + 1)  for i in range(1, train\_x.shape[1] + 1):  rfe = RFE(model, n\_features\_to\_select = i)  fit = rfe.fit(train\_x, train\_y)  print(fit.support\_.tolist())  temp = fit.support\_.tolist()  fs = [idx for idx, x in enumerate(temp) if x==True]  print(fs) |

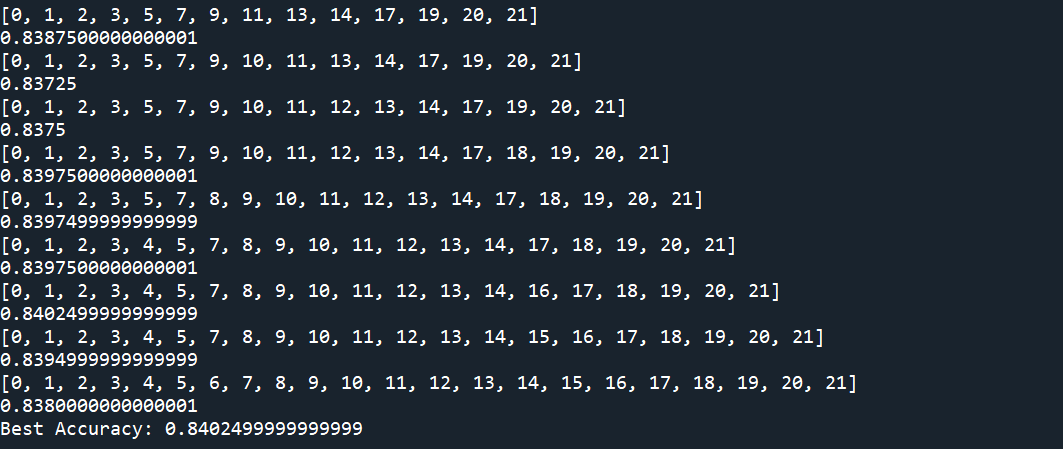
for문을 통해 몇 개의 feature를 선택할 것인지 설정하고 반복하였다. 다음과 같이 True와 False 값으로 선정된 feature가 무엇인지 구분이 된다. True와 False로 이루어진 배열에서 인덱스를 추출했다.



|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.model\_selection import cross\_val\_score  from sklearn.pipeline import make\_pipeline  from sklearn.preprocessing import MinMaxScaler  from sklearn.linear\_model import LogisticRegression  from sklearn.feature\_selection import SelectKBest  from sklearn.feature\_selection import chi2  from sklearn.feature\_selection import RFE  trainset = pd.read\_csv('C:/Users/82104/Desktop/deeplearning/competition/train\_open.csv')  train\_x = trainset.iloc[:, 0:22]  train\_y = trainset.iloc[:, -1]  scaler = MinMaxScaler()  model = LogisticRegression(solver='saga', max\_iter=10000)  train\_x = scaler.fit\_transform(train\_x)  acc\_selected = np.zeros(train\_x.shape[1] + 1)  for i in range(1, train\_x.shape[1] + 1):  rfe = RFE(model, n\_features\_to\_select = i)  fit = rfe.fit(train\_x, train\_y)  temp = fit.support\_.tolist()  fs = [idx for idx, x in enumerate(temp) if x==True]  print(fs)  train\_x\_selected = train\_x[:, fs]  scores = cross\_val\_score(model, train\_x\_selected, train\_y, cv = 10)  print(scores.mean())  acc\_selected[i] = scores.mean()  print("Best Accuracy: "+str(acc\_selected.max())) |

선택된 feature 조합으로 accuracy를 측정한 결과이다.



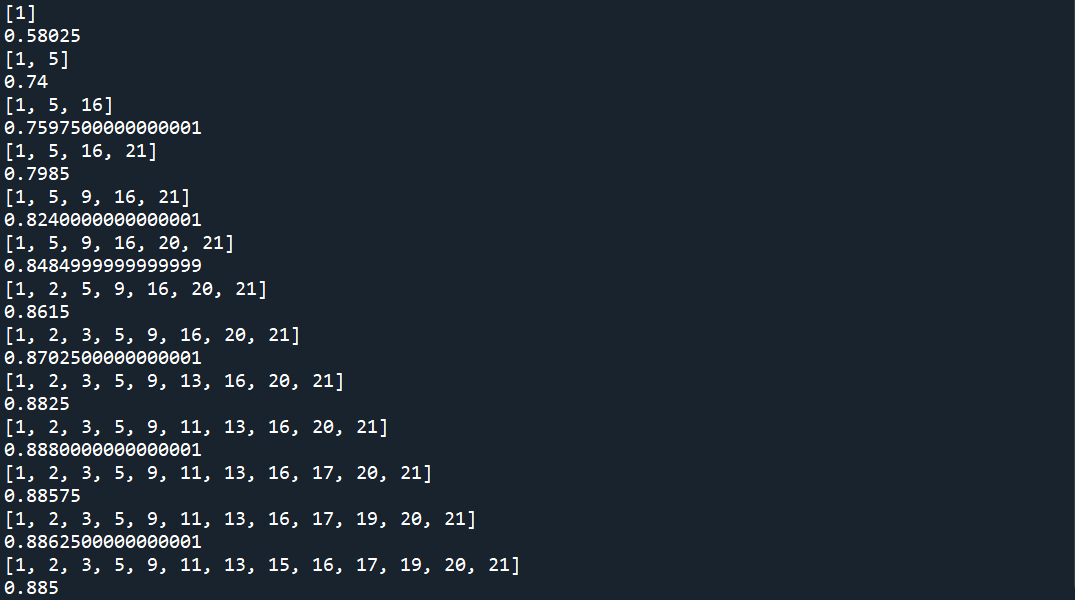


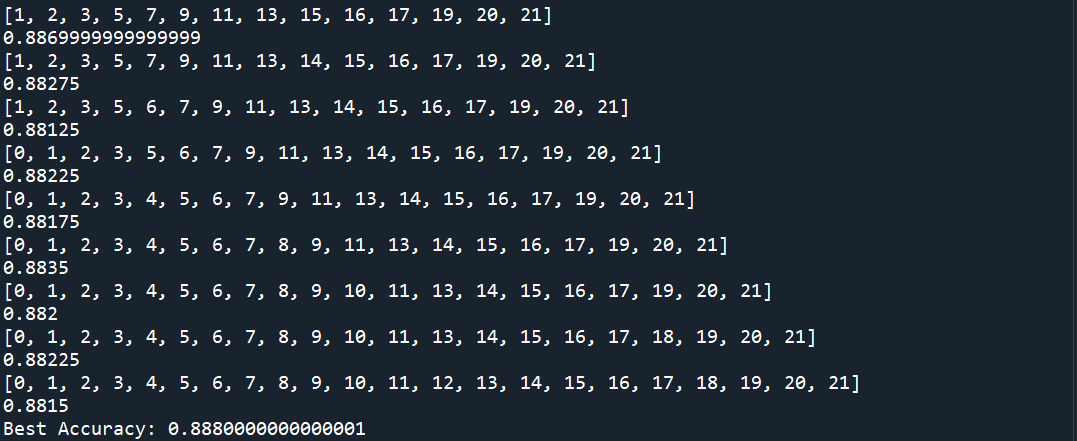
[0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18, 19, 20, 21] 일 때 0.84로, 6, 15를 제외했을 때 accuracy 가장 높은 것을 볼 수 있다. 몇 개의 feature를 가져와야 6, 15를 제외하는지 확인하고 적용시킨다.

|  |
| --- |
| print("Number of features: "+str(fit.n\_features\_)) |

Random Forest 모델에서 Backward elimination을 통한 feature selection을 진행했다.

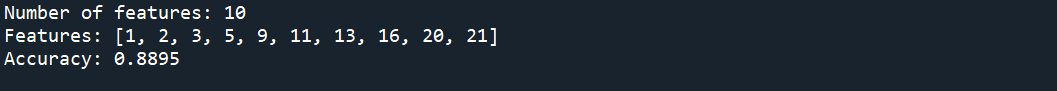
|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.model\_selection import train\_test\_split  from sklearn.model\_selection import cross\_val\_score  from sklearn.pipeline import make\_pipeline  from sklearn.preprocessing import MinMaxScaler  from sklearn.linear\_model import LogisticRegression  from sklearn.tree import DecisionTreeClassifier  from sklearn.neighbors import KNeighborsClassifier  from sklearn.ensemble import RandomForestClassifier  from sklearn.svm import SVC  from sklearn import model\_selection  from sklearn.feature\_selection import RFE  trainset = pd.read\_csv('C:/Users/82104/Desktop/deeplearning/competition/train\_open.csv')  train\_x = trainset.iloc[:, 0:22]  train\_y = trainset.iloc[:, -1]  scaler = MinMaxScaler()  model = RandomForestClassifier()  train\_x = scaler.fit\_transform(train\_x)  acc\_selected = np.zeros(train\_x.shape[1] + 1)  for i in range(1, train\_x.shape[1] + 1):  rfe = RFE(model, n\_features\_to\_select = i)  fit = rfe.fit(train\_x, train\_y)  temp = fit.support\_.tolist()  fs = [idx for idx, x in enumerate(temp) if x==True]  print(fs)  train\_x\_selected = train\_x[:, fs]  scores = cross\_val\_score(model, train\_x\_selected, train\_y, cv = 10)  print(scores.mean())  acc\_selected[i] = scores.mean()  print("Best Accuracy: "+str(acc\_selected.max())) |





[1, 2, 3, 5, 9, 11, 13, 16, 20, 21] 일 때, 0.89로 가장 높은 accuracy가 도출되었다. 0, 4, 6, 7, 8, 10, 12, 14, 15, 17, 18, 19번 feature가 제외되었다.

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.model\_selection import train\_test\_split  from sklearn.model\_selection import cross\_val\_score  from sklearn.pipeline import make\_pipeline  from sklearn.preprocessing import MinMaxScaler  from sklearn.linear\_model import LogisticRegression  from sklearn.tree import DecisionTreeClassifier  from sklearn.neighbors import KNeighborsClassifier  from sklearn.ensemble import RandomForestClassifier  from sklearn.svm import SVC  from sklearn import model\_selection  from sklearn.feature\_selection import RFE  trainset = pd.read\_csv('C:/Users/82104/Desktop/deeplearning/competition/train\_open.csv')  train\_x = trainset.iloc[:, 0:22]  train\_y = trainset.iloc[:, -1]  scaler = MinMaxScaler()  model = RandomForestClassifier()  train\_x = scaler.fit\_transform(train\_x)  rfe = RFE(model, n\_features\_to\_select = 10)  fit = rfe.fit(train\_x, train\_y)  temp = fit.support\_.tolist()  fs = [idx for idx, x in enumerate(temp) if x==True]  print("Number of features: " +str(fit.n\_features\_))  print("Features: ", end="")  print(fs)  train\_x\_selected = train\_x[:, fs]  scores = cross\_val\_score(model, train\_x\_selected, train\_y, cv = 10)  print("Accuracy: "+str(scores.mean())) |

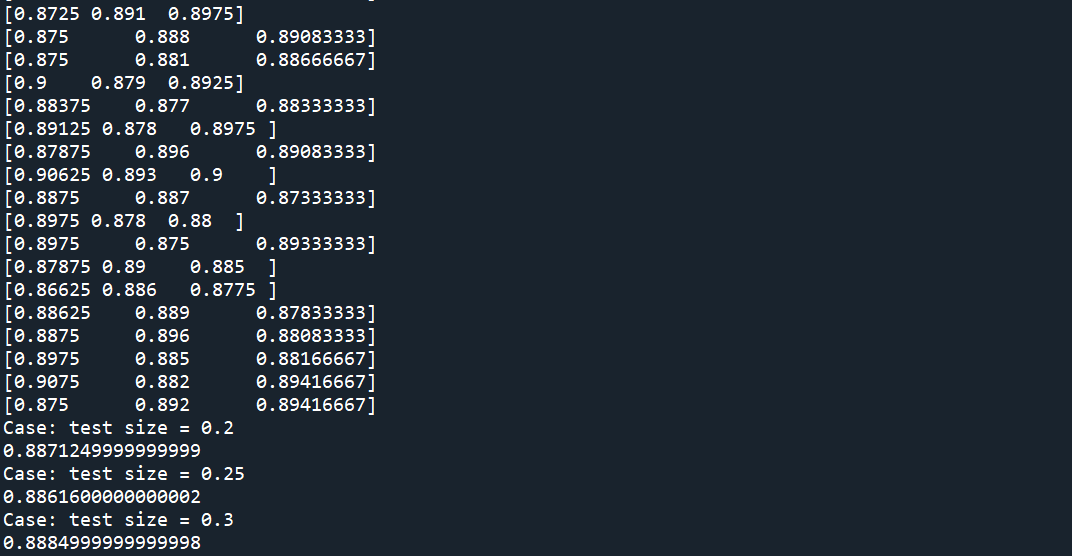


선정된 10개의 feature를 확인했다.

**\* test set size 선택**

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.model\_selection import train\_test\_split  from sklearn.model\_selection import cross\_val\_score  from sklearn.pipeline import make\_pipeline  from sklearn.preprocessing import MinMaxScaler  from sklearn.linear\_model import LogisticRegression  from sklearn.tree import DecisionTreeClassifier  from sklearn.neighbors import KNeighborsClassifier  from sklearn.ensemble import RandomForestClassifier  from sklearn.svm import SVC  from sklearn import model\_selection  from sklearn.feature\_selection import RFE  trainset = pd.read\_csv('C:/Users/82104/Desktop/deeplearning/competition/train\_open.csv')  train\_x = trainset.iloc[:, 0:22]  train\_y = trainset.iloc[:, -1]  scaler = MinMaxScaler()  model = RandomForestClassifier(random\_state=42)  train\_x = scaler.fit\_transform(train\_x)  rfe = RFE(model, n\_features\_to\_select = 10)  fit = rfe.fit(train\_x, train\_y)  temp = fit.support\_.tolist()  fs = [idx for idx, x in enumerate(temp) if x==True]  print("Number of features: " +str(fit.n\_features\_))  print("Features: ", end="")  print(fs)  train\_x = train\_x[:, fs]  size = [0.2, 0.25, 0.3]  train\_x\_all = train\_x  train\_y\_all = train\_y  temp = np.zeros([50, 3])  for i in range(50):  for j in range(3):  train\_x, test\_x, train\_y, test\_y = train\_test\_split(train\_x\_all, train\_y\_all, test\_size=size[j])  model.fit(train\_x, train\_y)  accuracy = model.score(test\_x, test\_y)  temp[i][j] = accuracy  print(temp[i])  avg = np.mean(temp, 0).tolist()  for idx, i in enumerate(avg):  print("Case: test size = "+str(size[idx]))  print(i) |

train set 과 test set이 어떻게 나누어지냐에 따라서 다른 결과를 보인다. 0.01 차이는 결과에 큰 영향이 없을 것 같으나 가장 accuracy 값이 좋게 나오는 0.3을 test set 사이즈의 비율로 선택했다.

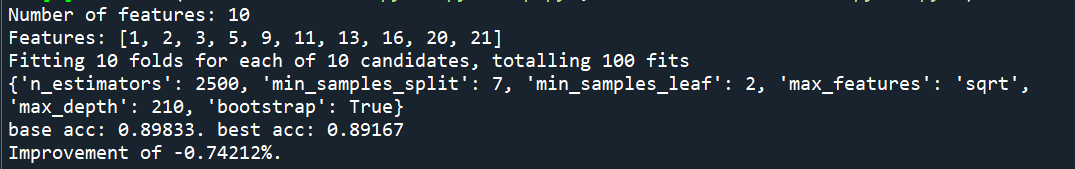


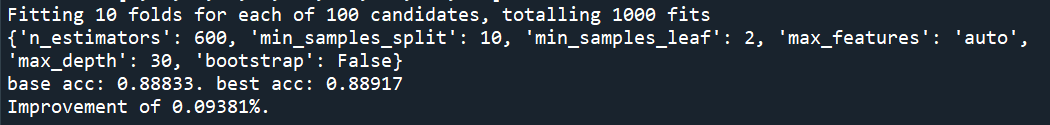
**4. Hyper parameter tuning**

RandomizedSearchCV와 GridSearchCV를 통해 하이퍼 파라미터 튜닝을 진행했다. 이 부분은 수행시간이 굉장히 오래걸려서 Spyder로 진행 중이던 개발을 Colab으로 옮겼다. 수행시간에 큰 차이가 있지는 않았지만 Colab은 진행경과를 표시해주어서 어느정도 수행시간을 예측할 수 있었기 때문이다.

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.model\_selection import train\_test\_split  from sklearn.model\_selection import cross\_val\_score  from sklearn.pipeline import make\_pipeline  from sklearn.preprocessing import MinMaxScaler  from sklearn.linear\_model import LogisticRegression  from sklearn.tree import DecisionTreeClassifier  from sklearn.neighbors import KNeighborsClassifier  from sklearn.ensemble import RandomForestClassifier  from sklearn.svm import SVC  from sklearn import model\_selection  from sklearn.model\_selection import RandomizedSearchCV  from sklearn.feature\_selection import RFE  trainset = pd.read\_csv('C:/Users/82104/Desktop/deeplearning/competition/train\_open.csv')  train\_x = trainset.iloc[:, 0:22]  train\_y = trainset.iloc[:, -1]  scaler = MinMaxScaler()  model = RandomForestClassifier(random\_state=42)  train\_x = scaler.fit\_transform(train\_x)  rfe = RFE(model, n\_features\_to\_select = 10)  fit = rfe.fit(train\_x, train\_y)  temp = fit.support\_.tolist()  fs = [idx for idx, x in enumerate(temp) if x==True]  print("Number of features: " +str(fit.n\_features\_))  print("Features: ", end="")  print(fs)  train\_x = train\_x[:, fs]  train\_x\_all = train\_x  train\_y\_all = train\_y  train\_x, test\_x, train\_y, test\_y = train\_test\_split(train\_x\_all, train\_y\_all, test\_size=0.3)  n\_estimators = [int(x) for x in np.linspace(start = 100, stop = 3000, num = 30)]  max\_features = ['auto', 'sqrt']  max\_depth = [int(x) for x in np.linspace(10, 300, num = 30)]  max\_depth.append(None)  min\_samples\_split = [2, 3, 5, 7, 8, 10]  min\_samples\_leaf = [1, 2, 3, 4, 5]  bootstrap = [True, False]  random\_grid = {'n\_estimators': n\_estimators,  'max\_features': max\_features,  'max\_depth': max\_depth,  'min\_samples\_split': min\_samples\_split,  'min\_samples\_leaf': min\_samples\_leaf,  'bootstrap': bootstrap  }  base\_model = RandomForestClassifier(random\_state=42)  base\_model.fit(train\_x, train\_y)  base\_accuracy = base\_model.score(test\_x, test\_y)  rf\_random = RandomizedSearchCV(estimator = base\_model, param\_distributions = random\_grid, n\_iter = 100, cv = 10, verbose=2, random\_state=42, n\_jobs = -1)  rf\_random.fit(train\_x, train\_y)  print(rf\_random.best\_params\_)  best\_random\_model = rf\_random.best\_estimator\_  best\_random\_accuracy = best\_random\_model.score(test\_x, test\_y)  print('base acc: {0:0.5f}. best acc: {1:0.5f}'.format(base\_accuracy, best\_random\_accuracy))  print('Improvement of {:0.5f}%.'.format(100 \* (best\_random\_accuracy - base\_accuracy) / base\_accuracy)) |

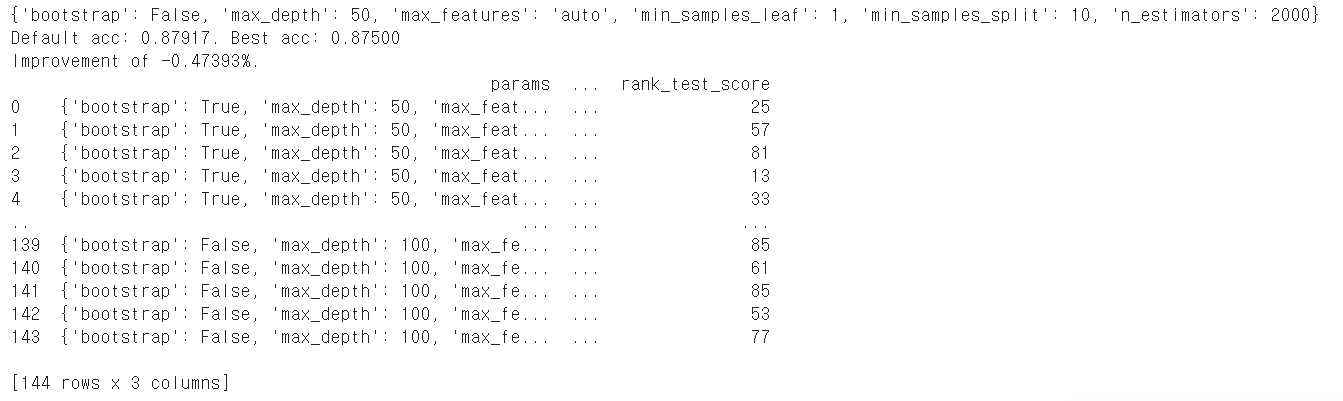
먼저 RandomizeSearchCV를 통해 하이퍼 파라미터 튜닝을 진행했다. 처음 100번의 fit을 진행했을 때는 오히려 기본 accuracy보다 결과가 좋지 않게 나왔다. 그래서 1000번의 fit을 진행했더니 0.09% accuracy가 향상했다.

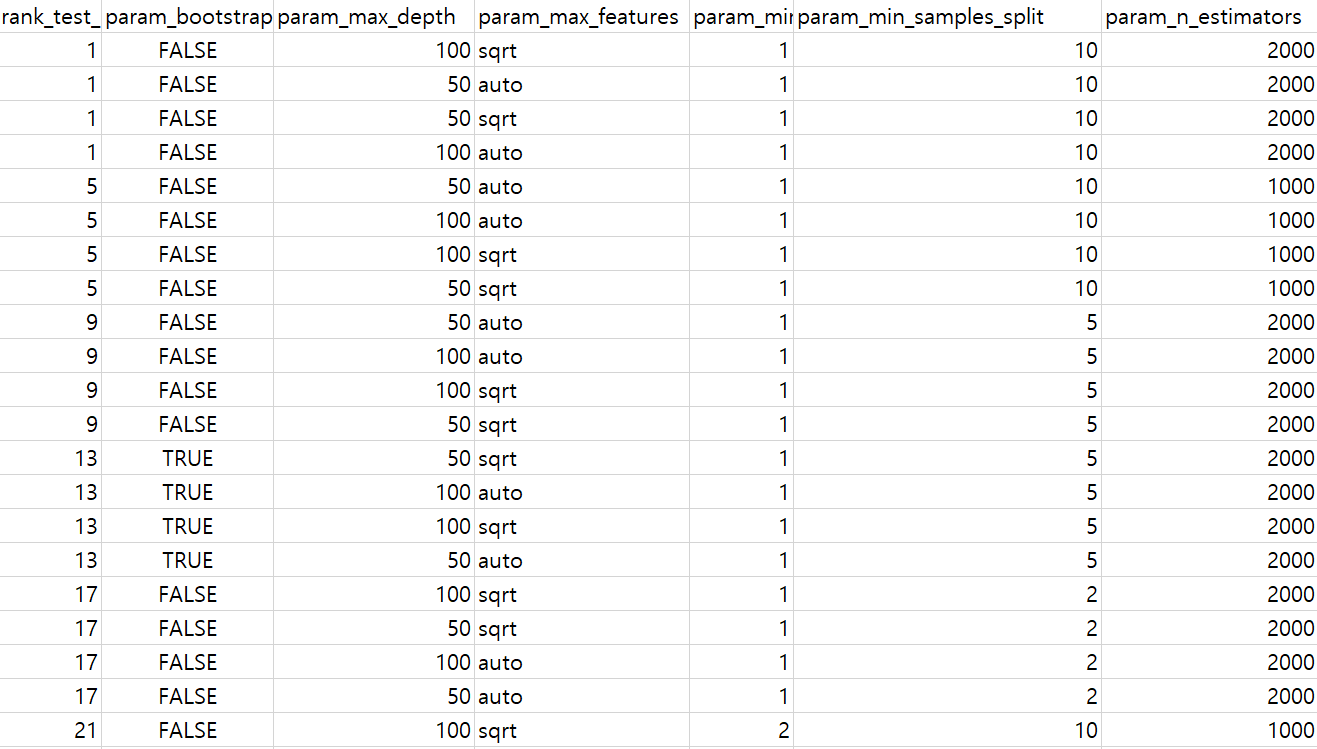




매개변수 값을 랜덤하게 추출하는 범위가 너무 넓어서 accuracy의 향상이 좋지 않은 것이라고 추측되어 어느정도 범위를 조정한 후 다시 랜덤하게 추출하기로 결정했다. GridSearchCV를 통해 하이퍼 파라미터 튜닝을 진행했다. 하이퍼 파라미터에 따른 accuracy 순위를 오름차순으로 정렬한 뒤 csv로 받았다.

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.model\_selection import train\_test\_split  from sklearn.model\_selection import cross\_val\_score  from sklearn.pipeline import make\_pipeline  from sklearn.preprocessing import MinMaxScaler  from sklearn.linear\_model import LogisticRegression  from sklearn.tree import DecisionTreeClassifier  from sklearn.neighbors import KNeighborsClassifier  from sklearn.ensemble import RandomForestClassifier  from sklearn.svm import SVC  from sklearn import model\_selection  from sklearn.model\_selection import RandomizedSearchCV  from sklearn.model\_selection import GridSearchCV    from sklearn.feature\_selection import RFE  trainset = pd.read\_csv('/content/drive/MyDrive/Colab Notebooks/train\_open.csv')  train\_x = trainset.iloc[:, 0:22]  train\_y = trainset.iloc[:, -1]  scaler = MinMaxScaler()  model = RandomForestClassifier(random\_state=42)  train\_x = scaler.fit\_transform(train\_x)  rfe = RFE(model, n\_features\_to\_select = 10)  fit = rfe.fit(train\_x, train\_y)  temp = fit.support\_.tolist()  fs = [idx for idx, x in enumerate(temp) if x==True]  print("Number of features: " +str(fit.n\_features\_))  print("Features: ", end="")  print(fs)  train\_x = train\_x[:, fs]  train\_x\_all = train\_x  train\_y\_all = train\_y  train\_x, test\_x, train\_y, test\_y = train\_test\_split(train\_x\_all, train\_y\_all, test\_size=0.3)  n\_estimators = [int(x) for x in np.linspace(start = 100, stop = 3000, num = 30)]  max\_features = ['auto', 'sqrt']  max\_depth = [int(x) for x in np.linspace(10, 300, num = 30)]  max\_depth.append(None)  min\_samples\_split = [2, 3, 5, 7, 8, 10]  min\_samples\_leaf = [1, 2, 3, 4, 5]  bootstrap = [True, False]  default\_model = RandomForestClassifier(random\_state=42)  default\_model.fit(train\_x, train\_y)  default\_accuracy = default\_model.score(test\_x, test\_y)  param\_grid = {      'n\_estimators': [1000, 2000],      'max\_depth': [50, 100],      'min\_samples\_split': [2, 5, 10],      'max\_features': ['auto', 'sqrt'],      'min\_samples\_leaf' : [1, 2, 3],      'bootstrap':[True, False]  }  grid\_search = GridSearchCV(estimator=default\_model, param\_grid=param\_grid, cv=5, n\_jobs= -1, scoring='f1\_weighted', verbose=10)  grid\_search.fit(train\_x, train\_y)  print(grid\_search.best\_params\_)  best\_random\_model = grid\_search.best\_estimator\_  best\_random\_accuracy = best\_random\_model.score(test\_x, test\_y)  print('Default acc: {0:0.5f}. Best acc: {1:0.5f}'.format(default\_accuracy, best\_random\_accuracy))  print('Improvement of {:0.5f}%.'.format(100 \* (best\_random\_accuracy - default\_accuracy) / default\_accuracy))  scores = pd.DataFrame(grid\_search.cv\_results\_)  print(scores[['params','mean\_test\_score', 'rank\_test\_score']])  scores.sort\_values(by=['rank\_test\_score'], axis=0, inplace=True)  scores.to\_csv('sorted\_scores.csv') |

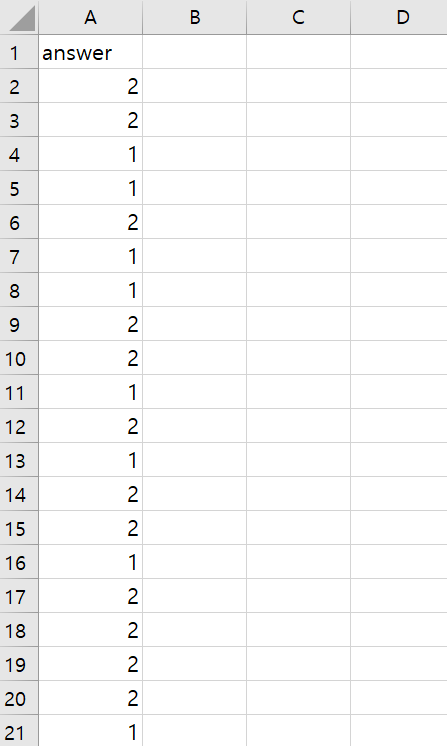




**5. 모델 평가**

주어진 test set으로 모델 평가를 진행하면서 하이퍼 파라미터 튜닝을 반복했다. 예측 결과를 csv로 받았다.

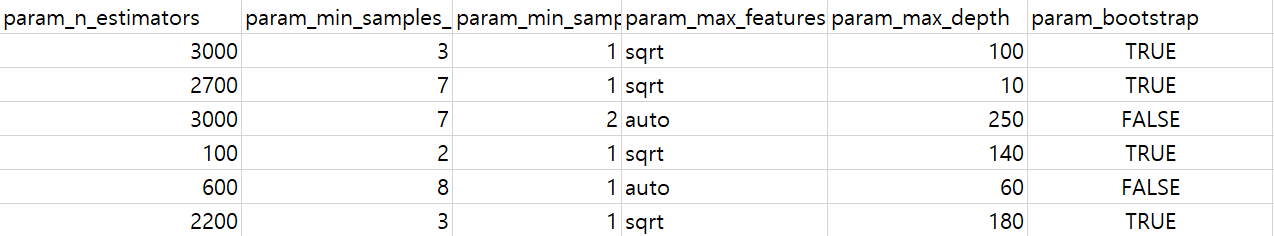
|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.model\_selection import train\_test\_split  from sklearn.model\_selection import cross\_val\_score  from sklearn.pipeline import make\_pipeline  from sklearn.preprocessing import MinMaxScaler  from sklearn.linear\_model import LogisticRegression  from sklearn.tree import DecisionTreeClassifier  from sklearn.neighbors import KNeighborsClassifier  from sklearn.ensemble import RandomForestClassifier  from sklearn.svm import SVC  from sklearn import model\_selection  from sklearn.model\_selection import RandomizedSearchCV  from sklearn.model\_selection import GridSearchCV    from sklearn.feature\_selection import RFE  trainset = pd.read\_csv('/content/drive/MyDrive/Colab Notebooks/train\_open.csv')  train\_x = trainset.iloc[:, 0:22]  train\_y = trainset.iloc[:, -1]  testset = pd.read\_csv('/content/drive/MyDrive/Colab Notebooks/test\_open.csv')  print(testset)  scaler = MinMaxScaler()  scaler.fit(train\_x)  train\_x = scaler.transform(train\_x)  testset = scaler.transform(testset)  print(testset)  model = RandomForestClassifier(random\_state=42)  rfe = RFE(model, n\_features\_to\_select = 10)  fit = rfe.fit(train\_x, train\_y)  temp = fit.support\_.tolist()  fs = [idx for idx, x in enumerate(temp) if x==True]  print("Number of features: " +str(fit.n\_features\_))  print("Features: ", end="")  print(fs)  # print(train\_x)  train\_x = train\_x[:, fs]  # print(train\_x, test\_x.iloc[:, fs])  testset = testset[:, fs]  print(train\_x, train\_y)  model = RandomForestClassifier(bootstrap=False, max\_depth=100, max\_features='sqrt', min\_samples\_split=10, min\_samples\_leaf=1, n\_estimators = 2000, random\_state=1234)  model.fit(train\_x, train\_y)  def save\_submission2(pred):   file\_name = "32191197\_김채은.csv"   np.savetxt(file\_name, pred, fmt="%s", delimiter=",")  pred = model.predict(testset)  submission = pd.DataFrame(pred)  save\_submission2(submission) |



위와 같은 결과가 나왔다.

RandomizedSearchCV를 통한 하이퍼 파라미터 튜닝을 다시 진행했다.

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.model\_selection import train\_test\_split  from sklearn.model\_selection import cross\_val\_score  from sklearn.pipeline import make\_pipeline  from sklearn.preprocessing import MinMaxScaler  from sklearn.linear\_model import LogisticRegression  from sklearn.tree import DecisionTreeClassifier  from sklearn.neighbors import KNeighborsClassifier  from sklearn.ensemble import RandomForestClassifier  from sklearn.svm import SVC  from sklearn import model\_selection  from sklearn.model\_selection import RandomizedSearchCV  from sklearn.model\_selection import GridSearchCV  from sklearn.feature\_selection import RFE  trainset = pd.read\_csv('/content/drive/MyDrive/Colab Notebooks/train\_open.csv')  train\_x = trainset.iloc[:, 0:22]  train\_y = trainset.iloc[:, -1]  scaler = MinMaxScaler()  model = RandomForestClassifier(random\_state=42)  train\_x = scaler.fit\_transform(train\_x)  rfe = RFE(model, n\_features\_to\_select = 10)  fit = rfe.fit(train\_x, train\_y)  temp = fit.support\_.tolist()  fs = [idx for idx, x in enumerate(temp) if x==True]  print("Number of features: " +str(fit.n\_features\_))  print("Features: ", end="")  print(fs)  train\_x = train\_x[:, fs]  train\_x\_all = train\_x  train\_y\_all = train\_y  train\_x, test\_x, train\_y, test\_y = train\_test\_split(train\_x\_all, train\_y\_all, test\_size=0.3)  n\_estimators = [int(x) for x in np.linspace(start = 100, stop = 3000, num = 30)]  max\_features = ['auto', 'sqrt']  max\_depth = [int(x) for x in np.linspace(10, 300, num = 30)]  max\_depth.append(None)  min\_samples\_split = [2, 3, 5, 7, 8, 10]  min\_samples\_leaf = [1, 2, 3, 4, 5]  bootstrap = [True, False]  random\_grid = {'n\_estimators': n\_estimators,   'max\_features': max\_features,   'max\_depth': max\_depth,   'min\_samples\_split': min\_samples\_split,   'min\_samples\_leaf': min\_samples\_leaf,   'bootstrap': bootstrap  }  default\_model = RandomForestClassifier(random\_state=42)  default\_model.fit(train\_x, train\_y)  default\_accuracy = default\_model.score(test\_x, test\_y)  param\_grid = {      'n\_estimators': [2000, 2500],      'max\_depth':  [int(x) for x in np.linspace(50, 110, num = 11)] ,      'min\_samples\_split': [10, 15],      'max\_features': ['sqrt'],      'min\_samples\_leaf' : [1],      'bootstrap':[False]  }  rf = RandomForestClassifier(random\_state=1234)  rf\_random = RandomizedSearchCV(estimator = rf, param\_distributions = random\_grid, n\_iter = 100,  cv = 10, verbose=2, random\_state=42, n\_jobs = -1)  rf\_random.fit(train\_x, train\_y)  best\_random\_model = rf\_random.best\_estimator\_  best\_random\_accuracy = best\_random\_model.score(test\_x, test\_y)  print('Default acc: {0:0.5f}. Best acc: {1:0.5f}'.format(default\_accuracy, best\_random\_accuracy))  print('Improvement of {:0.5f}%.'.format(100 \* (best\_random\_accuracy - default\_accuracy) / default\_accuracy))  scores = pd.DataFrame(rf\_random.cv\_results\_)  print(scores[['params','mean\_test\_score', 'rank\_test\_score']])  scores.sort\_values(by=['rank\_test\_score'], axis=0, inplace=True)  scores.to\_csv('sorted\_scores.csv') |



하이퍼 파라미터 값에 따른 accuracy를 표로 정리했다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n\_esimators | min\_samples\_split | min\_samples\_leaf | max\_features | max\_depth | bootstrap | accuracy |
| 3000 | 3 | 1 | Sqrt | 100 | True | 0.91 |
| 2700 | 7 | 1 | Sqrt | 10 | True | 0.92 |
| 3000 | 7 | 2 | Auto | 250 | False | 0.91 |
| 100 | 2 | 1 | Sqrt | 140 | True | 0.91 |
| 600 | 8 | 1 | Auto | 60 | False | 0.91 |
| 2200 | 3 | 1 | Sqrt | 180 | True | 0.91 |

하이퍼 파라미터 값이 바뀜에 따라 feature selection에 따른 결과도 달라질 수 있을 것 같았다.

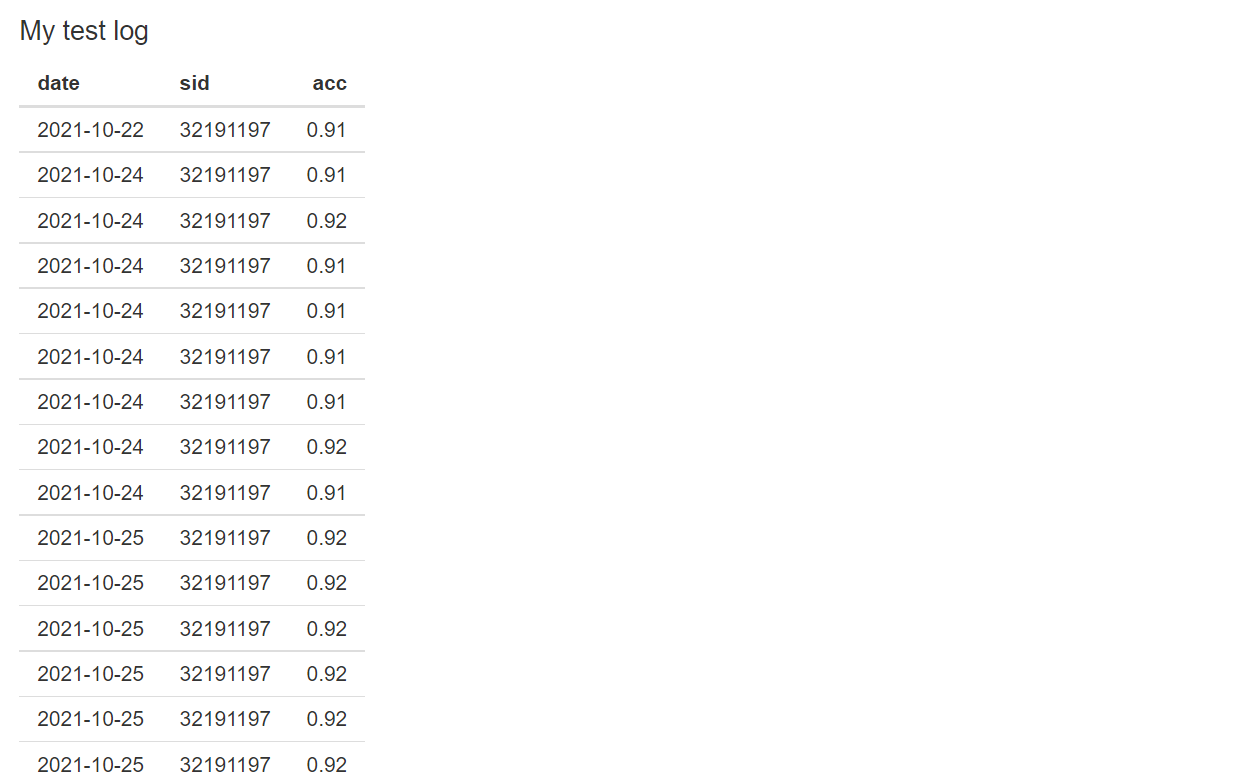
그래서 가장 accuracy가 높았던 하이퍼 파라미터에, 모든 feature를 선택하여 다시 테스트 해보았다.

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.model\_selection import train\_test\_split  from sklearn.model\_selection import cross\_val\_score  from sklearn.pipeline import make\_pipeline  from sklearn.preprocessing import MinMaxScaler  from sklearn.linear\_model import LogisticRegression  from sklearn.tree import DecisionTreeClassifier  from sklearn.neighbors import KNeighborsClassifier  from sklearn.ensemble import RandomForestClassifier  from sklearn.svm import SVC  from sklearn import model\_selection  from sklearn.model\_selection import RandomizedSearchCV  from sklearn.model\_selection import GridSearchCV  from sklearn.feature\_selection import RFE  trainset = pd.read\_csv('/content/drive/MyDrive/Colab Notebooks/train\_open.csv')  train\_x = trainset.iloc[:, 0:22]  train\_y = trainset.iloc[:, -1]  testset = pd.read\_csv('/content/drive/MyDrive/Colab Notebooks/test\_open.csv')  print(testset)  scaler = MinMaxScaler()  scaler.fit(train\_x)  train\_x = scaler.transform(train\_x)  testset = scaler.transform(testset)  print(testset)  model = RandomForestClassifier(random\_state=42)  rfe = RFE(model, n\_features\_to\_select = 22)  fit = rfe.fit(train\_x, train\_y)  temp = fit.support\_.tolist()  fs = [idx for idx, x in enumerate(temp) if x==True]  print("Number of features: " +str(fit.n\_features\_))  print("Features: ", end="")  print(fs)  # print(train\_x)  train\_x = train\_x[:, fs]  # print(train\_x, test\_x.iloc[:, fs])  testset = testset[:, fs]  print(train\_x, train\_y)  model = RandomForestClassifier(bootstrap=True, max\_depth=10, max\_features='sqrt', min\_samples\_split=7, min\_samples\_leaf=1, n\_estimators = 2700, random\_state=1234)  model.fit(train\_x, train\_y)  def save\_submission2(pred):   file\_name = "32191197\_김채은.csv"   np.savetxt(file\_name, pred, fmt="%s", delimiter=",")  pred = model.predict(testset)  submission = pd.DataFrame(pred)  save\_submission2(submission) |

놀랍게도 지금까지의 결과 중 가장 높은 값이 나왔다. 모든 feature를 선택하고 하이퍼 파라미터 튜닝을 다시 진행했다. 다음은 1위부터 4위까지의 테스트 결과이다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n\_estimators | min\_samples\_split | min\_samples\_leaf | max\_features | Max\_depth | bootstrap | accuracy |
| 1400 | 2 | 1 | sqrt | 180 | false | 0.92 |
| 2300 | 3 | 1 | sqrt | 120 | false | 0.92 |
| 600 | 8 | 1 | auto | 60 | false | 0.92 |
| 2100 | 2 | 1 | auto | 60 | false | 0.92 |
| 2500 | 3 | 1 | auto | 170 | false | 0.92 |

모든 feature selection에 대해 하이퍼 파라미터 튜닝을 진행할 시간이 없어서 이대로 마무리 지었다. 다음은 나의 테스트 로그이다.



**■ 경진대회 참여소감**

딥러닝/클라우드를 수강하면서 매주 많은 양의 과제가 제공되었다. 과제를 할 당시에는 지치고 힘들었는데, 경진대회를 위한 모델을 개발하면서 과제를 하면서 실습해보았던 내용을 많이 이용할 수 있었다. 과제를 통해 미리 실습해보지 않았다면 이번 경진대회를 진행하는 데 훨씬 많은 시간이 소요되고 진행하기가 굉장히 어려웠을 것 같다.

처음에는 accuracy가 너무 낮게 나와 막막했는데, 진행 과정을 거쳐가면서 조금씩 조금씩 향상되는 accuracy를 보는 게 큰 재미였던 것 같다. 과제를 진행할수록 어떻게 하면 좋은 예측 결과를 낼 수 있을지, 더 많이 고민하게 되고 열정이 생겼던 것 같다. 특히 마지막 단계에서 feature selection과 hyper parameter tuning을 반복 진행했으면 더 높은 accuracy를 찾을 수 있었을 것 같은데, 시간적인 한계와 업로드 수 제한으로 모두 테스트해보지 못한 것이 아쉽다.

딥러닝 모델을 가지고 accuracy를 테스트해본 경험은 있었지만, feature를 선택하는 것부터 하이퍼 파라미터 튜닝까지 전 과정을 진행해본 것은 이번이 처음이다. 결과가 어떻게 나올지 기다리면서, 오랜만에 가슴을 두근거리게 만드는 개발 과제를 할 수 있어 기뻤다.