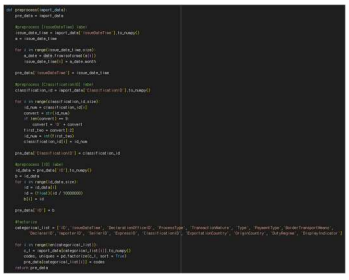
20190349 신지호

Summary report

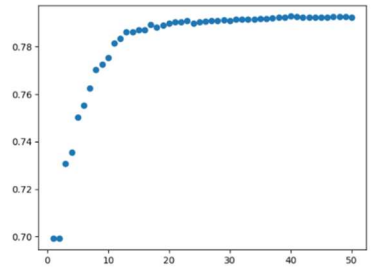
The purpose of the Term project part 3 was to create a model that best predicts whether the test dataset is fake or not, given the train dataset. The dataset given to us had a total of 21 features. A total of 20 features could be used in this task, excluding 'Fake', which is the label to be predicted.

I thought about which algorithm to use first and decided to try the kneighbors algorithm used in part 2 first. As we did in Part 2, I handled 'IssueDateTime' data contain only data corresponding to the month, and 'ClassificationID' data contain only data corresponding to the first two digits. For the 'ID' data, only the first digit was left. Also, 'TaxRate', 'TotalGrossMassMeasure(KG)', and 'AdValoremTaxBaseAmount(Won)' data with real data values were not touched, and the remaining 17 types of data were factorized using the pd.factorize function. At this point, I did not drop specific features because I did not yet know which data had a significant impact on accuracy. This process is implemented as a function called 'preprocess'

After bringing the train.csv data to Colab, I preprocessed it using the preprocess function, divided it into train data and test data by using the train_test_split function of Scikit-Learn, and increased the n_neighbors parameter of the kneighbors algorithm from 1 to 50. After testing with the cross validate function, I plotted the test score according to each n value as a scatter plot function. The result of the test score was lower than 80%.



```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_validate
import matplotlib.pyplot as plt

import_data = pd.read_csv('train.csv')
import_data.head()

pre_data = preprocess(import_data)
analyze_data = pre_data.astype('float')

data_input = analyze_data.drop(labels = 'Fake', axis = 1).to_numpy()
data_target = analyze_data['Fake'].to_numpy()

train_input, test_input, train_target, test_target = train_test_split(data_input, data_target, random_state = 42)

score = []
index = []
for i in range(50):
    kn = KNeighborsClassifier(n_neighbors= i +1, weights = 'distance')
    scores = cross_validate(kn, train_input, train_target, return_train_score = True, n_jobs = -1)

    kn.fit(train_input, train_target)
    index.append(i+1)
    score.append(kn.score(test_input, test_target))

plt.scatter(index, score)
```



I thought about the reasons for these results. First, I thought that the accuracy was low because the real data values were used in the algorithm without normalization. Second, I thought the accuracy was low because I used features that had nothing to do with whether they were fake or not. In order to improve the first cause, the real data data 'TaxRate', TotalGrossMassMeasure(KG)', and 'AdValoremTax BaseAmount(Won)' data were normalized using standardcaler and then used again in the algorithm. However, this did not improve the test score. In order to improve the second cause, I thought about how to select relatively more important characteristics.

While researching about Decision Tree Algorithms, I found that when using Random Forest Algorithm, the importance of each feature is stored numerically in the feature_importances variable. I decided to try this one. Among the 20 features, the last two features (TotalGrossMass Measure(KG)', 'AdValoremTaxBaseAmount(Won)') were found to have higher importance than the other features.

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_jobs = -1)
scores = cross_validate(rf, train_input, train_target, return_train_score = True, n_jobs = -1)
print(np.mean(scores['train_score']), np.mean(scores['test_score']))
rf.fit(train_input, train_target)
print(rf.feature_importances_)
```

```
0.9999911691981632 0.7990815966089722
[0.04822137 0.05379736 0.04498852 0.00197093 0.02195395 0.01883452
 0.02648588 0.02018098 0.08846788 0.09220622 0.08663459 0.02311874
 0.06500522 0.0416295  0.04130315 0.03346183 0.02541549 0.03539557
 0.1142871  0.1166412 ]
```

Before using the kneighbors algorithm again, in order to use the kneighbors algorithm more efficiently, I implemented 'test_data' function that automatically outputs the results of the kneighbors algorithm when the desired data label to use and n_neighbors variable value are specified. The 20 features were assigned numbers in the order shown in the train data as follows.

```
# 'ID' : 0, 'IssueDateTime' : 1, 'DeclarationOfficeID' : 2,
'ProcessType' : 3, 'TransactionNature' : 4, 'Type' : 5
# 'PaymentType' : 6, 'BorderTransportMeans' : 7, 'DeclarerID' : 8,
'ImporterID' : 9, 'SellerID' : 10
# 'ExpressID' : 11, 'ClassificationID' : 12, 'ExportationCountry' : 13,
'OriginCountry' : 14, 'TaxRate' : 15
# 'DutyRegime' : 16, 'DisplayIndicator' : 17,
'TotalGrossMassMeasure(KG)' : 18, 'AdValoremTaxBaseAmount(Won)' : 19,
'Fake' : 20
```

```
def test_data(pre_data, labels, index, lower, upper):
    data = make_data(pre_data, labels, index)
    data_input = data.to_numpy()
    data_target = pre_data['Fake'].to_numpy()

    train_input, test_input, train_target, test_target = train_test_split(data_input, data_target, random_state = 42)

    score = []
    indexs = []

    for i in range(lower, upper):
        kn = KNeighborsClassifier(n_neighbors= i, weights = 'distance')
        scores = cross_validate(kn, train_input, train_target, return_train_score = True, n_jobs = -1)

        kn.fit(train_input, train_target)
        indexs.append(i)
        score.append(kn.score(test_input, test_target))

    print('feartures :', end = '')
    for j in range(len(index)):
        print(labels[index[j]], ' ', end = '')
    print('%n', lower + np.argmax(score), np.max(score))

    plt.scatter(indexs, score)
    return
```

TotalGrossMassMeasure(KG)' and 'AdValoremTax BaseAmount(Won)' characteristics were fixedly included, and other characteristics were changed one by one to obtain test score values when n_neighbors was 400 to 410.

```
test_data(pre_data, labels, [0, 18, 19], 400, 410)
test_data(pre_data, labels, [1, 18, 19], 400, 410)
test_data(pre_data, labels, [2, 18, 19], 400, 410)
test_data(pre_data, labels, [3, 18, 19], 400, 410)
test_data(pre_data, labels, [4, 18, 19], 400, 410)
test_data(pre_data, labels, [5, 18, 19], 400, 410)
test_data(pre_data, labels, [6, 18, 19], 400, 410)
test_data(pre_data, labels, [7, 18, 19], 400, 410)
test_data(pre_data, labels, [8, 18, 19], 400, 410)
test_data(pre_data, labels, [9, 18, 19], 400, 410)
test_data(pre_data, labels, [10, 18, 19], 400, 410)
test_data(pre_data, labels, [11, 18, 19], 400, 410)
test_data(pre_data, labels, [12, 18, 19], 400, 410)
test_data(pre_data, labels, [13, 18, 19], 400, 410)
test_data(pre_data, labels, [14, 18, 19], 400, 410)
test_data(pre_data, labels, [15, 18, 19], 400, 410)
test_data(pre_data, labels, [16, 18, 19], 400, 410)
test_data(pre_data, labels, [17, 18, 19], 400, 410)
```

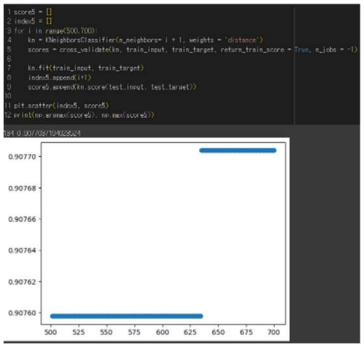
```
400 0.8026911533926269
feartures :ImporterID TotalGrossMassMeasure(KG) AdValoremTaxBaseAmount(Won)
 400 0.7928367065804811
feartures :SellerID TotalGrossMassMeasure(KG) AdValoremTaxBaseAmount(Won)
 400 0.7984528981667903
feartures :ExpressID TotalGrossMassMeasure(KG) AdValoremTaxBaseAmount(Won)
 401 0.8793048638338455
feartures :ClassificationID TotalGrossMassMeasure(KG) AdValoremTaxBaseAmount(Won)
 400 0.9022994595740171
feartures :ExportationCountry TotalGrossMassMeasure(KG) AdValoremTaxBaseAmount(Won)
 400 0.9070679241284306
feartures :OriginCountry TotalGrossMassMeasure(KG) AdValoremTaxBaseAmount(Won)
 400 0.9063261629755219
feartures :TaxRate TotalGrossMassMeasure(KG) AdValoremTaxBaseAmount(Won)
 400 0.9066440606124828
feartures :DutyRegime TotalGrossMassMeasure(KG) AdValoremTaxBaseAmount(Won)
 400 0.9060082653385609
feartures :DisplayIndicator TotalGrossMassMeasure(KG) AdValoremTaxBaseAmount(Won)
 402 0.8697679347250186
```

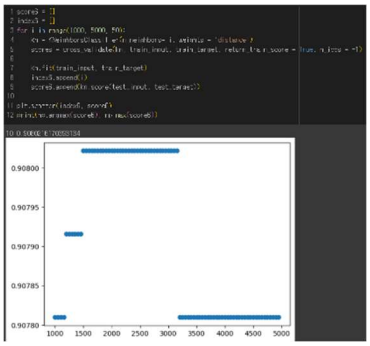
In five indicated cases out of 18 cases, scores exceeded 90%. Among them, I chose 'TaxRate' to include in the fixed characteristics and the test score value was obtained when n_neighbors was 400 to 410 in the same way as above.

```
test_data(pre_data, labels, [0, 15, 18, 19], 400, 410)
test_data(pre_data, labels, [1, 15, 18, 19], 400, 410)
test_data(pre_data, labels, [2, 15, 18, 19], 400, 410)
test_data(pre_data, labels, [3, 15, 18, 19], 400, 410)
test_data(pre_data, labels, [4, 15, 18, 19], 400, 410)
test_data(pre_data, labels, [5, 15, 18, 19], 400, 410)
test_data(pre_data, labels, [6, 15, 18, 19], 400, 410)
test_data(pre_data, labels, [7, 15, 18, 19], 400, 410)
test_data(pre_data, labels, [8, 15, 18, 19], 400, 410)
test_data(pre_data, labels, [9, 15, 18, 19], 400, 410)
test_data(pre_data, labels, [10, 15, 18, 19], 400, 410)
test_data(pre_data, labels, [11, 15, 18, 19], 400, 410)
test_data(pre_data, labels, [12, 15, 18, 19], 400, 410)
test_data(pre_data, labels, [13, 15, 18, 19], 400, 410)
test_data(pre_data, labels, [14, 15, 18, 19], 400, 410)
test_data(pre_data, labels, [15, 16, 18, 19], 400, 410)
test_data(pre_data, labels, [15, 17, 18, 19], 400, 410)
```

```
feartures :SellerID TaxRate TotalGrossMassMeasure(KG) AdValoremTaxBaseAmount(Won)
 400 0.7995125569566599
feartures :ExpressID TaxRate TotalGrossMassMeasure(KG) AdValoremTaxBaseAmount(Won)
 409 0.8829077037194023
feartures :ClassificationID TaxRate TotalGrossMassMeasure(KG) AdValoremTaxBaseAmount(Won)
 400 0.9072798558864046
feartures :ExportationCountry TaxRate TotalGrossMassMeasure(KG) AdValoremTaxBaseAmount(Won)
 400 0.9103528663770266
feartures :OriginCountry TaxRate TotalGrossMassMeasure(KG) AdValoremTaxBaseAmount(Won)
 400 0.9099290028610787
feartures :TaxRate DutyRegime TotalGrossMassMeasure(KG) AdValoremTaxBaseAmount(Won)
 401 0.9070679241284306
feartures :TaxRate DisplayIndicator TotalGrossMassMeasure(KG) AdValoremTaxBaseAmount(Won)
```

I trained the kneighbors algorithm using the feature combination with the highest Test score: ExportationCountry, TaxRate, TotalGrossMassMeasure (KG), and AdValoremTaxBaseAmount (Won), and submitted the prediction data for test.csv to kaggle. However, the Kaggle result value was about 0.83855. Concluding that this feature combination was not a very good combination, I tried the same with the next highest-scoring feature combination, OriginCountry, TaxRate, TotalGross MassMeasure (KG), and AdValoremTaxBaseAmount (Won). This combination made similar results as well. Next, I tried the high-scoring feature combinations ClassificationID, TaxRate, TotalGross MassMeasure (KG), and AdValoremTaxBaseAmount (Won). In this case, unlike the previous cases, the test score increased as the size of the kneighbors increased. The test score increased until the kneighbors size was approximately 3000. When I submitted it to Kaggle, it came out about 91.047%.



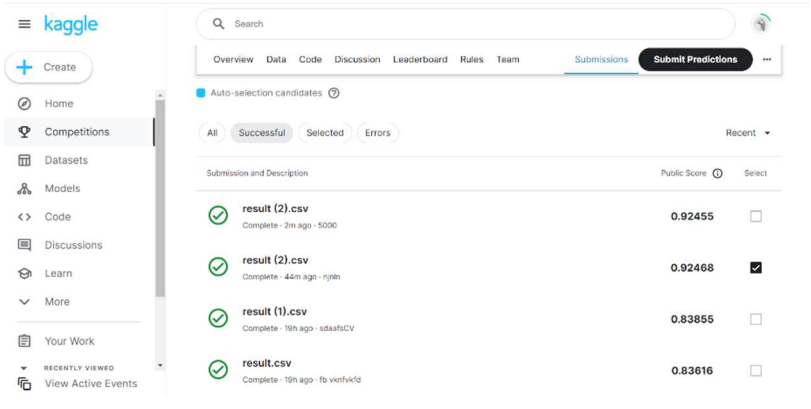


To further increase the accuracy, I also used support vector machines and tree ensembles, which are said to have high performance in classification problems, but it did not get any better at 91.047%. In the meantime, I suddenly thought of testing test.csv after training the kneighbors algorithm using all the train data once without using the train_test_split function, so I tried it. Then, the Kaggle result was 0.92468. I concluded that it was more efficient to use the whole data to classify test.csv data.

```
pre_data = preprocess(import_data)

analyze_data = pre_data.astype('float')
data_input = analyze_data.drop(labels = 'Fake', axis = 1).to_numpy()
data_target = analyze_data['Fake'].to_numpy()

kn = KNeighborsClassifier(n_neighbors= 3500, weights = 'distance')
kn.fit(data_input, data_target)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3500, weights='distance')
```

```
import_test = pd.read_csv('test.csv')
pre_data2 = preprocess(import_test)
analyze_data = pre_data2.astype('float')

test_data = analyze_data.to_numpy()
result = kn.predict(test_data)

df = pd.DataFrame(result)

df.to_csv('result.csv', index=False)
```



I thought about it a little more because I wanted to improve my score. While using the kneighbors algorithm a few times, when I factorized the 'exportationcountry' data and used it for model training, the training data showed good performance, but when I submitted it to kaggle, I got a low score. I thought about this issue. I think the cause of this problem is that the number of types of exportationcountry in train.csv data and test.csv data is different. This is because different values are assigned to the same country code when factorizing is performed if the number of types is different. Since not only 'exportationcountry' but also 'origincountry' features gave good scores when used for model training, I decided to use the kneighbors algorithm including both features. As follows, the two characteristics were divided into country codes that have both train.csv and test.csv, and country codes that only exist in one of them. To find out, I created a function called classify.

```python
import pandas as pd
import numpy as np
from datetime import date
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')

def classify(train, test, label):
    c_l1 = train[label].to_numpy()
    c_l2 = test[label].to_numpy()

    codes1, uniques1 = pd.factorize(c_l1, sort = True)
    codes2, uniques2 = pd.factorize(c_l2, sort = True)
    print(label, 'train :', len(uniques1), 'test :', len(uniques2))

    only_train = set(uniques1)
    only_test = set(uniques2)
    both = only_train.intersection(only_test)
    only_train = list(only_train - both)
    only_test = list(only_test - both)
    both = list(both)

    print('both :', both)
    print('only train :', only_train)
    print('only test :', only_test)

    return both, only_train, only_test
```

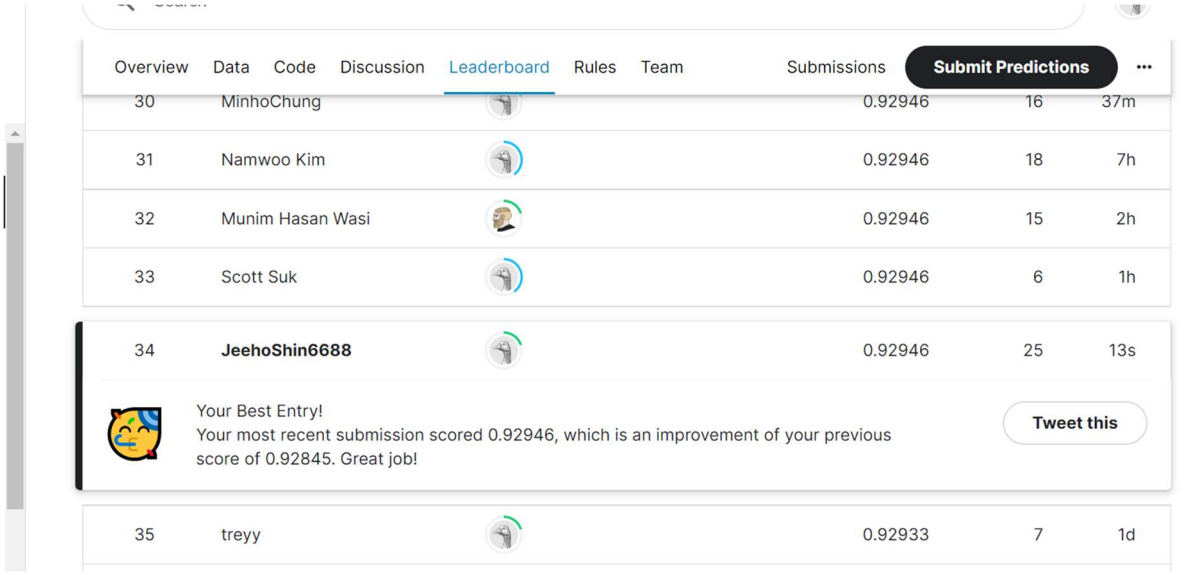
```
ExportationCountry train : 110 test : 96
both : ['RU', 'IE', 'NZ', 'JP', 'CZ', 'IN', 'HK', 'BE', 'PR', 'GB', 'MN', 'MX', 'IT', 'KE', 'SL', 'PE', 'SD', 'SV', 'GR', 'FR', 'AE', 'CN', 'CA', 'HU', 'NI', 'VN', 'NP', 'NG', 'PY', 'PG', 'GT', 'LR', 'ML', 'ZA', 'BH', 'DJ',
only train : ['SR', 'JM', 'GH', 'GE', 'ET', 'RS', 'RE', 'KG', 'MT', 'SO', 'ME', 'UY', 'LI', 'SN', 'MP', 'IS']
only test : ['AS', 'BJ']
OriginCountry train : 116 test : 106
both : ['RU', 'IE', 'NZ', 'JP', 'CZ', 'IN', 'HK', 'BE', 'PR', 'GB', 'MN', 'MX', 'IT', 'KE', 'SL', 'PE', 'SV', 'GR', 'FR', 'AE', 'CN', 'CA', 'HU', 'ET', 'NI', 'VN', 'NP', 'NG', 'MG', 'PY', 'PG', 'GT', 'LR', 'IR', 'ML', 'ZA',
only train : ['SR', 'JM', 'GH', 'AL', 'BH', 'YE', 'MD', 'TG', 'UG', 'SO', 'BW', 'UY', 'MP', 'LB']
only test : ['AS', 'MT', 'MK', 'BJ']
[654, 1659, 3281, 5766, 5992, 7699, 8645, 13126, 16735, 17313, 18455, 18936, 20972, 22446, 22824, 23080, 25530, 26788, 27282, 29877, 30643, 30943, 34442, 34449, 34619, 36152, 37705]
[654, 1659, 5410, 7699, 8645, 10037, 12972, 13126, 14822, 16735, 18355, 18356, 18531, 18936, 19621, 22446, 22824, 23080, 24122, 24335, 25950, 26788, 26991, 27781, 27823, 28764, 28928, 29877, 30885, 31271, 33537, 34449, 3524
47
```

For the exportationcountry feature, there were 94 values with both train.csv and test.csv, 16 values with only train.csv, and 2 values with only test.csv. As for the origincountry feature, there were 102 values with both train.csv and test.csv, 14 values with only train.csv, and 4 values with only test.csv. Rows corresponding to values appearing only in Train.csv were deleted using the drop function. Rows corresponding to values that appear only in Test.csv are also deleted using the drop function. After preprocessing in this way, the kneighbors algorithm was used for n = 3000, and the Kaggle score was 0.92946.

```python
labels = ['ExportationCountry', 'OriginCountry']
both1, only_train1, only_test1 = classify(train, test, labels[0])
both2, only_train2, only_test2 = classify(train, test, labels[1])

exportation1 = pre_data['ExportationCountry'].to_numpy()
exportation2 = pre_data['OriginCountry'].to_numpy()

index1 = []
index2 = []

for i in range(exportation1.size):
    for j in range(len(only_train1)):
        if exportation1[i] == only_train1[j]:
            index1.append(i)
print(index1)

for i in range(exportation2.size):
    for j in range(len(only_train2)):
        if exportation2[i] == only_train2[j]:
            index2.append(i)
print(index2)

index = list(set(index1 + index2))
print(len(index))

pre_data.drop(index, axis=0, inplace=True)
pre_data.head()
```

```python
#only test1 : ['AS', 'BJ'] exportation
#only test2 : ['AS', 'MT', 'MK', 'BJ'] origin

exportation3 = pre_test['ExportationCountry'].to_numpy()
exportation4 = pre_test['OriginCountry'].to_numpy()

index3 = []
index4 = []

for i in range(exportation3.size):
    for j in range(len(only_test1)):
        if exportation3[i] == only_test1[j]:
            index3.append(i)
print(index3)

for i in range(exportation4.size):
    for j in range(len(only_test2)):
        if exportation4[i] == only_test2[j]:
            index4.append(i)
print(index4)

index5 = list(set(index3 + index4))
print(index5)

keep = pre_test.loc[index5]
keep.head()

pre_test.drop(index5, axis=0, inplace=True)
#pre_test.tail()
```

```python
a = pre_data['ExportationCountry'].to_numpy()
c1, u1 = pd.factorize(a, sort = True)
pre_data['ExportationCountry'] = c1

b = pre_test['ExportationCountry'].to_numpy()
c2, u2 = pd.factorize(b, sort = True)
pre_test['ExportationCountry'] = c2

c = pre_data['OriginCountry'].to_numpy()
c3, u3 = pd.factorize(c, sort = True)
pre_data['OriginCountry'] = c3

d = pre_test['OriginCountry'].to_numpy()
c4, u4 = pd.factorize(d, sort = True)
pre_test['OriginCountry'] = c4

analyze_data = pre_data.astype('float')
data_input = analyze_data.drop(labels = 'Fake', axis = 1).to_numpy()
data_target = analyze_data['Fake'].to_numpy()

kn = KNeighborsClassifier(n_neighbors = 3000, weights = 'distance')
kn.fit(data_input, data_target)

test_input = pre_test.astype('float').to_numpy()
result = kn.predict(test_input)

print(result.size)

df = pd.DataFrame(result)

df.to_csv('result.csv', index=False)
```



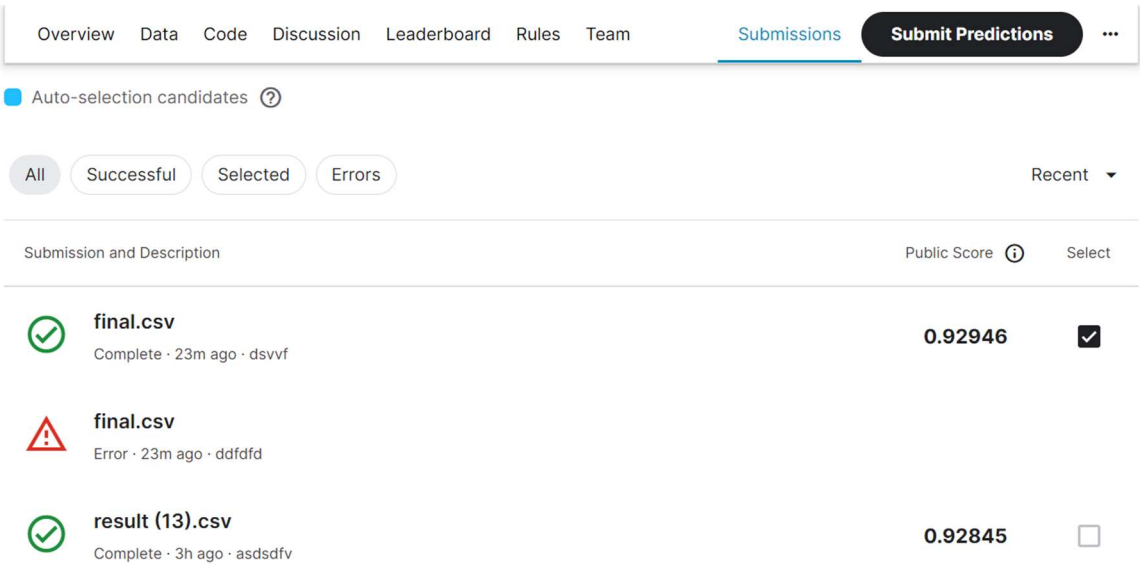
Overview Data Code Discussion Leaderboard Rules Team Submissions **Submit Predictions** ...

30	MinhoChung	0.92946	16	37m
31	Namwoo Kim	0.92946	18	7h
32	Munim Hasan Wasi	0.92946	15	2h
33	Scott Suk	0.92946	6	1h
34	**JeehoShin6688**	0.92946	25	13s

Your Best Entry!
Your most recent submission scored 0.92946, which is an improvement of your previous score of 0.92845. Great job!

Tweet this

35	treyy	0.92933	7	1d



Overview Data Code Discussion Leaderboard Rules Team Submissions **Submit Predictions** ...

Auto-selection candidates

All Successful Selected Errors Recent

Submission and Description	Public Score	Select
final.csv Complete · 23m ago · dsvvf	**0.92946**	☑
final.csv Error · 23m ago · ddfdfd		
result (13).csv Complete · 3h ago · asdsdfv	**0.92845**	☐