딥러닝/클라우드

중간 레포트

32183164 이석현

Dankook University

2020 Fall

딥러닝/클라우드 중간 레포트

Seokhyeon Lee(이석현) 32183164

목차

딥러닝/클라우드 중간 레포트

Seokhyeon Lee(이석현) 32183164

1. Basic Setting & Concept

Dataset의 feature구분을 쉽게 하기위해 header를 추가해주었고 train dataset의 label 값을 마지막 column으로 옮겨 주었다.

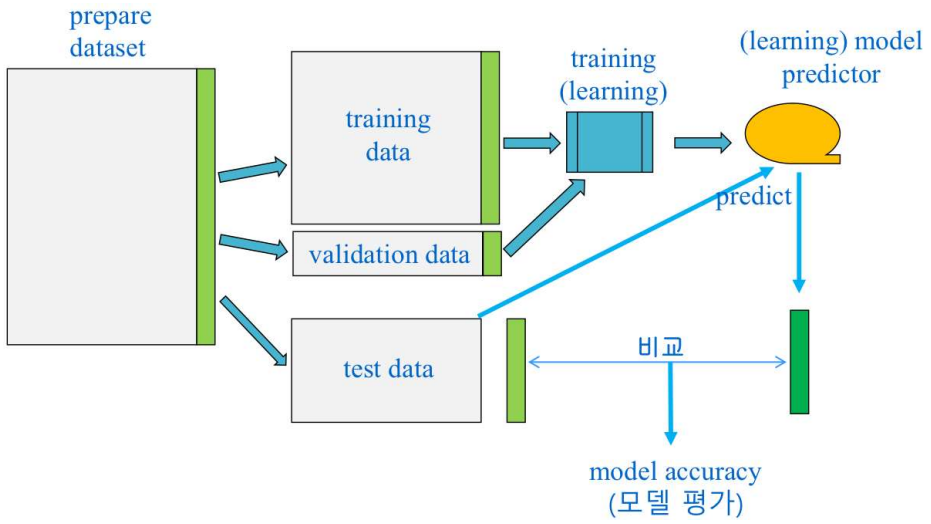
	A	B	C	D	E	F	G	H	I
1	Class1	Class2	Class3	Class4	Class5	Class6	Class7	Class8	Class9
2	188	128	95	114	143	108	88	103	113
3	174	112	88	104	119	92	74	79	88
4	175	138	106	105	135	109	75	95	113
5	176	111	80	106	131	96	76	99	104
6	182	144	111	100	151	119	67	106	114
7	143	159	156	73	71	139	43	31	133
8	166	100	69	96	119	76	66	79	80
9	171	95	62	94	108	71	60	71	71
10	146	148	145	76	71	133	46	31	139
11	175	117	82	109	128	97	79	88	93
12	159	92	63	93	106	75	63	70	75
13	163	117	93	93	124	93	63	79	89
14	171	150	111	101	148	114	63	88	97
15	174	117	90	100	119	82	60	63	74

▲trainset.csv

	A	B	C	D	E	F	G	H	I
1	Class1	Class2	Class3	Class4	Class5	Class6	Class7	Class8	Class9
2	185	130	100	119	142	110	89	102	114
3	178	139	100	104	142	114	63	97	114
4	182	125	97	112	137	105	78	97	105
5	174	113	83	97	115	89	63	63	82
6	167	144	107	97	146	124	67	111	114
7	174	134	100	97	146	114	63	102	114
8	180	148	109	106	156	122	71	112	122
9	192	139	100	122	148	119	92	108	110
10	156	117	89	82	113	93	52	66	86
11	182	124	88	112	136	104	82	96	100
12	163	95	63	89	106	72	63	66	75
13	155	97	72	85	88	74	55	51	67
14	188	131	97	122	146	115	88	102	111
15	159	124	102	97	139	109	67	87	100
16	167	94	72	90	103	74	57	56	74

▲testset.csv

답이 있는 데이터를 활용해 데이터를 학습시키는 것이므로, 아래와 같은 방식으로 지도학습(Supervised Learning)을 수행한다.



딥러닝/클라우드 중간 레포트

Seokhyeon Lee(이석현) 32183164

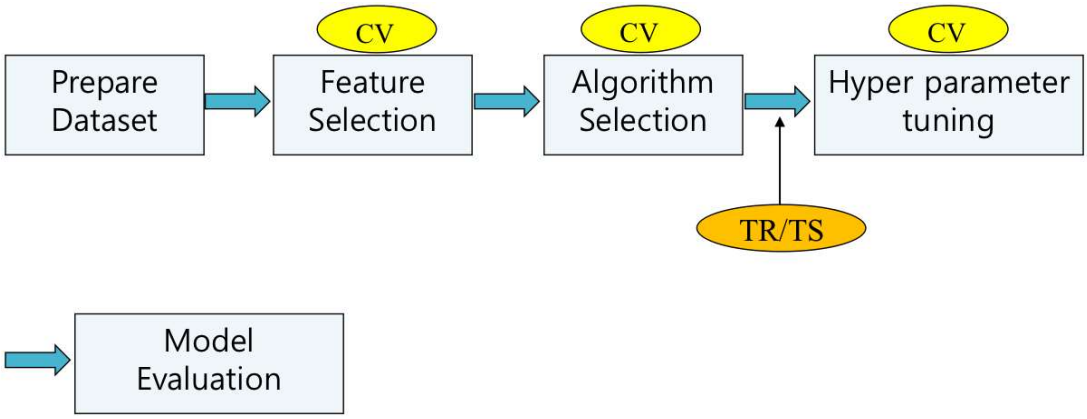
Training data: 과거 데이터의 역할

Validation data: 학습 과정에서 만들어지는 모델을 평가하는데 사용되는 데이터

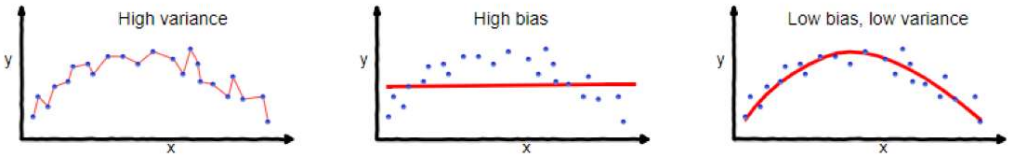
Test data: 미래 데이터의 역할. 미래 예측 시 모델이 어느 정도의 성능을 보일지 판단.

이 dataset 은 범주나 그룹을 예측하는 것이므로 분류(classification) 방식을 사용한다.

프로젝트는 아래의 순서를 따라 진행하였다.



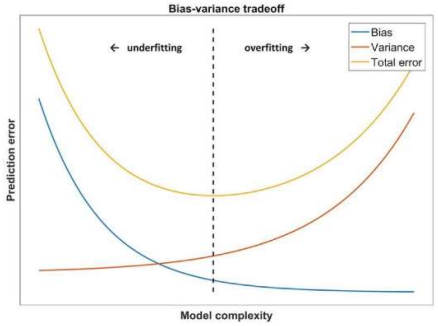
딥러닝을 수행하며 Classification model error 가 발생할 수 있다. 이는 Noise, Bias, Variance 가 합쳐져 나타나게 된다. Bias 는 데이터 내의 모든 정보를 고려하지 않음으로 인해, 잘못된 것을 학습하게 되는 경향을 말하며 Variance 는 데이터를 너무 세세하게 학습하여 모델을 만들면서 모델의 변동성이 커지게 되는 것이다. 이는 Overfitting 의 원인이 되기도 한다. Bias 와 Variance 는 반비례 관계로, Bias 를 줄이려고 하면 Variance 가 증가하고 Variance 를 줄이려고 하면 Bias 가 증가하게 된다. 따라서 이 둘이 적절하게 균형을 이루는 지점에서 모델을 택하는 것이 중요하다.



overfitting
너무 복잡한 모델

underfitting
너무 단순한 모델

Good balance



2. Feature Selection

LogisticRegression을 이용하며, feature 값에 따른 accuracy를 비교해주어 사용하기에 적절한 feature 값을 찾아내는 것을 목표로 한다. 좋은 feature는 명료한 class boundary를 갖는 feature를 말하며, Filter method, Forward Selection, Backward Elimination 방식을 이용해 feature값을 추가하고 제거하며 최적의 feature들을 찾아낸다.

```
import pandas as pd

import numpy as np

from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import cross_val_score

# prepare the dataset

df = pd.read_csv('D:/data/middle_test/trainset.csv')

df_X = df.loc[:, df.columns != 'label']

df_y = df['label']

# whole features

model = LogisticRegression(solver='lbfgs', max_iter=500)

scores = cross_val_score(model, df_X, df_y, cv=5)

print("Acc: "+str(scores.mean()))
```

⇨ `Acc: 0.8364497323516756`

```
# feature selection by filter method

# feature evaluation method : chi-square

from sklearn.feature_selection import SelectKBest

from sklearn.feature_selection import chi2



test = SelectKBest(score_func=chi2, k=df_X.shape[1])

print(df_X.shape[1])
```

```
fit = test.fit(df_X, df_y)

print(np.round(fit.scores_, 3))

f_order = np.argsort(-fit.scores_)  # sort index by decreasing order

sorted_columns = df.columns[f_order]

# test classification accuracy by selected features (KNN)

model = LogisticRegression(solver='lbfgs', max_iter=500)


for i in range(1, df_X.shape[1]+1):

    fs = sorted_columns[0:i]

    df_X_selected = df_X[fs]

    scores = cross_val_score(model, df_X_selected, df_y, cv=5)

    print(fs.tolist())

    print(np.round(scores.mean(), 3))

    ⇨ 중요도 순으로 26개의 feature를 사용할 때 accuracy가 0.837로 가장 높게 나
       왔다.
```

```
# Backward elimination (Recursive Feature Elimination) - 17

from sklearn.feature_selection import RFE


model = LogisticRegression(solver='lbfgs', max_iter=500)

rfe = RFE(model, n_features_to_select=17)

fit = rfe.fit(df_X, df_y)

print("Num Features: %d" % fit.n_features_)

fs = df_X.columns[fit.support_].tolist()    # selected features
```

```
print("Selected Features: %s" % fs)

scores = cross_val_score(model, df_X[fs], df_y, cv=5)

print("Acc: "+str(scores.mean()))
```

⇨ Feature 개수가 17개 일때, Acc: 0.8359805144152801로 가장 높은 값을 가졌다.

```
# Forward selection

from mlxtend.feature_selection import SequentialFeatureSelector as SFS



model = LogisticRegression(solver='lbfgs', max_iter=500)

sfs1 = SFS(model,

           k_features=19,

           verbose=2,

           scoring='accuracy',

           cv=5)

sfs1 = sfs1.fit(df_X, df_y, custom_feature_names=df_X.columns)

sfs1.subsets_                 # selection process

sfs1.k_feature_idx_           # selected feature index

sfs1.k_feature_names_         # selected feature name



scores = cross_val_score(model, df_X[list(sfs1.k_feature_names_)], df_y, cv=5)

print("Acc: "+str(scores.mean()))
```

⇨ Acc: 0.8348141781485772

필터 메소드를 이용한 방식은 총 31개의 feature 중 26개를 사용하게 되므로 너무 많은 것 같다는 생각이 들어 배제하였고 남은 두가지 방식 중에 accuracy가 조금 더 높은 Backward Elimination 방식을 사용하였다.

3. Model Selection

LinearRegression, KNN, DecisionTree, RandomForest, SVM 각각의 알고리즘을 사용해 각 모델의 예측 accuracy 중간 값과 범위를 알아본 후, 중간 값이 높고 범위가 좁은 알고리즘을 택해 학습모델로 삼는다.

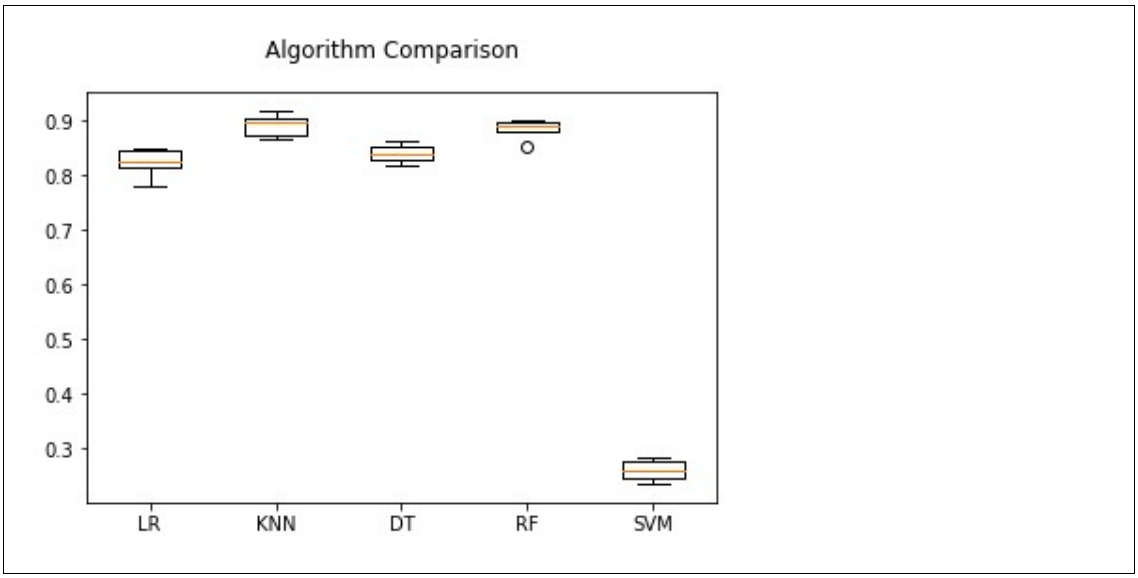
```
import matplotlib.pyplot as plt
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder
```

```
from sklearn.feature_selection import RFE

model = LogisticRegression(solver='lbfgs', max_iter=500)
rfe = RFE(model, n_features_to_select=17)
fit = rfe.fit(df_X_selected, df_y)
fs = df_X.columns[fit.support_].tolist()

encoder = LabelEncoder()
encoder.fit(df_y)
df_y = encoder.transform(df_y)

# prepare configuration for cross validation test harness
seed = 7
# prepare models
models = []
models.append(('LR', LogisticRegression(max_iter=500)))
models.append(('KNN', KNeighborsClassifier()))
models.append(('DT', DecisionTreeClassifier()))
models.append(('RF', RandomForestClassifier()))
models.append(('SVM', SVC()))
```

```
# evaluate each model in turn
results = []
names = []
scoring = 'accuracy'
for name, model in models:
	kfold = model_selection.KFold(n_splits=10, random_state=seed, shuffle=True)
	cv_results = model_selection.cross_val_score(model, df_X[fs], df_y, cv=kfold, scoring=scoring)
	results.append(cv_results)
	names.append(name)
	msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
	print(msg)

print(results)
# average accuracy of classifiers
for i in range(0,len(results)):
    print(names[i] + "\t" + str(round(np.mean(results[i]),4)))

# boxplot algorithm comparison
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```

⇨

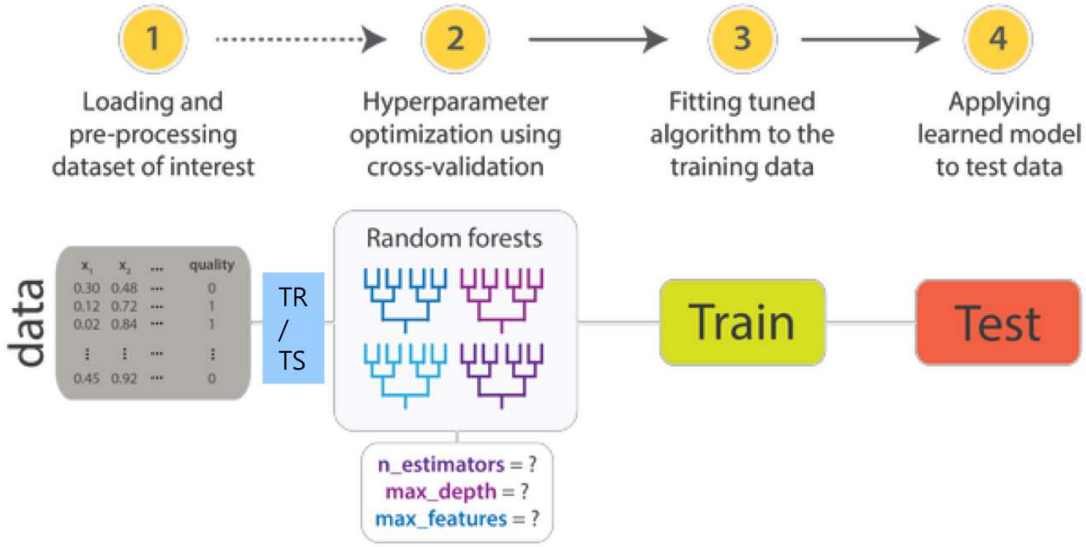
```
SVM: 0.261215 (0.017195)
[array([0.81775701, 0.82476636, 0.8411215 , 0.82009346, 0.84345794,
       0.84579439, 0.81074766, 0.77803738, 0.81308411, 0.84813084]), array([0.89719626, 0.87383178, 0.91588785, 0.86915888, 0.86915888,
       0.90420561, 0.9088785 , 0.86448598, 0.89485981, 0.90186916]), array([0.8411215 , 0.85280374, 0.8411215 , 0.83411215, 0.8317757 ,
       0.85747664, 0.82476636, 0.81542056, 0.81542056, 0.86214953]), array([0.89953271, 0.88551402, 0.89018692, 0.87850467, 0.85280374,
       0.89953271, 0.89485981, 0.87850467, 0.88317757, 0.89485981]), array([0.25      , 0.2546729 , 0.26401869, 0.28504673, 0.24299065,
       0.2453271 , 0.28037383, 0.28504673, 0.26869159, 0.23598131])]
LR      0.8243
KNN     0.89
DT      0.8376
RF      0.8857
SVM     0.2612
```



Feature selection을 한 후, 여태껏 수업에서 배운 알고리즘을 이용하여 정확도를 계산하였을 때, KNN과 RF가 비교적 높은 accuracy를 가지고 있었고 가장 높은 accuracy와 가장 낮은 accuracy의 차가 적게 나와 두 알고리즘을 모두 사용하기로 하였다.

4. Hyper parameter Tuning

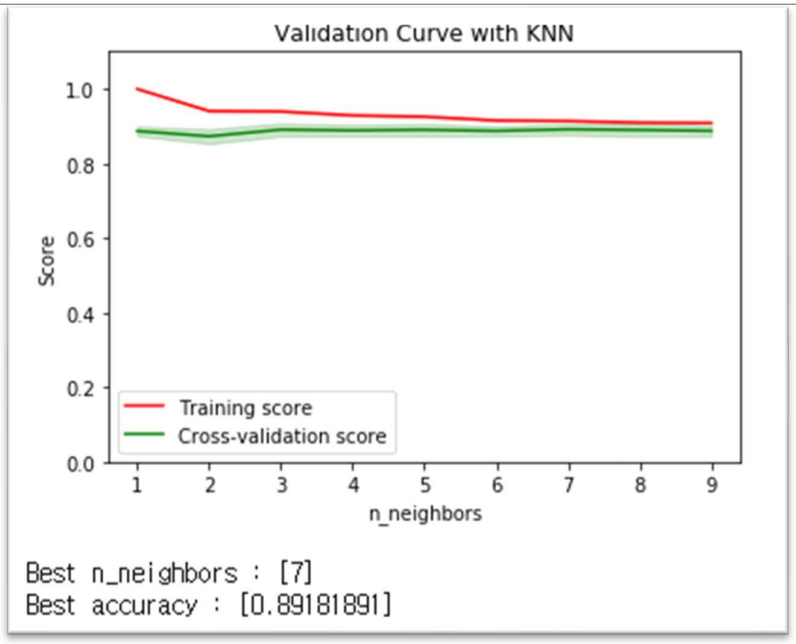
Hyper parmeter는 모델링할 때 사용자가 직접 세팅해주는 파라미터 값을 의미한다. 대부분의 classification 알고리즘에서 hyper parameter는 모델의 수행능력에 많은 영향을 준다. 따라서 hyper parameter를 tuning하는 작업은 중요하고 많은 시간이 소요된다. 데이터셋 전처리를 마친 후 grid search cv, random search cross-validation을 이용해 hyper parameter를 최적화해준다.



```
# KNN hyper parameter tuning Example
# using validation_curve (for single parameter)

from sklearn import datasets
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import validation_curve
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
import numpy as np
import matplotlib.pyplot as plt

df = pd.read_csv('D:/data/middle_test/trainset.csv')
df_X = df.loc[:, df.columns != 'label']
df_y = df['label']

# Backward elimination (Recursive Feature Elimination) 17
from sklearn.feature_selection import RFE

model = LogisticRegression(solver='lbfgs', max_iter=500)
rfe = RFE(model, n_features_to_select=17)
fit = rfe.fit(df_X, df_y)
print("Num Features: %d" % fit.n_features_)
fs = df_X.columns[fit.support_].tolist()

# Load the diabetes dataset
df_X, df_y = df_X[fs], df_y

# scaling input data
scaler = StandardScaler()
scaler.fit(df_X)
df_X = scaler.transform(df_X)
```

```
# hyper parameter tuning
param_range = np.array([1,2,3,4,5,6,7,8,9])

train_scores, test_scores = \
    validation_curve(KNeighborsClassifier(), df_X, df_y,
                     param_name="n_neighbors", param_range=param_range,
                     cv=10, scoring="accuracy", n_jobs=1)

train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)

plt.plot(param_range,  train_scores_mean,  label="Training  score",
color="r")
plt.fill_between(param_range, train_scores_mean - train_scores_std,
                 train_scores_mean   +   train_scores_std,   alpha=0.2,
color="r")
plt.plot(param_range, test_scores_mean,
             label="Cross-validation score", color="g")
plt.fill_between(param_range, test_scores_mean - test_scores_std,
                 test_scores_mean    +    test_scores_std,    alpha=0.2,
color="g")
plt.legend(loc="best")
plt.title("Validation Curve with KNN")
plt.xlabel("n_neighbors")
plt.ylabel("Score")
plt.ylim(0.0, 1.1)
plt.show()

# find best parameter value
ch = np.where(test_scores_mean == np.amax(test_scores_mean))
print('Best n_neighbors :', param_range[ch])
print('Best accuracy :', test_scores_mean[ch])
```

⇨




```
# Random Forest tuning Example
# using: GridSearchCV

from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import pandas as pd
import pprint

pp = pprint.PrettyPrinter(width=80, indent=4)

# Split the data into training/testing sets
train_X, test_X, train_y, test_y = \
    train_test_split(df_X[fs], df_y, test_size=0.3,\
                     random_state=1234)

# base model
base_model = RandomForestClassifier(random_state=1234)
base_model.fit(train_X, train_y)
```

```
base_accuracy = base_model.score(test_X, test_y)


## GridSearchCV

# hyper parameter tuning
param_grid = {
    'bootstrap': [True],
    'max_depth': [80, 90, 100, 110],
    'max_features': [2, 3, 'auto', 'sqrt'],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 200, 300, 1000]
}

# Create a based model
rf = RandomForestClassifier(random_state=1234)

# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                      cv = 5, n_jobs = -1, verbose = 2)
# cv: cross validation
# n_jobs  : # of CPU core. -1: using all core
# verbose : print message (the higher, the more messages.)

# Fit the grid search to the data
grid_search.fit(train_X, train_y)

# best parameters
pp.pprint(grid_search.best_params_)

# best model
best_model = grid_search.best_estimator_
best_accuracy = best_model.score(test_X, test_y)
```

```
print('base acc: {0:0.2f}. best acc : {1:0.2f}'.format( base_accuracy, best_accuracy))
print('Improvement of {:0.2f}%.'.format( 100 * (best_accuracy - base_accuracy) / base_accuracy))
```

⇨

```
{   'bootstrap': True,
    'max_depth': 80,
    'max_features': 3,
    'min_samples_leaf': 3,
    'min_samples_split': 8,
    'n_estimators': 200}
base acc: 0.89. best acc : 0.90
Improvement of 1.31%.
```

```
# Random Forest tuning Example
# using: RandomizedSearchCV

from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
import pprint

pp = pprint.PrettyPrinter(width=80, indent=4)



# Split the data into training/testing sets
train_X, test_X, train_y, test_y = \
   train_test_split(df_X[fs], df_y, test_size=0.3,\
                    random_state=1234)

# base model
base_model = RandomForestClassifier(random_state=1234)
base_model.fit(train_X, train_y)
base_accuracy = base_model.score(test_X, test_y)
```

```
## RandomizedSearchCV
# define range of parameter values
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000,
num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]

# Create the random grid
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}
pp.pprint(random_grid)

# Use the random grid to search for best hyperparameters
# Random search of parameters, using 5 fold cross validation,
#  search across 100 different combinations, and use all available
cores
rf = RandomForestClassifier(random_state=1234)
rf_random = RandomizedSearchCV(estimator = rf, param_distributions =
random_grid, n_iter = 100, cv = 5, verbose=2, random_state=42, n_jobs
= -1)
```

```
# Fit the random search model
rf_random.fit(train_X, train_y)

# best parameters
pp.pprint(rf_random.best_params_)

# best model
best_random_model = rf_random.best_estimator_
best_random_accuracy = best_random_model.score(test_X, test_y)

print('base acc: {0:0.2f}. best acc : {1:0.2f}'.format( base_accuracy, best_random_accuracy))
print('Improvement of {:0.2f}%.'.format( 100 * (best_random_accuracy - base_accuracy) / base_accuracy))
```

⇨

```
{   'bootstrap': [True, False],
    'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
    'max_features': ['auto', 'sqrt'],
    'min_samples_leaf': [1, 2, 4],
    'min_samples_split': [2, 5, 10],
    'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
Fitting 5 folds for each of 100 candidates, totalling 500 fits

[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   36.9s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:  3.8min
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed:  7.9min
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed: 10.6min finished

{   'bootstrap': False,
    'max_depth': 90,
    'max_features': 'sqrt',
    'min_samples_leaf': 1,
    'min_samples_split': 5,
    'n_estimators': 800}
base acc: 0.89. best acc : 0.91
Improvement of 2.19%.
```

KNN 은 neighbor 의 수를 7 로 할 때, 최적임을 알 수 있었다.

5. Prediction

(1) KNN(n_neighbors = 7) 이용

```
import pandas as pd
from sklearn.linear_model import LogisticRegression

df = pd.read_csv('D:/data/middle_test/trainset.csv')
df_X = df.loc[:, df.columns != 'label']
df_y = df['label']

# Backward elimination (Recursive Feature Elimination) 17
from sklearn.feature_selection import RFE

model = LogisticRegression(solver='lbfgs', max_iter=500)
rfe = RFE(model, n_features_to_select=17)
fit = rfe.fit(df_X, df_y)
print("Num Features: %d" % fit.n_features_)
fs = df_X.columns[fit.support_].tolist()   # selected features
#======================Prediction=========================
from sklearn import datasets
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import numpy as np

# Load dataset
train_X = df_X[fs]
train_y = df_y
model =  KNeighborsClassifier(n_neighbors=7)
model.fit(train_X, train_y)
df = pd.read_csv('D:/data/middle_test/testset.csv')
df_test_X = df.loc[:,:]
df_test_X = df_test_X[fs]
```

```
len(df)
len(df_test_X)
pred_y = model.predict(df_test_X)

df_pred = pd.DataFrame(pred_y)
print(df_pred)
df_pred.to_csv('32183164_이석현.csv', header = None, index=None)
```

⇨ Accuracy: 0.90

(2) Random Forest – grid search tuning 이용

```
import pandas as pd
#import numpy as np
from sklearn.linear_model import LogisticRegression
#from sklearn.model_selection import cross_val_score

df = pd.read_csv('D:/data/middle_test/trainset.csv')
df_X = df.loc[:, df.columns != 'label']
df_y = df['label']

# Backward elimination (Recursive Feature Elimination) 17
from sklearn.feature_selection import RFE

model = LogisticRegression(solver='lbfgs', max_iter=500)
rfe = RFE(model, n_features_to_select=17)
fit = rfe.fit(df_X, df_y)
print("Num Features: %d" % fit.n_features_)
fs = df_X.columns[fit.support_].tolist()   # selected features
'''
print("Selected Features: %s" % fs)
#print("Feature Ranking: %s" % fit.ranking_)

scores = cross_val_score(model, df_X[fs], df_y, cv=5)
```

```
print("Acc: "+str(scores.mean()))
'''
#============================Algorithm=============================
from sklearn.ensemble import RandomForestClassifier
#from sklearn.model_selection import train_test_split
#from sklearn.metrics import confusion_matrix
import pandas as pd

# Load dataset
train_X = df_X[fs]
train_y = df_y

# Define learning model
model = RandomForestClassifier(n_estimators = 200, max_depth = 80,
max_features = 3, min_samples_split = 8, bootstrap = 'True',
random_state=1234)
# Train the model using the training sets
model.fit(train_X, train_y)

# prepare the dataset
df = pd.read_csv('D:/data/middle_test/testset.csv')

df_test_X = df.loc[:,:]
df_test_X = df_test_X[fs]

pred_y = model.predict(df_test_X)
print(pred_y)
len(pred_y)

df_pred = pd.DataFrame(pred_y)
print(df_pred)
df_pred.to_csv('32183164_이석현.csv', header = None, index=None)
```

⇨ Accuracy: 0.901

(3) Random Forest – random grid 이용

```
import pandas as pd
#import numpy as np
from sklearn.linear_model import LogisticRegression
#from sklearn.model_selection import cross_val_score

df = pd.read_csv('D:/data/middle_test/trainset.csv')
df_X = df.loc[:, df.columns != 'label']
df_y = df['label']

# Backward elimination (Recursive Feature Elimination) 17
from sklearn.feature_selection import RFE

model = LogisticRegression(solver='lbfgs', max_iter=500)
rfe = RFE(model, n_features_to_select=17)
fit = rfe.fit(df_X, df_y)
print("Num Features: %d" % fit.n_features_)
fs = df_X.columns[fit.support_].tolist()   # selected features

#============================Algorithm============================
from sklearn.ensemble import RandomForestClassifier
#from sklearn.model_selection import train_test_split
#from sklearn.metrics import confusion_matrix
import pandas as pd

# Load dataset
train_X = df_X[fs]
train_y = df_y

# Define learning model
model = RandomForestClassifier(n_estimators = 800, max_depth = 90,
max_features = 'sqrt', min_samples_split = 5, bootstrap = 'False',
random_state=1234)
```

```
# Train the model using the training sets
model.fit(train_X, train_y)

# prepare the dataset
df = pd.read_csv('D:/data/middle_test/testset.csv')

df_test_X = df.loc[:,:]
df_test_X = df_test_X[fs]

pred_y = model.predict(df_test_X)
print(pred_y)
len(pred_y)

df_pred = pd.DataFrame(pred_y)
print(df_pred)
df_pred.to_csv('32183164_이석현.csv', header = None, index=None)
```

⇨ Accuracy: 0.902

6. Additional Implementation

정확도를 보다 높이기 위해서 새로운 알고리즘을 공부하였다. Ensemble Learning이란 여러 개의 classifier를 생성하고 그 예측을 결합함으로써 보다 정확한 최종 예측을 도출하는 기법을 말한다. Ensemble Learning에는 수업시간에 배운 RandomForest 외에도 캐글의 XGboost, LightGBM 등이 있다. 이러한 Ensembel Learning은 Voting, Bagging, Boosting의 세가지 유형이 있다. Voting과 Bagging은 투표를 통해 최종 예측 결과를 결정하는 방식이라는 공통 점이 있지만, Voting의 경우 일반적으로 서로 다른 알고리즘을 가진 classifier를 결합하는 것이고 Bagging은 각각의 classifier가 비슷한 알고리즘 기반이지만 데이터 샘플링을 서로 다르게 가져가며 학습을 수행하는 것이다. Bagging의 대표적인 예로 RandomForest가 있다. Boosting은 여러 개의 weak learner를 순차적으로 학습 및 예측하면서 잘못 에측한 데이터에 가중치를 부여해 오류를 개선하며 학습하는 방식으로 Boosting계열의 알고리즘에는 LightGBM, XGboost가 있다. 이 중 LightGBM은 XGboost의 학습

시간이 길다는 것과 메모리 사용량이 많다는 단점을 보완하여 만들어진 알고리즘이며, 적은 dataset에 적용할 경우 overfitting이 만들어지기 쉽다는 단점을 가지고 있다. LightGBM은 Leaf Wise 방식을 사용하여 트리의 균형을 맞추지 않고, max delta loss를 가지는 leaf 노드를 지속적으로 분할하면서 트리의 깊이가 깊어지고 비대칭적인 규칙 트리를 만들게 된다. 이러한 학습을 반복할수록 예측 오류 손실을 최소화할 수 있다는 장점이 있다.

lightGBM알고리즘을 이용하여 다음과 같이 코드를 짜보았다.

```
import lightgbm as lgb
from matplotlib import pyplot as plt
from matplotlib import rcParams
import numpy as np
from pathlib import Path
import pandas as pd
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import seaborn as sns
import warnings
#import kaggler

import pandas as pd
#import numpy as np
from sklearn.linear_model import LogisticRegression
#from sklearn.model_selection import cross_val_score

df = pd.read_csv('D:/data/middle_test/trainset.csv')
df_X = df.loc[:, df.columns != 'label']
df_y = df['label']

# Backward elimination (Recursive Feature Elimination) 17
from sklearn.feature_selection import RFE

model = LogisticRegression(solver='lbfgs', max_iter=500)
```

```
rfe = RFE(model, n_features_to_select=17)
fit = rfe.fit(df_X, df_y)
print("Num Features: %d" % fit.n_features_)
fs = df_X.columns[fit.support_].tolist()

from sklearn.ensemble import RandomForestClassifier
#from sklearn.model_selection import train_test_split
#from sklearn.metrics import confusion_matrix
import pandas as pd
from lightgbm import LGBMClassifier as lgb
model = lgb(n_estimators = 400)

train_X = df_X[fs]
train_y = df_y

evals = [(train_X, train_y)]
model.fit(train_X,    train_y,    early_stopping_rounds    =    100,
eval_metric="logloss", eval_set = evals, verbose = True)

df = pd.read_csv('D:/data/middle_test/testset.csv')

df_test_X = df.loc[:,:]
df_test_X = df_test_X[fs]

pred_y = model.predict(df_test_X)
print(pred_y)
len(pred_y)



df_pred = pd.DataFrame(pred_y)
print(df_pred)
df_pred.to_csv('32183164_이석현.csv', header = None, index=None)
```

⇨ Accuracy: 0.91

Seokhyeon Lee(이석현) 32183164

Hyper parameter Tuning 을 하기위해 parameter 의 적절한 값을 찾는 코드를 구현했다.

```
import lightgbm as lgb
from matplotlib import pyplot as plt
from matplotlib import rcParams
import numpy as np
from pathlib import Path
import pandas as pd
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import seaborn as sns
import warnings
#import kaggler

import pandas as pd
#import numpy as np
from sklearn.linear_model import LogisticRegression
#from sklearn.model_selection import cross_val_score
df = pd.read_csv('D:/data/middle_test/trainset.csv')
df_X = df.loc[:, df.columns != 'label']
df_y = df['label']

# Backward elimination (Recursive Feature Elimination) 17
from sklearn.feature_selection import RFE

model = LogisticRegression(solver='lbfgs', max_iter=500)
rfe = RFE(model, n_features_to_select=17)
fit = rfe.fit(df_X, df_y)
print("Num Features: %d" % fit.n_features_)
fs = df_X.columns[fit.support_].tolist()

from sklearn.ensemble import RandomForestClassifier
#from sklearn.model_selection import train_test_split
#from sklearn.metrics import confusion_matrix
```

```
import pandas as pd
from lightgbm import LGBMClassifier as lgb

ftr = df_X[fs]
target = df_y

X_train, X_test, y_train, y_test = train_test_split(ftr, target,
test_size=0.2, random_state=156)

from sklearn.model_selection import GridSearchCV

model = lgb(n_estimators = 200)

params = {'num_leaves': [32,64],
         'max_depth':[128, 160],
         'min_child_samples':[60, 100],
         'subsample':[0.8, 1]}

gridcv = GridSearchCV(model, param_grid=params, cv=3)
gridcv.fit(X_train, y_train)

print('GridSearchCV 최적 파라미터:', gridcv.best_params_)
```

```
⇨ GridSearchCV 최적 파라미터: {'max_depth': 128, 'min_child_samples': 60, 'num_leav
es': 32, 'subsample': 0.8}
```

최적 파라미터를 이용해 다시한번 코딩을 하였다.

```
import lightgbm as lgb
from matplotlib import pyplot as plt
from matplotlib import rcParams
import numpy as np
from pathlib import Path
import pandas as pd
```

```
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import seaborn as sns
import warnings
#import kaggler

import pandas as pd
#import numpy as np
from sklearn.linear_model import LogisticRegression
#from sklearn.model_selection import cross_val_score

df = pd.read_csv('D:/data/middle_test/trainset.csv')
df_X = df.loc[:, df.columns != 'label']
df_y = df['label']

# Backward elimination (Recursive Feature Elimination) 17
from sklearn.feature_selection import RFE

model = LogisticRegression(solver='lbfgs', max_iter=500)
rfe = RFE(model, n_features_to_select=17)
fit = rfe.fit(df_X, df_y)
print("Num Features: %d" % fit.n_features_)
fs = df_X.columns[fit.support_].tolist()

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import pandas as pd
from lightgbm import LGBMClassifier as lgb

ftr = df_X[fs]
target = df_y

X_train, X_test, y_train, y_test = train_test_split(ftr, target,
test_size=0.2, random_state=156)
```

```
from sklearn.model_selection import GridSearchCV

params = {'num_leaves': 32,
         'max_depth':128,
         'min_child_samples':60,
         'subsample':0.8}

model = lgb(n_estimators = 200, num_leaves = 32, max_depth = 128,
min_child_samples = 60, subsample = 0.8)

evals = [(X_test, y_test)]
model.fit(X_train,    y_train,    early_stopping_rounds    =    100,
eval_metric="logloss", eval_set = evals, verbose = True)
df = pd.read_csv('D:/data/middle_test/testset.csv')

df_test_X = df.loc[:,:]
df_test_X = df_test_X[fs]

pred_y = model.predict(df_test_X)
print(pred_y)
len(pred_y)

df_pred = pd.DataFrame(pred_y)
print(df_pred)
df_pred.to_csv('32183164_이석현.csv', header = None, index=None)
```

⇨ Accuracy: 0.917

7. Personal Feeling

6주차까지 수업에서 머신러닝의 개념 및 각 알고리즘의 원리와 코딩, 보다 정확도를 높일 수 있는 K-fold cross validation 등을 배웠습니다. 7주차에서는 feature 선택, model 선택, hyper parameter 값 조정 등의 내용을 배우고 이를 토대로 competition을 진행하였습니다. 이전에는 dataset을 train data와 test data로 나누고 train data로 학습을 통해 test data에서이 prediction이 얼마나 높은 accuracy를 가지고 있는지 구하는 내용이었지만, 이번 프로젝트에서는 처음부터 주어진 train set을 이용해 알아서 학습 모델을 구현하고 이 모델을 평가하는 것이었습니다. 수업을 듣

딥러닝/클라우드 중간 레포트

Seokhyeon Lee(이석현) 32183164

고 이 competition을 위한 coding을 시작하면서 정말 막막한 느낌이 들었습니다. 매주 열심히 공부했지만 어디서부터 시작하여야 할지 감이 오지 않았고, feature selection의 개념이 쉽게 이해되지 않아 어려움이 있었습니다. 강의를 반복해서 재생하고 구글링 및 책을 찾아보며 조금씩 이해를 할 수 있게 되었고 feature selection부분의 코딩을 시작할 수 있었습니다. Filter method, Forward Selection, Backward Elimination 각각으로 코딩하여 Logistic Regression을 통해 각각의 accuracy를 비교하였을 때 Filter method의 accuracy가 가장 높은 값을 가졌지만 너무 많은 feature를 사용한다는 생각이 들어 어느 것을 사용할지 더 공부를 하였고, 결과적으로 feature개수가 너무 많으면 불필요한 feature도 학습에 이용되게 되고 그로 인해 악영향이 생길 것을 우려하여 accuracy가 조금 낮지만 훨씬 적은 feature만을 사용하는 Backward Elimination을 사용하게 되었습니다. Model selection은 비교적 쉽게 할 수 있었습니다. 앞서 선택된 feature만을 이용해서 여러 모델링을 진행하면서 각각의 accuracy의 중간 값과 range를 비교하게 되었고 중간 값이 높고 range가 좁은 RandomForest, KNN을 사용하게 되었습니다. 마지막으로 hyper parameter tuning에 상당히 많은 시간을 소요하였습니다. Hyper parameter에 직접 여러 값을 넣어 보기도 하고 gridcv의 best parameter를 찾는 메소드로 최적 값을 찾아보기도 했습니다. 최적 값을 method의 parameter값으로 사용할 때 오류가 발생하여 어려움도 있었지만 구글링을 하며 이를 해결하였습니다. 하지만 계속해서 accuracy값이 높아지도록 시도했음에도 아무리 바꾸어 봐도 prediction의 accuracy가 0.91을 넘지 못했습니다. 어떤 방법이 있을지 계속해서 고민하고 구글링하던 중, 수업시간에 배우지 않은 Boosting방식의 model이 있다는 것을 알게 되었고 보다 늦게 만들어진 알고리즘인 LightGBM에 대해 공부하였고 이를 이용해 코딩을 하였습니다. 그 후 hyper parameter를 tuning해주면서 보다 accuracy를 높였고 최종적으로 0.917의 accuracy를 얻을 수 있었습니다. 처음부터 끝까지 학습을 시키고 테스팅을 하는 과정이 처음이다 보니, 코딩을 하고 수정하며 더 좋은 모델을 찾는 시간이 부족했던 것 같아 아쉬움이 남습니다. 하지만 이번 프로젝트를 통해 딥러닝에 자신감을 가질 수 있는 계기가 되었던 것 같습니다. 평소에는 어려움이 있을 때 다른 사람의 도움을 받아 극복하는 나태한 태도가 있었는데 이번 수업은 아는 사람없이 수업을 들어 많은 어려움을 스스로 학습하며 극복하였기에 순위는 높지 않지만 수업 내용이상의 공부를 스스로 하는 제 자신의 모습을 보며 뿌듯함을 느낄 수 있었습니다. 또 스스로 오류를 해결하고 더 높은 순위를 위해 도전하면서 대학에 입학하고 여러 코딩을 하면서도 느끼지 못했던 재미와 희열을 느낄 수 있었습니다. 비록 뛰어난 성적을 거두지는 못하였지만, 이 과정을 통해 제 스스로 성장하는 계기가 되었다고 생각하며 남은 수업도 열심히 수강하고 기말에는 노력뿐만 아니라 좋은 성적도 얻을 수 있도록 노력하겠습니다.