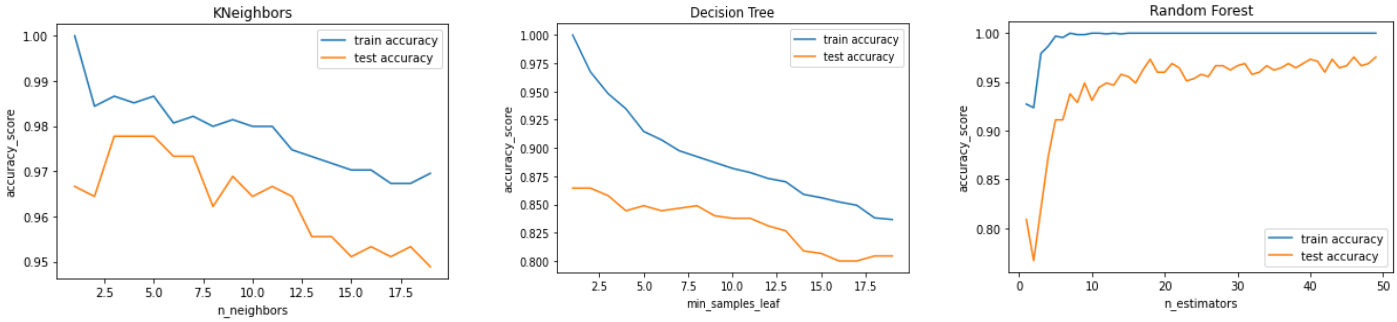
Assignment 2: Comparison of Supervised Learning Algorithms

"Optical recognition of handwritten digits dataset" 을 load_digits 를 사용해 불러왔다. 이 데이터셋은 sample 이 1797 개, dimensionality 가 64 개, class 수는 10 개인 multi-class classification 문제이다. 모든 알고리즘에 대한 데이터 전처리를 실행했다. train_test_split 을 사용했다. test_size 는 default 인 0.25 를 사용하였으며, stratify=data.target 을 사용해 class 의 비율을 공평하게 유지했고, random_state=2021 을 지정해 랜덤 값들을 고정했다. 그 이후, standard scaler 를 사용해 X_train 과 X_test 모두 scalining 했다. 그 후, 하이퍼 파라미터 값을 변경해가며 각 learning algorithm 에서의 최적의 파라미터를 찾은 후, 모든 learning algorithm 을 비교해 최적의 algorithm 을 찾아냈다. 평가지표는 sklearn.metrics 의 accuracy_score 를 사용했다.

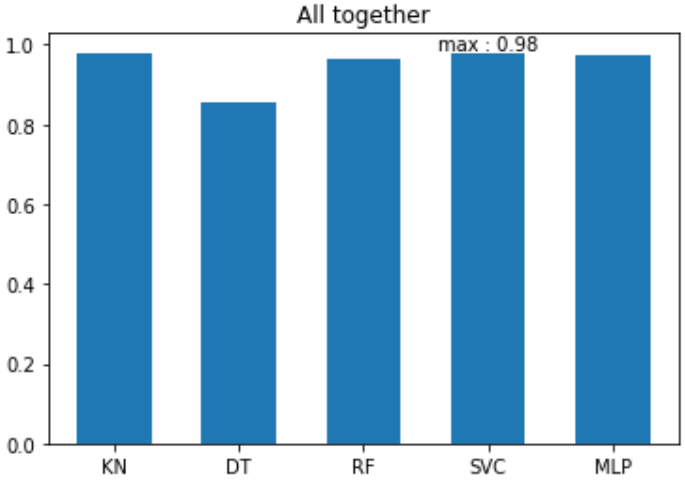


KNeighbors 는 k=3 일 때, Decision Tree 는 min_samples_leaf =3 일 때, Random Forest 는 n_estimators=21 일 때 최고의 accuracy_score 를 보였다.

SVC 는 C 를 0.01, 1, 100 으로 바꿔가며, gamma 값은 0.01, 0.1, 1 로 바꿔가며 실행해본 결과, C =100, gamma = 0.01 일 때 가장 높은 accuracy_score 를 보였다.

MLP 는 max_iter 가 141 까지 'ConvergenceWarning: Stochastic Optimizer: Maximum iterations (141) reached and the optimization hasn't converged yet.'의 오류를 내며 수렴하지 않았다. 142 부터 accuracy_score 를 측정했으며, 143 에서 최고의 accuracy_score 를 보였다.

모든 learning algorithm 을 최적의 parameter 로 학습해 결과를 낸 후, accuracy_score 를 비교해 보았다.



그리고 가장 높은 수치의 값에 수치를 표시하도록 코딩했다. SVC 의 accuracy_score 가 가장 높은 것을 볼 수 있었다.

결론적으로, digits dataset 의 최적의 learning algorithm 은 SVC 이며 Hypermarameter 는 C=100. Gamma 0.01 이었다.

```
import numpy as np
from matplotlib import pyplot as plt
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
digits = load_digits()

X_train, X_test, y_train, y_test = train_test_split(digits.data,digits.target,stratify=digits.target, random_state =2021)
scaler = StandardScaler()
scaler.fit(X_train)
X_train_sc = scaler.transform(X_train)
X_test_sc = scaler.transform(X_test)

values = dict()

k = 3
KNclf = KNeighborsClassifier(n_neighbors=k)
KNclf.fit(X_train_sc, y_train)
values[accuracy_score(y_test, KNclf.predict(X_test_sc))] = 'KN'

k = 3
DTclf = DecisionTreeClassifier(min_samples_leaf = k)
DTclf.fit(X_train_sc, y_train)
```

```
values[accuracy_score(y_test, DTclf.predict(X_test_sc))] = 'DT'


k = 21
RFclf = RandomForestClassifier(n_estimators=k)
RFclf.fit(X_train_sc, y_train)
values[accuracy_score(y_test, RFclf.predict(X_test_sc))] = 'RF'


C = 100
gamma = 0.01
SVCclf = SVC(C=C, kernel = 'rbf', gamma=gamma)
SVCclf.fit(X_train_sc, y_train)
values[accuracy_score(y_test, SVCclf.predict(X_test_sc))] = 'SVC'


k = 143
MLPclf = MLPClassifier(max_iter=k, random_state=2021)
MLPclf.fit(X_train_sc, y_train)
values[accuracy_score(y_test, MLPclf.predict(X_test_sc))] = 'MLP'


plt.bar(list(values.values()), list(values.keys()), width=0.6)
plt.title('All together')
plt.text(values[max(values.keys())],max(values.keys()),
        'max : ' + str(format(max(values.keys()), ".2f")),
        verticalalignment='bottom' , horizontalalignment='center')
plt.show()
```