# 머신러닝과 딥러닝

## Report7


소프트웨어학과

2016312568 정희윤

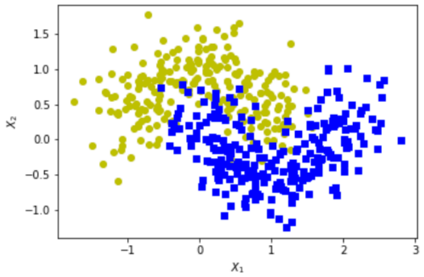

## 1. MNIST_knn분류

```
In [2]: X, y = mnist["data"], mnist["target"]
        X.shape
Out[2]: (70000, 784)

In [3]: y.shape
Out[3]: (70000,)

In [4]: X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]

In [5]: X_train.shape
Out[5]: (60000, 784)

In [6]: X_test.shape
Out[6]: (10000, 784)

In [7]: #Training set 순서 섞기 (shuffling)
        import numpy as np

        shuffle_index = np.random.permutation(60000)
        X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]

In [8]: shuffle_index
Out[8]: array([46203, 26510, 36046, ..., 32543, 42987, 52998])

In [9]: from sklearn.neighbors import KNeighborsClassifier
        knn_clf = KNeighborsClassifier(n_jobs=-1, weights='distance', n_neighbors=4)
        knn_clf.fit(X_train, y_train)
Out[9]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                   metric_params=None, n_jobs=-1, n_neighbors=4, p=2,
                   weights='distance')

In [10]: y_knn_pred = knn_clf.predict(X_test)

In [11]: from sklearn.metrics import accuracy_score
         accuracy_score(y_test, y_knn_pred)
Out[11]: 0.9714

In [ ]:
```

sklearn에서 MNIST original 데이터를 들고와서 60000개까지 knn으로 학습을 시킨다. 60000개 이후부터의 데이터셋으로 테스트를 하면 정확도가 97.14%정도 나온다.

## 2. makemoons_iris_ensemble

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=500, noise=0.30, random_state=42)
plt.plot(X[:, 0][y==0], X[:, 1][y==0], "yo")
plt.plot(X[:, 0][y==1], X[:, 1][y==1], "bs")
plt.xlabel("$X_1$")
plt.ylabel("$X_2$")
plt.show()
```




In [3]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

In [4]:
```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

bag_clf = BaggingClassifier(
    DecisionTreeClassifier(random_state=42), n_estimators=500,
    max_samples=100, bootstrap=True, n_jobs=-1, random_state=42)
bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```

In [5]:
```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

```
0.904
```

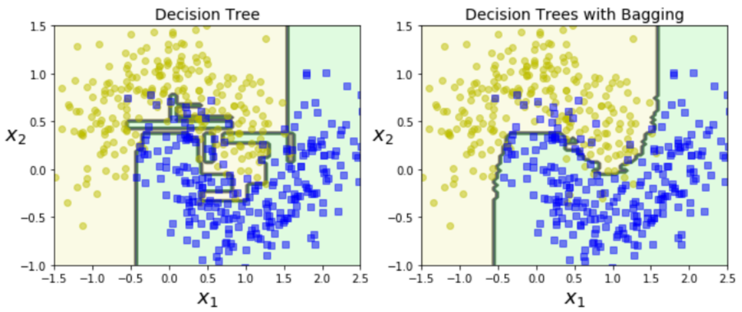
In [6]:
```
tree_clf = DecisionTreeClassifier(random_state=42)
tree_clf.fit(X_train, y_train)
y_pred_tree = tree_clf.predict(X_test)
print(accuracy_score(y_test, y_pred_tree))
```

```
0.856
```

make_moons dataset을 들고와서 Bagging을 사용하여 학습을 시킨다. 학습을 시킨뒤 테스트 데이터로 정확도를 보면 90.4%로 단일 decision tree의 정확도인 85.6%비해 높음을 알 수있다.

```
In [7]: from matplotlib.colors import ListedColormap

def plot_decision_boundary(clf, X, y, axes=[-1.5, 2.5, -1, 1.5], alpha=0.5, contour=True):
    x1s = np.linspace(axes[0], axes[1], 100)
    x2s = np.linspace(axes[2], axes[3], 100)
    x1, x2 = np.meshgrid(x1s, x2s)
    X_new = np.c_[x1.ravel(), x2.ravel()]
    y_pred = clf.predict(X_new).reshape(x1.shape)
    custom_cmap = ListedColormap(['#fafab0','#9898ff','#a0faa0'])
    plt.contourf(x1, x2, y_pred, alpha=0.3, cmap=custom_cmap)
    if contour:
        custom_cmap2 = ListedColormap(['#7d7d58','#4c4c7f','#507d50'])
        plt.contour(x1, x2, y_pred, cmap=custom_cmap2, alpha=0.8)
    plt.plot(X[:, 0][y==0], X[:, 1][y==0], "yo", alpha=alpha)
    plt.plot(X[:, 0][y==1], X[:, 1][y==1], "bs", alpha=alpha)
    plt.axis(axes)
    plt.xlabel(r"$x_1$", fontsize=18)
    plt.ylabel(r"$x_2$", fontsize=18, rotation=0)
```

```
In [8]: plt.figure(figsize=(11,4))
plt.subplot(121)
plot_decision_boundary(tree_clf, X, y)
plt.title("Decision Tree", fontsize=14)
plt.subplot(122)
plot_decision_boundary(bag_clf, X, y)
plt.title("Decision Trees with Bagging", fontsize=14)
plt.show()
```




학습된 결과를 표현한 결과이다. Bagging을 사용시 overfitting들이 많이 보완되었을 알 수 있다.

```
In [9]: bag_clf = BaggingClassifier(
            DecisionTreeClassifier(splitter="random", max_leaf_nodes=16, random_state=42),
            n_estimators=500, max_samples=1.0, bootstrap=True, n_jobs=-1, random_state=42)

        bag_clf.fit(X_train, y_train)
        y_pred = bag_clf.predict(X_test)
```

```
In [10]: from sklearn.ensemble import RandomForestClassifier

         rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16, n_jobs=-1, random_state=42)
         rnd_clf.fit(X_train, y_train)

         y_pred_rf = rnd_clf.predict(X_test)
```

```
In [11]: print(accuracy_score(y_test, y_pred))

         0.92
```

```
In [12]: print(accuracy_score(y_test, y_pred_rf))

         0.912
```

```
In [13]: np.sum(y_pred == y_pred_rf) / len(y_pred)
Out[13]: 0.976
```

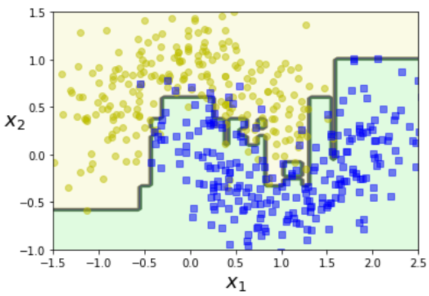
```
In [14]: from sklearn.datasets import load_iris
         from sklearn.ensemble import RandomForestClassifier

         iris = load_iris()
         rnd_clf = RandomForestClassifier(n_estimators=500, n_jobs=-1, random_state=42)
         rnd_clf.fit(iris["data"], iris["target"])
         for name, score in zip(iris["feature_names"], rnd_clf.feature_importances_):
             print(name, score)

         sepal length (cm) 0.11249225099876374
         sepal width (cm) 0.023119288282510326
         petal length (cm) 0.44103046436395765
         petal width (cm) 0.4233579963547681
```

```
In [15]: rnd_clf.feature_importances_
Out[15]: array([0.11249225, 0.02311929, 0.44103046, 0.423358  ])
```

위는 Iris의 데이터셋에 Randomforest를 활용하여 나온 결과이다.

```
In [17]: from sklearn.ensemble import AdaBoostClassifier

         ada_clf = AdaBoostClassifier(
             DecisionTreeClassifier(max_depth=1), n_estimators=200,
             algorithm="SAMME.R", learning_rate=0.5, random_state=42)
         ada_clf.fit(X_train, y_train)
Out[17]: AdaBoostClassifier(algorithm='SAMME.R',
                   base_estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=1,
                     max_features=None, max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, presort=False, random_state=None,
                     splitter='best'),
                   learning_rate=0.5, n_estimators=200, random_state=42)
```

```
In [18]: plot_decision_boundary(ada_clf, X, y)
```




위는 AdaBoost 분류를 사용하여 학습을 시킨 결과이다.