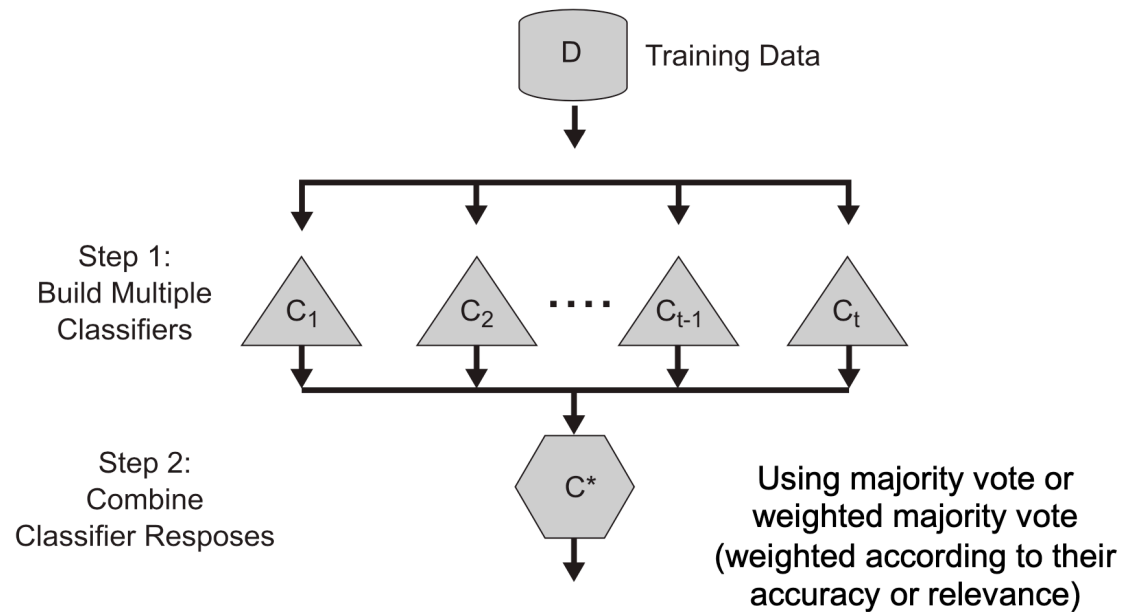


```
In [1]: import numpy as np
import matplotlib.pyplot as plt
```

Basic Idea of ensemble learning



Types of Ensemble learning

Ensemble learning is usually divided in 2 types

1. Bagging
2. Boosting

Bagging

The algorithm can be expressed as a set of following general steps:

- 1: Let k be the number of bootstrap samples.
- 2: **for** $i = 1$ to k **do**
- 3: Create a bootstrap sample of size N , D_i .
- 4: Train a base classifier C_i on the bootstrap sample D_i .
- 5: **end for**
- 6: $C^*(x) = \operatorname{argmax}_y \sum_i \delta(C_i(x) = y)$.
 $\{\delta(\cdot) = 1$ if its argument is true and 0 otherwise. $\}$

```
In [19]: from sklearn.datasets import load_breast_cancer
         from sklearn.model_selection import train_test_split
```

```
In [20]: cancer = load_breast_cancer()
```

```
In [21]: X = cancer.data
         y = cancer.target
         X.shape, y.shape
```

```
Out[21]: ((569, 30), (569,))
```

```
In [22]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ran
```

```
In [23]: len(y_train), len(y_test)
```

```
Out[23]: (398, 171)
```

Bagging

```
In [26]: from sklearn.neighbors import KNeighborsClassifier
```

```
In [27]: ## bootstrap samples
         k = 10
         nbh = [5, 10, 15, 20, 50, 100, 200, 250, 300, 398]
```

```
In [28]: pred_array = []
         for i in range(k):
             ind = [np.random.randint(len(y_train)) for _ in range(len(y_train))]
             D_x = X_train[ind,:]
             D_y = y_train[ind]
             nn = KNeighborsClassifier(n_neighbors=nbh[i])
             nn.fit(D_x,D_y)
             pred = nn.predict(X_test)
             pred_array.append(pred)

         pred_array = np.array(pred_array)
```

```
In [30]: pred_array.shape
```

```
Out[30]: (10, 171)
```

```
In [33]: sum_array = np.mean(pred_array,axis = 0)
         sum_array.shape
```

```
Out[33]: (171,)
```

In [34]: `sum_array`

Out[34]: `array([0.6, 1. , 1. , 0.7, 1. , 1. , 1. , 1. , 1. , 1. , 1. , 1. , 1. , 1. ,
' 0.4, 0.9, 0.4, 1. , 0.1, 0.3, 0.1, 0.5, 0.4, 1. , 1. , 0.2, 1. ,
' 1. , 1. , 1. , 0.2, 1. , 0.1, 1. , 0.4, 1. , 0.5, 1. , 0.3, 1. ,
' 0.4, 0.5, 1. , 0.4, 1. , 0.8, 0.1, 1. , 1. , 1. , 0.4, 0.3, 1. ,
' 0.3, 1. , 1. , 1. , 1. , 1. , 1. , 0.1, 0.7, 0.3, 1. , 1. , 0. ,
3, 1. , 0.1, 0.3, 0.2, 1. , 0.7, 0.3, 1. , 1. , 0.3, 1. , 1. , 1. ,
' 1. , 1. , 0.4, 0.2, 0.3, 1. , 0.4, 1. , 1. , 1. , 0.3, 0.1, 1. ,
' 0.4, 0.9, 0.5, 1. , 1. , 0.2, 1. , 1. , 1. , 1. , 1. , 1. , 1. ,
' 0.3, 1. , 0.3, 0.9, 0.4, 0.4, 1. , 0.3, 0.3, 0.9, 1. , 1. , 0. ,
4, 1. , 1. , 1. , 1. , 1. , 0.9, 1. , 0.2, 1. , 0.7, 1. , 1. , 1. ,
' 0.3, 1. , 1. , 1. , 1. , 1. , 1. , 0.4, 0.1, 1. , 1. , 1. , 0. ,
1, 1. , 1. , 0.2, 0.9, 0.4, 1. , 1. , 1. , 1. , 1. , 1. , 1. , 0. ,
5, 1. , 0.6, 1. , 0.3, 0.2, 1. , 1. , 0.3, 1. , 0.3, 0.5, 0.3, 1. ,
' 1. , 1.])`

In [37]: `final_pred = np.round(sum_array)`

In [36]: `from sklearn.metrics import accuracy_score`

In [38]: `accuracy_score(y_test, final_pred)`

Out[38]: `0.9473684210526315`

In []: