Support Vector Machine (SVM)

Support Vector Machine (SVM) is a supervised learning algorithm commonly used for classification and regression tasks. The key idea behind SVM is to find a **hyperplane** that best separates different classes in the feature space while maximizing the margin between the nearest data points (support vectors) from each class. The optimal hyperplane is the one that leaves the largest margin between the support vectors, which helps improve the model's generalization ability.

Support Vector Classifier (SVC)

- **SVC** is the classification variant of the SVM algorithm. It works by finding the optimal hyperplane that separates different classes in the dataset.
- The decision boundary is represented as a hyperplane in higher-dimensional space, and SVC tries to maximize the margin between the classes.
- It supports both **linear** and **non-linear classification**. In the case of non-linearly separable data, SVC uses **kernel functions** (like RBF, polynomial, sigmoid) to project the data into a higher dimension where a linear separator can be applied.

Support Vector Regressor (SVR)

- **SVR** is the regression variant of the SVM algorithm. Instead of finding a hyperplane to classify data, SVR tries to fit the best line (or hyperplane) within a margin such that most of the data points lie within a specified distance from it.
- It aims to minimize the error, considering the margin of tolerance (epsilon). This makes SVR robust to outliers in the dataset.

Support Vectors

- **Support Vectors** are the data points that are closest to the hyperplane (decision boundary). These points are crucial because they are the ones that determine the position and orientation of the hyperplane.
- The **margin** is the distance between the hyperplane and the nearest support vector from either class. SVM tries to maximize this margin to improve model robustness and generalization.

Difference Between Logistic Regression and Support Vector Classifier (SVC)

1. **Decision Boundary**:
    - **Logistic Regression**: Logistic regression creates a **linear decision boundary** between classes by estimating the probabilities of class membership and then applying a threshold (usually 0.5).

- **SVC**: SVC can create both **linear and non-linear decision boundaries**. If the data is linearly separable, it creates a linear hyperplane. For non-linear data, it uses kernel functions to map the data to a higher-dimensional space.

2. **Optimization Objective**:

   - **Logistic Regression**: Logistic regression minimizes the **log loss** (or cross-entropy loss), which is a probabilistic model focusing on fitting the data and estimating the likelihood of each class.
   - **SVC**: SVC focuses on maximizing the **margin** between classes. It finds the hyperplane that maximizes the margin between the support vectors, improving the classification's robustness.

3. **Probabilistic vs. Non-Probabilistic**:

   - **Logistic Regression**: It is a **probabilistic** classifier that predicts the probability of a data point belonging to a class.
   - **SVC**: SVC is a **non-probabilistic** classifier, and its output is based on the decision boundary (hyperplane). However, SVC can output probabilities using techniques like **Platt scaling**.

4. **Kernel Trick**:

   - **Logistic Regression**: Logistic regression does not use kernel functions and is typically limited to **linear** decision boundaries unless feature transformations are applied manually.
   - **SVC**: SVC can handle **non-linear** data through the **kernel trick**, which implicitly maps data into a higher dimension to find a linear separator.

5. **Handling Outliers**:

   - **Logistic Regression**: It is sensitive to **outliers**, as they can influence the fit of the decision boundary.
   - **SVC**: SVC is more **robust to outliers** due to its focus on maximizing the margin between support vectors, which are often not outliers.

6. **Application**:

   - **Logistic Regression**: Used for binary classification problems, especially where the classes are linearly separable or the relationship between features and output is mostly linear.
   - **SVC**: Used in cases where the data may not be linearly separable, and kernel methods can be employed to map data into a higher-dimensional space to find the best separating hyperplane.

Key Parameters of SVM

1. **C (Regularization Parameter)**:

   - **C** controls the trade-off between having a smooth decision boundary and classifying training points correctly. A **small C** creates a larger margin hyperplane but allows more misclassifications. A **large C** will try to classify all training examples correctly, leading to a smaller margin.

2. **Kernel**:
    - The kernel function is used to transform the input data into a higher-dimensional space where it is easier to separate the classes. Popular kernels include:
        - **Linear Kernel**: For linearly separable data.
        - **Polynomial Kernel**: For polynomial decision boundaries.
        - **Radial Basis Function (RBF) Kernel**: For complex non-linear data.
        - **Sigmoid Kernel**: For certain specific problems.
3. **Gamma**:
    - **Gamma** defines how far the influence of a single training example reaches. A low gamma value means that data points far away from the hyperplane will be considered in determining the boundary, while a high gamma value means only nearby points will affect the boundary.

Ensemble Techniques: Bagging

Support Vector Machines can be combined with ensemble methods, but SVM itself is not inherently an ensemble model. **Bagging (Bootstrap Aggregation)**, an ensemble method, can be applied to SVM by training multiple SVMs on different bootstrap samples and then aggregating their predictions.

Advantages of SVM

- **Effective in high-dimensional spaces**.
- **Memory efficient** because only a subset of the training points (support vectors) is used to determine the hyperplane.
- **Versatile**: SVM can be adapted for both classification and regression problems, and with the right kernel, it works well for non-linear data.

Disadvantages of SVM

- **Not suitable for large datasets**: The training time complexity is higher, which makes SVM slow for large datasets.
- **Sensitive to the choice of hyperparameters**: Parameters like `C`, `gamma`, and the kernel type significantly impact performance.
- **Poor performance with noisy data**: SVM can struggle with overlapping classes and data with noise.

Normalization

- **Normalization** is crucial when using SVM, as it helps in handling features with different scales. Without normalization, features with larger numerical ranges may dominate the decision boundary.
- Techniques like **StandardScaler** (standardizing features by removing the mean and scaling to unit variance) or **MinMaxScaler** (scaling features to a given range,

typically [0, 1]) are often used before training SVM models.

Conclusion

- **SVC** and **SVR** are powerful variants of SVM for classification and regression tasks, respectively. Their ability to find an optimal hyperplane and handle non-linear data using kernels makes them highly flexible.
- **Logistic Regression** is more interpretable and works well for linearly separable problems, while **SVC** is more powerful for complex, non-linear problems with the help of kernel functions.

The Mathematics Behind Support Vector Machines (SVM)

Support Vector Machines (SVMs) are a class of supervised learning models that are used for both classification and regression tasks. The core idea behind SVMs is to find a hyperplane in a high-dimensional space that separates data points of different classes with the maximum margin.

Linearly Separable Case

In the simplest case where the data is linearly separable, SVM aims to find the hyperplane that maximizes the distance between the closest points of the two classes. This distance is called the **margin**.

Equation of a Hyperplane:

```
w^T * x + b = 0
```

where:

- `w` is the normal vector to the hyperplane
- `x` is a data point
- `b` is the bias term

Margin:

The margin is defined as the distance between the hyperplane and the closest data points. The goal of SVM is to maximize this margin.

Optimization Problem:

The SVM optimization problem can be formulated as follows:

```
maximize: margin
subject to: y_i * (w^T * x_i + b) >= 1 for all i
```

where:

- `y_i` is the class label of data point `i`
- `x_i` is the feature vector of data point `i`

Non-Linearly Separable Case

In many real-world scenarios, the data may not be linearly separable. To handle such cases, SVM uses a technique called **kernel trick**. The kernel trick maps the data into a higher-dimensional feature space where it might become linearly separable.

Kernels: Common kernels include:

- Linear kernel
- Polynomial kernel
- Radial basis function (RBF) kernel

Regularization

To prevent overfitting, SVM often incorporates regularization. This involves adding a penalty term to the optimization problem that controls the complexity of the model.

Regularized SVM:

```
maximize: margin
subject to: y_i * (w^T * x_i + b) >= 1 for all i
minimize: ||w||^2
```

where `||w||^2` is the L2 norm of the weight vector.

In summary, SVMs are powerful machine learning models that can handle both linear and non-linear classification and regression tasks. The underlying mathematics involves finding the optimal hyperplane that separates the data with the maximum margin, often using techniques like kernels and regularization to improve performance.

Import all the required frameworsk

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Load the data

In [2]:
```
data = pd.read_csv('./WineQT.csv')
data.head()
```

Out[2]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulp
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	

Note: EDA implementation is skipped in this pipeline.

Data Splitting: Split the data into input features and target feature

In [3]:
```
## input features
X = data.drop('quality', axis=1)
## target feature
y = data['quality']
```

Data Splitting: Split the data into train and test set

In [4]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
```

Feature Scaling: StandardScaler, convert the entire feature set into standard normal form. It simply means that mean = 0 and std = 1

In [5]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Data Modeling: SVC classifier

In [6]:
```
from sklearn.svm import SVC
svm_classifier = SVC()
svm_classifier.fit(X_train, y_train)
svm_classifier.predict(X_test)
```

Out[6]:
```
array([5, 6, 6, 7, 5, 6, 5, 7, 6, 6, 5, 6, 6, 6, 5, 6, 6, 6, 5, 5, 5, 5,
       5, 5, 5, 6, 6, 5, 6, 6, 6, 5, 7, 5, 6, 5, 6, 6, 6, 5, 5, 5, 6, 5,
       5, 5, 5, 6, 5, 5, 5, 6, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6, 6,
       5, 7, 5, 6, 6, 6, 5, 7, 6, 6, 5, 5, 5, 5, 6, 6, 5, 6, 5, 6, 6, 6,
       5, 5, 6, 5, 5, 6, 5, 5, 5, 5, 6, 6, 6, 6, 5, 6, 5, 6, 6, 6, 5, 6,
       6, 6, 6, 5, 6, 5, 6, 5, 7, 5, 5, 6, 5, 7, 5, 6, 5, 6, 6, 5, 6, 6,
       6, 5, 6, 6, 5, 6, 7, 6, 5, 6, 6, 5, 5, 5, 5, 5, 5, 6, 5, 5, 6, 6,
       5, 5, 6, 5, 5, 5, 5, 6, 5, 6, 5, 5, 5, 6, 7, 7, 5, 5, 5, 6, 6, 5,
       7, 5, 6, 5, 6, 5, 6, 5, 5, 5, 6, 7, 6, 6, 6, 6, 6, 7, 5, 5, 7, 7,
       6, 6, 6, 5, 5, 5, 5, 6, 5, 5, 5, 5, 6, 5, 6, 5, 5, 5, 5, 5, 6, 5,
       5, 6, 6, 6, 6, 6, 5, 7, 6])
```

In [7]:
```
svm_classifier.score(X_test, y_test)
```

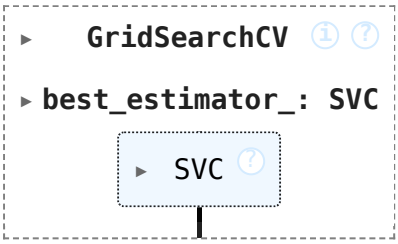
Out[7]:
```
0.6200873362445415
```

Hyperparameter tuning: GridSearchCV or RandomizedSearchCV

In [8]:
```
param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf', 'poly'],
    'gamma': ['scale', 'auto'],
    'degree': [2, 3, 4]
}

## to apply gridsearch cv
from sklearn.model_selection import GridSearchCV
grid_search = GridSearchCV(svm_classifier, param_grid, cv=5)
grid_search.fit(X_train, y_train)
```

Out[8]:

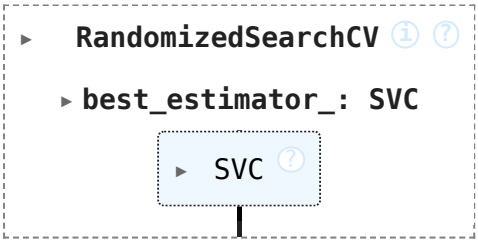


In [9]:
```
grid_search.best_params_
```

Out[9]:
```
{'C': 1, 'degree': 2, 'gamma': 'scale', 'kernel': 'rbf'}
```

In [12]:
```
## to apply the randomized search cv
from sklearn.model_selection import RandomizedSearchCV
random_search = RandomizedSearchCV(svm_classifier, param_grid, cv=5)
random_search.fit(X_train, y_train)
```

Out[12]:

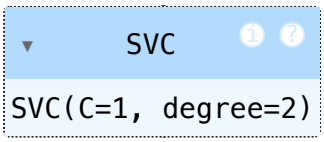


In [13]:
```
random_search.best_params_
```

Out[13]:
```
{'kernel': 'linear', 'gamma': 'scale', 'degree': 3, 'C': 1}
```

In [14]:
```
## train the model with the best optimal parameters
svm_classifier_optimizedparams = SVC(C=1, gamma='scale', kernel='rbf', de
svm_classifier_optimizedparams.fit(X_train, y_train)
```

Out[14]:



In [15]:
```
svm_classifier_optimizedparams.score(X_test, y_test)
```

Out[15]:
```
0.6200873362445415
```

One can work on Classification report: Accuracy, Recall, Precision and many more