Introduction to Support Vector Machine

`Support Vector Machine` is a supervised learning algorithm which can be used for both classification and regression challenges. However, it is mostly used in classification problems. In this article, I will be discussing about the `Support Vector Machine` algorithm and how it works.

- LinkedIn
- YouTube
- gtihub
- Gmail
- discord

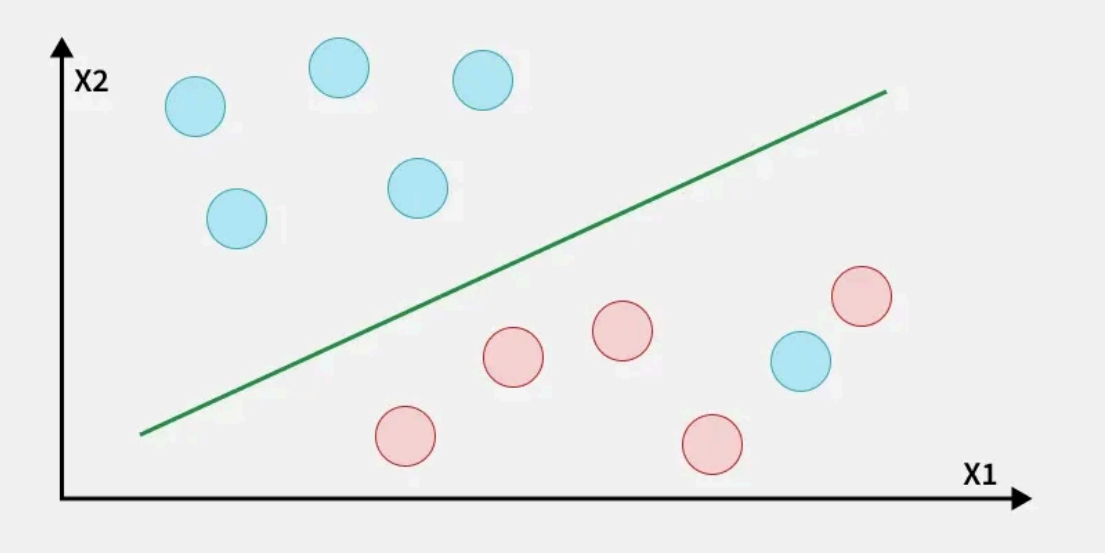
Introduction

`Support Vector Machine (SVM)` is a simple yet powerful machine learning algorithm that can be used for both `classification` and `regression` tasks.

In most cases, `SVM` is introduced as a `binary classification` algorithm. Given a set of training examples, where each example belongs to one of `two classes`, the `SVM` algorithm learns a rule that assigns new, unseen examples to one class or the other. Since the decision is made based on geometry rather than probabilities, `SVM` is often described as a `non-probabilistic linear classifier`.

An `SVM` model represents data points as vectors in a feature space. The main idea is to position these points in such a way that the two classes can be separated by a boundary, while keeping that boundary as far away as possible from the closest points of each class. Once this boundary is learned, new data points are mapped into the same space and classified based on which side of the boundary they fall on.

More formally, consider a dataset that looks like this:



Here, we have `orange` and `blue` points representing two different classes. At first glance, it seems easy to separate them by drawing a straight line. However, if you look closely, you'll notice that there are actually `many possible lines` that could separate these two classes.

So the natural question becomes:

Which line should we choose?

The line chosen by `SVM` is the one that `maximizes the distance between the two classes`. This optimal separating line is called the `maximum margin hyperplane`. The distance between this line and the closest data points from each class is known as the `margin`.

The data points that lie closest to the hyperplane are especially important. These points are called `support vectors`, and they are the only points that directly influence the position of the hyperplane. The goal of `SVM` is to choose a hyperplane that results in the `largest possible margin`, making the model more robust to small variations in the data.

At this point, you might wonder:

What if the data is not `linearly separable`?

In real-world problems, data is often messy and cannot be perfectly separated by a straight line. To handle this situation, `SVM` uses a technique called the `kernel trick`. Instead of separating the data in its original space, `SVM` transforms the data into a `higher-dimensional space` where a linear separation becomes possible. This process is commonly referred to as `kernelization`.

In this article, we will discuss:

- how the `SVM` algorithm works conceptually
- how to implement `SVM` in `python` using `scikit-learn`
- and how the `kernel trick` helps `SVM` handle non-linear data

How `SVM` Works

The `SVM` algorithm tries to find a `hyperplane` that best separates the dataset into two classes. In a two-dimensional space, this hyperplane is simply a line. In higher dimensions, it becomes a plane or, more generally, a `hyperplane`.

The hyperplane is chosen such that the `margin between the two classes` is maximized. The margin is defined as the distance between the hyperplane and the closest data points from each class. As mentioned earlier, these closest points are called `support vectors`.

Mathematically, the hyperplane can be represented as:

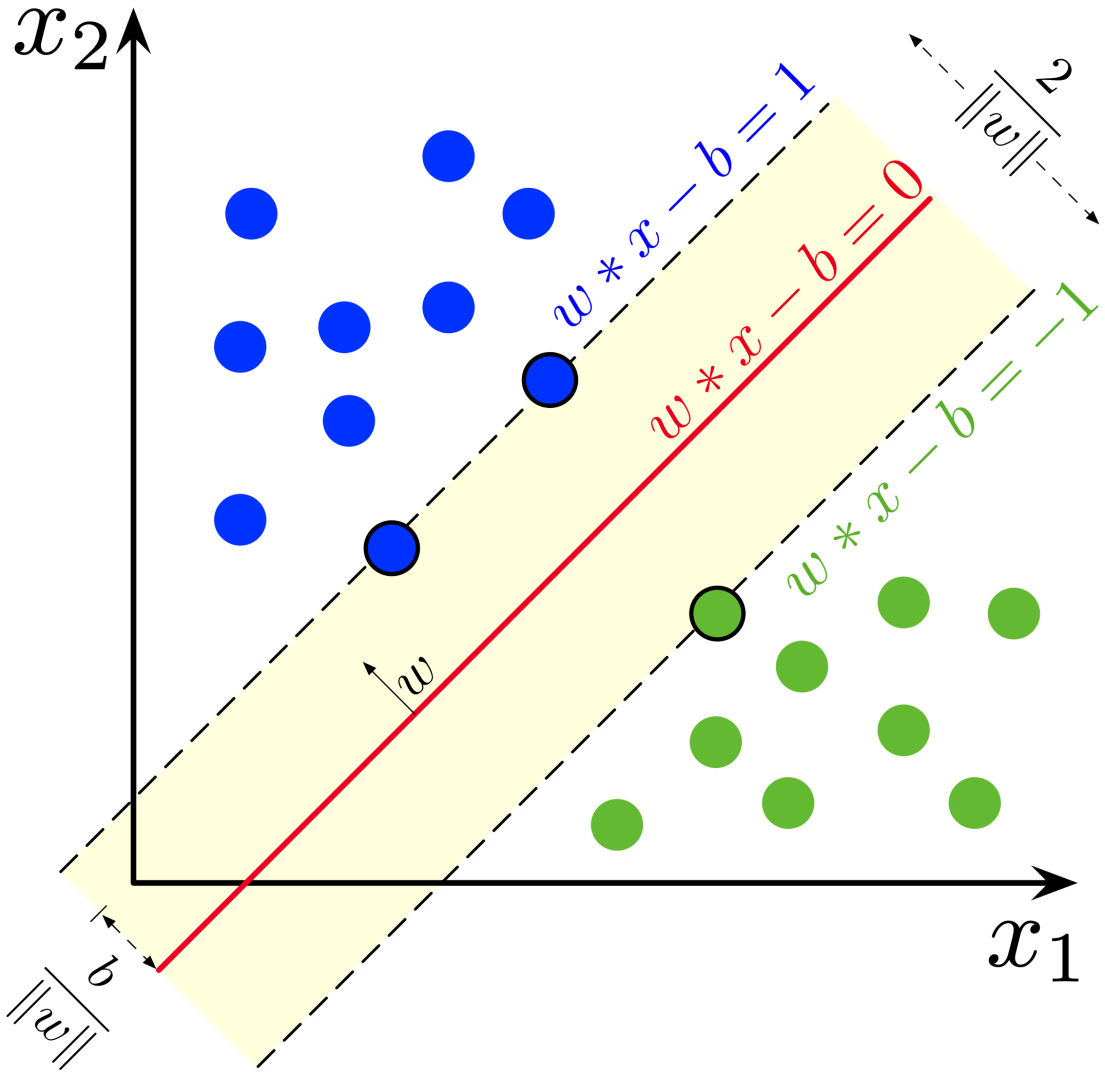
$$w^T x - b = 0$$

Here,

- `w` is the normal vector perpendicular to the hyperplane
- `b` is the bias term
- `x` represents a point in the feature space

This equation divides the feature space into two regions. Points lying on one side of the hyperplane are assigned to one class, while points on the other side are assigned to the second class.

The following visualization can help clarify this idea:



There is a considerable amount of math involved in the `SVM` algorithm, especially when optimization and kernels are discussed. However, the core intuition remains simple: `SVM` tries to place the decision boundary as far away as possible from the nearest data points of each class.

During training, the algorithm iteratively searches for this optimal hyperplane by maximizing the margin while allowing small misclassifications when necessary. This balance between `margin maximization` and `error tolerance` is what makes `SVM` both powerful and reliable.

> Explaining the maths without any visualization is pretty hard.

> The internal mathematics of `SVM` are complex, but the core idea remains simple: `SVM` tries to place the decision boundary as far away as possible from the nearest data points of each class. You can watch an amazing 3 part video StatQuest with Josh Starmer. Amazing channel btw.

Implementing `SVM` in `python`

First we need Data

In [56]:
```
import pandas as pd
import numpy as np
```

I'll use the `breast cancer dataset` from `scikit-learn` to train the model.

In [57]:
```
from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer(as_frame=True)
```

The `dataset` is leaded and stored in the `cancer` variable. But we need to extract it from the variable.

> it's a dictionary-like object that holds all the data and some metadata about the data. This data is stored in the `.data` member, which is a `n_samples`, `n_features` array. In the case of supervised problem, one or more response variables are stored in the `.target` member.

We can check out it's keys as it is a dictionary-like object.

In [58]:
```
cancer.keys()
```

Out[58]:
```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_n
ames', 'filename', 'data_module'])
```

The dataset has `data`, `target`, `frame`, `target_names`, `DESCR`, `feature_names`, `filename` etc. Not all of them are important for us. We will only use `data` and `target` to train the model and `feature_names` and `target_names` to make the data more readable.

SO, I'll start by seeing and `description` of the dataset.

In [59]:
```
print(cancer['DESCR'])
```

.. _breast_cancer_dataset:

Breast cancer Wisconsin (diagnostic) dataset

Data Set Characteristics:

:Number of Instances: 569

:Number of Attributes: 30 numeric, predictive attributes and the class

:Attribute Information:
    - radius (mean of distances from center to points on the perimeter)
    - texture (standard deviation of gray-scale values)
    - perimeter
    - area
    - smoothness (local variation in radius lengths)
    - compactness (perimeter^2 / area - 1.0)
    - concavity (severity of concave portions of the contour)
    - concave points (number of concave portions of the contour)
    - symmetry
    - fractal dimension ("coastline approximation" - 1)

    The mean, standard error, and "worst" or largest (mean of the three
    worst/largest values) of these features were computed for each image,
    resulting in 30 features. For instance, field 0 is Mean Radius, field
    10 is Radius SE, field 20 is Worst Radius.

    - class:
            - WDBC-Malignant
            - WDBC-Benign

:Summary Statistics:

	Min	Max
radius (mean):	6.981	28.11
texture (mean):	9.71	39.28
perimeter (mean):	43.79	188.5
area (mean):	143.5	2501.0
smoothness (mean):	0.053	0.163
compactness (mean):	0.019	0.345
concavity (mean):	0.0	0.427
concave points (mean):	0.0	0.201
symmetry (mean):	0.106	0.304
fractal dimension (mean):	0.05	0.097
radius (standard error):	0.112	2.873
texture (standard error):	0.36	4.885
perimeter (standard error):	0.757	21.98
area (standard error):	6.802	542.2
smoothness (standard error):	0.002	0.031
compactness (standard error):	0.002	0.135
concavity (standard error):	0.0	0.396
concave points (standard error):	0.0	0.053
symmetry (standard error):	0.008	0.079
fractal dimension (standard error):	0.001	0.03
radius (worst):	7.93	36.04
texture (worst):	12.02	49.54
perimeter (worst):	50.41	251.2
area (worst):	185.2	4254.0

```
smoothness (worst):                   0.071  0.223
compactness (worst):                  0.027  1.058
concavity (worst):                    0.0    1.252
concave points (worst):               0.0    0.291
symmetry (worst):                     0.156  0.664
fractal dimension (worst):            0.055  0.208
==================================== ====== ======

:Missing Attribute Values: None

:Class Distribution: 212 - Malignant, 357 - Benign

:Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian

:Donor: Nick Street

:Date: November, 1995

This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.
https://goo.gl/U2Uwz2

Features are computed from a digitized image of a fine needle
aspirate (FNA) of a breast mass.  They describe
characteristics of the cell nuclei present in the image.

Separating plane described above was obtained using
Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree
Construction Via Linear Programming." Proceedings of the 4th
Midwest Artificial Intelligence and Cognitive Science Society,
pp. 97-101, 1992], a classification method which uses linear
programming to construct a decision tree.  Relevant features
were selected using an exhaustive search in the space of 1-4
features and 1-3 separating planes.

The actual linear program used to obtain the separating plane
in the 3-dimensional space is that described in:
[K. P. Bennett and O. L. Mangasarian: "Robust Linear
Programming Discrimination of Two Linearly Inseparable Sets",
Optimization Methods and Software 1, 1992, 23-34].

This database is also available through the UW CS ftp server:

ftp ftp.cs.wisc.edu
cd math-prog/cpo-dataset/machine-learn/WDBC/

.. dropdown:: References

  - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extract
ion
    for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on
    Electronic Imaging: Science and Technology, volume 1905, pages 861-870,
    San Jose, CA, 1993.
  - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis
and
    prognosis via linear programming. Operations Research, 43(4), pages 570
-577,
    July-August 1995.
  - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techn
iques
    to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77
(1994)
```

163-171.

The description shows that the dataset has `569` samples and `30` features. The features are the characteristics of the cell nuclei present in the breast cancer. The target is `binary` and represents the diagnosis of the cancer. `0` represents `malignant` and `1` represents `benign`.

Now let's get the data.

In [60]:
```
data = cancer.frame
data
```

Out[60]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430
...	...	...	...	...	...	...	...	...
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000

569 rows × 31 columns

The dataframe is made. Now let's see so details.

In [61]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
 30  target                   569 non-null    int64  
dtypes: float64(30), int64(1)
memory usage: 137.9 KB
```

The data is already clean and ready to be used and as there is a lot of features, I'll be lazy and not do any kind of preprocessing.

Let's split the data

In [62]:
```
from sklearn.model_selection import train_test_split

features = data.drop(columns=['target'])
target = data['target']

X_train, X_test, Y_train, Y_test = train_test_split(features, target, test
```

Now let's train the model.

Training the model

In [63]:
```
from sklearn.svm import SVC
```

```
model = SVC()

model.fit(X_train, Y_train)
```

Out[63]:

SVC

Parameters

The model is trained and and we can now do some predictions and see how well the model performs.

In [64]:
```
# predicting
predictions = model.predict(X_test)

# Evaluating the model
from sklearn.metrics import classification_report, confusion_matrix

print(confusion_matrix(Y_test, predictions))
print('\n')
print(classification_report(Y_test, predictions))
```

```
[[ 56  10]
 [  3 102]]


              precision    recall  f1-score   support

           0       0.95      0.85      0.90        66
           1       0.91      0.97      0.94       105

    accuracy                           0.92       171
   macro avg       0.93      0.91      0.92       171
weighted avg       0.93      0.92      0.92       171
```

Well, the model has an accuracy of `92%`. This is a good accuracy and most of the time you will get a good accuracy with `SVM` model.

But is this the best accuracy? should we change the hyperparameters and see if we can get a better accuracy?

It's a good choice to actually try out different hyperparameters and see if we can get a better accuracy. But there is a issue.

Let's see how many parameters are there in the `SVM` model.

In [65]:
```
SVC().get_params()
```

Out[65]:
```
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
```

That's a lot of parameters.

First let me explain what these hyperparameters are.

- `C` is the `penalty` parameter of the `error term`. It controls the trade off between smooth decision boundary and classifying the training points correctly. A large `C` value gives you low bias and high variance and a small `C` value gives you high bias and low variance.
- `gamma` is the `kernel` coefficient for `rbf`, `poly` and `sigmoid`. It controls the influence of a single training example. A large `gamma` value gives you high bias and low variance and a small `gamma` value gives you low bias and high variance. (You have to learn the math behind these to understand these fancy terms)
- `kernel` is the `kernel` type to be used in the algorithm. It can be `linear`, `poly`, `rbf` and `sigmoid`.
- `degree` is the blah blah blah.
- blah blah blah.
- blah

> You have the task to find out which parameter does what.

There you go. You have the knowledge of the hyperparameters.

So, tuning these parameters and testing them one by one would be a hastle right?

This is where another cross validation method comes into play.

It's called `GRID SEARCH!`.

> This is a cross validation method as well and i've talked about cross validation in a previous article. Please have a look at it. For now, cross validation validates the models performance for the whole dataset by splitting the dataset into multiple folds and testing the model on each fold and then averaging the results.

This is a automated process of testing out all the possible combinations of hyperparameters from a `list of HP(hyperparameters)` and finding the best combination of hyperparameters that gives the best accuracy.

Sklearn provides a `grid search` algorithm to find the best hyperparameters for the model.

In [66]:
```
#importing the grid search cv
from sklearn.model_selection import GridSearchCV
```

Now how do we use `GridSearchCV` to find the best hyperparameters for the model?

- We will first make a dictionary of the hyperparameters and their values that we want to try as a list.
- Then we will make a `grid` variable using the `GridSearchCV` class and pass the model, the hyperparameters and the number of `cross validation` to it.

Now let's make a parameter grid.

In [67]:
```
param_grid = {'C': [0.1, 1, 10, 100, 1000], 'gamma': [1, 0.1, 0.01, 0.001,
```

Now, we can pass this parameter grid to the `grid search` algorithm using the `GridSearchCV` class.

In [68]:
```
gridSVM = GridSearchCV(
    model,
    param_grid,
    verbose=3
)
```

First we made a dictionary of the hyperparameters:

`param_grid = {'C': [0.1, 1, 10, 100, 1000], 'gamma': [1, 0.1, 0.01, 0.001, 0.0001]}`
Here we are trying `5` different values for `C` and `5` different values for `gamma`. So, we are trying `25` different combinations of hyperparameters.

Which will be tested by the grid search algorithm using cross validation. So, the model will be trained `25` * `k` = `125` times.

> K stands for `k-fold cross validation`.

Then we define the `grid` variable using the `GridSearchCV` class.

First parameter is the model, second parameter is the hyperparameters, third parameter is the `refit` parameter which is set to `True` and the fourth parameter is the `verbose` parameter which is set to `3`.

The `refit` parameter is set to `True` because we want to refit the model with the best hyperparameters on the whole dataset.

The `verbose` parameter is set to `3` to see the progress of the `grid search` algorithm.

Now we can fit the `grid` variable to the training data.

In [69]:
```
gridSVM.fit(features, target)
```

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits
[CV 1/5] END ..................C=0.1, gamma=1;, score=0.623 total time=
0.0s
[CV 2/5] END ..................C=0.1, gamma=1;, score=0.623 total time=
0.0s
[CV 3/5] END ..................C=0.1, gamma=1;, score=0.632 total time=
0.0s
[CV 4/5] END ..................C=0.1, gamma=1;, score=0.632 total time=
0.0s
[CV 5/5] END ..................C=0.1, gamma=1;, score=0.628 total time=
0.0s
[CV 1/5] END ................C=0.1, gamma=0.1;, score=0.623 total time=
0.0s
[CV 2/5] END ................C=0.1, gamma=0.1;, score=0.623 total time=
0.0s
[CV 3/5] END ................C=0.1, gamma=0.1;, score=0.632 total time=
0.0s
[CV 4/5] END ................C=0.1, gamma=0.1;, score=0.632 total time=
0.0s
[CV 5/5] END ................C=0.1, gamma=0.1;, score=0.628 total time=
0.0s
[CV 1/5] END ...............C=0.1, gamma=0.01;, score=0.623 total time=
0.0s
[CV 2/5] END ...............C=0.1, gamma=0.01;, score=0.623 total time=
0.0s
[CV 3/5] END ...............C=0.1, gamma=0.01;, score=0.632 total time=
0.0s
[CV 4/5] END ...............C=0.1, gamma=0.01;, score=0.632 total time=
0.0s
[CV 5/5] END ...............C=0.1, gamma=0.01;, score=0.628 total time=
0.0s
[CV 1/5] END ..............C=0.1, gamma=0.001;, score=0.623 total time=
0.0s
[CV 2/5] END ..............C=0.1, gamma=0.001;, score=0.623 total time=
0.0s
[CV 3/5] END ..............C=0.1, gamma=0.001;, score=0.632 total time=
0.0s
[CV 4/5] END ..............C=0.1, gamma=0.001;, score=0.632 total time=
0.0s
[CV 5/5] END ..............C=0.1, gamma=0.001;, score=0.628 total time=
0.0s
[CV 1/5] END .............C=0.1, gamma=0.0001;, score=0.886 total time=
0.0s
[CV 2/5] END .............C=0.1, gamma=0.0001;, score=0.930 total time=
0.0s
[CV 3/5] END .............C=0.1, gamma=0.0001;, score=0.912 total time=
0.0s
[CV 4/5] END .............C=0.1, gamma=0.0001;, score=0.956 total time=
0.0s
[CV 5/5] END .............C=0.1, gamma=0.0001;, score=0.938 total time=
0.0s
[CV 1/5] END ..................C=1, gamma=1;, score=0.623 total time=
0.0s
[CV 2/5] END ..................C=1, gamma=1;, score=0.623 total time=
0.0s
[CV 3/5] END ..................C=1, gamma=1;, score=0.632 total time=
0.0s
[CV 4/5] END ..................C=1, gamma=1;, score=0.632 total time=
0.0s
[CV 5/5] END ..................C=1, gamma=1;, score=0.628 total time=
0.0s
```

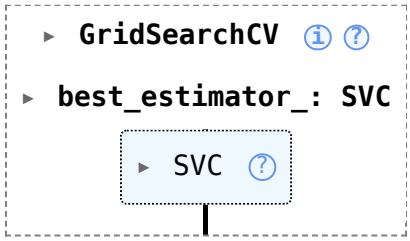
```
[CV 1/5] END ..................C=1, gamma=0.1;, score=0.623 total time=
0.0s
[CV 2/5] END ..................C=1, gamma=0.1;, score=0.623 total time=
0.0s
[CV 3/5] END ..................C=1, gamma=0.1;, score=0.632 total time=
0.0s
[CV 4/5] END ..................C=1, gamma=0.1;, score=0.632 total time=
0.0s
[CV 5/5] END ..................C=1, gamma=0.1;, score=0.628 total time=
0.0s
[CV 1/5] END .................C=1, gamma=0.01;, score=0.623 total time=
0.0s
[CV 2/5] END .................C=1, gamma=0.01;, score=0.623 total time=
0.0s
[CV 3/5] END .................C=1, gamma=0.01;, score=0.632 total time=
0.0s
[CV 4/5] END .................C=1, gamma=0.01;, score=0.632 total time=
0.0s
[CV 5/5] END .................C=1, gamma=0.01;, score=0.619 total time=
0.0s
[CV 1/5] END ................C=1, gamma=0.001;, score=0.930 total time=
0.0s
[CV 2/5] END ................C=1, gamma=0.001;, score=0.921 total time=
0.0s
[CV 3/5] END ................C=1, gamma=0.001;, score=0.921 total time=
0.0s
[CV 4/5] END ................C=1, gamma=0.001;, score=0.947 total time=
0.0s
[CV 5/5] END ................C=1, gamma=0.001;, score=0.894 total time=
0.0s
[CV 1/5] END ...............C=1, gamma=0.0001;, score=0.904 total time=
0.0s
[CV 2/5] END ...............C=1, gamma=0.0001;, score=0.939 total time=
0.0s
[CV 3/5] END ...............C=1, gamma=0.0001;, score=0.939 total time=
0.0s
[CV 4/5] END ...............C=1, gamma=0.0001;, score=0.956 total time=
0.0s
[CV 5/5] END ...............C=1, gamma=0.0001;, score=0.938 total time=
0.0s
[CV 1/5] END ..................C=10, gamma=1;, score=0.623 total time=
0.0s
[CV 2/5] END ..................C=10, gamma=1;, score=0.623 total time=
0.0s
[CV 3/5] END ..................C=10, gamma=1;, score=0.632 total time=
0.0s
[CV 4/5] END ..................C=10, gamma=1;, score=0.632 total time=
0.0s
[CV 5/5] END ..................C=10, gamma=1;, score=0.628 total time=
0.0s
[CV 1/5] END ..................C=10, gamma=0.1;, score=0.623 total time=
0.0s
[CV 2/5] END ..................C=10, gamma=0.1;, score=0.623 total time=
0.0s
[CV 3/5] END ..................C=10, gamma=0.1;, score=0.632 total time=
0.0s
[CV 4/5] END ..................C=10, gamma=0.1;, score=0.632 total time=
0.0s
[CV 5/5] END ..................C=10, gamma=0.1;, score=0.628 total time=
0.0s
[CV 1/5] END .................C=10, gamma=0.01;, score=0.623 total time=
```

```
0.0s
[CV 2/5] END .................C=10, gamma=0.01;, score=0.623 total time=
0.0s
[CV 3/5] END .................C=10, gamma=0.01;, score=0.632 total time=
0.0s
[CV 4/5] END .................C=10, gamma=0.01;, score=0.640 total time=
0.0s
[CV 5/5] END .................C=10, gamma=0.01;, score=0.619 total time=
0.0s
[CV 1/5] END ................C=10, gamma=0.001;, score=0.895 total time=
0.0s
[CV 2/5] END ................C=10, gamma=0.001;, score=0.904 total time=
0.0s
[CV 3/5] END ................C=10, gamma=0.001;, score=0.921 total time=
0.0s
[CV 4/5] END ................C=10, gamma=0.001;, score=0.939 total time=
0.0s
[CV 5/5] END ................C=10, gamma=0.001;, score=0.885 total time=
0.0s
[CV 1/5] END ...............C=10, gamma=0.0001;, score=0.895 total time=
0.0s
[CV 2/5] END ...............C=10, gamma=0.0001;, score=0.939 total time=
0.0s
[CV 3/5] END ...............C=10, gamma=0.0001;, score=0.947 total time=
0.0s
[CV 4/5] END ...............C=10, gamma=0.0001;, score=0.956 total time=
0.0s
[CV 5/5] END ...............C=10, gamma=0.0001;, score=0.920 total time=
0.0s
[CV 1/5] END ..................C=100, gamma=1;, score=0.623 total time=
0.0s
[CV 2/5] END ..................C=100, gamma=1;, score=0.623 total time=
0.0s
[CV 3/5] END ..................C=100, gamma=1;, score=0.632 total time=
0.0s
[CV 4/5] END ..................C=100, gamma=1;, score=0.632 total time=
0.0s
[CV 5/5] END ..................C=100, gamma=1;, score=0.628 total time=
0.0s
[CV 1/5] END ................C=100, gamma=0.1;, score=0.623 total time=
0.0s
[CV 2/5] END ................C=100, gamma=0.1;, score=0.623 total time=
0.0s
[CV 3/5] END ................C=100, gamma=0.1;, score=0.632 total time=
0.0s
[CV 4/5] END ................C=100, gamma=0.1;, score=0.632 total time=
0.0s
[CV 5/5] END ................C=100, gamma=0.1;, score=0.628 total time=
0.0s
[CV 1/5] END ...............C=100, gamma=0.01;, score=0.623 total time=
0.0s
[CV 2/5] END ...............C=100, gamma=0.01;, score=0.623 total time=
0.0s
[CV 3/5] END ...............C=100, gamma=0.01;, score=0.632 total time=
0.0s
[CV 4/5] END ...............C=100, gamma=0.01;, score=0.640 total time=
0.0s
[CV 5/5] END ...............C=100, gamma=0.01;, score=0.619 total time=
0.0s
[CV 1/5] END ..............C=100, gamma=0.001;, score=0.895 total time=
0.0s
```

```
[CV 2/5] END ................C=100, gamma=0.001;, score=0.904 total time=
0.0s
[CV 3/5] END ................C=100, gamma=0.001;, score=0.921 total time=
0.0s
[CV 4/5] END ................C=100, gamma=0.001;, score=0.939 total time=
0.0s
[CV 5/5] END ................C=100, gamma=0.001;, score=0.885 total time=
0.0s
[CV 1/5] END ...............C=100, gamma=0.0001;, score=0.947 total time=
0.0s
[CV 2/5] END ...............C=100, gamma=0.0001;, score=0.939 total time=
0.0s
[CV 3/5] END ...............C=100, gamma=0.0001;, score=0.956 total time=
0.0s
[CV 4/5] END ...............C=100, gamma=0.0001;, score=0.939 total time=
0.0s
[CV 5/5] END ...............C=100, gamma=0.0001;, score=0.912 total time=
0.0s
[CV 1/5] END ..................C=1000, gamma=1;, score=0.623 total time=
0.0s
[CV 2/5] END ..................C=1000, gamma=1;, score=0.623 total time=
0.0s
[CV 3/5] END ..................C=1000, gamma=1;, score=0.632 total time=
0.0s
[CV 4/5] END ..................C=1000, gamma=1;, score=0.632 total time=
0.0s
[CV 5/5] END ..................C=1000, gamma=1;, score=0.628 total time=
0.0s
[CV 1/5] END .................C=1000, gamma=0.1;, score=0.623 total time=
0.0s
[CV 2/5] END .................C=1000, gamma=0.1;, score=0.623 total time=
0.0s
[CV 3/5] END .................C=1000, gamma=0.1;, score=0.632 total time=
0.0s
[CV 4/5] END .................C=1000, gamma=0.1;, score=0.632 total time=
0.0s
[CV 5/5] END .................C=1000, gamma=0.1;, score=0.628 total time=
0.0s
[CV 1/5] END ................C=1000, gamma=0.01;, score=0.623 total time=
0.0s
[CV 2/5] END ................C=1000, gamma=0.01;, score=0.623 total time=
0.0s
[CV 3/5] END ................C=1000, gamma=0.01;, score=0.632 total time=
0.0s
[CV 4/5] END ................C=1000, gamma=0.01;, score=0.640 total time=
0.0s
[CV 5/5] END ................C=1000, gamma=0.01;, score=0.619 total time=
0.0s
[CV 1/5] END ...............C=1000, gamma=0.001;, score=0.895 total time=
0.0s
[CV 2/5] END ...............C=1000, gamma=0.001;, score=0.904 total time=
0.0s
[CV 3/5] END ...............C=1000, gamma=0.001;, score=0.921 total time=
0.0s
[CV 4/5] END ...............C=1000, gamma=0.001;, score=0.939 total time=
0.0s
[CV 5/5] END ...............C=1000, gamma=0.001;, score=0.885 total time=
0.0s
[CV 1/5] END ..............C=1000, gamma=0.0001;, score=0.939 total time=
0.0s
[CV 2/5] END ..............C=1000, gamma=0.0001;, score=0.912 total time=
```

```
0.0s
[CV 3/5] END .............C=1000, gamma=0.0001;, score=0.939 total time=
0.0s
[CV 4/5] END .............C=1000, gamma=0.0001;, score=0.930 total time=
0.0s
[CV 5/5] END .............C=1000, gamma=0.0001;, score=0.903 total time=
0.0s
```

Out[69]:



AAAANNND here you go. The `grid search` algorithm has found the best hyperparameters for the model.

We can see the best hyperparameters by using the `best_params_` attribute of the `grid` variable.

In [70]:
```
gridSVM.best_params_ # we are getting the best parameters
```

Out[70]:
```
{'C': 100, 'gamma': 0.0001}
```

The best hyperparameters are `{'C': 100, 'gamma': 0.0001}`.

meaning that the best `C` or `penalty` parameter of the `error term` is `100` and the best `gamma` or `kernel` coefficient for `rbf` is `0.0001`.

We can also see the best estimator by using the `best_estimator_` attribute of the `grid` variable.

In [71]:
```
model = gridSVM.best_estimator_ # we are getting the best estimator model

model.get_params() # we can see the parameters of the model
```

Out[71]:
```
{'C': 100,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 0.0001,
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
```

This is the best performing model with the best hyperparameters.

Now we can make `predictions` using this model directly or make a new model using the best hyperparameters and then make predictions. But why would we do that? We

can just use the best performing model to make predictions.

In [72]:
```
grid_predictions = gridSVM.predict(X_test) # we are predicting the model
```

Let's see if the model performs better with the best hyperparameters.

In [73]:
```
# we are printing the classification report and the confusion matrix
print(confusion_matrix(Y_test, grid_predictions))
print('\n')
print(classification_report(Y_test, grid_predictions))
```

```
[[ 64   2]
 [  1 104]]


              precision    recall  f1-score   support

           0       0.98      0.97      0.98        66
           1       0.98      0.99      0.99       105

    accuracy                           0.98       171
   macro avg       0.98      0.98      0.98       171
weighted avg       0.98      0.98      0.98       171
```

AS you can see we have a better accuracy of `98%` with the best hyperparameters. This is better than the previous accuracy of `92%` with the default hyperparameters.

So, the `grid search` algorithm has indeed found the best hyperparameters for the model.

> Grid search can be used more effectively to find the best hyperparameters for the model. AS this is a demo, I have used only `C` and `gamma` hyperparameters and I hope you have a rough idea of how to use `grid search` to find the best hyperparameters for the model.

> Grid search is not only used in `SVM` but also in all the other models to find the best hyperparameters for the model.

Final Words

Non-probabilistic algorithms are hard to understand but easy to master.

Happy learning!

In [ ]:
```
```