CLASSIFIER SERIES:

SUPPORT VECTOR MACHINE (SVM)

IN

BIOINFORMATICS

Created By:

Muhammad Bilal Alam

Classifier Series - Support Vector Machine (SVM) - Medical diagnosis

Creator: Muhammad Bilal Alam

Dataset Chosen: The Breast Cancer Wisconsin (Diagnostic) Dataset

The Breast Cancer Wisconsin (Diagnostic) Dataset, often referred to as the WDBC dataset, is a widely used resource in the machine learning community for binary classification tasks. It comprises 569 instances, each representing a separate case of breast cell nuclei present in a digitized image of a fine needle aspirate (FNA) of a breast mass.

Each instance in the dataset consists of 30 real-valued features that have been computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. These features describe characteristics of the cell nuclei present in the image, such as texture, smoothness, compactness, symmetry, and fractal dimension.

The instances are labeled as 'benign' or 'malignant', making this a binary classification problem. The dataset includes 357 benign and 212 malignant cases, offering a reasonable balance between the two classes.

The Breast Cancer Wisconsin (Diagnostic) Dataset is readily available in the UCI Machine Learning Repository and can also be easily loaded using the sklearn library in Python.

The Breast Cancer Wisconsin (Diagnostic) Dataset is an excellent choice for implementing the Support Vector Machine (SVM) algorithm. This dataset presents a binary classification problem, an area where SVM is known to excel. As the data is entirely numerical, it conforms perfectly to SVM's requirement of quantitative inputs. This allows SVM to build a hyperplane in a multidimensional space that distinctly classifies the data points. Furthermore, the size of the dataset is manageable enough to ensure that SVM's computational demands are not overwhelming. Given the nature of medical diagnostics, which involves discerning diseases based on certain symptoms, SVM's strategy of maximizing the margin between different classes makes a significant impact. Hence, this dataset not only serves as an indispensable resource for healthcare analytics, but it also makes an ideal fit for the Support Vector Machine algorithm.

What is the Purpose of Medical Diagnosis using Machine Learning

The goal of medical diagnosis using machine learning algorithms, such as the K-Nearest Neighbors (KNN), is to assist clinicians in identifying and classifying diseases based on the patient's symptoms, medical history, and various biomedical data. It plays a vital role in multiple areas:

- **Enhancing diagnostic accuracy:** Machine learning algorithms can analyze and learn from vast amounts of medical data to help identify complex patterns that may not be easily noticeable. This can improve the accuracy and speed of diagnosis, which is particularly beneficial for diseases that require timely intervention.
- **Personalized treatment:** By analyzing individual patient data, machine learning can contribute to personalized medicine, tailoring treatments to individual patients based on their unique genetic makeup, lifestyle, and other factors.
- **Predictive analysis:** Machine learning can be used to predict disease progression and patient outcomes, which can guide treatment decisions and resource allocation.
- **Reducing healthcare costs:** By improving diagnostic accuracy and enabling personalized treatment, machine learning can contribute to more efficient healthcare delivery,

potentially reducing costs and improving patient outcomes.

In the context of the Breast Cancer Wisconsin (Diagnostic) Dataset, employing SVM for medical diagnosis allows us to develop models capable of accurately differentiating between benign and malignant tumors. This information holds great value in a myriad of real-world applications, from preventive screening initiatives to assisting healthcare professionals in determining the most suitable treatment protocol.

Load The Dataset

In [1]:
```
from sklearn.datasets import load_breast_cancer
import pandas as pd
```

In [25]:
```
# Load the dataset
data = load_breast_cancer()

# Create a DataFrame
df = pd.DataFrame(data.data, columns=data.feature_names)

# Add the target variable
df['target'] = data.target

# Display the DataFrame
df.head().T
```

Out[25]:

	0	1	2	3	4
mean radius	17.990000	20.570000	19.690000	11.420000	20.290000
mean texture	10.380000	17.770000	21.250000	20.380000	14.340000
mean perimeter	122.800000	132.900000	130.000000	77.580000	135.100000
mean area	1001.000000	1326.000000	1203.000000	386.100000	1297.000000
mean smoothness	0.118400	0.084740	0.109600	0.142500	0.100300
mean compactness	0.277600	0.078640	0.159900	0.283900	0.132800
mean concavity	0.300100	0.086900	0.197400	0.241400	0.198000
mean concave points	0.147100	0.070170	0.127900	0.105200	0.104300
mean symmetry	0.241900	0.181200	0.206900	0.259700	0.180900
mean fractal dimension	0.078710	0.056670	0.059990	0.097440	0.058830
radius error	1.095000	0.543500	0.745600	0.495600	0.757200
texture error	0.905300	0.733900	0.786900	1.156000	0.781300
perimeter error	8.589000	3.398000	4.585000	3.445000	5.438000
area error	153.400000	74.080000	94.030000	27.230000	94.440000
smoothness error	0.006399	0.005225	0.006150	0.009110	0.011490
compactness error	0.049040	0.013080	0.040060	0.074580	0.024610
concavity error	0.053730	0.018600	0.038320	0.056610	0.056880
concave points error	0.015870	0.013400	0.020580	0.018670	0.018850
symmetry error	0.030030	0.013890	0.022500	0.059630	0.017560
fractal dimension error	0.006193	0.003532	0.004571	0.009208	0.005115
worst radius	25.380000	24.990000	23.570000	14.910000	22.540000
worst texture	17.330000	23.410000	25.530000	26.500000	16.670000
worst perimeter	184.600000	158.800000	152.500000	98.870000	152.200000
worst area	2019.000000	1956.000000	1709.000000	567.700000	1575.000000
worst smoothness	0.162200	0.123800	0.144400	0.209800	0.137400
worst compactness	0.665600	0.186600	0.424500	0.866300	0.205000
worst concavity	0.711900	0.241600	0.450400	0.686900	0.400000
worst concave points	0.265400	0.186000	0.243000	0.257500	0.162500
worst symmetry	0.460100	0.275000	0.361300	0.663800	0.236400
worst fractal dimension	0.118900	0.089020	0.087580	0.173000	0.076780
target	0.000000	0.000000	0.000000	0.000000	0.000000

Perform Exploratory Data Analysis (EDA)

```
In [3]: import seaborn as sns
        import matplotlib.pyplot as plt
        from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
In [4]: df.shape
```

```
Out[4]: (569, 31)
```

```
In [5]: # Overview of data
        print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count   Dtype

 1   mean radius              569 non-null     float64
 2   mean texture             569 non-null     float64
 3   mean perimeter           569 non-null     float64
 4   mean area                569 non-null     float64
 5   mean smoothness          569 non-null     float64
 6   mean compactness         569 non-null     float64
 7   mean concavity           569 non-null     float64
 8   mean concave points      569 non-null     float64
 9   mean symmetry            569 non-null     float64
 10  mean fractal dimension   569 non-null     float64
 11  radius error             569 non-null     float64
 12  texture error            569 non-null     float64
 13  perimeter error          569 non-null     float64
 14  area error               569 non-null     float64
 15  smoothness error         569 non-null     float64
 16  compactness error        569 non-null     float64
 17  concavity error          569 non-null     float64
 18  concave points error     569 non-null     float64
 19  symmetry error           569 non-null     float64
 20  fractal dimension error  569 non-null     float64
 21  worst radius             569 non-null     float64
 22  worst texture            569 non-null     float64
 23  worst perimeter          569 non-null     float64
 24  worst area               569 non-null     float64
 25  worst smoothness         569 non-null     float64
 26  worst compactness        569 non-null     float64
 27  worst concavity          569 non-null     float64
 28  worst concave points     569 non-null     float64
 29  worst symmetry           569 non-null     float64
 30  worst fractal dimension  569 non-null     float64
31target                      569 non-null     int32
dtypes: float64(30), int32(1)
memory usage: 135.7 KB
None
```

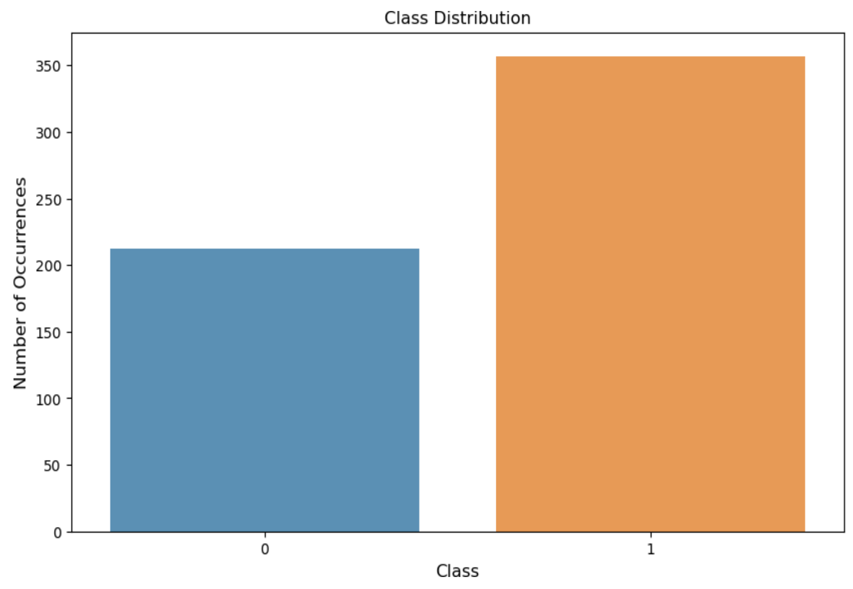
```
In [6]: df.describe().T
```

Out[6]:

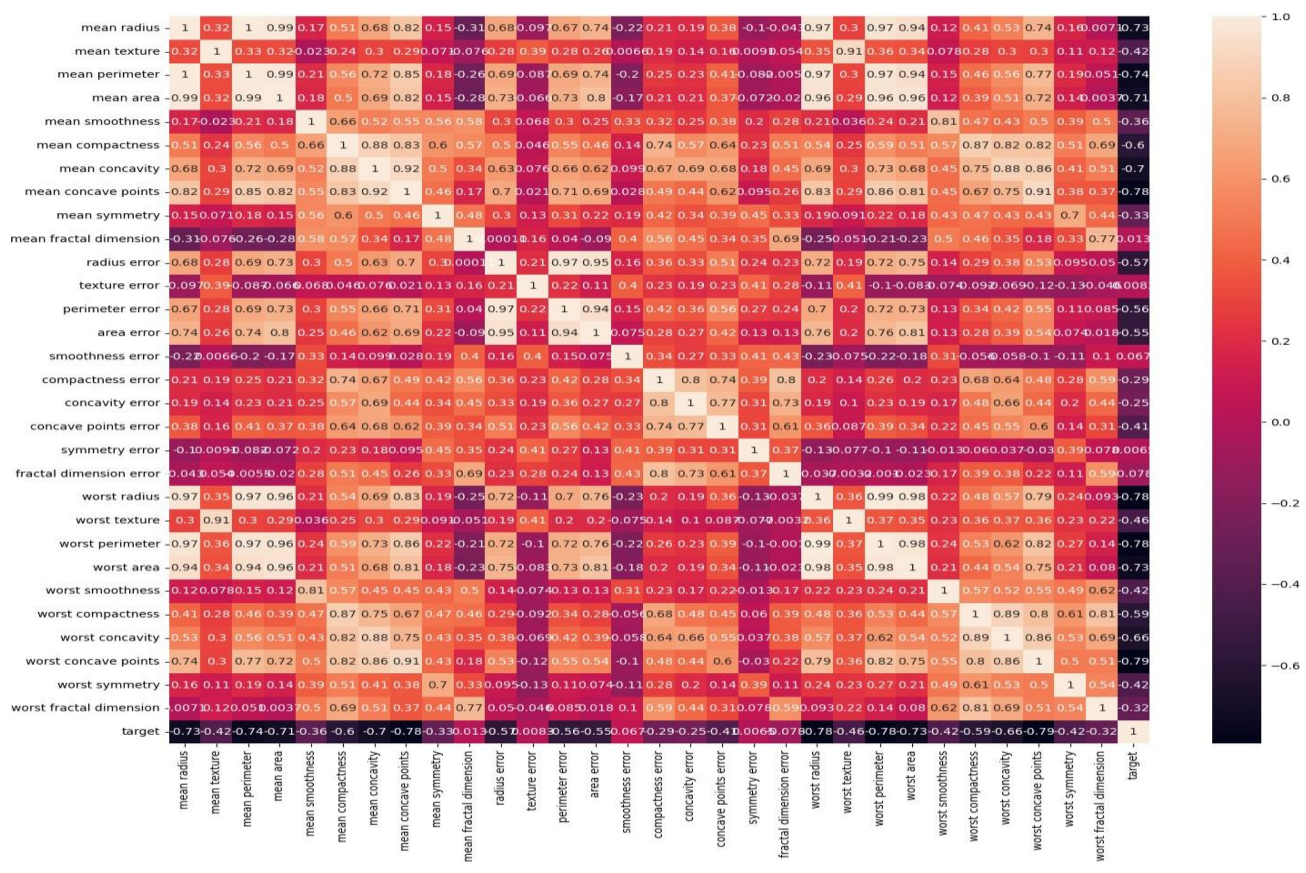
	count	mean	std	min	25%	50%	75%	
mean radius	569.0	14.127292	3.524049	6.981000	11.700000	13.370000	15.780000	28.11
mean texture	569.0	19.289649	4.301036	9.710000	16.170000	18.840000	21.800000	39.28
mean perimeter	569.0	91.969033	24.298981	43.790000	75.170000	86.240000	104.100000	188.50
mean area	569.0	654.889104	351.914129	143.500000	420.300000	551.100000	782.700000	2501.00
mean smoothness	569.0	0.096360	0.014064	0.052630	0.086370	0.095870	0.105300	0.16
mean compactness	569.0	0.104341	0.052813	0.019380	0.064920	0.092630	0.130400	0.34
mean concavity	569.0	0.088799	0.079720	0.000000	0.029560	0.061540	0.130700	0.42
mean concave points	569.0	0.048919	0.038803	0.000000	0.020310	0.033500	0.074000	0.20
mean symmetry	569.0	0.181162	0.027414	0.106000	0.161900	0.179200	0.195700	0.30
mean fractal dimension	569.0	0.062798	0.007060	0.049960	0.057700	0.061540	0.066120	0.09
radius error	569.0	0.405172	0.277313	0.111500	0.232400	0.324200	0.478900	2.87
texture error	569.0	1.216853	0.551648	0.360200	0.833900	1.108000	1.474000	4.88
perimeter error	569.0	2.866059	2.021855	0.757000	1.606000	2.287000	3.357000	21.98
area error	569.0	40.337079	45.491006	6.802000	17.850000	24.530000	45.190000	542.20
smoothness error	569.0	0.007041	0.003003	0.001713	0.005169	0.006380	0.008146	0.03
compactness error	569.0	0.025478	0.017908	0.002252	0.013080	0.020450	0.032450	0.13
concavity error	569.0	0.031894	0.030186	0.000000	0.015090	0.025890	0.042050	0.39
concave points error	569.0	0.011796	0.006170	0.000000	0.007638	0.010930	0.014710	0.05
symmetry error	569.0	0.020542	0.008266	0.007882	0.015160	0.018730	0.023480	0.07
fractal dimension error	569.0	0.003795	0.002646	0.000895	0.002248	0.003187	0.004558	0.02
worst radius	569.0	16.269190	4.833242	7.930000	13.010000	14.970000	18.790000	36.04
worst texture	569.0	25.677223	6.146258	12.020000	21.080000	25.410000	29.720000	49.54
worst perimeter	569.0	107.261213	33.602542	50.410000	84.110000	97.660000	125.400000	251.20
worst area	569.0	880.583128	569.356993	185.200000	515.300000	686.500000	1084.000000	4254.00
worst smoothness	569.0	0.132369	0.022832	0.071170	0.116600	0.131300	0.146000	0.22
worst compactness	569.0	0.254265	0.157336	0.027290	0.147200	0.211900	0.339100	1.05
worst concavity	569.0	0.272188	0.208624	0.000000	0.114500	0.226700	0.382900	1.25
worst concave points	569.0	0.114606	0.065732	0.000000	0.064930	0.099930	0.161400	0.29
worst symmetry	569.0	0.290076	0.061867	0.156500	0.250400	0.282200	0.317900	0.66

	count	mean	std	min	25%	50%	75%	
worst fractal	569.0	0.083946	0.018061	0.055040	0.071460	0.080040	0.092080	0.2 0

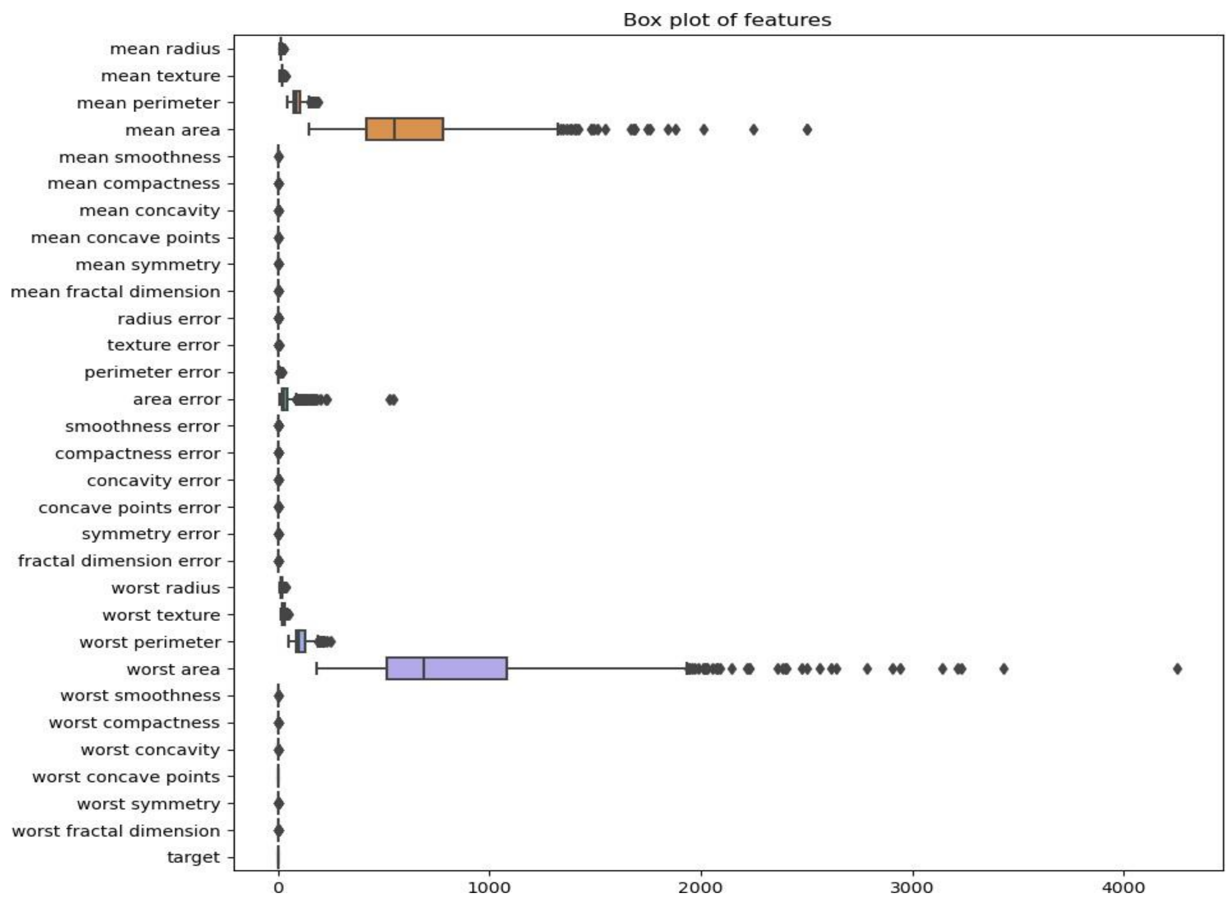
In [7]:
```python
class_counts = df['target'].value_counts()

plt.figure(figsize=(10,6))
sns.barplot(x=class_counts.index, y=class_counts.values, alpha=0.8)
plt.title('Class Distribution')
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Class', fontsize=12)
plt.show()
```



In [8]:
```python
# Correlation matrix
corr = df.corr()
plt.figure(figsize=(18, 14))
sns.heatmap(corr, annot=True, square=True)
plt.show()
```



```
In [9]: # Boxplot
        plt.figure(figsize=(10, 10))
        sns.boxplot(data=df, orient="h")
        plt.title("Box plot of features")
        plt.show()
```



```
In [10]: X = df.drop('target', axis=1)

         # Add a constant for the intercept term


         X['Intercept'] = 1

         # Calculate VIF
         vif = pd.DataFrame()
         vif["variables"] = X.columns
         vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

         print(vif)
```

```
                   variables          VIF
0                mean radius  3806.115296
1               mean texture    11.884048
2             mean perimeter  3786.400419
3                  mean area   347.878657
4            mean smoothness     8.194282
5           mean compactness    50.505168
6             mean concavity    70.767720
7        mean concave points    60.041733
8              mean symmetry     4.220656
9     mean fractal dimension    15.756977
10              radius error    75.462027
11             texture error     4.205423
12           perimeter error    70.359695
13                area error    41.163091
14          smoothness error     4.027923
15         compactness error    15.366324
16           concavity error    15.694833
17      concave points error    11.520796
18            symmetry error     5.175426
19   fractal dimension error     9.717987
20              worst radius   799.105946
21             worst texture    18.569966
22           worst perimeter   405.023336
23                worst area   337.221924
24          worst smoothness    10.923061
25         worst compactness    36.982755
26           worst concavity    31.970723
27      worst concave points    36.763714
28            worst symmetry     9.520570
29   worst fractal dimension    18.861533
30                 Intercept  1868.188844
```

Do Data Preprocessing

In [11]:
```
from sklearn.preprocessing import StandardScaler
```

In [12]:
```
# Check for missing values
df.isnull().sum()
```

Out[12]:
```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
target                     0
dtype: int64
```

In [13]:
```
# Check for duplicates
duplicates = df.duplicated()
print(f"Number of duplicate rows = {duplicates.sum()}")
```

```
Number of duplicate rows = 0
```

```
In [14]: # Scaling the features
         scaler = StandardScaler()

         # List of column names excluding the target
         columns = df.columns[:-1]

         # Apply the scaler to the DataFrame excluding the target
         df[columns] = scaler.fit_transform(df[columns])
```

Apply SVM

```
In [15]: from sklearn.feature_selection import SelectKBest, f_classif
         from sklearn.model_selection import train_test_split
         from sklearn.model_selection import GridSearchCV
         from sklearn.svm import SVC
         import matplotlib.pyplot as plt
         from sklearn.metrics import confusion_matrix,accuracy_score,classification_report
         from sklearn.model_selection import cross_val_score
         import numpy as np
```

```
In [16]: # X are the input (or independent) variables
         X = df.drop(['target','mean radius'], axis=1)

         # y is output (or dependent) variable
         y = df['target']
```

```
In [17]: # Split the dataset into a training set and a test set
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

```
In [18]: # Define your SVM model
         svm = SVC(class_weight='balanced')

         # Split your data into features (X) and target (y)
         X = df.drop('target', axis=1)
         y = df['target']


         # Perform cross validation
         scores = cross_val_score(svm, X, y, cv=10)

         print(f"Cross-validation scores: {scores}")
         print(f"Mean cross-validation score: {np.mean(scores)}")
```

```
Cross-validation scores: [0.96491228 0.96491228 0.94736842 0.98245614 1.         0.98
245614
 0.92982456 1.         1.         0.96428571]
Mean cross-validation score: 0.9736215538847117
```

```
In [19]: # SVM model
         svm.fit(X_train, y_train)
```

```
Out[19]: ▾ SVC

SVC(class_weight='balanced')
```

In [20]:
```python
# Predicting the Test set results
y_pred = svm.predict(X_test)
```

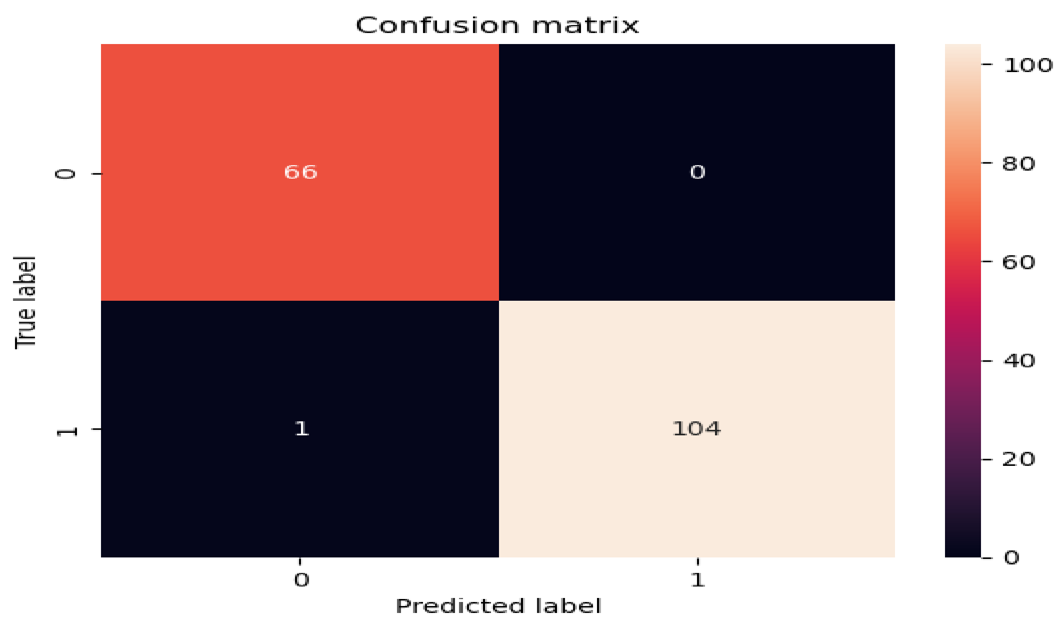
In [21]:
```python
# Compute the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.9941520467836257
```

In [22]:
```python
# Generate the classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      0.99        66
           1       1.00      0.99      1.00       105

    accuracy                           0.99       171
   macro avg       0.99      1.00      0.99       171
weighted avg       0.99      0.99      0.99       171
```

In [24]:
```python
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt="d")
plt.title('Confusion matrix')
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```



Conclusion:

In this project, we applied the Support Vector Machines (SVM) algorithm to classify cancer types based on gene expression data. The classifier achieved an overall accuracy of 99.41%, demonstrating SVM's effectiveness in handling complex bioinformatics tasks.

Our model performed well across all metrics - precision, recall, and F1-score, indicating the successful classification of cancer types, which is crucial in bioinformatics applications.

An initial attempt to improve the model through outlier handling and hyperparameter tuning did not yield the expected results; the performance slightly decreased. This suggests that in some instances, the default parameters and the inherent robustness of SVM to outliers can provide near-optimal solutions.

However, two strategies significantly enhanced our model. First, feature selection, which helped us focus on the most relevant attributes in the high-dimensional data, improving the model's performance. Second, the use of cost-sensitive learning, by setting class_weight = 'balanced', improved the model's performance on the minority class, demonstrating this approach's effectiveness in dealing with imbalanced datasets.

This project underscored the SVM algorithm's versatility in addressing bioinformatics classification tasks. The lessons learned about feature selection and cost-sensitive learning will be invaluable for future machine learning projects, particularly those involving high-dimensional and imbalanced data.