Support Vector Machine Classifier in Predicting a Pulsar Star

SVMs are supervised machine learning learning algorithms that are used for classification and regression purposes. In this project, I am going to predict the given star is **Pulsar Star** or not.

1. Introduction to SVM

Support Vector Machines (SVMs in short) are machine learning algorithms that are used for classification and regression purposes. An SVM classifier builds a model that assigns new data points to one of the given categories. Thus, it can be viewed as a non-probabilistic binary linear classifier.

SVMs can be used for linear classification purposes. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using the kernel trick. It enable us to implicitly map the inputs into high dimensional feature spaces.

2. SVMs Intution

Some SVM terminology

Hyperplane:

Hyperplane is the decision boundary that is used to separate the data points of different classes in a feature space. In the case of linear classifications, it will be a linear equation i.e. wx+b = 0.

Support Vectors:

Support vectors are the closest data points to the hyperplane, which makes a critical role in deciding the hyperplane and margin.

Margin:

Margin is the distance between the support vector and hyperplane. The main objective of the support vector machine algorithm is to maximize the margin. The wider margin indicates better classification performance.

Kernel:

Kernel is the mathematical function, which is used in SVM to map the original input data points into high-dimensional feature spaces, so, that the hyperplane can be easily found out even if the data points are not linearly separable in the original input space. Some of the common kernel functions are linear, polynomial, radial basis function(RBF), and sigmoid.

Hard Margin:

The maximum-margin hyperplane or the hard margin hyperplane is a hyperplane that properly separates the data points of different categories without any misclassifications.

Soft Margin:

When the data is not perfectly separable or contains outliers, SVM permits a soft margin technique. Each data point has a slack variable introduced by the soft-margin SVM formulation, which softens the strict margin requirement and permits certain misclassifications or violations. It discovers a compromise between increasing the margin and reducing violations.

C:

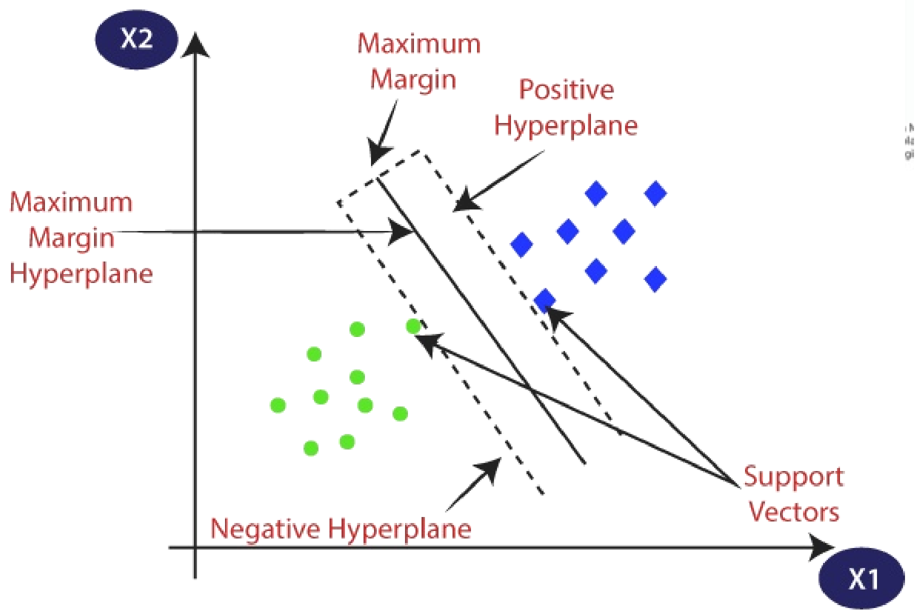
Margin maximisation and misclassification fines are balanced by the regularisation parameter C in SVM. The penalty for going over the margin or misclassifying data items is decided by it. A stricter penalty is imposed with a greater value of C, which results in a smaller margin and perhaps fewer misclassifications.

Hinge Loss:

A typical loss function in SVMs is hinge loss. It punishes incorrect classifications or margin violations. The objective function in SVM is frequently formed by combining it with the regularisation term.

Dual Problem:

A dual Problem of the optimisation problem that requires locating the Lagrange multipliers related to the support vectors can be used to solve SVM. The dual formulation enables the use of kernel tricks and more effective computing.



3. Kernal Trick

Kernel trick is a technique used in machine learning, especially with support vector machines (SVMs), to transform data into a higher-dimensional space without explicitly calculating the coordinates in that space. This transformation is performed using a kernel function.

Here's a simple way to understand it:

1. Imagine you have some data points that are not easily separable in their current form in a two-dimensional space (like on a piece of paper).
2. The kernel trick allows you to mathematically "lift" these data points into a higher-dimensional space (imagine they're now above the paper), where they might become more separable.
3. In this higher-dimensional space, it's easier to draw a straight line or any other shape to separate the data into different categories.
4. Even though you can't visualize this higher-dimensional space, the kernel trick allows the SVM (or another machine learning algorithm) to work with this transformed data without actually calculating the positions of the data points in the higher-dimensional space. It uses a mathematical function (the kernel) to determine how they relate to each other.

So, the kernel trick is a way to make complex data separable by transforming it into a higher-dimensional space while keeping the calculations efficient in the original space. It's a powerful tool in machine learning for solving problems where the data is not easily separated using simple lines or curves in its original form.

Types of Kernel trick:-

1. Linear Kernel: The linear kernel is the simplest kernel and represents a linear transformation. It's often used when the data is already linearly separable.

   Equation: K(x, y) = x * y

2. Polynomial Kernel: The polynomial kernel is used to map data into a higher-dimensional space with a polynomial function.

   Equation: K(x, y) = (x * y + c)^d

   where 'c' is a constant and 'd' is the degree of the polynomial.

3. Radial Basis Function (RBF) Kernel: The RBF kernel is also known as the Gaussian kernel. It maps data into an infinite-dimensional space, making it highly flexible for capturing complex relationships.

   Equation: K(x, y) = exp(-γ * ||x - y||^2)

   where 'γ' is a positive constant.

4. Sigmoid Kernel: The sigmoid kernel maps data into a higher-dimensional space using a hyperbolic tangent function.

   Equation: K(x, y) = tanh(α *x* y + c)

   where 'α' and 'c' are constants.

5. Custom Kernels: In addition to the standard kernels, you can create custom kernels tailored to your specific data and problem. These custom kernels are designed to capture the unique characteristics of your dataset.

TABLE I. DIFFERENT KERNEL FUNCTIONS OF SVM

	Formula	Parameters	Merits
Linear	$K(x, x_i) = x \cdot x_i$	/	It is only used when the sample is separable in low dimensional space.
Polynomial	$K(x, x_i) = [\gamma * (x \cdot x_i) + coef]^d$	γ, *coef*, *d*	global kernels
RBF	$K(x, x_i) = \exp(-\gamma * \|x - x_i\|^2)$	γ.	good local performance
Sigmoid	$K(x, x_i) = \tanh(\gamma(x \cdot x_i) + coef)$	γ, *coef*	needs to meet certain conditions

The choice of kernel function depends on the nature of your data and the problem you are trying to solve. Different kernels can have a significant impact on the performance of your machine learning model, so it's essential to experiment and choose the one that works best for your specific task.

4. Dataset description

I have used the **Predicting a Pulsar Star** dataset for this project.

This dataset is about identifying pulsar stars, which are a rare type of neutron star that emits detectable radio waves here on Earth. Pulsars are interesting for scientific research as they can provide insights into space-time, the interstellar medium, and states of matter. In this dataset, the goal is to use classification algorithms to distinguish between two types of examples: real pulsar stars (positive class) and fake signals caused by interference or noise (negative class). The dataset contains information about eight different characteristics for each candidate, like statistics from the pulse profile and DM-SNR curve, and it uses these attributes to classify whether a candidate is a pulsar (1) or not (0)..

The data set shared here contains 16,259 spurious examples caused by RFI/noise, and 1,639 real pulsar examples. Each row lists the variables first, and the class label is the final entry. The class labels used are 0 (negative) and 1 (positive).

Attribute Information:

Each candidate is described by 8 continuous variables, and a single class variable. The first four are simple statistics obtained from the integrated pulse profile. The remaining four variables are similarly obtained from the DM-SNR curve . These are summarised below:

1. Mean of the integrated profile.
2. Standard deviation of the integrated profile.
3. Excess kurtosis of the integrated profile.
4. Skewness of the integrated profile.
5. Mean of the DM-SNR curve.
6. Standard deviation of the DM-SNR curve.
7. Excess kurtosis of the DM-SNR curve.
8. Skewness of the DM-SNR curve.
9. Class

5. Import Libraries

In [1]:

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

6. Import dataset

In [2]:

```
df = pd.read_csv("pulsar_stars.csv")
```

7. Exploratory data analysis

In [3]:
```
df.shape
```
Out[3]:
```
(17898, 9)
```

In [4]:
```
df.head()
```
Out[4]:

	Mean of the integrated profile	Standard deviation of the integrated profile	Excess kurtosis of the integrated profile	Skewness of the integrated profile	Mean of the DM-SNR curve	Standard deviation of the DM-SNR curve	Excess kurtosis of the DM-SNR curve	Skewness of the DM-SNR curve	target_class
0	140.562500	55.683782	-0.234571	-0.699648	3.199833	19.110426	7.975532	74.242225	0
1	102.507812	58.882430	0.465318	-0.515088	1.677258	14.860146	10.576487	127.393580	0
2	103.015625	39.341649	0.323328	1.051164	3.121237	21.744669	7.735822	63.171909	0
3	136.750000	57.178449	-0.068415	-0.636238	3.642977	20.959280	6.896499	53.593661	0
4	88.726562	40.672225	0.600866	1.123492	1.178930	11.468720	14.269573	252.567306	0

In [5]:
```
df.columns
```
Out[5]:
```
Index([' Mean of the integrated profile',
       ' Standard deviation of the integrated profile',
       ' Excess kurtosis of the integrated profile',
       ' Skewness of the integrated profile', ' Mean of the DM-SNR curve',
       ' Standard deviation of the DM-SNR curve',
       ' Excess kurtosis of the DM-SNR curve', ' Skewness of the DM-SNR curve',
       'target_class'],
      dtype='object')
```

In [6]:
```
# remove leading spaces from column names

df.columns = df.columns.str.strip()
```

In [7]:
```
df.columns
```
Out[7]:
```
Index(['Mean of the integrated profile',
       'Standard deviation of the integrated profile',
       'Excess kurtosis of the integrated profile',
       'Skewness of the integrated profile', 'Mean of the DM-SNR curve',
       'Standard deviation of the DM-SNR curve',
       'Excess kurtosis of the DM-SNR curve', 'Skewness of the DM-SNR curve',
       'target_class'],
      dtype='object')
```

In [8]:
```
# rename column names

df.columns = ['IP Mean', 'IP Sd', 'IP Kurtosis', 'IP Skewness',
              'DM-SNR Mean', 'DM-SNR Sd', 'DM-SNR Kurtosis', 'DM-SNR Skewness', 'target_class']
```

In [9]:
```
# check distribution of target_class column

df['target_class'].value_counts()
```
Out[9]:
```
0    16259
1     1639
Name: target_class, dtype: int64
```

In [10]:
```
# view the percentage distribution of target_class column

df['target_class'].value_counts() / float(len(df))
```
Out[10]:
```
0    0.908426
1    0.091574
Name: target_class, dtype: float64
```

Here we can see that class label 0 and 1 is 90.84% and 9.16%. So, this is a class imbalanced problem.

In [11]:
```
# view summary of dataset

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17898 entries, 0 to 17897
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   IP Mean           17898 non-null  float64
 1   IP Sd             17898 non-null  float64
 2   IP Kurtosis       17898 non-null  float64
 3   IP Skewness       17898 non-null  float64
 4   DM-SNR Mean       17898 non-null  float64
 5   DM-SNR Sd         17898 non-null  float64
 6   DM-SNR Kurtosis   17898 non-null  float64
 7   DM-SNR Skewness   17898 non-null  float64
 8   target_class      17898 non-null  int64  
dtypes: float64(8), int64(1)
memory usage: 1.2 MB
```

In [12]:

```
# check for missing values in variables

df.isnull().sum()
```

Out[12]:

```
IP Mean            0
IP Sd              0
IP Kurtosis        0
IP Skewness        0
DM-SNR Mean        0
DM-SNR Sd          0
DM-SNR Kurtosis    0
DM-SNR Skewness    0
target_class       0
dtype: int64
```

Summary fo dataset

- There are 9 numerical variables in the dataset.
- 8 are continuous variables and 1 is discrete variable.
- The discrete variable is `target_class` variable. It is also the target variable.
- There are no missing values in the dataset.

In [13]:

```
# view summary statistics in numerical variables

round(df.describe(),2)
```

Out[13]:

	IP Mean	IP Sd	IP Kurtosis	IP Skewness	DM-SNR Mean	DM-SNR Sd	DM-SNR Kurtosis	DM-SNR Skewness	target_class
count	17898.00	17898.00	17898.00	17898.00	17898.00	17898.00	17898.00	17898.00	17898.00
mean	111.08	46.55	0.48	1.77	12.61	26.33	8.30	104.86	0.09
std	25.65	6.84	1.06	6.17	29.47	19.47	4.51	106.51	0.29
min	5.81	24.77	-1.88	-1.79	0.21	7.37	-3.14	-1.98	0.00
25%	100.93	42.38	0.03	-0.19	1.92	14.44	5.78	34.96	0.00
50%	115.08	46.95	0.22	0.20	2.80	18.46	8.43	83.06	0.00
75%	127.09	51.02	0.47	0.93	5.46	28.43	10.70	139.31	0.00
max	192.62	98.78	8.07	68.10	223.39	110.64	34.54	1191.00	1.00

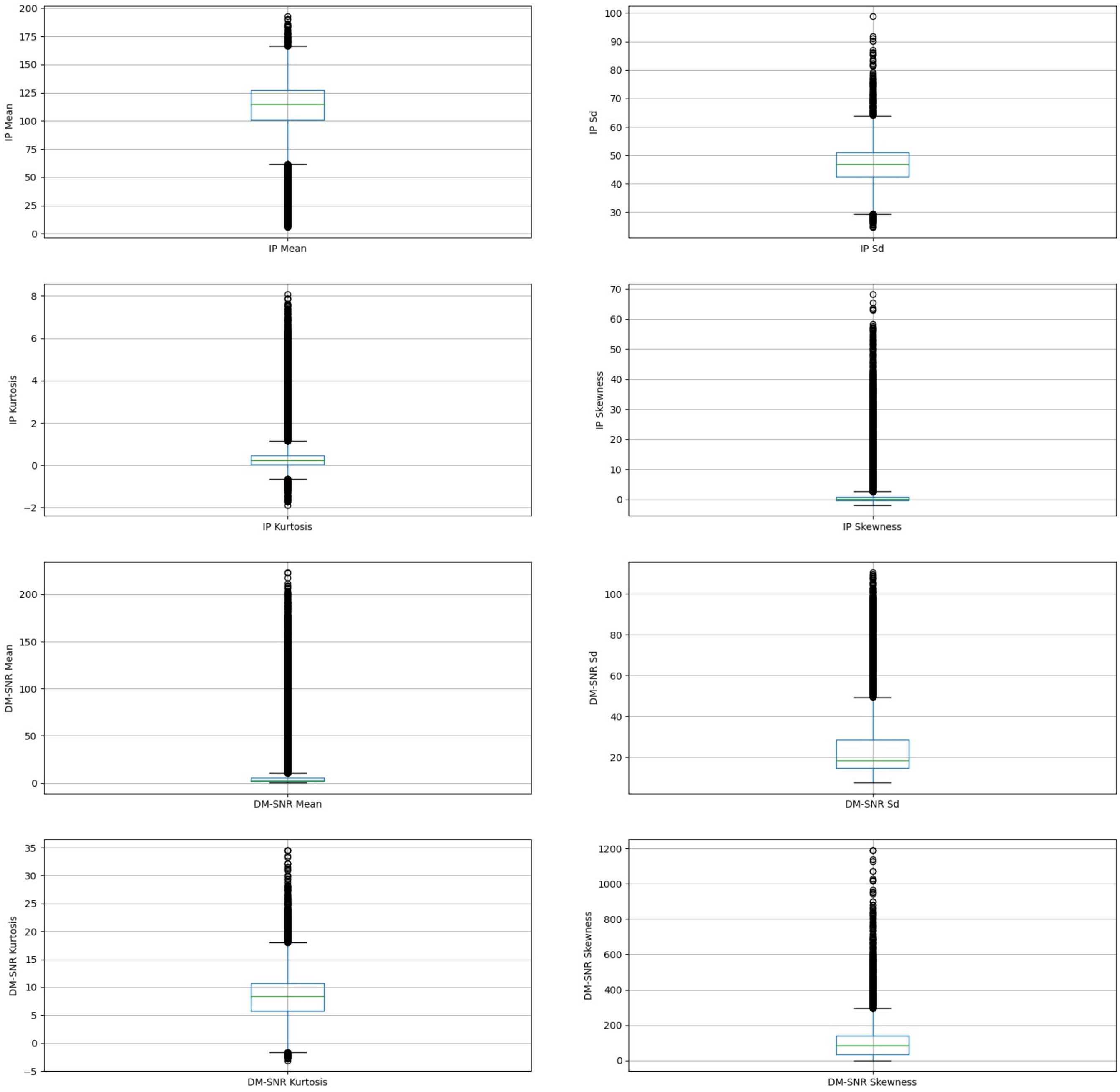
In [14]:

```
# draw boxplots to visualize outliers

plt.figure(figsize = (20,20))

plt.subplot(4,2,1)
fig = df.boxplot(column = 'IP Mean')
fig.set_ylabel('IP Mean')

plt.subplot(4,2,2)
fig = df.boxplot(column = 'IP Sd')
fig.set_ylabel('IP Sd')

plt.subplot(4,2,3)
fig = df.boxplot(column = 'IP Kurtosis')
fig.set_ylabel('IP Kurtosis')

plt.subplot(4,2,4)
fig = df.boxplot(column = 'IP Skewness')
fig.set_ylabel('IP Skewness')

plt.subplot(4,2,5)
fig = df.boxplot(column = 'DM-SNR Mean')
fig.set_ylabel('DM-SNR Mean')
```

```
plt.subplot(4, 2, 6)
fig = df.boxplot(column='DM-SNR Sd')
fig.set_ylabel('DM-SNR Sd')

plt.subplot(4,2,7)
fig = df.boxplot(column = 'DM-SNR Kurtosis')
fig.set_ylabel('DM-SNR Kurtosis')

plt.subplot(4, 2, 8)
fig = df.boxplot(column='DM-SNR Skewness')
fig.set_ylabel('DM-SNR Skewness')
```

Out[14]:

```
Text(0, 0.5, 'DM-SNR Skewness')
```



Handel outliers with SVMs

Hard-Margin SVM is strict and doesn't tolerate mistakes or outliers. Soft-Margin SVM is more flexible, allowing some mistakes with a penalty controlled by 'C'. When you have outliers, use a high 'C' in the Soft-Margin SVM to handle them.

Check the distribution of Variables

If they are normal or skewed.

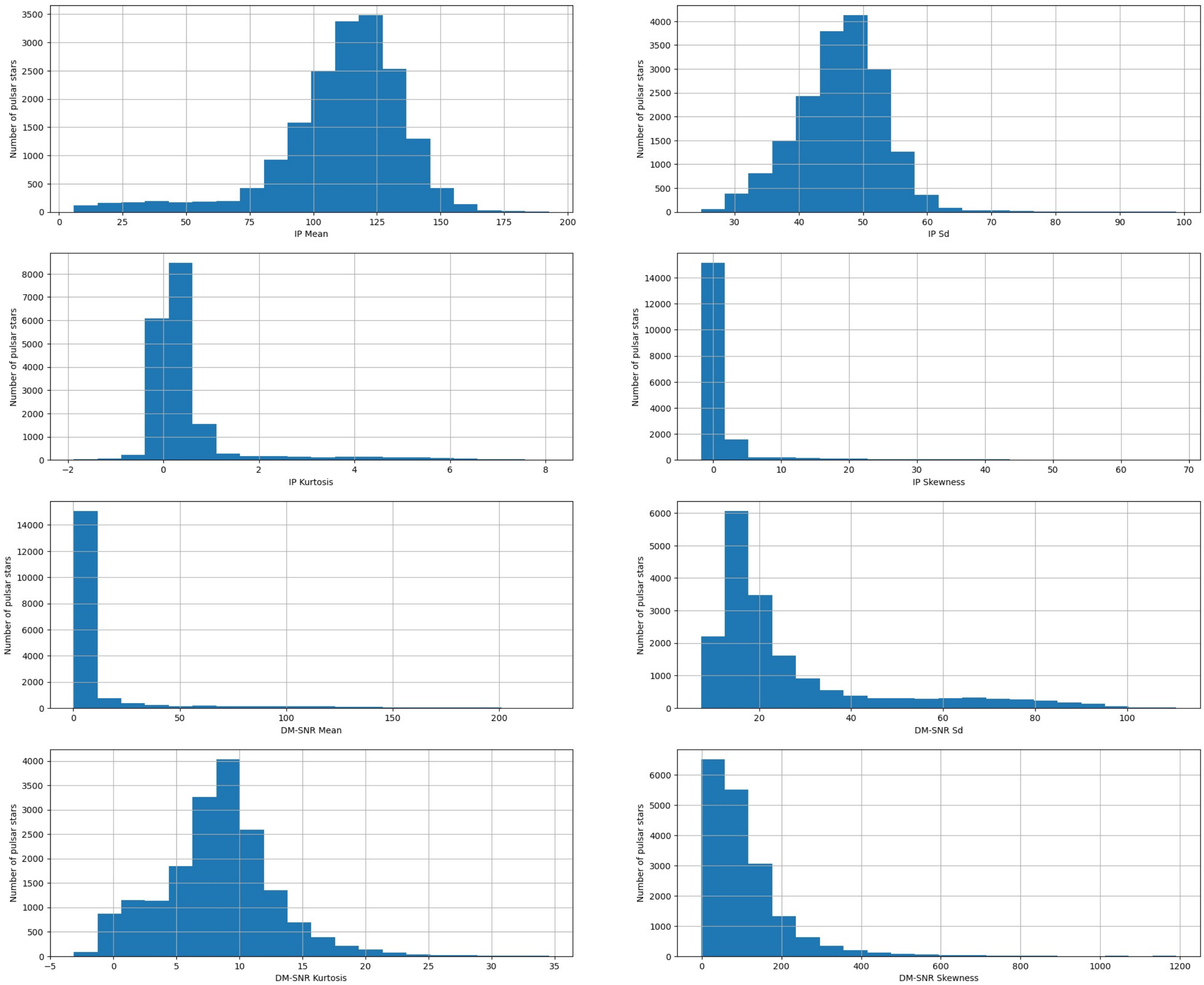
In [15]:

```
# plot histogram to check distribution


plt.figure(figsize=(24,20))


plt.subplot(4, 2, 1)
fig = df['IP Mean'].hist(bins=20)
```

```python
fig.set_xlabel('IP Mean')
fig.set_ylabel('Number of pulsar stars')


plt.subplot(4, 2, 2)
fig = df['IP Sd'].hist(bins=20)
fig.set_xlabel('IP Sd')
fig.set_ylabel('Number of pulsar stars')


plt.subplot(4, 2, 3)
fig = df['IP Kurtosis'].hist(bins=20)
fig.set_xlabel('IP Kurtosis')
fig.set_ylabel('Number of pulsar stars')



plt.subplot(4, 2, 4)
fig = df['IP Skewness'].hist(bins=20)
fig.set_xlabel('IP Skewness')
fig.set_ylabel('Number of pulsar stars')



plt.subplot(4, 2, 5)
fig = df['DM-SNR Mean'].hist(bins=20)
fig.set_xlabel('DM-SNR Mean')
fig.set_ylabel('Number of pulsar stars')



plt.subplot(4, 2, 6)
fig = df['DM-SNR Sd'].hist(bins=20)
fig.set_xlabel('DM-SNR Sd')
fig.set_ylabel('Number of pulsar stars')



plt.subplot(4, 2, 7)
fig = df['DM-SNR Kurtosis'].hist(bins=20)
fig.set_xlabel('DM-SNR Kurtosis')
fig.set_ylabel('Number of pulsar stars')


plt.subplot(4, 2, 8)
fig = df['DM-SNR Skewness'].hist(bins=20)
fig.set_xlabel('DM-SNR Skewness')
fig.set_ylabel('Number of pulsar stars')
```

Out[15]:

```
Text(0, 0.5, 'Number of pulsar stars')
```



8. Declare feature vector and target variable

In [16]:
```
X = df.drop(['target_class'], axis=1)

y = df['target_class']
```

9. Split data into separate training and test set

In [17]:
```
# split X and y into training and testing sets

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, random_state = 0)
```

In [18]:
```
# check the shape of X_train and X_test

X_train.shape, X_test.shape
```

Out[18]:
```
((14318, 8), (3580, 8))
```

10. Feature Scaling

In [19]:
```
cols = X_train.columns

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

In [20]:
```
X_train = pd.DataFrame(X_train, columns=[cols])
X_test = pd.DataFrame(X_test, columns=[cols])
```

11. Run SVM with default hyperparameters

Default hyperparameter means C=1.0, kernel= `rbf` and gamma= `auto` among other parameters.

In [21]:
```
# import SVC classifier
from sklearn.svm import SVC


# import metrics to compute accuracy
from sklearn.metrics import accuracy_score


# instantiate classifier with default hyperparameters
svc=SVC()


# fit classifier to training set
svc.fit(X_train,y_train)


# make predictions on test set
y_pred=svc.predict(X_test)


# compute and print accuracy score
print('Model accuracy score with default hyperparameters: {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with default hyperparameters: 0.9827
```

Run SVM with rbf kernel and C=100.0

We have seen that there are outliers in our dataset. So, we should increase the value of C as higher C means fewer outliers. So, I will run SVM with kernel= `rbf` and C=100.0.

In [22]:
```
# instantiate classifier with rbf kernel and C=100
svc=SVC(C=100.0)


# fit classifier to training set
svc.fit(X_train,y_train)


# make predictions on test set
y_pred=svc.predict(X_test)
```

```
# compute and print accuracy score
print('Model accuracy score with rbf kernel and C=100.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with rbf kernel and C=100.0 : 0.9832
```

We can see that we obtain a higher accuracy with C=100.0 as higher C means less outliers.

Now, I will further increase the value of C=1000.0 and check accuracy.

Run SVM with rbf kernel and C=1000.0

In [23]:
```
# instantiate classifier with rbf kernel and C=1000
svc=SVC(C=1000.0)


# fit classifier to training set
svc.fit(X_train,y_train)


# make predictions on test set
y_pred=svc.predict(X_test)


# compute and print accuracy score
print('Model accuracy score with rbf kernel and C=1000.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with rbf kernel and C=1000.0 : 0.9816
```

Accuracy had decreased when C=1000

12. Run SVM with linear kernel

Run SVM with kernel and C = 1.0

In [24]:
```
# instantiate classifier with linear kernel and C=1.0
linear_SVC = SVC(kernel = 'linear',C=1.0)

#fit classifier to training set
linear_SVC.fit(X_train, y_train)

#Make predictions on test set
y_pred_test = linear_SVC.predict(X_test)

#Compute and print accuracy score
print('Model accuracy score with linear Kernel and C=1.0 : {0:0.4f}'.format(accuracy_score(y_test,y_pred_test))
```

```
Model accuracy score with linear Kernel and C=1.0 : 0.9830
```

SVM with linear kernel and C=100.0

In [26]:
```
linear_svc100 = SVC(kernel='linear',C=100.0)

linear_svc100.fit(X_train, y_train)

y_pred_test = linear_svc100.predict(X_test)

print('Model accuracy score with linear Kernel and C=100.0 : {0:0.4f}'.format(accuracy_score(y_test,y_pred_test
```

```
Model accuracy score with linear Kernel and C=1.0 : 0.9832
```

SVM with linear kernel and C=1000.0

In [28]:
```
# instantiate classifier with linear kernel and C=1.0
linear_SVC1000 = SVC(kernel = 'linear',C=1000.0)

#fit classifier to training set
linear_SVC1000.fit(X_train, y_train)

#Make predictions on test set
y_pred_test = linear_SVC1000.predict(X_test)

#Compute and print accuracy score
print('Model accuracy score with linear Kernel and C=1000.0 : {0:0.4f}'.format(accuracy_score(y_test,y_pred_tes
```

```
Model accuracy score with linear Kernel and C=1.0 : 0.9832
```

Now we can see that the accuracy is increased when c=100 and 1000

Comparing the train accuracy

In [30]:
```
y_pred_train = linear_SVC.predict(X_train)
```

```
print('Training accuracy : {0:0.4f}'.format(accuracy_score(X_train,y_pred_train)))
```

```
Training accuracy : 0.9785
```

Checking for overfitting and underfitting

In [34]:
```
# print the scores on training and test set

print('Training set score: {:.4f}'.format(linear_SVC.score(X_train, y_train))) # This is same as above calculat

print('Test set score: {:.4f}'.format(linear_SVC.score(X_test, y_test)))
```

```
Training set score: 0.9785
Test set score: 0.9832
```

There is no proble of overfitting and underfitting

Comparing model with null accuracy

The model accuracy is 0.9832 but we cannot say our model is very good based on the accuracy only so we compare it with null accuracy.

In simple words, null accuracy in machine learning is like the easiest way to guess the outcome. It's the accuracy you'd achieve if you just guessed the most common answer for every question. It's a basic reference point to see if your machine learning model is doing better than just guessing the obvious answer. If your model's accuracy is not much better than the null accuracy, it may not be very useful.

In [35]:
```
# check class distribution in test set

y_test.value_counts()
```

Out[35]:
```
0    3306
1     274
Name: target_class, dtype: int64
```

We can see that the occurences of most frequent class 0 is 3306. So, we can calculate null accuracy by dividing 3306 by total number of occurences.

In [37]:
```
# Check null accuracy score

null_accuracy = (3306/(3306+274))

print('Null accuracy score : {0:0.4f}'.format(null_accuracy))
```

```
Null accuracy score : 0.9235
```

We can see that our model accuracy score is 0.9830 but null accuracy score is 0.9235. So, we can conclude that our SVM classifier is doing a very good job in predicting the class labels.

13. SVM with polynomial kernel

Polynomial kernel and C=1.0

In [39]:
```
# instantiate classifier with polynomial kernel and C=1.0
poly_svc = SVC(kernel = 'poly', C=1.0)

# fit classifier to training set
poly_svc.fit(X_train, y_train)

# Make predictions om test set
y_pred = poly_svc.predict(X_test)

# Compute and print the accuracy

print('Model accuracy with polynomial kernel and C=1.0 is : {0:0.4f}'.format(accuracy_score(y_test,y_pred)))
```

```
Model accuracy with polynomial kernel and C=1.0 is : 0.9807
```

Polynomial kernel with C=100.0

In [41]:
```
#instantiate classifier with polynomial kernel and C=100.0
poly_svc100 = SVC(kernel = 'poly',C=100.0)

# Fit classifier to training set
poly_svc100.fit(X_train, y_train)

# prediction
y_pred = poly_svc100.predict(X_test)

#Print accuracy
print("Accuracy with polynomial kernel and C= 100.0 is : {0:0.4f}".format(accuracy_score(y_pred,y_test)))
```

```
Accuracy with polynomial kernel and C= 100.0 is : 0.9824
```

14. SVM with sigmoid kernel

Sigmoid kernel and C=1.0

In [42]:
```python
# instantiate classifier with sigmoid kernel and C=1.0
sigmoid_svc=SVC(kernel='sigmoid', C=1.0)


# fit classifier to training set
sigmoid_svc.fit(X_train,y_train)


# make predictions on test set
y_pred=sigmoid_svc.predict(X_test)


# compute and print accuracy score
print('Model accuracy score with sigmoid kernel and C=1.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with sigmoid kernel and C=1.0 : 0.8858
```

Sigmoid kernel and C=100.0

In [43]:
```python
# instantiate classifier with sigmoid kernel and C=100.0
sigmoid_svc100=SVC(kernel='sigmoid', C=100.0)


# fit classifier to training set
sigmoid_svc100.fit(X_train,y_train)


# make predictions on test set
y_pred=sigmoid_svc100.predict(X_test)


# compute and print accuracy score
print('Model accuracy score with sigmoid kernel and C=100.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred))
```

```
Model accuracy score with sigmoid kernel and C=100.0 : 0.8855
```

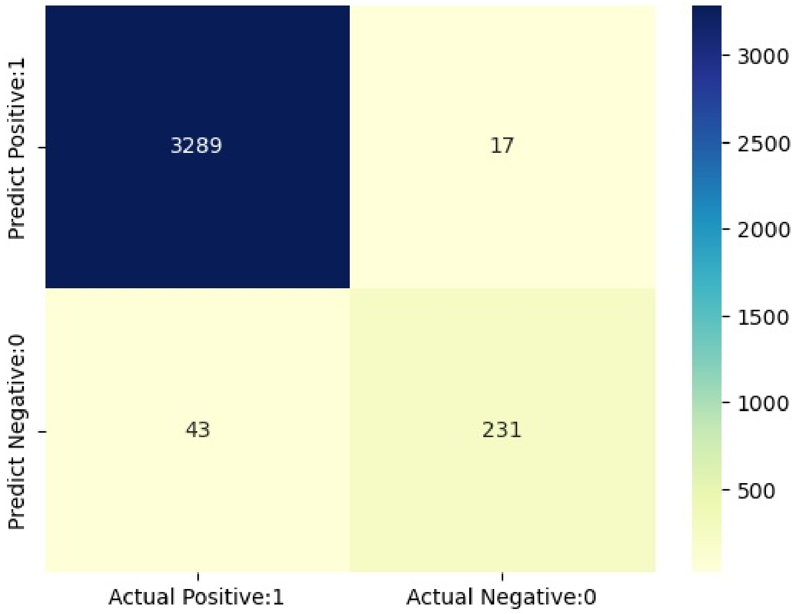
From above observations we can see taht rbf anf linear kernel with C=100.0 gives higher accuracy. Based on the above analysis we can conclude that our classification model accuracy is very good. Our model is doing a very good job in terms of predicting the class labels.

But this is not true because we have imbalanced dataset. In an imbalanced dataset, accuracy can be misleading. A better metric to evaluate model performance is the confusion matrix, which helps us understand the distribution of true positive, true negative, false positive, and false negative predictions. It provides more insight into the classifier's performance, especially in scenarios where one class is dominant.

15. Confusion Matrix

In [45]:
```python
# visualize confusion matrix with seaborn heatmap
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred_test)


cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative:0'],
                                 index=['Predict Positive:1', 'Predict Negative:0'])

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

Out[45]:
```
<Axes: >
```



16. Classification metrices

In [46]:

```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      3306
           1       0.93      0.84      0.89       274

    accuracy                           0.98      3580
   macro avg       0.96      0.92      0.94      3580
weighted avg       0.98      0.98      0.98      3580
```

17. Hyperparameter Optimization using GridSearch CV

In [47]:

```
# import GridSearchCV
from sklearn.model_selection import GridSearchCV


# import SVC classifier
from sklearn.svm import SVC


# instantiate classifier with default hyperparameters with kernel=rbf, C=1.0 and gamma=auto
svc=SVC()



# declare parameters for hyperparameter tuning
parameters = [ {'C':[1, 10, 100, 1000], 'kernel':['linear']},
               {'C':[1, 10, 100, 1000], 'kernel':['rbf'], 'gamma':[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
               {'C':[1, 10, 100, 1000], 'kernel':['poly'], 'degree': [2,3,4] ,'gamma':[0.01,0.02,0.03,0.04,0.05
              ]




grid_search = GridSearchCV(estimator = svc,
```

```
                              param_grid = parameters,
                              scoring = 'accuracy',
                              cv = 5,
                              verbose=0)


grid_search.fit(X_train, y_train)
```

Out[47]:

```
▸ GridSearchCV
▸ estimator: SVC
    ▸ SVC
```

In [48]:

```
# examine the best model


# best score achieved during the GridSearchCV
print('GridSearch CV best score : {:.4f}\n\n'.format(grid_search.best_score_))


# print parameters that give the best results
print('Parameters that give the best results :','\n\n', (grid_search.best_params_))


# print estimator that was chosen by the GridSearch
print('\n\nEstimator that was chosen by the search :','\n\n', (grid_search.best_estimator_))
```

```
GridSearch CV best score : 0.9793


Parameters that give the best results :

 {'C': 10, 'gamma': 0.3, 'kernel': 'rbf'}


Estimator that was chosen by the search :

 SVC(C=10, gamma=0.3)
```

In [49]:

```
# calculate GridSearch CV score on test set

print('GridSearch CV score on test set: {0:0.4f}'.format(grid_search.score(X_test, y_test)))
```

```
GridSearch CV score on test set: 0.9835
```

18. Conclusion

1. Accuracy is higher with rbf and linear kernel with C=100 which is 0.9832 and also from confusion matrix we get the different accuracy
2. After performing hyperparameter tuning the accuracy is increased form 0.9832 to 0.9835.So, GridSearch CV helps to identify the parameters that will improve the performance for this particular model.

19. Reference

https://www.kaggle.com/code/prashant111/svm-classifier-tutorial#12.-Run-SVM-with-default-hyperparameters-

Thank You !!

In [ ]: