****

**Hamdard University Islamabad**

**Assignment#4 Report**

**Title: Comparative Analysis of Nu-Support Vector Machines (Nu-SVM) and C-Support Vector Machines (C-SVM)**

**Members:**

Ali Najam

Aqsa Farooq

Arbab Hussain

M Hassan Farid

**Submitted To:** Sir Dr. Shaheer

1. **Introduction:**

Support Vector Machines (SVM) are powerful machine learning algorithms used for classification and regression tasks. Nu-SVM and C-SVM are two variants of SVM that differ in their formulation and parameterization. This report provides a comparative analysis of Nu-SVM and C-SVM, highlighting their key differences, advantages, and applications.

1. **Overview of Nu-SVM and C-SVM:**

**2.1 Nu-Support Vector Machines (Nu-SVM):** Nu-SVM is an extension of the traditional C-SVM that uses a different parameterization. Instead of the regularization parameter C, Nu-SVM uses a new parameter, denoted as ν (nu), which represents an upper bound on the fraction of margin errors and a lower bound on the fraction of support vectors. The nu parameter offers a more intuitive way to control the trade-off between training error and model complexity.

**2.2 C-Support Vector Machines (C-SVM**): C-SVM is the classical formulation of SVM, where the regularization parameter C is used to control the trade-off between achieving a low training error and maintaining a simple decision boundary. C-SVM aims to find a hyperplane that maximizes the margin between classes while penalizing misclassifications.

1. **Comparative Analysis:**

**3.1 Model Complexity:** Nu-SVM allows for a more direct control over the complexity of the model through the nu parameter. The nu parameter serves as an upper bound on the fraction of support vectors, providing a clear interpretation of the model complexity. In contrast, C-SVM uses the regularization parameter C, which indirectly influences model complexity.

**3.2 Interpretability:** Nu-SVM offers better interpretability due to the direct control over the fraction of support vectors. Users can set a desired upper bound on the fraction of margin errors, allowing for a more intuitive specification of model constraints. C-SVM, on the other hand, may require tuning the regularization parameter C to achieve similar control, which might be less straightforward.

**3.3 Robustness to Outliers:** Nu-SVM tends to be more robust to outliers than C-SVM. The nu parameter explicitly limits the influence of outliers on the model by constraining the fraction of margin errors. C-SVM, with its reliance on the regularization parameter C, might be more sensitive to outliers.

1. **Applications:**

**4.1 Nu-SVM Applications:** Nu-SVM is particularly suitable for scenarios where interpretability and explicit control over the fraction of support vectors are crucial. It is often preferred in applications where outliers are present, and robustness to noise is essential, such as in bioinformatics, finance, and outlier detection.

**4.2 C-SVM Applications**: C-SVM is widely used in various applications, including image classification, text categorization, and speech recognition. It is suitable for scenarios where a balance between achieving a high level of accuracy and controlling model complexity is desired.

1. **Conclusion:**

In conclusion, both Nu-SVM and C-SVM are powerful tools with distinct advantages. The choice between them depends on the specific requirements of the problem at hand, with Nu-SVM offering more interpretability and robustness to outliers, and C-SVM being a versatile choice for a wide range of applications. Understanding the characteristics of each algorithm is crucial for making informed decisions in machine learning tasks.

**Code:**

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.preprocessing import StandardScaler  
from sklearn.svm import SVC, NuSVC  
from sklearn.metrics import accuracy\_score, classification\_report

df = pd.read\_csv('diabetes\_binary\_5050split\_health\_indicators\_BRFSS2021.csv')  
df

Diabetes\_binary HighBP HighChol CholCheck BMI Smoker Stroke   
0 0.0 1 0.0 1 33.0 0.0 0.0 \  
1 0.0 0 1.0 1 27.0 1.0 0.0   
2 0.0 0 1.0 1 26.0 1.0 0.0   
3 0.0 0 0.0 1 19.0 1.0 0.0   
4 0.0 1 0.0 1 37.0 0.0 0.0   
... ... ... ... ... ... ... ...   
67131 1.0 1 0.0 1 27.0 0.0 0.0   
67132 1.0 1 1.0 1 26.0 0.0 0.0   
67133 1.0 1 1.0 1 32.0 0.0 0.0   
67134 1.0 1 1.0 1 33.0 0.0 0.0   
67135 1.0 1 1.0 1 21.0 0.0 0.0   
  
 HeartDiseaseorAttack PhysActivity Fruits ... AnyHealthcare   
0 0.0 1 1 ... 1 \  
1 0.0 1 0 ... 1   
2 0.0 0 0 ... 1   
3 0.0 1 1 ... 1   
4 0.0 1 1 ... 1   
... ... ... ... ... ...   
67131 0.0 1 1 ... 1   
67132 0.0 0 1 ... 1   
67133 1.0 1 0 ... 1   
67134 0.0 0 0 ... 1   
67135 0.0 1 1 ... 1   
  
 NoDocbcCost GenHlth MentHlth PhysHlth DiffWalk Sex Age   
0 0.0 2.0 15.0 0.0 1.0 1 7 \  
1 0.0 2.0 1.0 2.0 0.0 1 7   
2 0.0 3.0 0.0 30.0 0.0 1 13   
3 0.0 3.0 0.0 0.0 0.0 0 11   
4 0.0 2.0 0.0 0.0 0.0 0 5   
... ... ... ... ... ... ... ...   
67131 0.0 3.0 0.0 0.0 0.0 1 11   
67132 0.0 4.0 0.0 0.0 0.0 0 11   
67133 1.0 2.0 10.0 0.0 0.0 1 8   
67134 0.0 2.0 0.0 0.0 1.0 1 10   
67135 0.0 4.0 0.0 0.0 0.0 1 10   
  
 Education Income   
0 6.0 9.0   
1 6.0 6.0   
2 4.0 3.0   
3 5.0 7.0   
4 5.0 3.0   
... ... ...   
67131 5.0 6.0   
67132 4.0 2.0   
67133 6.0 6.0   
67134 4.0 5.0   
67135 2.0 3.0   
  
[67136 rows x 22 columns]

# **Exploring Data**

df.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 67136 entries, 0 to 67135  
Data columns (total 22 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Diabetes\_binary 67136 non-null float64  
 1 HighBP 67136 non-null int64   
 2 HighChol 67136 non-null float64  
 3 CholCheck 67136 non-null int64   
 4 BMI 67136 non-null float64  
 5 Smoker 67136 non-null float64  
 6 Stroke 67136 non-null float64  
 7 HeartDiseaseorAttack 67136 non-null float64  
 8 PhysActivity 67136 non-null int64   
 9 Fruits 67136 non-null int64   
 10 Veggies 67136 non-null int64   
 11 HvyAlcoholConsump 67136 non-null int64   
 12 AnyHealthcare 67136 non-null int64   
 13 NoDocbcCost 67136 non-null float64  
 14 GenHlth 67136 non-null float64  
 15 MentHlth 67136 non-null float64  
 16 PhysHlth 67136 non-null float64  
 17 DiffWalk 67136 non-null float64  
 18 Sex 67136 non-null int64   
 19 Age 67136 non-null int64   
 20 Education 67136 non-null float64  
 21 Income 67136 non-null float64  
dtypes: float64(13), int64(9)  
memory usage: 11.3 MB

df.describe()

Diabetes\_binary HighBP HighChol CholCheck   
count 67136.000000 67136.000000 67136.000000 67136.000000 \  
mean 0.500000 0.548320 0.500238 0.976227   
std 0.500004 0.497663 0.500004 0.152341   
min 0.000000 0.000000 0.000000 0.000000   
25% 0.000000 0.000000 0.000000 1.000000   
50% 0.500000 1.000000 1.000000 1.000000   
75% 1.000000 1.000000 1.000000 1.000000   
max 1.000000 1.000000 1.000000 1.000000   
  
 BMI Smoker Stroke HeartDiseaseorAttack   
count 67136.000000 67136.000000 67136.000000 67136.000000 \  
mean 30.288340 0.440151 0.058866 0.136633   
std 7.095737 0.496409 0.235375 0.343462   
min 12.000000 0.000000 0.000000 0.000000   
25% 26.000000 0.000000 0.000000 0.000000   
50% 29.000000 0.000000 0.000000 0.000000   
75% 34.000000 1.000000 0.000000 0.000000   
max 99.000000 1.000000 1.000000 1.000000   
  
 PhysActivity Fruits ... AnyHealthcare NoDocbcCost   
count 67136.000000 67136.000000 ... 67136.000000 67136.000000 \  
mean 0.717260 0.605919 ... 0.967260 0.066522   
std 0.450334 0.488656 ... 0.177955 0.249194   
min 0.000000 0.000000 ... 0.000000 0.000000   
25% 0.000000 0.000000 ... 1.000000 0.000000   
50% 1.000000 1.000000 ... 1.000000 0.000000   
75% 1.000000 1.000000 ... 1.000000 0.000000   
max 1.000000 1.000000 ... 1.000000 1.000000   
  
 GenHlth MentHlth PhysHlth DiffWalk Sex   
count 67136.000000 67136.000000 67136.000000 67136.000000 67136.000000 \  
mean 2.774756 4.230845 5.136752 0.231202 0.493431   
std 1.073759 8.323138 9.593837 0.421605 0.499961   
min 1.000000 0.000000 0.000000 0.000000 0.000000   
25% 2.000000 0.000000 0.000000 0.000000 0.000000   
50% 3.000000 0.000000 0.000000 0.000000 0.000000   
75% 3.000000 4.000000 5.000000 0.000000 1.000000   
max 5.000000 30.000000 30.000000 1.000000 1.000000   
  
 Age Education Income   
count 67136.000000 67136.000000 67136.000000   
mean 8.501743 5.035912 6.563885   
std 3.019624 0.981610 2.422641   
min 1.000000 1.000000 1.000000   
25% 7.000000 4.000000 5.000000   
50% 9.000000 5.000000 7.000000   
75% 11.000000 6.000000 8.000000   
max 13.000000 6.000000 11.000000   
  
[8 rows x 22 columns]

df.duplicated().sum()

737

# **Data Cleaning**

# Dropping unnecessary features  
  
df.drop(['Income', 'Education'], axis = 1, inplace = True)  
df

Diabetes\_binary HighBP HighChol CholCheck BMI Smoker Stroke   
0 0.0 1 0.0 1 33.0 0.0 0.0 \  
1 0.0 0 1.0 1 27.0 1.0 0.0   
2 0.0 0 1.0 1 26.0 1.0 0.0   
3 0.0 0 0.0 1 19.0 1.0 0.0   
4 0.0 1 0.0 1 37.0 0.0 0.0   
... ... ... ... ... ... ... ...   
67131 1.0 1 0.0 1 27.0 0.0 0.0   
67132 1.0 1 1.0 1 26.0 0.0 0.0   
67133 1.0 1 1.0 1 32.0 0.0 0.0   
67134 1.0 1 1.0 1 33.0 0.0 0.0   
67135 1.0 1 1.0 1 21.0 0.0 0.0   
  
 HeartDiseaseorAttack PhysActivity Fruits Veggies HvyAlcoholConsump   
0 0.0 1 1 1 0 \  
1 0.0 1 0 0 0   
2 0.0 0 0 0 0   
3 0.0 1 1 1 0   
4 0.0 1 1 1 0   
... ... ... ... ... ...   
67131 0.0 1 1 1 0   
67132 0.0 0 1 1 0   
67133 1.0 1 0 0 0   
67134 0.0 0 0 1 0   
67135 0.0 1 1 1 0   
  
 AnyHealthcare NoDocbcCost GenHlth MentHlth PhysHlth DiffWalk Sex   
0 1 0.0 2.0 15.0 0.0 1.0 1 \  
1 1 0.0 2.0 1.0 2.0 0.0 1   
2 1 0.0 3.0 0.0 30.0 0.0 1   
3 1 0.0 3.0 0.0 0.0 0.0 0   
4 1 0.0 2.0 0.0 0.0 0.0 0   
... ... ... ... ... ... ... ...   
67131 1 0.0 3.0 0.0 0.0 0.0 1   
67132 1 0.0 4.0 0.0 0.0 0.0 0   
67133 1 1.0 2.0 10.0 0.0 0.0 1   
67134 1 0.0 2.0 0.0 0.0 1.0 1   
67135 1 0.0 4.0 0.0 0.0 0.0 1   
  
 Age   
0 7   
1 7   
2 13   
3 11   
4 5   
... ...   
67131 11   
67132 11   
67133 8   
67134 10   
67135 10   
  
[67136 rows x 20 columns]

# Dropping Duplicates  
  
df = df.drop\_duplicates()  
print(df.duplicated().sum())

0

# Converting datatypes of some features  
  
df['Diabetes\_binary']=df['Diabetes\_binary'].astype(int)  
df['Age']=df['Age'].astype(int)  
df['Sex']=df['Sex'].astype(int)  
df['Smoker']=df['Smoker'].astype(int)  
df['Stroke']=df['Stroke'].astype(int)

C:\Users\hassa\AppData\Local\Temp\ipykernel\_10000\3309364861.py:3: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 df['Diabetes\_binary']=df['Diabetes\_binary'].astype(int)  
C:\Users\hassa\AppData\Local\Temp\ipykernel\_10000\3309364861.py:4: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 df['Age']=df['Age'].astype(int)  
C:\Users\hassa\AppData\Local\Temp\ipykernel\_10000\3309364861.py:5: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 df['Sex']=df['Sex'].astype(int)  
C:\Users\hassa\AppData\Local\Temp\ipykernel\_10000\3309364861.py:6: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 df['Smoker']=df['Smoker'].astype(int)  
C:\Users\hassa\AppData\Local\Temp\ipykernel\_10000\3309364861.py:7: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 df['Stroke']=df['Stroke'].astype(int)

df.shape

(61709, 20)

df['Diabetes\_binary'].value\_counts()

Diabetes\_binary  
1 31768  
0 29941  
Name: count, dtype: int64

# **Data Preprocessing**

X = df[df.columns[1:]]

Y = df['Diabetes\_binary']

from sklearn.model\_selection import train\_test\_split  
xtrain, xtest, ytrain, ytest = train\_test\_split(X,Y, test\_size = 0.2, random\_state = 50)

columnss = X.columns

# Use MinMaxScaler to scale the features in the DataFrame 'X'  
from sklearn.preprocessing import MinMaxScaler  
  
# Create MinMaxScaler object  
min\_max\_scaler = MinMaxScaler()  
  
# Scale the features and create a new DataFrame 'X'  
x\_scaled\_minmax = min\_max\_scaler.fit\_transform(X)  
X = pd.DataFrame(x\_scaled\_minmax, columns = columnss)  
  
# Display the head of the scaled features DataFrame  
X.head()

HighBP HighChol CholCheck BMI Smoker Stroke   
0 1.0 0.0 1.0 0.241379 0.0 0.0 \  
1 0.0 1.0 1.0 0.172414 1.0 0.0   
2 0.0 1.0 1.0 0.160920 1.0 0.0   
3 0.0 0.0 1.0 0.080460 1.0 0.0   
4 1.0 0.0 1.0 0.287356 0.0 0.0   
  
 HeartDiseaseorAttack PhysActivity Fruits Veggies HvyAlcoholConsump   
0 0.0 1.0 1.0 1.0 0.0 \  
1 0.0 1.0 0.0 0.0 0.0   
2 0.0 0.0 0.0 0.0 0.0   
3 0.0 1.0 1.0 1.0 0.0   
4 0.0 1.0 1.0 1.0 0.0   
  
 AnyHealthcare NoDocbcCost GenHlth MentHlth PhysHlth DiffWalk Sex   
0 1.0 0.0 0.25 0.500000 0.000000 1.0 1.0 \  
1 1.0 0.0 0.25 0.033333 0.066667 0.0 1.0   
2 1.0 0.0 0.50 0.000000 1.000000 0.0 1.0   
3 1.0 0.0 0.50 0.000000 0.000000 0.0 0.0   
4 1.0 0.0 0.25 0.000000 0.000000 0.0 0.0   
  
 Age   
0 0.500000   
1 0.500000   
2 1.000000   
3 0.833333   
4 0.333333

# **C-SVC**

### Linear

svm = SVC(C=1, kernel='linear', random\_state=42 , decision\_function\_shape='ovr')

svm.fit(xtrain, ytrain)

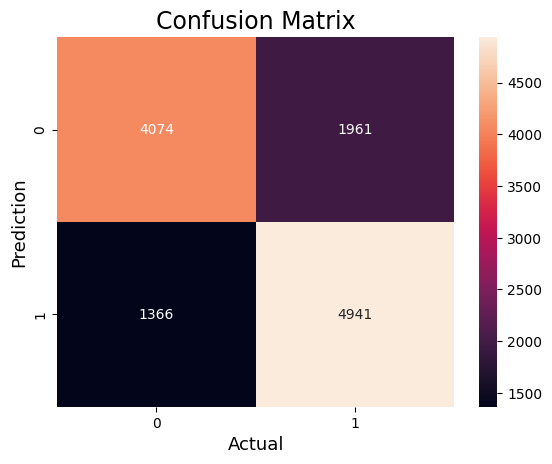
SVC(C=1, kernel='linear', random\_state=42)

pred = svm.predict(xtest)

c\_lr\_acc = accuracy\_score(ytest, pred)  
print(f'Accuracy: {c\_lr\_acc}')  
print(classification\_report(ytest,pred))

Accuracy: 0.7304326689353428  
 precision recall f1-score support  
  
 0 0.75 0.68 0.71 6035  
 1 0.72 0.78 0.75 6307  
  
 accuracy 0.73 12342  
 macro avg 0.73 0.73 0.73 12342  
weighted avg 0.73 0.73 0.73 12342

from sklearn.metrics import confusion\_matrix  
cm = confusion\_matrix(ytest,pred)  
sns.heatmap(cm, annot=True,fmt='g')  
plt.ylabel('Prediction',fontsize=13)  
plt.xlabel('Actual',fontsize=13)  
plt.title('Confusion Matrix',fontsize=17)  
plt.show()



### Polynomial

svm = SVC(C=1, kernel='poly', random\_state=42 , decision\_function\_shape='ovr')

svm.fit(xtrain, ytrain)

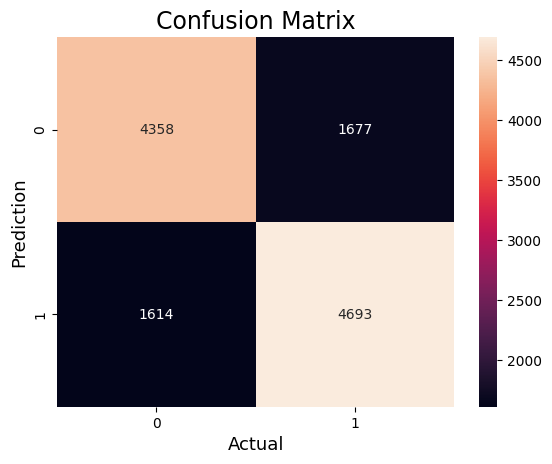
SVC(C=1, kernel='poly', random\_state=42)

pred = svm.predict(xtest)

c\_poly\_acc = accuracy\_score(ytest, pred)  
print(f'Accuracy: {c\_poly\_acc}')  
print(classification\_report(ytest,pred))

Accuracy: 0.7333495381623724  
 precision recall f1-score support  
  
 0 0.73 0.72 0.73 6035  
 1 0.74 0.74 0.74 6307  
  
 accuracy 0.73 12342  
 macro avg 0.73 0.73 0.73 12342  
weighted avg 0.73 0.73 0.73 12342

from sklearn.metrics import confusion\_matrix  
cm = confusion\_matrix(ytest,pred)  
sns.heatmap(cm, annot=True,fmt='g')  
plt.ylabel('Prediction',fontsize=13)  
plt.xlabel('Actual',fontsize=13)  
plt.title('Confusion Matrix',fontsize=17)  
plt.show()



### Sigmoid

svm = SVC(C=1, kernel='sigmoid', random\_state=42 , decision\_function\_shape='ovr')

svm.fit(xtrain, ytrain)

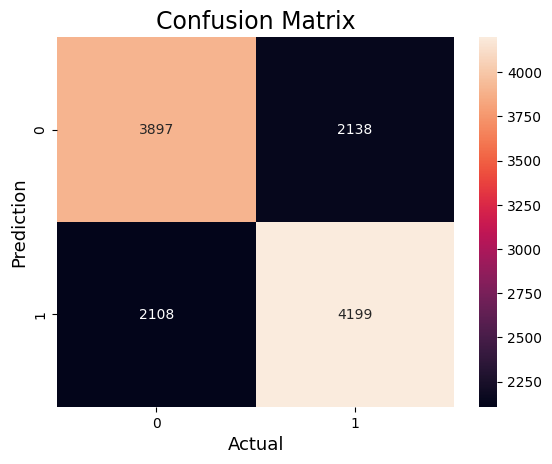
SVC(C=1, kernel='sigmoid', random\_state=42)

pred = svm.predict(xtest)

c\_sigmoid\_acc = accuracy\_score(ytest, pred)  
print(f'Accuracy: {c\_sigmoid\_acc}')  
print(classification\_report(ytest,pred))

Accuracy: 0.6559714795008913  
 precision recall f1-score support  
  
 0 0.65 0.65 0.65 6035  
 1 0.66 0.67 0.66 6307  
  
 accuracy 0.66 12342  
 macro avg 0.66 0.66 0.66 12342  
weighted avg 0.66 0.66 0.66 12342

from sklearn.metrics import confusion\_matrix  
cm = confusion\_matrix(ytest,pred)  
sns.heatmap(cm, annot=True,fmt='g')  
plt.ylabel('Prediction',fontsize=13)  
plt.xlabel('Actual',fontsize=13)  
plt.title('Confusion Matrix',fontsize=17)  
plt.show()



### RBF

svm = SVC(C=1, kernel='rbf', random\_state=42 , decision\_function\_shape='ovr')

svm.fit(xtrain, ytrain)

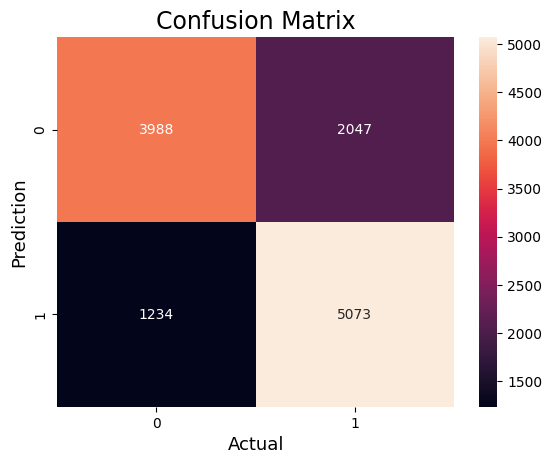
SVC(C=1, random\_state=42)

pred = svm.predict(xtest)

c\_rbf\_acc = accuracy\_score(ytest, pred)  
print(f'Accuracy: {c\_rbf\_acc}')  
print(classification\_report(ytest,pred))

Accuracy: 0.7341597796143251  
 precision recall f1-score support  
  
 0 0.76 0.66 0.71 6035  
 1 0.71 0.80 0.76 6307  
  
 accuracy 0.73 12342  
 macro avg 0.74 0.73 0.73 12342  
weighted avg 0.74 0.73 0.73 12342

from sklearn.metrics import confusion\_matrix  
cm = confusion\_matrix(ytest,pred)  
sns.heatmap(cm, annot=True,fmt='g')  
plt.ylabel('Prediction',fontsize=13)  
plt.xlabel('Actual',fontsize=13)  
plt.title('Confusion Matrix',fontsize=17)  
plt.show()



# **Nu-SVM**

### Linear

nu\_svr = NuSVC(kernel='linear', nu=0.1, random\_state=42, verbose=True, decision\_function\_shape='ovr') # You can adjust the 'nu' parameter  
nu\_svr.fit(xtrain, ytrain)

[LibSVM]

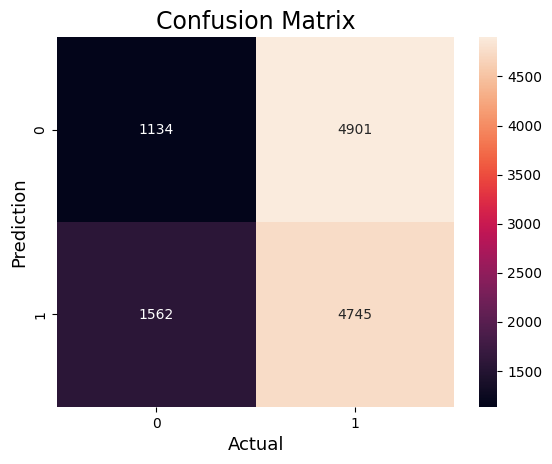
NuSVC(kernel='linear', nu=0.1, random\_state=42, verbose=True)

pred = nu\_svr.predict(xtest)

nu\_lr\_acc = accuracy\_score(ytest, pred)  
print(f'Accuracy: {nu\_lr\_acc}')  
print(classification\_report(ytest,pred))

Accuracy: 0.4763409496029817  
 precision recall f1-score support  
  
 0 0.42 0.19 0.26 6035  
 1 0.49 0.75 0.59 6307  
  
 accuracy 0.48 12342  
 macro avg 0.46 0.47 0.43 12342  
weighted avg 0.46 0.48 0.43 12342

from sklearn.metrics import confusion\_matrix  
cm = confusion\_matrix(ytest,pred)  
sns.heatmap(cm, annot=True,fmt='g')  
plt.ylabel('Prediction',fontsize=13)  
plt.xlabel('Actual',fontsize=13)  
plt.title('Confusion Matrix',fontsize=17)  
plt.show()



### Polynomial

nu\_svr = NuSVC(kernel='poly', nu=0.1, random\_state=42, verbose=True, decision\_function\_shape='ovr') # You can adjust the 'nu' parameter  
nu\_svr.fit(xtrain, ytrain)

[LibSVM]

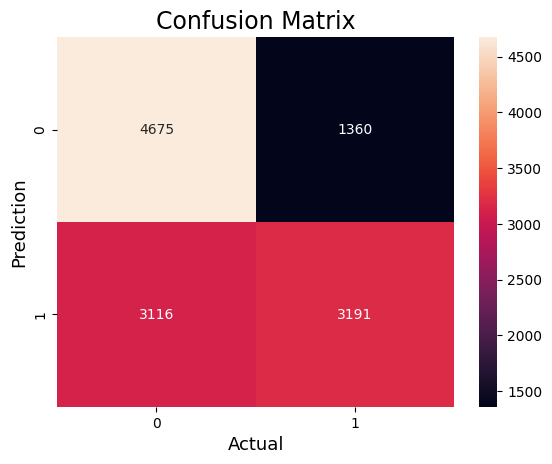
NuSVC(kernel='poly', nu=0.1, random\_state=42, verbose=True)

pred = nu\_svr.predict(xtest)

nu\_poly\_acc = accuracy\_score(ytest, pred)  
print(f'Accuracy: {nu\_poly\_acc}')  
print(classification\_report(ytest,pred))

Accuracy: 0.6373359261059796  
 precision recall f1-score support  
  
 0 0.60 0.77 0.68 6035  
 1 0.70 0.51 0.59 6307  
  
 accuracy 0.64 12342  
 macro avg 0.65 0.64 0.63 12342  
weighted avg 0.65 0.64 0.63 12342

from sklearn.metrics import confusion\_matrix  
cm = confusion\_matrix(ytest,pred)  
sns.heatmap(cm, annot=True,fmt='g')  
plt.ylabel('Prediction',fontsize=13)  
plt.xlabel('Actual',fontsize=13)  
plt.title('Confusion Matrix',fontsize=17)  
plt.show()



### Sigmoid

nu\_svr = NuSVC(kernel='sigmoid', nu=0.1, random\_state=42, verbose=True, decision\_function\_shape='ovr') # You can adjust the 'nu' parameter  
nu\_svr.fit(xtrain, ytrain)

[LibSVM]

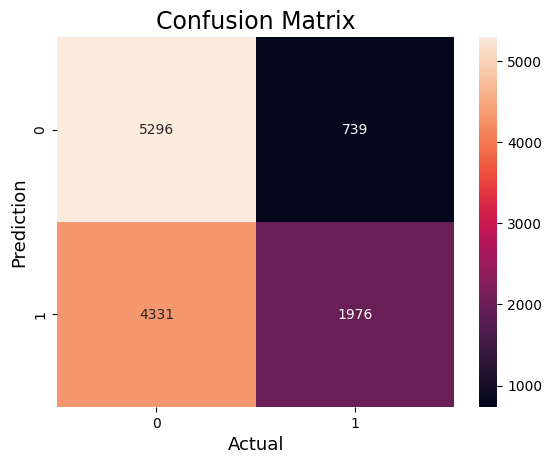
NuSVC(kernel='sigmoid', nu=0.1, random\_state=42, verbose=True)

pred = nu\_svr.predict(xtest)

nu\_sigmoid\_acc = accuracy\_score(ytest, pred)  
print(f'Accuracy: {nu\_sigmoid\_acc}')  
print(classification\_report(ytest,pred))

Accuracy: 0.5892075838599903  
 precision recall f1-score support  
  
 0 0.55 0.88 0.68 6035  
 1 0.73 0.31 0.44 6307  
  
 accuracy 0.59 12342  
 macro avg 0.64 0.60 0.56 12342  
weighted avg 0.64 0.59 0.55 12342

from sklearn.metrics import confusion\_matrix  
cm = confusion\_matrix(ytest,pred)  
sns.heatmap(cm, annot=True,fmt='g')  
plt.ylabel('Prediction',fontsize=13)  
plt.xlabel('Actual',fontsize=13)  
plt.title('Confusion Matrix',fontsize=17)  
plt.show()



### RBF

nu\_svr = NuSVC(kernel='rbf', nu=0.1, random\_state=42, verbose=True, decision\_function\_shape='ovr') # You can adjust the 'nu' parameter  
nu\_svr.fit(xtrain, ytrain)

[LibSVM]

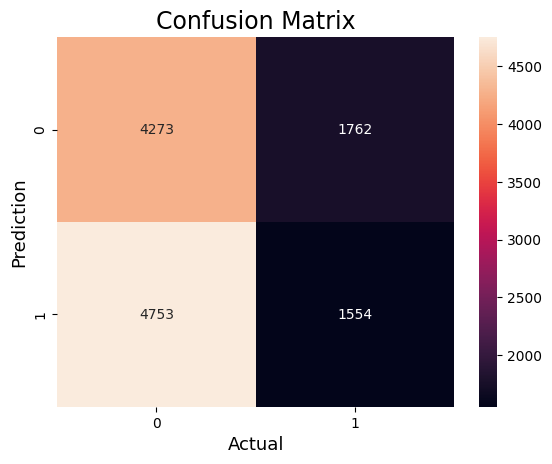
NuSVC(nu=0.1, random\_state=42, verbose=True)

pred = nu\_svr.predict(xtest)

nu\_rbf\_acc = accuracy\_score(ytest, pred)  
print(f'Accuracy: {nu\_rbf\_acc}')  
print(classification\_report(ytest,pred))

Accuracy: 0.47212769405282773  
 precision recall f1-score support  
  
 0 0.47 0.71 0.57 6035  
 1 0.47 0.25 0.32 6307  
  
 accuracy 0.47 12342  
 macro avg 0.47 0.48 0.45 12342  
weighted avg 0.47 0.47 0.44 12342

from sklearn.metrics import confusion\_matrix  
cm = confusion\_matrix(ytest,pred)  
sns.heatmap(cm, annot=True,fmt='g')  
plt.ylabel('Prediction',fontsize=13)  
plt.xlabel('Actual',fontsize=13)  
plt.title('Confusion Matrix',fontsize=17)  
plt.show()



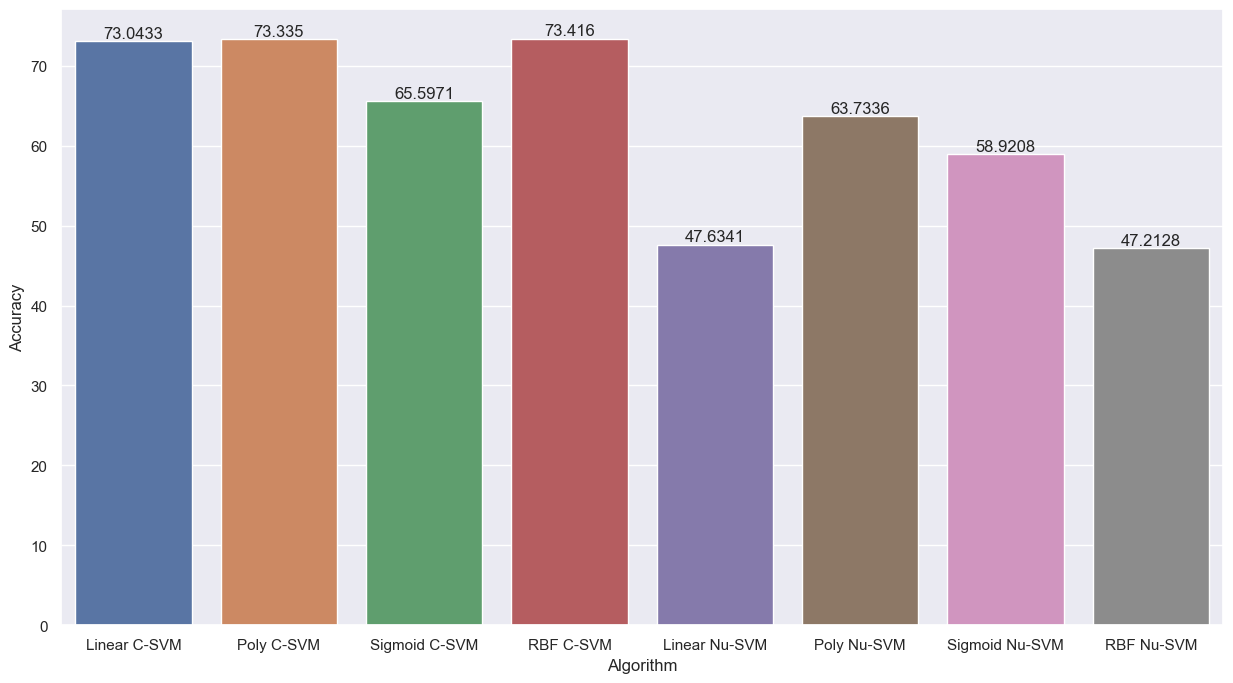
# **Comparison**

comparison\_dict={"Algorithm":["Linear C-SVM","Poly C-SVM","Sigmoid C-SVM","RBF C-SVM","Linear Nu-SVM","Poly Nu-SVM","Sigmoid Nu-SVM","RBF Nu-SVM"],  
 "Accuracy":[c\_lr\_acc\*100,c\_poly\_acc\*100,c\_sigmoid\_acc\*100,c\_rbf\_acc\*100,nu\_lr\_acc\*100,nu\_poly\_acc\*100,nu\_sigmoid\_acc\*100,nu\_rbf\_acc\*100],  
   
 }

comparison = pd.DataFrame(comparison\_dict)  
comparison.sort\_values(['Accuracy'], ascending=False)

Algorithm Accuracy  
3 RBF C-SVM 73.415978  
1 Poly C-SVM 73.334954  
0 Linear C-SVM 73.043267  
2 Sigmoid C-SVM 65.597148  
5 Poly Nu-SVM 63.733593  
6 Sigmoid Nu-SVM 58.920758  
4 Linear Nu-SVM 47.634095  
7 RBF Nu-SVM 47.212769

ax = sns.barplot(x='Algorithm', y='Accuracy', data=comparison )  
sns.set(rc={'figure.figsize':(15,9)})  
for bars in ax.containers:  
 ax.bar\_label(bars)



From the above table its shows that the C-SVM perform well rather than Nu-SVM on large and unbalance dataset.