WEEK 3

AIM:

Write a program to implement a SVM model to perform classification on a data stored in a .CSV file

DESCRIPTION:

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

Hyperplane: There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

Support Vectors: The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.

SVM Kernel:

The SVM kernel is a function that takes low dimensional input space and transforms it into higher-dimensional space, ie it converts not separable problem to separable problem. It is mostly useful in non-linear separation problems. Simply put the kernel, it does some extremely complex data transformations then finds out the process to separate the data based on the labels or outputs defined.

Advantages of SVM:

- Effective in high dimensional cases
- Its memory efficient as it uses a subset of training points in the decision function called support vectors
- Different kernel functions can be specified for the decision functions and its possible to specify custom kernels

CODE:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

df = pd.read_csv("/content/creditcard.csv")

df.head(10)

```
[ ] df.head(10)
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0
5	2.0	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.260314	-0.568671	...	-0.208254	-0.559825	-0.026398	-0.371427	-0.232794	0.105915	0.253844	0.081080	3.67	0
6	4.0	1.229658	0.141004	0.045371	1.202613	0.191881	0.272708	-0.005159	0.081213	0.464960	...	-0.167716	-0.270710	-0.154104	-0.780055	0.750137	-0.257237	0.034507	0.005168	4.99	0
7	7.0	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.807864	0.615375	...	1.943465	-1.015455	0.057504	-0.649709	-0.415267	-0.051634	-1.206921	-1.085339	40.80	0
8	7.0	-0.894286	0.286157	-0.113192	-0.271526	2.669599	3.721818	0.370145	0.851084	-0.392048	...	-0.073425	-0.268092	-0.204233	1.011592	0.373205	-0.384157	0.011747	0.142404	93.20	0
9	9.0	-0.338262	1.119593	1.044367	-0.222187	0.499361	-0.246761	0.651583	0.069539	-0.736727	...	-0.246914	-0.633753	-0.120794	-0.385050	-0.069733	0.094199	0.246219	0.083076	3.68	0

10 rows × 31 columns

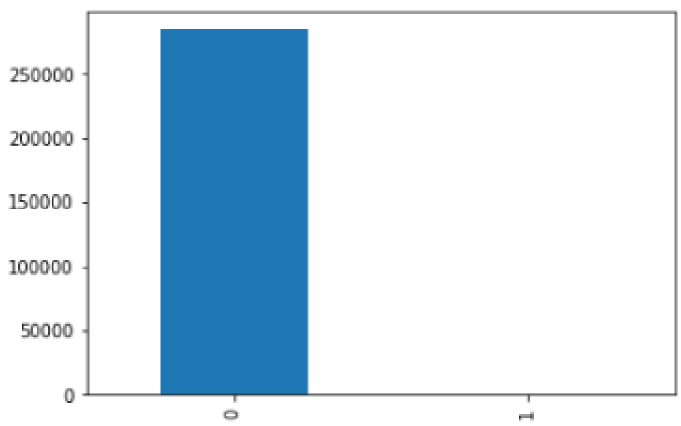
count_class=pd.value_counts(df["Class"], sort= True)

count_class.plot(kind= 'bar')

Plotting transactions with different classes

```
count_class=pd.value_counts(df["Class"], sort= True)
count_class.plot(kind= 'bar')
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f3eec580a10>



```
frauds= len(df[df["Class"]==1])
normal= len(df[df["Class"]==0])
print("The number of fraud transactions( Class = 1): ",frauds)
print("The number of normal transactions( Class = 0): ",normal)
```

Checking for no. of Fraud transactions and normal transactions in the data

```
[ ]  frauds= len(df[df["Class"]==1])
     normal= len(df[df["Class"]==0])
     print("The number of fraud transactions( Class = 1): ",frauds)
     print("The number of normal transactions( Class = 0): ",normal)

     The number of fraud transactions( Class = 1):  492
     The number of normal transactions( Class = 0):  284315
```

```
#getting the indices of normal and fraud transactions
fraud_index= np.array(df[df["Class"]==1].index)
normal_index= df[df["Class"]==0].index
random_normal_indices= np.random.choice(normal_index, 500, replace= False)
random_normal_indices= np.array(random_normal_indices)


new_indices= np.concatenate([fraud_index, random_normal_indices])
```

#use the undersampled indices to build the undersampled_data dataframe

new_data= df.iloc[new_indices, :]

print("No of Fraud Tranasactions:",len(new_data[new_data["Class"]== 1]))

print("No of Normal Tranasactions:",len(new_data[new_data["Class"]== 0]))

print(new_data.head(10))

Creating a new balanced dataset

```
#getting the indices of normal and fraud transactions
fraud_index= np.array(df[df["Class"]==1].index)
normal_index= df[df["Class"]==0].index

#choosing 500 random normal transaction indices
random_normal_indices= np.random.choice(normal_index, 500, replace= False)
random_normal_indices= np.array(random_normal_indices)

# concatenate fraud index and normal index to create a list of indices
new_indices= np.concatenate([fraud_index, random_normal_indices])

#use the undersampled indices to build the undersampled_data dataframe
new_data= df.iloc[new_indices, :]

print("No of Fraud Tranasactions:",len(new_data[new_data["Class"]== 1]))
print("No of Normal Tranasactions:",len(new_data[new_data["Class"]== 0]))

print(new_data.head(10))
```

```
No of Fraud Tranasactions: 492
No of Normal Tranasactions: 500
        Time        V1        V2        V3        V4        V5        V6  \
541    406.0 -2.312227  1.951992 -1.609851  3.997906 -0.522188 -1.426545
623    472.0 -3.043541 -3.157307  1.088463  2.288644  1.359805 -1.064823
4920  4462.0 -2.303350  1.759247 -0.359745  2.330243 -0.821628 -0.075788
6108  6986.0 -4.397974  1.358367 -2.592844  2.679787 -1.128131 -1.706536
6329  7519.0  1.234235  3.019740 -4.304597  4.732795  3.624201 -1.357746
6331  7526.0  0.008430  4.137837 -6.240697  6.675732  0.768307 -3.353060
6334  7535.0  0.026779  4.132464 -6.560600  6.348557  1.329666 -2.513479
6336  7543.0  0.329594  3.712889 -5.775935  6.078266  1.667359 -2.420168
6338  7551.0  0.316459  3.809076 -5.615159  6.047445  1.554026 -2.651353
6427  7610.0  0.725646  2.300894 -5.329976  4.007683 -1.730411 -1.732193

            V7        V8        V9  ...       V21       V22       V23  \
541  -2.537387  1.391657 -2.770089  ...  0.517232 -0.035049 -0.465211
```

new_data = new_data.drop(["Time","Amount"], axis= 1)

new_data.head(10)

```python
new_data = new_data.drop(["Time","Amount"], axis= 1)

new_data.head(10)
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V20	V21	V22	V23	V24	V25	V26	V27	V28	Class
541	-2.312227	1.951992	-1.609851	3.997906	-0.522188	-1.426545	-2.537387	1.391657	-2.770089	-2.772272	...	0.126911	0.517232	-0.035049	-0.465211	0.320198	0.044519	0.177840	0.261145	-0.143276	1
623	-3.043541	-3.157307	1.088463	2.288644	1.359805	-1.064823	0.325574	-0.067794	-0.270953	-0.838587	...	2.102339	0.661696	0.435477	1.375966	-0.293803	0.279798	-0.145362	-0.252773	0.035764	1
4920	-2.303350	1.759247	-0.359745	2.330243	-0.821628	-0.075788	0.562320	-0.399147	-0.238253	-1.525412	...	-0.430022	-0.294166	-0.932391	0.172726	-0.087330	-0.156114	-0.542628	0.039566	-0.153029	1
6108	-4.397974	1.358367	-2.592844	2.679787	-1.128131	-1.706536	-3.496197	-0.248778	-0.247768	-4.801637	...	-0.171608	0.573574	0.176968	-0.436207	-0.053502	0.252405	-0.657488	-0.827136	0.849573	1
6329	1.234235	3.019740	-4.304597	4.732795	3.624201	-1.357746	1.713445	-0.496358	-1.282858	-2.447469	...	0.009061	-0.379068	-0.704181	-0.656805	-1.632653	1.488901	0.566797	-0.010016	0.146793	1
6331	0.008430	4.137837	-6.240697	6.675732	0.768307	-3.353060	-1.631735	0.154612	-2.795892	-6.187891	...	0.488378	0.364514	-0.608057	-0.539528	0.128940	1.488481	0.507963	0.735822	0.513574	1
6334	0.026779	4.132464	-6.560600	6.348557	1.329666	-2.513479	-1.689102	0.303253	-3.139409	-6.045468	...	0.587743	0.370509	-0.576752	-0.669605	-0.759908	1.605056	0.540675	0.737040	0.496699	1
6336	0.329594	3.712889	-5.775935	6.078266	1.667359	-2.420168	-0.812891	0.133080	-2.214311	-5.134454	...	0.269773	0.156617	-0.652450	-0.551572	-0.716522	1.415717	0.555265	0.530507	0.404474	1
6338	0.316459	3.809076	-5.615159	6.047445	1.554026	-2.651353	-0.746579	0.055586	-2.678679	-4.959493	...	0.388307	0.208828	-0.511747	-0.583813	-0.219845	1.474753	0.491192	0.518868	0.402528	1
6427	0.725646	2.300894	-5.329976	4.007683	-1.730411	-1.732193	-3.968593	1.063728	-0.486097	-4.624985	...	0.504646	0.589669	0.109541	0.601045	-0.364700	-1.843078	0.351909	0.594550	0.099372	1

10 rows × 29 columns

X = new_data.iloc[:,new_data.columns != 'Class'].values

Y = new_data.iloc[:,new_data.columns == 'Class'].values

from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

from mlxtend.plotting import plot_confusion_matrix

from sklearn.metrics import confusion_matrix

x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size= 0.25, random_state= 0)

print("x_train: ", len(x_train), "y_train: ",len(y_train))

print("x_test: ", len(x_test), "y_test: ",len(y_test))

```python
[ ]  x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size= 0.25, random_state= 0)

     print("x_train: ", len(x_train), "y_train: ",len(y_train))
     print("x_test: ", len(x_test), "y_test: ",len(y_test))
```

```
x_train:  744 y_train:  744
x_test:  248 y_test:  248
```

classifier= SVC(C= 1, kernel= 'rbf', random_state= 0)

classifier.fit(x_train, y_train.ravel())

Training SVD Classifier

```
[ ] classifier= SVC(C= 1, kernel= 'rbf', random_state= 0)
    classifier.fit(x_train, y_train.ravel())

    SVC(C=1, random_state=0)
```

y_pred = classifier.predict(x_test)

cm = confusion_matrix(y_test, y_pred)

print(cm)

Confusion Matrix

```
[ ] y_pred = classifier.predict(x_test)
    cm = confusion_matrix(y_test, y_pred)
    print(cm)

    [[118   4]
     [ 18 108]]
```

from sklearn import metrics

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

Model Accuracy

```
[ ] from sklearn import metrics

    print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

    Accuracy: 0.9112903225806451
```