Induction-Motor-Fault-Detection-using-Support-Vector-Machine
/ Induction-Motor-Fault-Detection-using-Support-Vector-Machine.ipynb

Rafeul1997 Add files via upload eaaa53d · 6 minutes ago

1860 lines (1860 loc) · 219 KB

Preview Code Blame Raw

In [1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # for data visualization
import seaborn as sns # for statistical data visualization
%matplotlib inline
```

In [2]:
```
import warnings

warnings.filterwarnings('ignore')
```

In [3]:
```
data = 'im_data.csv'

df = pd.read_csv(data)
```

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

Out[4]:

	temp	vib	speed	current	voltage	output
0	26.4	1.12	0	4.5	230	0
1	29.0	1.41	1430	4.5	230	0
2	29.5	1.32	1423	4.5	230	0
3	28.2	1.38	1400	4.5	230	0
4	28.9	1.40	1420	4.5	230	0

In [5]:
```
df.shape
```

Out[5]:
```
(1000, 6)
```

In [6]:
```
df.head()
```

Out[6]:

	temp	vib	speed	current	voltage	output
0	26.4	1.12	0	4.5	230	0
1	29.0	1.41	1430	4.5	230	0
2	29.5	1.32	1423	4.5	230	0
3	28.2	1.38	1400	4.5	230	0
4	28.9	1.40	1420	4.5	230	0

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
```

```
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   temp     1000 non-null   float64
 1   vib      1000 non-null   float64
 2   speed    1000 non-null   int64
 3   current  1000 non-null   float64
 4   voltage  1000 non-null   int64
 5   output   1000 non-null   int64
dtypes: float64(3), int64(3)
memory usage: 47.0 KB
```

In [8]:
```
cat_cols = [col for col in df.columns if df[col].dtype == 'object']
num_cols = [col for col in df.columns if df[col].dtype != 'object']
```

In [9]:
```
for col in cat_cols:
    print(f"{col} has {df[col].unique()} values\n")
```

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

Out[10]:

	temp	vib	speed	current	voltage	output
0	26.4	1.12	0	4.5	230	0
1	29.0	1.41	1430	4.5	230	0
2	29.5	1.32	1423	4.5	230	0
3	28.2	1.38	1400	4.5	230	0
4	28.9	1.40	1420	4.5	230	0

In [11]:
```
df = df.replace(to_replace = {'': '0'})
```

In [12]:
```
df.head()
```

Out[12]:

	temp	vib	speed	current	voltage	output
0	26.4	1.12	0	4.5	230	0
1	29.0	1.41	1430	4.5	230	0
2	29.5	1.32	1423	4.5	230	0
3	28.2	1.38	1400	4.5	230	0
4	28.9	1.40	1420	4.5	230	0

In [13]:
```
df = df.fillna(0)
```

In [14]:
```
df.head()
```

Out[14]:

	temp	vib	speed	current	voltage	output
0	26.4	1.12	0	4.5	230	0
1	29.0	1.41	1430	4.5	230	0
2	29.5	1.32	1423	4.5	230	0
3	28.2	1.38	1400	4.5	230	0
4	28.9	1.40	1420	4.5	230	0

In [15]:
```
df.shape
```

Out[15]: (1000, 6)

In [16]:
```
df.describe()
```

Out[16]:

	temp	vib	speed	current	voltage	outpu
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000
mean	35.628500	1.260400	1053.439000	10.440000	226.400000	0.48800
std	7.835648	0.208313	462.237532	7.923963	4.802402	0.50010
min	25.400000	0.820000	0.000000	4.500000	220.000000	0.00000
25%	29.500000	1.120000	500.000000	4.500000	220.000000	0.00000
50%	32.300000	1.300000	1380.000000	4.500000	230.000000	0.00000
75%	41.100000	1.390000	1420.000000	21.000000	230.000000	1.00000
max	56.300000	1.890000	1474.000000	21.000000	230.000000	1.00000

In [17]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   temp     1000 non-null   float64
 1   vib      1000 non-null   float64
 2   speed    1000 non-null   int64  
 3   current  1000 non-null   float64
 4   voltage  1000 non-null   int64  
 5   output   1000 non-null   int64  
dtypes: float64(3), int64(3)
memory usage: 47.0 KB
```

In [18]:
```
df = df.apply(pd.to_numeric)
df = df.astype('float')
```

In [19]:
```
np.any(np.isnan(df))
df.isnull().sum()
```

Out[19]:
```
temp       0
vib        0
speed      0
current    0
voltage    0
output     0
dtype: int64
```

In [20]:
```
round(df.describe(),2)
```

Out[20]:

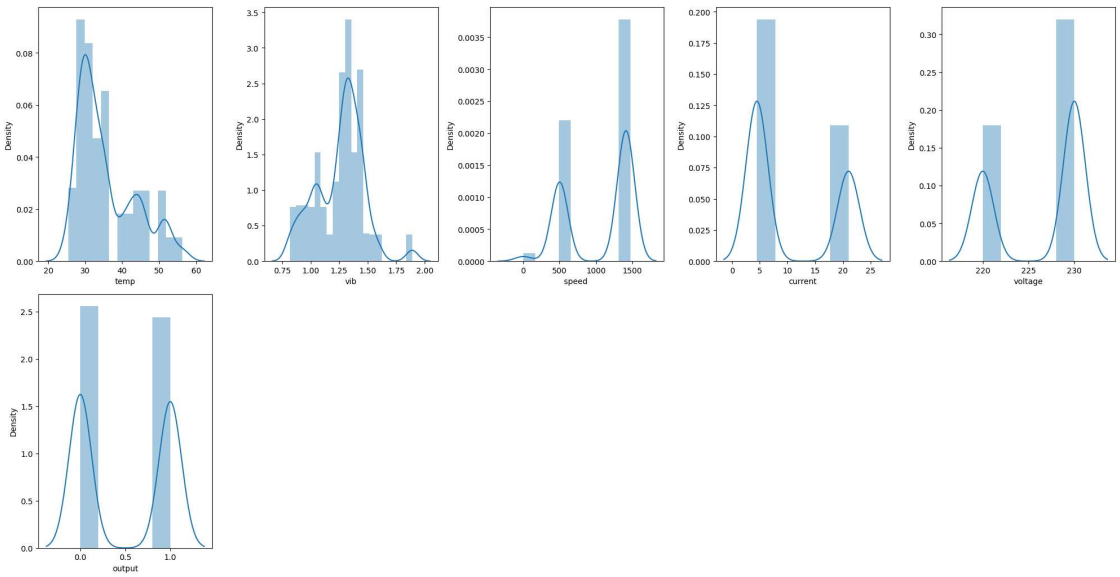
	temp	vib	speed	current	voltage	output
count	1000.00	1000.00	1000.00	1000.00	1000.0	1000.00
mean	35.63	1.26	1053.44	10.44	226.4	0.49
std	7.84	0.21	462.24	7.92	4.8	0.50
min	25.40	0.82	0.00	4.50	220.0	0.00
25%	29.50	1.12	500.00	4.50	220.0	0.00
50%	32.30	1.30	1380.00	4.50	230.0	0.00
75%	41.10	1.39	1420.00	21.00	230.0	1.00
max	56.30	1.89	1474.00	21.00	230.0	1.00

In [21]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   temp     1000 non-null   float64
 1   vib      1000 non-null   float64
 2   speed    1000 non-null   float64
 3   current  1000 non-null   float64
 4   voltage  1000 non-null   float64
 5   output   1000 non-null   float64
dtypes: float64(6)
memory usage: 47.0 KB
```

In [22]:
```
# checking numerical features distribution

plt.figure(figsize = (20, 15))
plotnumber = 1

for column in num_cols:
    if plotnumber <= 25:
        ax = plt.subplot(3, 5, plotnumber)
        sns.distplot(df[column])
```

```
        plt.xlabel(column)

    plotnumber += 1

plt.tight_layout()
plt.show()
```



In [23]:
```
X = df.drop(['output'], axis=1)

y = df['output']
```

In [24]:
```
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,
```

In [25]:
```
X_train.shape, X_test.shape
```

Out[25]: ((800, 5), (200, 5))

In [26]:
```
cols = X_train.columns
```

In [27]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

In [28]:
```
X_train = pd.DataFrame(X_train, columns=[cols])
```

In [29]:
```
X_test = pd.DataFrame(X_test, columns=[cols])
```

In [30]:
```
X_train.describe()
```

Out[30]:

	temp	vib	speed	current	voltage
count	8.000000e+02	8.000000e+02	8.000000e+02	8.000000e+02	8.000000e+02
mean	2.708944e-16	-2.409184e-16	-2.597922e-16	-4.218847e-17	-1.232348e-15
std	1.000626e+00	1.000626e+00	1.000626e+00	1.000626e+00	1.000626e+00
min	-1.309466e+00	-2.083190e+00	-2.239385e+00	-7.602034e-01	-1.315437e+00
25%	-7.940100e-01	-8.537507e-01	-1.167069e+00	-7.602034e-01	-1.315437e+00
50%	-3.414143e-01	1.392576e-01	7.202079e-01	-7.602034e-01	7.602034e-01
75%	6.643539e-01	6.121187e-01	8.059932e-01	1.315437e+00	7.602034e-01
max	2.575314e+00	2.976424e+00	9.218034e-01	1.315437e+00	7.602034e-01

In [31]:
```
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
```

```
Model accuracy score with default hyperparameters: 1.0000
```

In [32]:
```
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
```

```
Model accuracy score with rbf kernel and C=100.0 : 1.0000
```

In [33]:

```
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}'. forma
```

```
Model accuracy score with rbf kernel and C=1000.0 : 1.0000
```

In [34]:

```
# 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}'. forma
```

```
Model accuracy score with linear kernel and C=1.0 : 1.0000
```

In [35]:

```
# instantiate classifier with linear kernel and C=100.0
linear_svc100=SVC(kernel='linear', C=100.0)


# fit classifier to training set
linear_svc100.fit(X_train, y_train)


# make predictions on test set
y_pred=linear_svc100.predict(X_test)


# compute and print accuracy score
print('Model accuracy score with linear kernel and C=100.0 : {0:0.4f}'. for
```

```
Model accuracy score with linear kernel and C=100.0 : 1.0000
```

In [36]:

```
# instantiate classifier with linear kernel and C=1000.0
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=linear_svc1000.predict(X_test)


# compute and print accuracy score
print('Model accuracy score with linear kernel and C=1000.0 : {0:0.4f}'. fo
```

```
Model accuracy score with linear kernel and C=1000.0 : 1.0000
```

In [37]:

```
y_pred_train = linear_svc.predict(X_train)

y_pred_train
```

Out[37]:

```
array([0., 0., 1., 1., 0., 0., 1., 0., 0., 0., 1., 0., 1., 1., 1., 1., 0.,
       0., 1., 1., 0., 0., 0., 0., 1., 0., 1., 0., 1., 0., 1., 1., 0., 0.,
       1., 1., 1., 1., 1., 0., 1., 1., 0., 0., 0., 0., 1., 0., 1., 0., 0.,
       1., 1., 1., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 0., 1., 0., 1.,
       1., 0., 0., 1., 0., 1., 1., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
       1., 1., 1., 0., 0., 0., 1., 0., 1., 1., 1., 1., 0., 0., 0., 0., 1.,
       1., 0., 1., 1., 1., 0., 0., 0., 0., 1., 1., 1., 0., 1., 0., 1., 0.,
       0., 0., 1., 1., 1., 1., 0., 1., 1., 1., 0., 1., 1., 1., 1., 0., 1.,
       0., 1., 1., 1., 0., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 0., 1.,
       0., 1., 0., 0., 1., 0., 0., 1., 1., 1., 0., 1., 0., 0., 0., 1., 1.,
       0., 0., 1., 0., 0., 0., 1., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 1., 0., 0., 1., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0., 1.,
       1., 1., 1., 0., 1., 1., 0., 1., 1., 0., 1., 1., 0., 1., 1., 1., 1.,
       1., 0., 0., 1., 0., 1., 0., 1., 0., 0., 0., 1., 0., 0., 0., 0., 1.,
       1., 1., 1., 0., 1., 0., 0., 1., 0., 0., 1., 1., 1., 1., 1., 1., 0.,
       1., 1., 0., 1., 1., 0., 1., 0., 0., 1., 0., 0., 1., 0., 0., 0., 0.,
       0., 0., 0., 1., 1., 0., 0., 1., 0., 1., 1., 0., 0., 0., 1., 0., 0.,
       1., 1., 1., 1., 1., 0., 0., 1., 1., 0., 0., 1., 1., 1., 1., 1., 0.,
       0., 1., 1., 0., 0., 1., 0., 1., 0., 1., 0., 1., 1., 1., 0., 0., 1.,
       0., 0., 0., 0., 1., 1., 0., 0., 1., 0., 0., 0., 1., 0., 1., 1., 0.,
       1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 1., 1., 1., 0., 0.,
       0., 0., 1., 1., 0., 1., 1., 0., 1., 1., 1., 1., 0., 1., 1., 1., 0.,
       1., 0., 1., 0., 0., 1., 0., 1., 1., 0., 1., 1., 0., 0., 0., 0., 0.,
       0., 0., 1., 1., 0., 0., 0., 0., 0., 1., 0., 1., 1., 1., 0., 0., 1.,
       0., 0., 1., 1., 0., 1., 0., 0., 0., 0., 1., 1., 0., 0., 0., 0., 1.,
       1., 1., 0., 0., 0., 1., 0., 1., 1., 1., 0., 0., 0., 0., 1., 0., 0.,
       1., 1., 0., 1., 0., 0., 0., 1., 1., 1., 1., 0., 1., 1., 1., 1., 0.,
       1., 1., 1., 1., 1., 1., 0., 1., 1., 1., 0., 0., 0., 1., 0., 1., 1.,
       0., 1., 1., 0., 0., 0., 1., 0., 0., 1., 1., 1., 1., 0., 0., 1., 1.,
       1., 0., 1., 1., 0., 1., 1., 1., 1., 1., 1., 0., 1., 1., 0., 0., 0.,
       1., 1., 0., 1., 1., 1., 0., 0., 0., 1., 1., 0., 0., 1., 0., 0., 1.,
       1., 1., 1., 0., 0., 1., 1., 0., 1., 0., 0., 0., 0., 1., 0., 0., 0.,
       0., 0., 0., 0., 1., 0., 0., 1., 1., 0., 1., 1., 0., 0., 0., 0., 1.,
       0., 1., 1., 0., 1., 1., 1., 0., 1., 1., 0., 0., 0., 1., 1., 0., 1.,
```

```
1., 1., 1., 0., 1., 0., 0., 1., 0., 0., 0., 1., 1., 0., 1., 0., 0.,
1., 0., 0., 1., 0., 0., 1., 0., 0., 1., 0., 1., 1., 1., 1., 1., 1.,
0., 0., 1., 1., 1., 0., 0., 0., 1., 1., 0., 0., 0., 0., 1., 0., 1.,
0., 1., 0., 1., 1., 1., 0., 0., 0., 0., 0., 1., 0., 0., 1., 1., 1.,
1., 0., 0., 1., 1., 1., 0., 1., 0., 0., 0., 0., 1., 0., 0., 0., 0.,
1., 0., 1., 1., 0., 0., 1., 0., 0., 1., 0., 1., 0., 0., 1., 0., 0.,
1., 0., 1., 0., 1., 0., 1., 0., 0., 1., 1., 0., 0., 0., 0., 1., 1.,
0., 0., 0., 1., 1., 1., 1., 0., 0., 1., 0., 0., 0., 1., 1., 0., 1.,
1., 0., 1., 1., 0., 0., 1., 0., 1., 0., 1., 1., 1., 1., 0., 0., 1.,
0., 1., 0., 0., 0., 1., 1., 0., 1., 0., 1., 1., 1., 0., 1., 0., 0.,
0., 0., 0., 1., 1., 0., 0., 0., 0., 1., 0., 1., 0., 1., 0., 1., 1.,
0., 1., 1., 0., 1., 0., 1., 1., 1., 1., 0., 0., 1., 1., 1., 1., 1.,
0., 1., 0., 0., 0., 1., 1., 0., 1., 0., 1., 0., 0., 0., 1., 1., 1.,
1.])
```

In [38]:
```
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_trai
```

```
Training-set accuracy score: 1.0000
```

In [39]:
```
print('Training set score: {:.4f}'.format(linear_svc.score(X_train, y_train

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

```
Training set score: 1.0000
Test set score: 1.0000
```

In [40]:
```
y_test.value_counts()
```

Out[40]:
```
output
0.0    107
1.0     93
Name: count, dtype: int64
```

In [41]:
```
null_accuracy = (3306/(3306+274))

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

```
Null accuracy score: 0.9235
```

In [42]:
```
# 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 on test set
y_pred=poly_svc.predict(X_test)


# compute and print accuracy score
print('Model accuracy score with polynomial kernel and C=1.0 : {0:0.4f}'. f
```

```
Model accuracy score with polynomial kernel and C=1.0 : 1.0000
```

In [43]:

```
# 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)


# make predictions on test set
y_pred=poly_svc100.predict(X_test)


# compute and print accuracy score
print('Model accuracy score with polynomial kernel and C=1.0 : {0:0.4f}'. f
```

```
Model accuracy score with polynomial kernel and C=1.0 : 1.0000
```

In [44]:

```
# 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}'. form
```

```
Model accuracy score with sigmoid kernel and C=1.0 : 0.9100
```

In [45]:

```
# 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}'. fo
```

```
Model accuracy score with sigmoid kernel and C=100.0 : 0.9000
```

In [46]:

```
# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion_matrix
```

```
cm = confusion_matrix(y_test, y_pred_test)


print('Confusion matrix\n\n', cm)

print('\nTrue Positives(TP) = ', cm[0,0])

print('\nTrue Negatives(TN) = ', cm[1,1])

print('\nFalse Positives(FP) = ', cm[0,1])

print('\nFalse Negatives(FN) = ', cm[1,0])
```

```
Confusion matrix

 [[107   0]
 [  0  93]]

True Positives(TP) =  107

True Negatives(TN) =  93

False Positives(FP) =  0

False Negatives(FN) =  0
```

In [47]:

```
import seaborn as sns
sns.heatmap(cm, annot=True, fmt='g')
plt.show()
```