Ex No. 3	Feature Selection Techniques in Machine Learning
Date:	

Aim

To implement feature subset selection techniques in machine learning.

Definition

Feature selection

Feature Selection is the method of reducing the input variable to your model by using only relevant data and getting rid of noise in data. It is the process of automatically choosing relevant features for your machine learning model based on the type of problem you are trying to solve.

Procedure

Open PyCharm Community Edition.

Go to File menu → New Project → Specify the project name → Press "Create" button.

Right Click on Project name → New → Python File → Specify the file name → Press Enter.

Type the following codes. Right click on file name or coding window → Select "Run" to view the result.

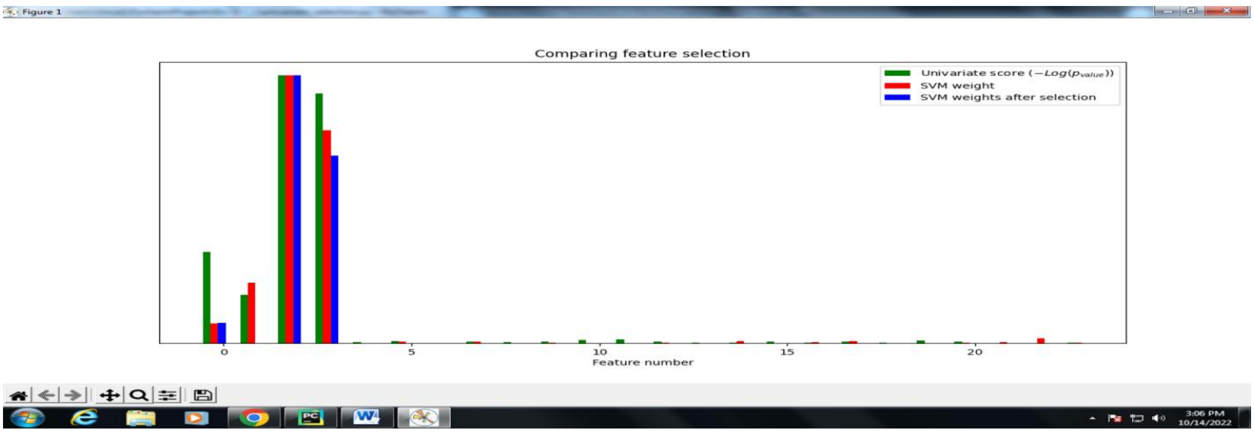
1. Univariate Feature Selection:

univariate_selection.py

```
print(__doc__)

import numpy as np

import matplotlib.pyplot as plt


from sklearn import datasets, svm

from sklearn.feature_selection import SelectPercentile, f_classif

iris = datasets.load_iris()

E = np.random.uniform(0, 0.1, size=(len(iris.data), 20))

X = np.hstack((iris.data, E))

y = iris.target

plt.figure(1)

plt.clf()

X_indices = np.arange(X.shape[-1])

selector = SelectPercentile(f_classif, percentile=10)

selector.fit(X, y)

scores = -np.log10(selector.pvalues_)

scores /= scores.max()

plt.bar(X_indices - .45, scores, width=.2,

label=r'Univariate score ($-Log(p_{value})$)', color='g')

clf = svm.SVC(kernel='linear')

clf.fit(X, y)
```

```
svm_weights = (clf.coef_ ** 2).sum(axis=0)

svm_weights /= svm_weights.max()

plt.bar(X_indices - .25, svm_weights, width=.2, label='SVM weight', color='r')

clf_selected = svm.SVC(kernel='linear')

clf_selected.fit(selector.transform(X), y)

svm_weights_selected = (clf_selected.coef_ ** 2).sum(axis=0)

svm_weights_selected /= svm_weights_selected.max()

plt.bar(X_indices[selector.get_support()] - .05, svm_weights_selected, width=.2, label='SVM
weights after selection', color='b')

plt.title("Comparing feature selection")

plt.xlabel('Feature number')

plt.yticks(())

plt.axis('tight')

plt.legend(loc='upper right')

plt.show()
```

Output



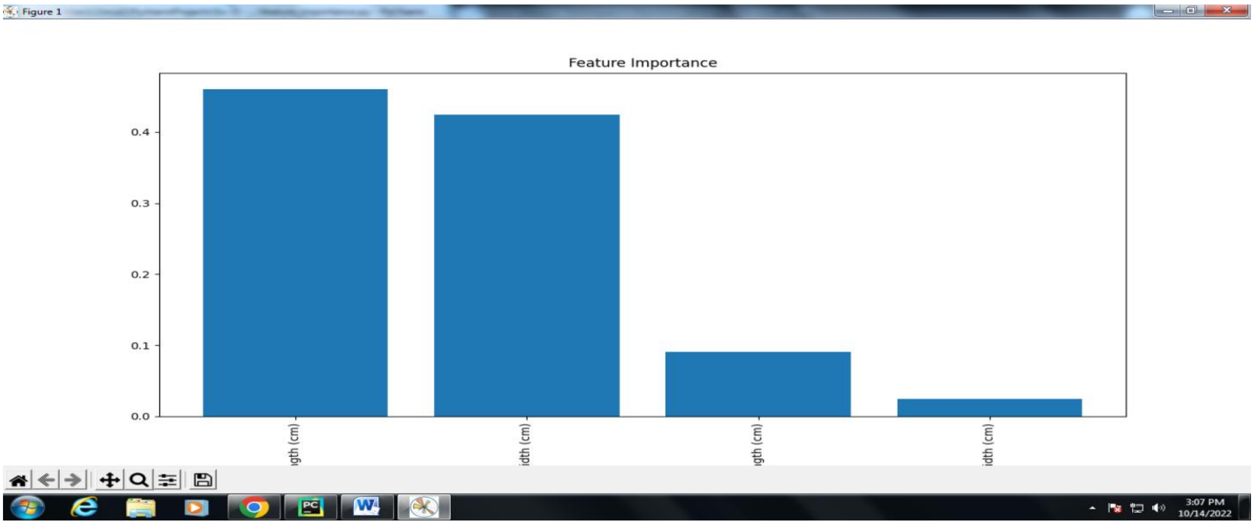
2. Feature Importance:

feature_importance.py

```
# Load libraries

from sklearn.ensemble import RandomForestClassifier

from sklearn import datasets
```

```
import numpy as np

import matplotlib.pyplot as plt

# Load data

iris = datasets.load_iris()

X = iris.data

y = iris.target

# Create decision tree classifer object

clf = RandomForestClassifier(random_state=0, n_jobs=-1)

# Train model

model = clf.fit(X, y)

# Calculate feature importances

importances = model.feature_importances_

# Sort feature importances in descending order

indices = np.argsort(importances)[::-1]

# Rearrange feature names so they match the sorted feature importances

names = [iris.feature_names[i] for i in indices]

# Create plot

plt.figure()

# Create plot title

plt.title("Feature Importance")

# Add bars

plt.bar(range(X.shape[1]), importances[indices])

# Add feature names as x-axis labels

plt.xticks(range(X.shape[1]), names, rotation=90)

# Show plot

plt.show()
```

Output



3. Correlation Matrix with Heatmap:

heatmap.py

```
# Load iris data
from sklearn.datasets import load_iris
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

iris = load_iris()

# Create features and target
X = iris.data
y = iris.target
# Convert feature matrix into DataFrame
df = pd.DataFrame(X)

# View the data frame
print(df)
# Create correlation matrix
corr_matrix = df.corr()
print(corr_matrix)
# Create correlation heatmap
plt.figure(figsize=(8,6))
plt.title('Correlation Heatmap of Iris Dataset')
a = sns.heatmap(corr_matrix, square=True, annot=True, fmt='.2f', linecolor='black')
a.set_xticklabels(a.get_xticklabels(), rotation=30)
a.set_yticklabels(a.get_yticklabels(), rotation=30)
plt.show()
 # Select upper triangle of correlation matrix
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))
# Find index of feature columns with correlation greater than 0.9
to_drop = [column for column in upper.columns if any(upper[column] > 0.9)]
print(to_drop)
# Drop Marked Features
df1 = df.drop(df.columns[to_drop], axis=1)
print(df1)
```

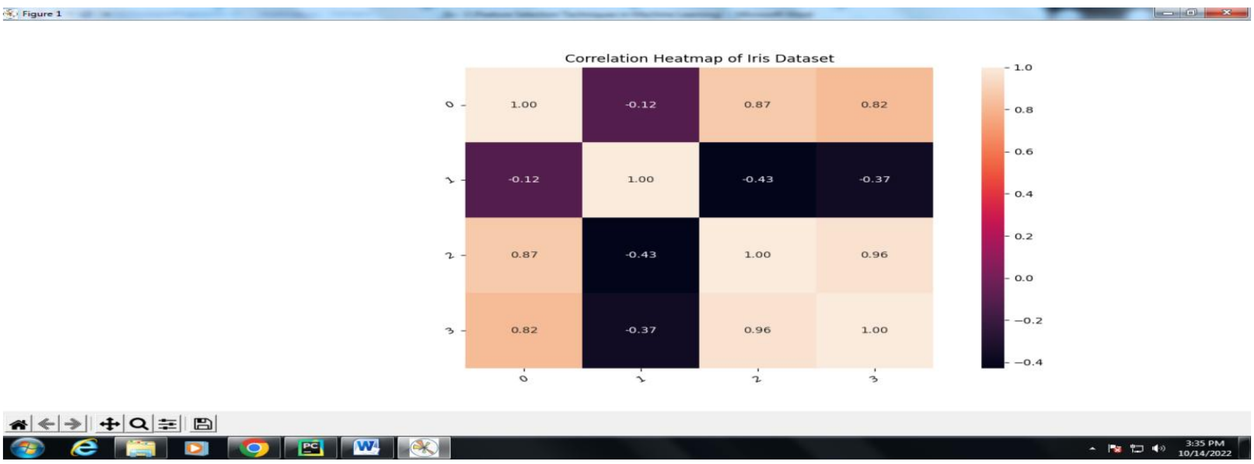
Output

C:\Users\2mca1\PycharmProjects\Ex-3\venv\Scripts\python.exe
C:/Users/2mca1/PycharmProjects/Ex-3/heatmap.py

	0	1	2	3
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
..	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

[150 rows x 4 columns]

	0	1	2	3
0	1.000000	-0.117570	0.871754	0.817941
1	-0.117570	1.000000	-0.428440	-0.366126
2	0.871754	-0.428440	1.000000	0.962865
3	0.817941	-0.366126	0.962865	1.000000



Result

Thus, feature subset selection techniques have been implemented successfully.