Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

BE-ETRX **UID:**2019110039 **Sub-Minor ML**
NAME: Devansh Palliyath

Ise 2

Aim: To implement logistic regression, K nearest neighbor and support vector machine algorithms on the given credit card dataset.

Observation:
- To identify and deal with different samplings.
- To justify and compare the output obtained through different algorithms.

Code:

```
[4] import numpy as np
    import pandas as pd
```

```
[5] from google.colab import drive
    drive.mount('/content/drive')

    Drive already mounted at /content/drive; to attemp
```

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

```
df.isnull().sum() #checking for null values
```

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       1
V19       1
V20       1
V21       1
V22       1
V23       1
V24       1
V25       1
V26       1
V27       1
V28       1
Amount    1
Class     1
dtype: int64
```

```python
df['Class'].value_counts()[0] #number of normal transactions
```

```
5970
```

```python
df['Class'].value_counts()[1] #number of fraud transactions
```

```
3
```

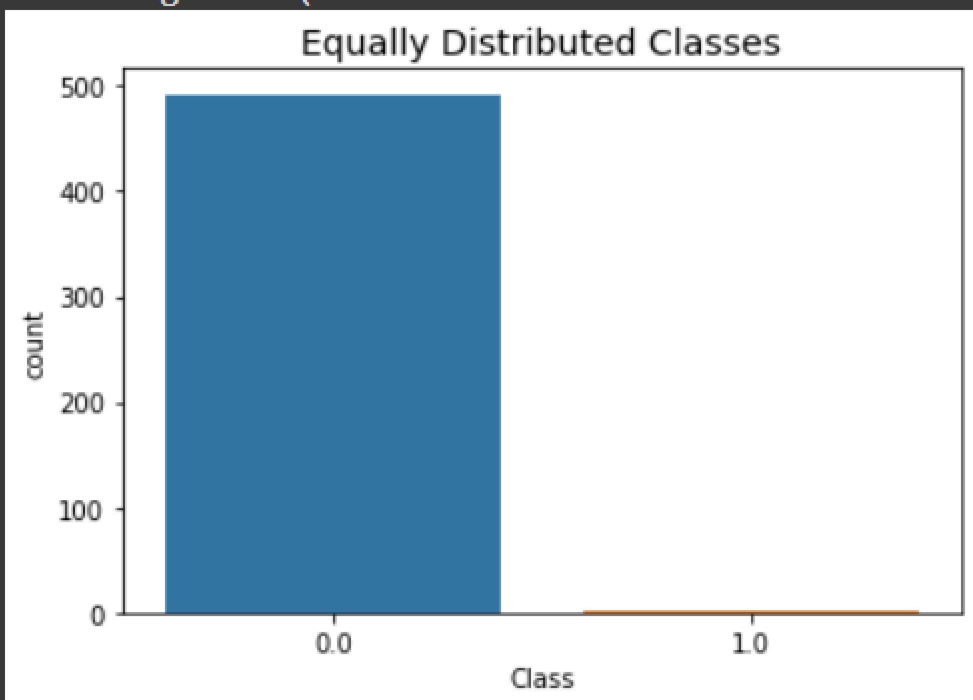
```python
from sklearn.preprocessing import RobustScaler
rob_scaler = RobustScaler()
df['scaled_amount'] =
rob_scaler.fit_transform(df['Amount'].values.reshape(-1,1))
df['scaled_time'] =
rob_scaler.fit_transform(df['Time'].values.reshape(-1,1))

df.drop(['Time','Amount'], axis=1, inplace=True)

df = df.sample(frac=1)

# amount of fraud classes 492 rows.
fraud_df = df.loc[df['Class'] == 1]
non_fraud_df = df.loc[df['Class'] == 0][:492]

normal_distributed_df = pd.concat([fraud_df, non_fraud_df])

# Shuffle dataframe rows
new_df = normal_distributed_df.sample(frac=1, random_state=42)

new_df.head()
```

```
print('No Frauds', round(df['Class'].value_counts()[0]/len(df) * 100,2), '% of the dataset')
print('Frauds', round(df['Class'].value_counts()[1]/len(df) * 100,2), '% of the dataset')

X = df.drop('Class', axis=1)
y = df['Class']
```

```
No Frauds 99.93 % of the dataset
Frauds 0.05 % of the dataset
```

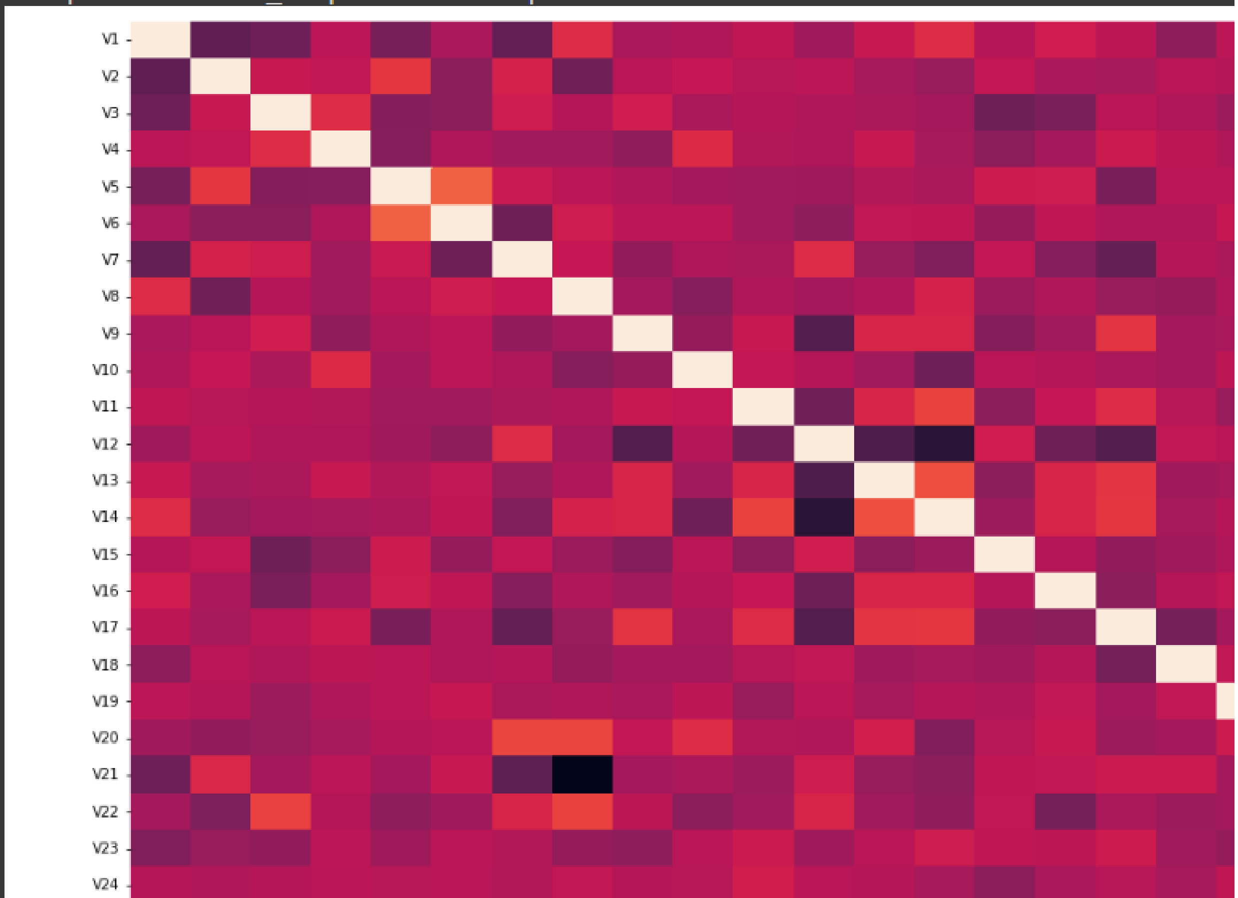
```
import seaborn as sns
import matplotlib.pyplot as plt
print('Distribution of the Classes in the subsample dataset')
print(new_df['Class'].value_counts()/len(new_df))
sns.countplot('Class', data=new_df)
plt.title('Equally Distributed Classes', fontsize=14)
plt.show()
```

```
warnings.warn(
```



```
fig, ax = plt.subplots(figsize=(30,15))
sub_sample_corr = new_df.corr()
sns.heatmap(sub_sample_corr)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd9f3cdf160>
```



```python
X = new_df.drop('Class', axis=1)
y = new_df['Class']
```

```python
from sklearn.model_selection import train_test_split

# This is explicitly used for undersampling.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```python
X_train = X_train.values
X_test = X_test.values
y_train = y_train.values
y_test = y_test.values
```

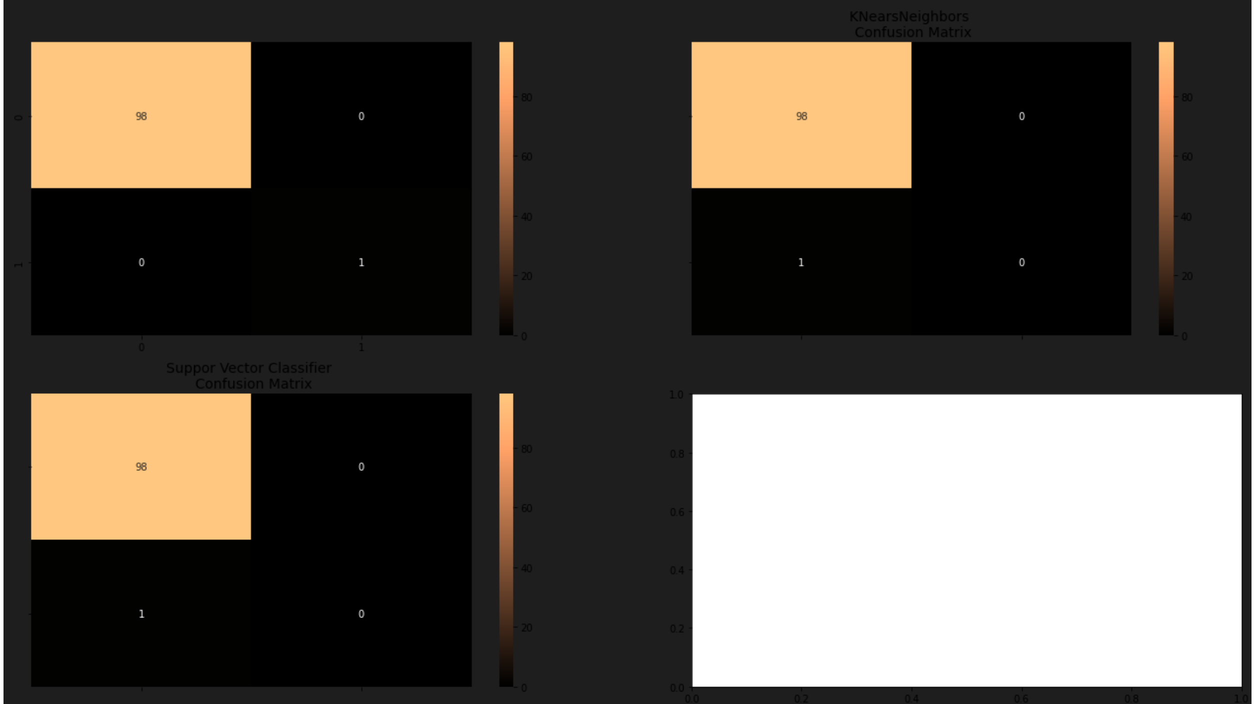
```python
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB

classifiers = {
    "LogisiticRegression": LogisticRegression(),
    "KNearest": KNeighborsClassifier(),
    "Support Vector Classifier": SVC(),
    "RandomForestClassifier": RandomForestClassifier(),
    "Naive Bayes": GaussianNB()
}
```

```python
from sklearn.model_selection import cross_val_score


for key, classifier in classifiers.items():
    classifier.fit(X_train, y_train)
    training_score = cross_val_score(classifier, X_train, y_train, cv=5)
    print("Classifiers: ", classifier.__class__.__name__, "Has a training
score of", round(training_score.mean(), 2) * 100, "% accuracy score")
from sklearn.model_selection import GridSearchCV

log_reg_params = {"penalty": ['l1', 'l2'], 'C': [0.001, 0.01, 0.1, 1, 10,
100, 1000]}
grid_log_reg = GridSearchCV(LogisticRegression(), log_reg_params)
grid_log_reg.fit(X_train, y_train)
log_reg = grid_log_reg.best_estimator_
```

```
knears_params = {"n_neighbors": list(range(2,5,1)), 'algorithm': ['auto',
'ball_tree', 'kd_tree', 'brute']}
grid_knears = GridSearchCV(KNeighborsClassifier(), knears_params)
grid_knears.fit(X_train, y_train)
knears_neighbors = grid_knears.best_estimator_

svc_params = {'C': [0.5, 0.7, 0.9, 1], 'kernel': ['rbf', 'poly',
'sigmoid', 'linear']}
grid_svc = GridSearchCV(SVC(), svc_params)
grid_svc.fit(X_train, y_train)
svc = grid_svc.best_estimator_
log_reg_score = cross_val_score(log_reg, X_train, y_train, cv=5)
print('Logistic Regression Cross Validation Score: ',
round(log_reg_score.mean() * 100, 2).astype(str) + '%')


knears_score = cross_val_score(knears_neighbors, X_train, y_train, cv=5)
print('Knears Neighbors Cross Validation Score', round(knears_score.mean()
* 100, 2).astype(str) + '%')

svc_score = cross_val_score(svc, X_train, y_train, cv=5)
print('Support Vector Classifier Cross Validation Score',
round(svc_score.mean() * 100, 2).astype(str) + '%')
from sklearn.metrics import roc_curve
from sklearn.model_selection import cross_val_predict

log_reg_pred = cross_val_predict(log_reg, X_train, y_train, cv=5,
                             method="decision_function")

knears_pred = cross_val_predict(knears_neighbors, X_train, y_train, cv=5)

svc_pred = cross_val_predict(svc, X_train, y_train, cv=5,
                             method="decision_function")
from sklearn.metrics import roc_auc_score

print('Logistic Regression: ', roc_auc_score(y_train, log_reg_pred))
print('KNears Neighbors: ', roc_auc_score(y_train, knears_pred))
print('Support Vector Classifier: ', roc_auc_score(y_train, svc_pred))
from sklearn.metrics import precision_recall_curve
```

```
precision, recall, threshold = precision_recall_curve(y_train,
log_reg_pred)

from sklearn.metrics import recall_score, precision_score, f1_score,
accuracy_score
y_pred = log_reg.predict(X_train)

# Overfitting Case
print('Recall Score: {:.2f}'.format(recall_score(y_train, y_pred)))
print('Precision Score: {:.2f}'.format(precision_score(y_train, y_pred)))
print('F1 Score: {:.2f}'.format(f1_score(y_train, y_pred)))
print('Accuracy Score: {:.2f}'.format(accuracy_score(y_train, y_pred)))
print('---' * 45)

from sklearn.metrics import confusion_matrix

clf = LogisticRegression(random_state=0).fit(X, y)
y_lr = clf.predict(X_test)
y_pred_knear = knears_neighbors.predict(X_test)
y_pred_svc = svc.predict(X_test)

lr_cf = confusion_matrix(y_test, y_lr)
kneighbors_cf = confusion_matrix(y_test, y_pred_knear)
svc_cf = confusion_matrix(y_test, y_pred_svc)

fig, ax = plt.subplots(2, 2,figsize=(22,12))

sns.heatmap(lr_cf, ax=ax[0][0], annot=True, cmap=plt.cm.copper)
ax[0][1].set_title("L.R. \n Confusion Matrix", fontsize=14)
ax[0][1].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[0][1].set_yticklabels(['', ''], fontsize=14, rotation=360)

sns.heatmap(kneighbors_cf, ax=ax[0][1], annot=True, cmap=plt.cm.copper)
ax[0][1].set_title("KNearsNeighbors \n Confusion Matrix", fontsize=14)
ax[0][1].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[0][1].set_yticklabels(['', ''], fontsize=14, rotation=360)

sns.heatmap(svc_cf, ax=ax[1][0], annot=True, cmap=plt.cm.copper)
ax[1][0].set_title("Suppor Vector Classifier \n Confusion Matrix",
fontsize=14)
```

```python
ax[1][0].set_xticklabels(['', ''], fontsize=14, rotation=90)
ax[1][0].set_yticklabels(['', ''], fontsize=14, rotation=360)
```




```python
from sklearn.metrics import classification_report


print('Logistic Regression:')
print(classification_report(y_test, y_lr))

print('KNears Neighbors:')
print(classification_report(y_test, y_pred_knear))

print('Support Vector Classifier:')
print(classification_report(y_test, y_pred_svc))
```

Output:

```
Logistic Regression:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        98
         1.0       1.00      1.00      1.00         1

    accuracy                           1.00        99
   macro avg       1.00      1.00      1.00        99
weighted avg       1.00      1.00      1.00        99
```

KNears Neighbors:

	precision	recall	f1-score	support
0.0	0.99	1.00	0.99	98
1.0	0.00	0.00	0.00	1
accuracy			0.99	99
macro avg	0.49	0.50	0.50	99
weighted avg	0.98	0.99	0.98	99

Support Vector Classifier:

	precision	recall	f1-score	support
0.0	0.99	1.00	0.99	98
1.0	0.00	0.00	0.00	1
accuracy			0.99	99
macro avg	0.49	0.50	0.50	99
weighted avg	0.98	0.99	0.98	99

Conclusion:

- We almost got the same results in both KNN and SVM.
- We got the maximum accuracy in the logistic regression algorithm.
- We successfully got the expected confusion matrix for all of the three algorithms.
- Logistic regression is a statistical method that is used for building machine learning models where the dependent variable is dichotomous: i.e. binary. Logistic regression is used to describe data and the relationship between one dependent variable and one or more independent variables.
- The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point.
- 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.