SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 of the UGC Act, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA
THANJAVUR | KUMBAKONAM | CHENNAI

SQL injection attack detection in network flow data

Students:

124003377 -> Srijith V
125003241 -> Raakul Aravind A R
125003195 -> Narendrapandian J

Guide:

Dr . Anu P (Assistant Professor),SOC

Layout of this Presentation

- Abstract
- Motivation
- Objective
- Base Paper
- Problem Statement
- Timeline
- Literature Survey
- Hardware and Software
- Data Set
- WorkFlow
- Modules
- Module 1: Data Preprocessing
- Module 2: Graphical Representation - Different Models
- Source Code and outputs
- Conclusion
- References

Abstract

- Research indicates that examining network datagrams can help identify and stop these kinds of attacks. Regretfully, detecting such things typically requires examining every packet that moves via a network of computers. Consequently, the solutions suggested in the literature are typically not applicable to routers responsible for routing large volumes of data. This paper shows that it is possible to identify SQL injection attacks using flow data derived from lightweight protocols. To that end, we collected two databases that gather flow information from many SQL injection assaults on the most widely used database engines.
- After testing a number of machine learning-based methods, we find that a Logistic Regression-based model has a detection rate of over 98.86% and a false alarm rate of less than 0.0081%.

Motivation

- The increasing menace of cyber-attacks poses a heightened worry for both companies and individuals.
- Web applications, essential for online functions, face vulnerability and exploitation, providing opportunities for SQL injections, a prominent security challenge.
- SQL injection attacks (SQLIA) grant unauthorized access to a web application's database, risking information theft or unauthorized modification/deletion of stored data.

Objective

- Develop an efficient machine learning approach for detecting SQL injection attacks using network flow data.
- To find a faster and direct approach using machine learning algorithms.
- Optimize the chosen model's parameters and feature selection techniques to enhance detection accuracy and reduce false positives.
- To compare different ML models and test its accuracy.

Base Paper

- Ignacio Samuel Crespo-Martínez , Adrián Campazas-Vega , Ángel Manuel Guerrero-Higueras , Virginia Riego-DelCastillo , Claudia Álvarez-Aparicio,Camino Fernández-Llamas, “SQL injection attack detection in network flow data”, Journal of Computers and Security, Elsevier, Volume 127, April 2023, 103093

- Base Paper Link:
  [SQL injection attack detection in network flow data - ScienceDirect](https://www.sciencedirect.com/science/article/pii/S0167404823000032)
  (https://www.sciencedirect.com/science/article/pii/S0167404823000032)

- Indexed in: Sciencedirect | SCI-E
  Year: 2023

Problem Statement

- Problem: Developing a robust method for detecting SQL injection attacks using lightweight protocol flow data.
- Challenge: Existing solutions require analyzing all network packets, which is impractical for high-traffic routers.
- Approach: Gathered datasets from SQL injection attacks on popular database engines and evaluated machine learning algorithms minimizing false positives and negatives.

Timeline

- 0th review (22-02-2024)
  - Literature survey and work plan for the existing problem statement
- 1st Review (08-03-2024)
  - Partial implementation of the existing work (at least 40%)
- 2nd Review (13-04-2024)
  - Completion of the existing work (100%) and documentation

LITERATURE SURVEY

Year	Author	Paper Name	Models Adapted	Accuracy
2023	**Ignacio et al.**	**SQL injection attack detection in network flow data, (Base Paper)**	**Logistic Regression(LR), Perceptron with Stochastic Gradient Descent(SGD), Voting Classifier(VC), Random Forest(RF),Linear Support Vector Classification(LSVC),K-Nearest Neighbors(KNN)**	<table><tr><th>Model</th><th>Accuracy</th></tr><tr><td>LR</td><td>97.3%</td></tr><tr><td>Perceptron +SGD</td><td>96.3%</td></tr><tr><td>VC</td><td>85.6%</td></tr><tr><td>RF</td><td>84.0%</td></tr><tr><td>LSVC</td><td>83.4%</td></tr><tr><td>KNN</td><td>71.8%</td></tr></table>
2022	Roy et al.	SQL injection attack detection by machine learning classifier.	Naive Bayes(NB), Logistic Regression(LR)	<table><tr><td>Naive Bayes Accuracy</td><td>Logistic Regression Accuracy</td></tr><tr><td>98.3%</td><td>92.7%</td></tr></table>

2021	Farooq et al.	Ensemble machine learning approaches for detection of SQL injection attack.	Ensemble machine Learning Algorithm - Gradient Boosting Machine (GBM), AdaBoost, Extended GBM (XGBM), Light GBM (LGBM)	Models Accuracy 99%
2020	Tripathy et al.	Detecting SQL injection attacks in cloud SaaS using machine learning.	Random Forest (RF), Boosted Tree Classifier, Adaptive Boosting Classifier (AdaBoost), Decision Tree (DT), SGD Classifier model	RF Accuracy: 99.8% Remaining Model Accuracy: 98.6%
2019	Hasan et al.	Detection of SQL injection attacks: a machine learning approach.	Boosted Trees and Bagged Trees Ensemble, Linear discriminate(LD),SVM Model	93.8% - Boosted Trees and Bagged Trees Ensemble

<table>
<tr><td>2018</td><td>Ross et al.</td><td>Multi-source data analysis and evaluation of machine learning techniques for SQL injection detection.</td><td>jRip, J48, Random Forest(RF), Support Vector Machine(SVM), Artificial Neural Network(ANN)</td><td><table><tr><td>RF Model Accuracy</td><td>ANN model Accuracy</td></tr><tr><td>98.05%</td><td>97.61%</td></tr></table></td></tr>
<tr><td>2017</td><td>Uwagbole et al.</td><td>Applied machine learning predictive analytics to SQL injection attack detection and prevention</td><td>Support Vector Machine (SVM)</td><td><table><tr><td>Accuracy</td><td>F1_Score</td></tr><tr><td>98.6%</td><td>98.5%</td></tr></table></td></tr>
</table>

Hardware & Software

Softwares components used:

- Windows 11 - 23H2
- Jupyter Notebook - 7.0.7
- Python - 3.12.1

Hardware components used:

- Processor - Intel (or) AMD x86 based processor
- RAM - Min 4 GB
- Hard disk with 1 GB free storage space

Dataset

- The two datasets containing flow data are retrieved from several SQL injection attacks on popular database engines (such as MySQL, PostgreSQL, and SQL Server) and using lightweight NetFlow V5 flows.
- Then the datasets are collected using DOROTHEA (A tool used by authors to collect flow data using NetFlow protocol)
- The datasets contains 10 attributes and over 68000 rows.

	A	B	C	D	E	F	G	H	I	J
1	dpkts	doctets	input	output	srcport	dstport	prot	tos	tcp_flags	Label
2	1	228	10	8	53	36602	17	0	0	0
3	1	70	8	10	51085	53	17	0	0	0
4	1	70	8	10	36602	53	17	0	0	0
5	1	228	10	8	53	51085	17	0	0	0
6	1	83	8	10	53358	53	17	0	0	0
7	1	73	10	8	53	33761	17	0	0	0
8	1	73	10	8	53	35098	17	0	0	0
9	1	70	8	10	52792	53	17	0	0	0
10	1	70	8	10	43092	53	17	0	0	0
11	1	228	10	8	53	52792	17	0	0	0
12	1	228	10	8	53	40757	17	0	0	0
13	1	57	8	10	34127	53	17	0	0	0
14	1	57	8	10	35098	53	17	0	0	0
15	1	73	10	8	53	34127	17	0	0	0
16	1	57	8	10	56449	53	17	0	0	0
17	1	228	10	8	53	43092	17	0	0	0
18	1	70	8	10	40757	53	17	0	0	0
19	1	70	8	10	36273	53	17	0	0	0
20	1	57	8	10	33761	53	17	0	0	0

Workflow (Architecture)

Dataset

→

Data Preprocessing:
Feature Cleaning
Dimensionality Reduction

⇩

Classification
Logistic Regression, Linear Support Vector Classification (LSVC), Perceptron with stochastic gradient descent (SGD), Random Forest (RF), K-Nearest Neighbors(KNN), Voting Classifier(VC)

⇩

Ensemble Techniques
Boosted Trees, Bagged Trees, Extended Gradient Boosting Machine(XGBM), Light Gradient Boosting Machine(LGBM)

→

Results

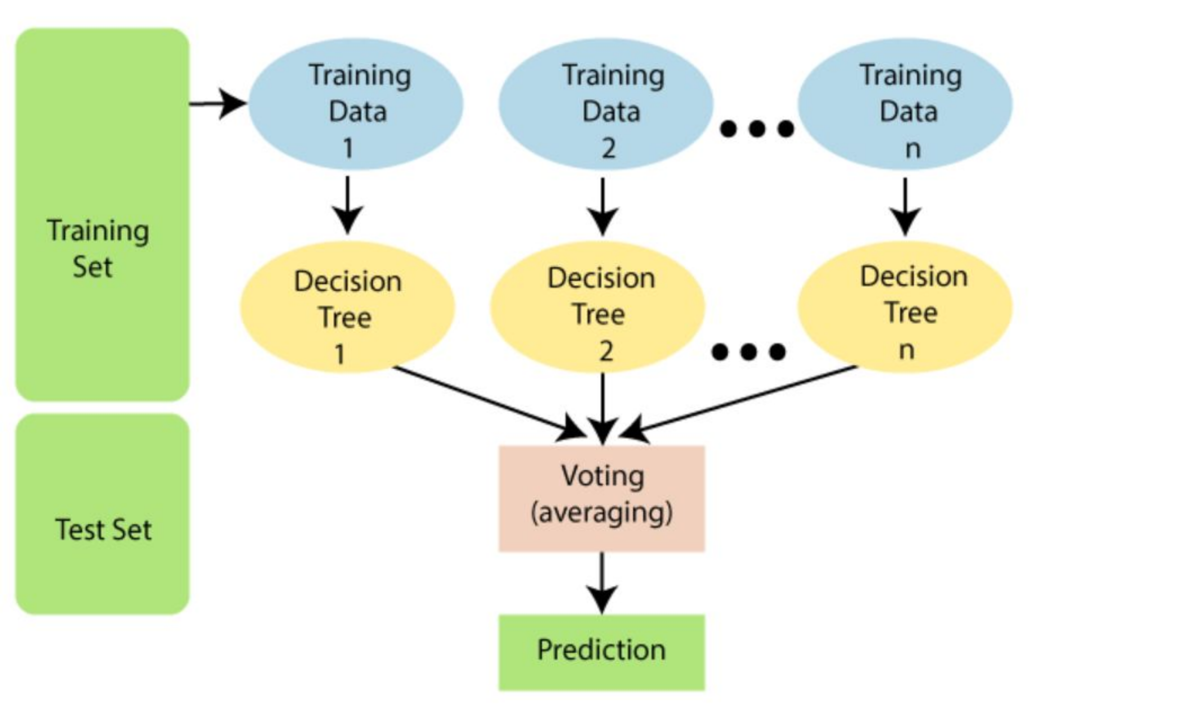
Modules

- Data preprocessing:
  - Data Cleaning, Dimensionality reduction Data Normalization
- Machine Learning algorithms:
  - Logistic Regression, Perceptron with Stochastic Gradient Descent(SGD), Voting Classifier(VC), Random Forest(RF), Linear Support Vector Classification(LSVC), K-Nearest Neighbors(KNN)
- Preparing DataSets:
  - DataSet split into - Training data 80% | Testing data 20%
- Testing & Documentation

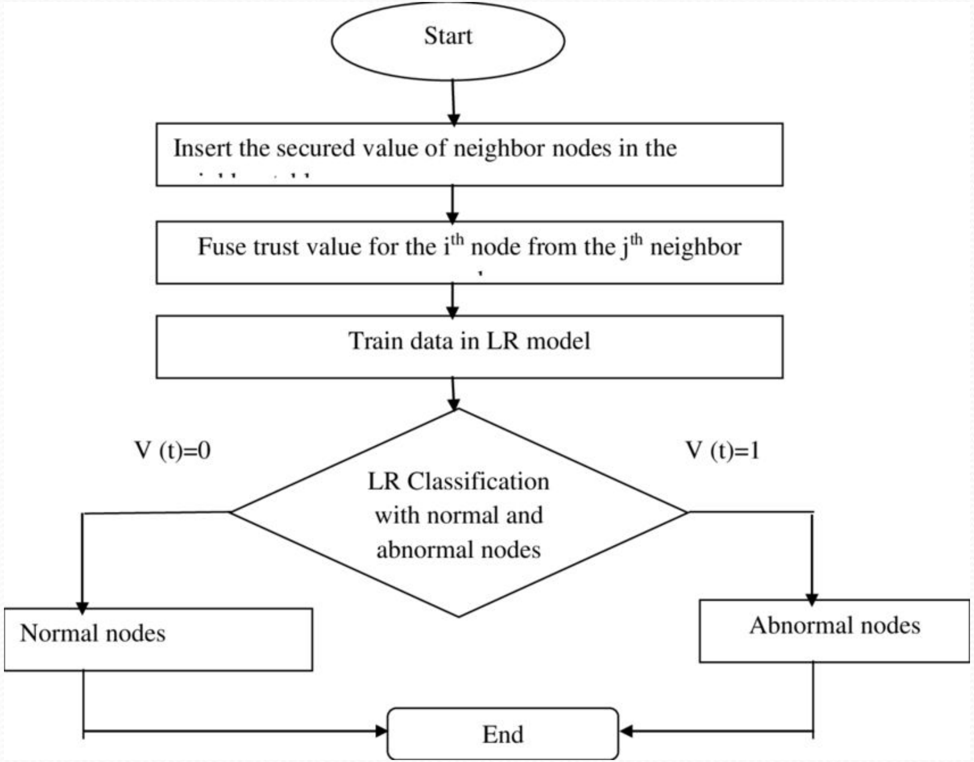
Module-I (Data Preprocessing)

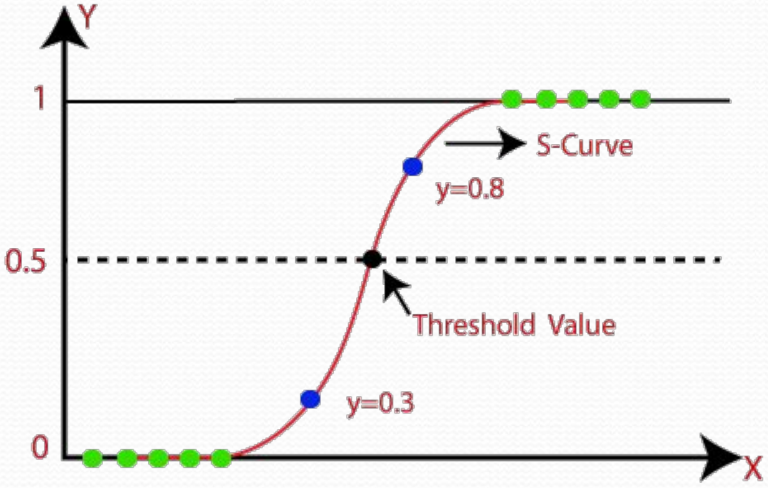
- Dimensionality Reduction
  - Remove bad columns
  - Remove features
  - Search Columns with no Variance
- Data Normalization
  - Min Max Scalar
  - Standard Scalar
  - MaxAbs Scalar
  - Robust Scalar
- Data Cleaning
  - Search Columns with no Variance
  - Search Null Data
  - Search Infinite Values
  - Search Negative Values
  - Remove Duplicate rows

Random Forest (RF)

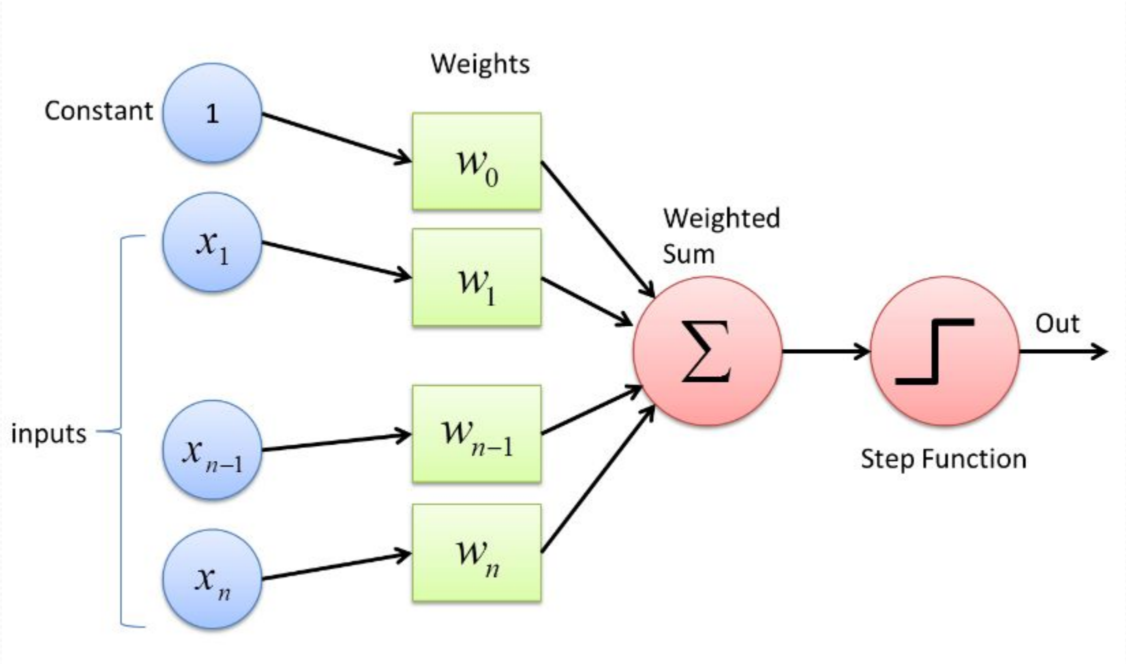


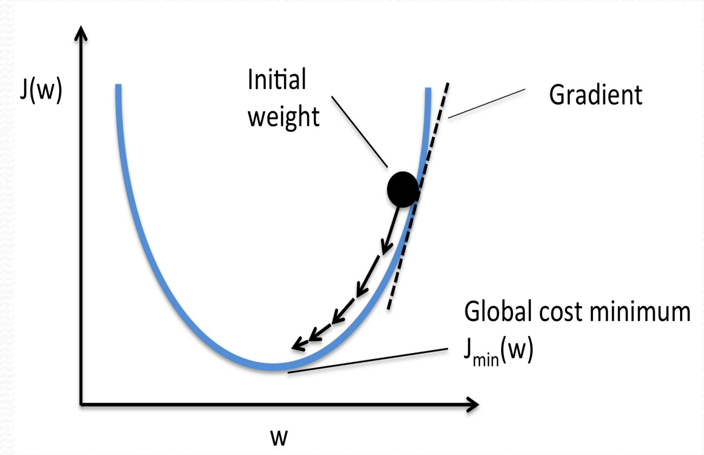
Logistics Regression(LR)



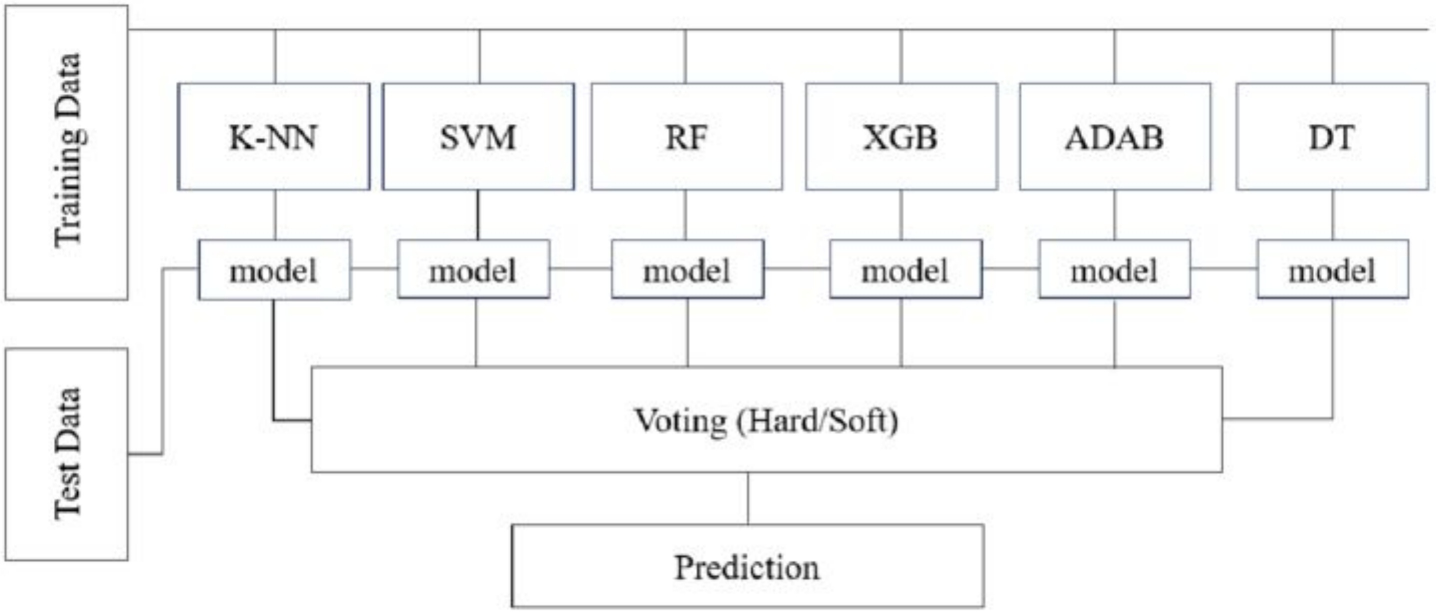


Perceptron+SGD

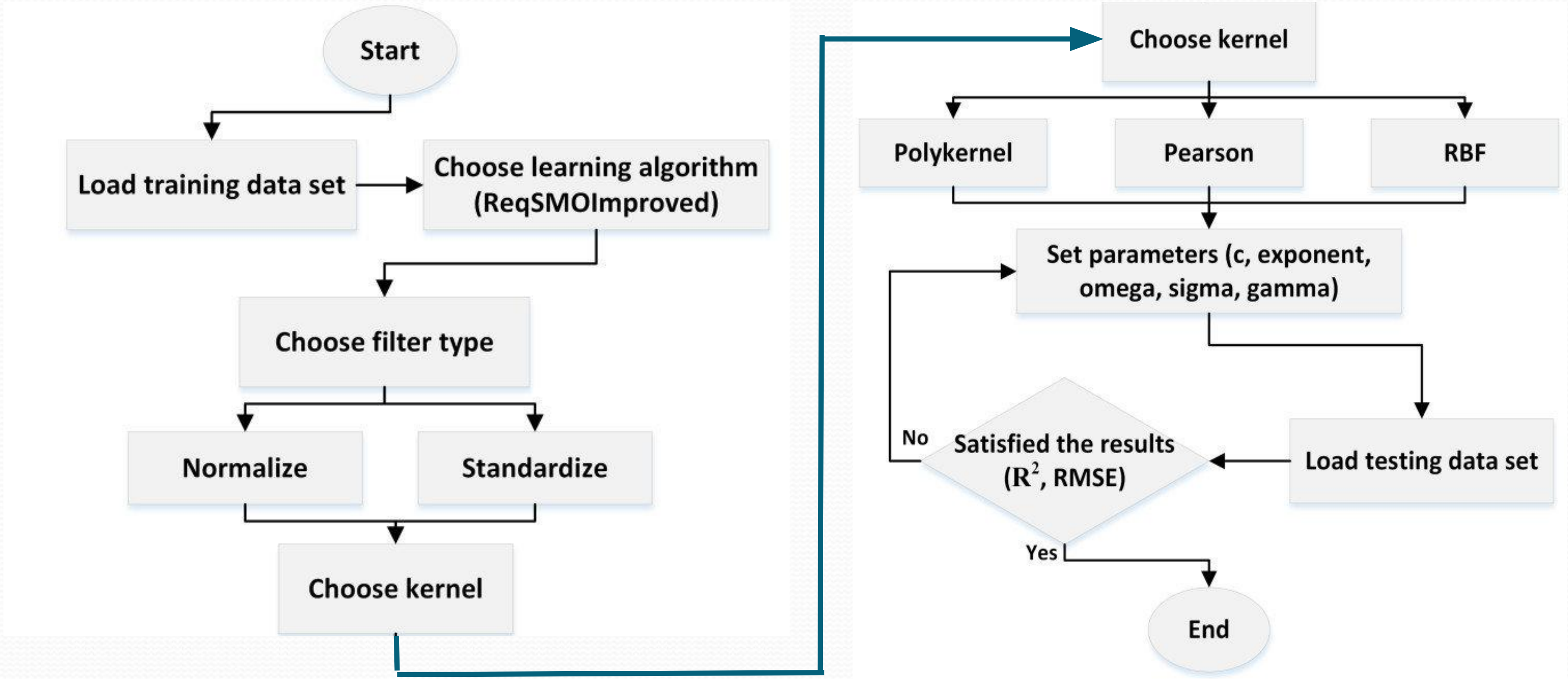




Voting Classifier

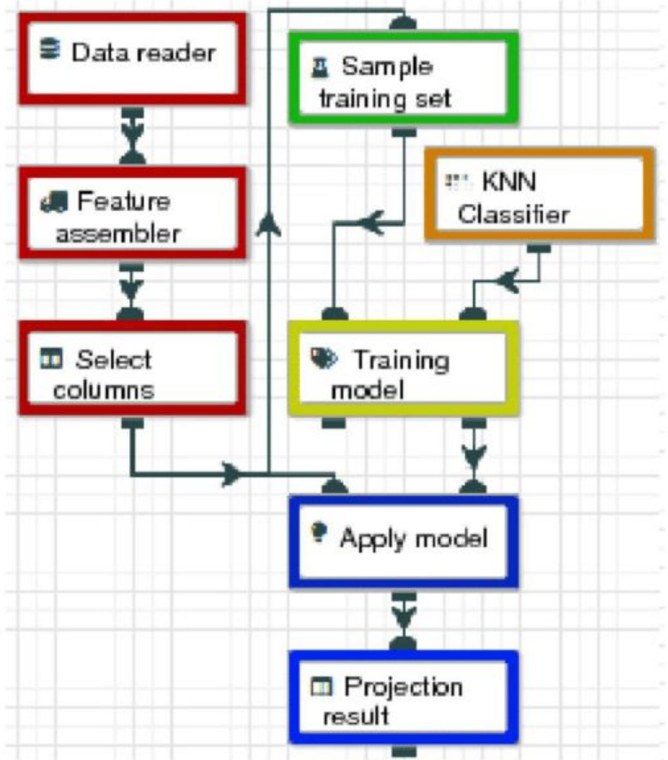


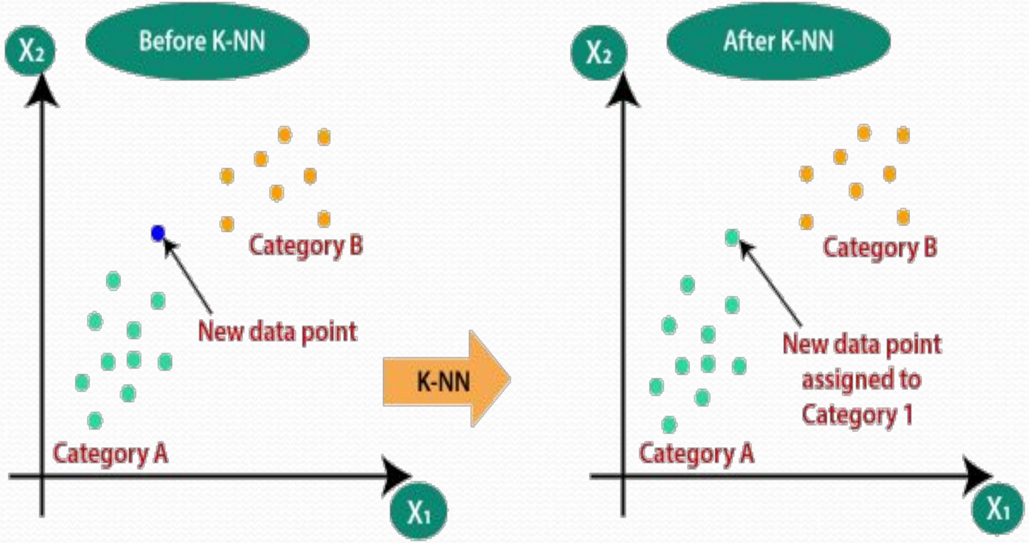
Linear Support Vector Classification(LSVC)



SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 of the UGC Act, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA
THANJAVUR | KUMBAKONAM | CHENNAI

Kth-Nearest Neighbors(KNN)





Source Code: Data Cleaning

```
class Cleaner:
    def search_unimportant(self, df):
        print("Searching for columns with no variance...")
        row_num=len(df)
        for col in df:
            aux=df[col][0]*row_num
            col_sum=sum(df[col])
            if aux!=0:
                if (col_sum)/(aux)==1:
                        print("All values in column",col,"are the
same:",float(df[col][0]))
                if col_sum==0:
                    print("Columns with zero:\n",col)
            if aux==0:
                if col_sum==0:
                    df.drop(col)
                    print("Columns with zero:\n",col)

        print(df)
```

```
def search_infinite(self, df, list):
        #row_num = len(df)
        for col in df:
            flag = False
            for i in list:
                if(col == i):
                    flag = True
            if not (flag):
                inf_list = np.isinf(df[col].astype('float64'))
                inf_num = inf_list.sum()
                if not inf_num == 0:
                    aux = df[col].astype('float64').replace([np.inf, -np.inf], np.nan)
                    aux = aux.dropna()
                    #Remove infinite values and find mean of that column
                    mean = aux.mean()

                    #Substitute infinite values with mean in orig dataframe
                    df[col] = df[col].astype('float64').replace([np.inf, -np.inf],
mean,inplace=True)
                    print("Column %s have %i infinite values" % (col, inf_num))
        return df
```

```
def search_null(self, df):
        row_num = len(df)
        for col in df:
            null_list = df[col].isnull()
            null_num = null_list.sum()
            if not null_num == 0:
                print("Column %s have %i Null values" % (col, nan_null))
        return df.fillna(0)
```

```
def search_negatives(self, df):
        row_num = len(df)
        for col in df:
            neg_list = df[col] < 0
            neg_num = neg_list.sum()
            if not neg_num == 0:
                print("Column %s have %i negative values" % (col, neg_num))
        print(df)
```

```
    def removeFeatures(self,
df,list=['nexthop','engine_id','engine_type','src_mask','dst_mask','src_a
s','dst_as','#:unix_secs','unix_nsecs','sysuptime','first','last', 'exaddr']):
        print("Removing features...")
        df = df.drop(columns=list)
        print(df)
```

After Data Cleaning

Data:

	dpkts	doctets	input	output	srcport	dstport	prot	tos	tcp_flags	Label
0	1	228	10	8	53	36602	17	0	0	0
1	1	70	8	10	51085	53	17	0	0	0
2	1	70	8	10	36602	53	17	0	0	0
3	1	228	10	8	53	51085	17	0	0	0
4	1	83	8	10	53358	53	17	0	0	0
...	...	...	...	...	...	...	...	...	...	...
57224	11	812	65535	4851	56298	22	6	8	25	1
57225	36	6370	4851	65535	22	56298	6	0	26	1
57226	27	3337	65535	4851	56298	22	6	0	26	1
57227	18	3185	4851	4851	22	57932	6	0	26	1
57228	5	456	4851	4851	22	57932	6	8	25	1

[57229 rows x 10 columns]

Data Information:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 57229 entries, 0 to 57228
Data columns (total 10 columns):

#	Column	Non-Null Count	Dtype
0	dpkts	57229 non-null	int64
1	doctets	57229 non-null	int64
2	input	57229 non-null	int64
3	output	57229 non-null	int64
4	srcport	57229 non-null	int64
5	dstport	57229 non-null	int64
6	prot	57229 non-null	int64
7	tos	57229 non-null	int64
8	tcp_flags	57229 non-null	int64
9	Label	57229 non-null	int64

dtypes: int64(10)
memory usage: 4.4 MB

Source Code: Normalization

```
class preprocess:
    def new_dataset(self,dataframe): # function to remove "Label" column
        lc=dataframe["Label"]
        f_lc=dataframe.drop("Label",axis=1)
        new_columns=dataframe.columns.drop("Label")
        print(lc,f_lc,new_columns)
    def minScalingNormal_train(self,dataframe):
        heading=dataframe.columns
        class_call=MinMaxScaler()
        #class_call.fit(dataframe)
        scaled=class_call.fit_transform(dataframe)
        scaled=pd.DataFrame(scaled,columns=heading)
        return (scaled)
    def minScalingNormal_test(self,dataframe,test_df):
        heading=dataframe.columns
        class_call=MinMaxScaler()
        #class_call.fit(dataframe)
        scaled_train=self.minScalingNormal_train(dataframe)
        scaled_test=class_call.fit_transform(test_df)
        scaled=pd.DataFrame(scaled_test,columns=heading)
        print(scaled)
```

```
def maxabsNormal_train(self,dataframe):
    heading=dataframe.columns
    class_call=MaxAbsScaler()
    trained=class_call.fit_transform(dataframe)
    trained=pd.DataFrame(trained,columns=heading)
    print(trained)
def maxabsNormal_test(self,dataframe,test_df):
    heading=dataframe.columns
    scale=MaxAbsScaler()
    scale_trained=self.maxabsNormal_train(dataframe)
    scale_test=scale.fit_transform(test_df)
    scaled_test=pd.DataFrame(scale_test,columns=heading)
    print(scale_test)
def robust_train(self,dataframe):
    heading=dataframe.columns
    class_call=RobustScaler()
    trained=class_call.fit_transform(dataframe)
    trained=pd.DataFrame(trained,columns=heading)
    print(trained)
```

```
def robust_test(self,dataframe,test_df):
        heading=dataframe.columns
        scale=RobustScaler()
        scale_trained=self.robust_train(dataframe)
        scale_test=scale.fit_transform(test_df)
        scaled_test=pd.DataFrame(scale_test,columns=heading)
        print(scale_test)

def variance_cal(self,dataframe,limit=0,list=["Label"]):
        for column in dataframe:
            f=False
            for itr in list:
                if itr==column:
                    f=True
            if f==False:
                variance=dataframe[column].var()
                print("Columns\n:",column,"Variance:\n",variance)  # Reference
                if variance==limit:
                    print("Columns\n:",column,"Variance:\n",variance)
                    dataframe=dataframe.drop(column,axis=1)
        print(dataframe)
```

After Normalization

MaxAbs Normalization Train:

	dpkts	doctets	input	output	srcport	dstport	prot \
0	0.000153	0.000024	0.000153	0.000122	0.000869	0.600052	1.000000
1	0.000153	0.000007	0.000122	0.000153	0.837486	0.000869	1.000000
2	0.000153	0.000007	0.000122	0.000153	0.600052	0.000869	1.000000
3	0.000153	0.000024	0.000153	0.000122	0.000869	0.837486	1.000000
4	0.000153	0.000009	0.000122	0.000153	0.874750	0.000869	1.000000
...	...	...	...	...	...	...	...
57224	0.001679	0.000085	1.000000	0.074022	0.922948	0.000361	0.352941
57225	0.005496	0.000668	0.074022	1.000000	0.000361	0.922948	0.352941
57226	0.004122	0.000350	1.000000	0.074022	0.922948	0.000361	0.352941
57227	0.002748	0.000334	0.074022	0.074022	0.000361	0.949736	0.352941
57228	0.000763	0.000048	0.074022	0.074022	0.000361	0.949736	0.352941

	tos	tcp_flags	Label
0	0.000000	0.000000	0.0
1	0.000000	0.000000	0.0
2	0.000000	0.000000	0.0
3	0.000000	0.000000	0.0
4	0.000000	0.000000	0.0
...	...	...	...
57224	0.041667	0.806452	1.0
57225	0.000000	0.838710	1.0
57226	0.000000	0.838710	1.0
57227	0.000000	0.838710	1.0
57228	0.041667	0.806452	1.0

[57229 rows x 10 columns]

Random Forest

```
print("Random Forest Algo....")
data=pd.read_csv("E:\\jupyter\\dataset\\MinScale\\minscaleTrain.csv")
data=data.sample(frac=1)
print(data)
x=data.drop(columns=['Label'])
y=data['Label']

x_train=pd.read_csv("E:\\jupyter\\dataset\\rf_Model\\x.train.csv")
x_test=pd.read_csv("E:\\jupyter\\dataset\\rf_Model\\x.test.csv")
y_train=pd.read_csv("E:\\jupyter\\dataset\\rf_Model\\y.train.csv")
y_test=pd.read_csv("E:\\jupyter\\dataset\\rf_Model\\y.test.csv")

rf=RandomForestClassifier(criterion='gini',n_estimators=80)
start_time=time.time()
rf.fit(x_train,y_train["Label"].values.ravel())
execution_time=time.time()-start_time
prediction=rf.predict(x_test)
m=classification_report(y_test["Label"].values.ravel(),prediction,digits=6)
print(m)
cohen_kappa=cohen_kappa_score(y_test["Label"].values.ravel(),prediction)
matthew_corr=matthews_corrcoef(y_test["Label"].values.ravel(),prediction)

print("Confusion Matrics\n")
cm = confusion_matrix(y_test["Label"].values.ravel(), prediction)
print(cm)

print("\nKappa Score");                print(cohen_kappa)
print("Matthew_correlation");          print(matthew_corr)
print("Execution Time:",execution_time,"Seconds")
```

Random Forest - Output

	Unnamed: 0	dpkts	doctets	input	output	srcport	dstport
36391	3406	0.000153	0.000011	0.000000	0.000031	0.882980	0.006904
36176	1532	0.000305	0.000012	0.000031	0.000000	0.007263	0.936959
784	54581	0.000611	0.000098	0.073908	0.073908	0.728942	0.000951
12593	3380	0.003054	0.000606	0.000000	0.000031	0.985704	0.006904
34158	10063	0.000000	0.000002	0.000000	0.000031	0.673547	0.000508
...	...	...	...	...	...	...	...
17446	11458	0.000458	0.000019	0.000031	0.000000	0.007263	0.850958
27488	32571	0.000611	0.000086	0.073908	0.073908	0.569166	0.006904
3312	32734	0.000611	0.000054	0.073908	0.073908	0.007263	0.537917
9272	40097	0.000611	0.000081	0.073908	0.073908	0.007263	0.842397
22321	49097	0.000611	0.000069	0.073908	0.073908	0.001312	0.545329

Random Forest - Output

```
         prot  tos  tcp_flags  Label
36391  0.3125  0.0   0.774194      0
36176  0.3125  0.0   0.548387      0
784    0.3125  0.0   0.870968      1
12593  0.3125  0.0   0.870968      0
34158  1.0000  0.0   0.000000      0
...       ...  ...        ...    ...
17446  0.3125  0.0   0.548387      0
27488  0.3125  0.0   0.870968      1
3312   0.3125  0.0   0.870968      1
9272   0.3125  0.0   0.870968      1
22321  0.3125  0.0   0.870968      1

[57229 rows x 11 columns]
```

```
[57229 rows x 11 columns]
              precision    recall  f1-score   support

           0   0.808883  0.873447  0.839926      5713
           1   0.862990  0.794349  0.827248      5733

    accuracy                       0.833828     11446
   macro avg   0.835937  0.833898  0.833587     11446
weighted avg   0.835984  0.833828  0.833576     11446

Confusion Matrics

[[4990  723]
 [1179 4554]]

Kappa Score
0.6677020729709133
Matthew_correlation
0.6698311433602062
Execution Time: 26.504883766174316 Seconds
```

KNN Classifier

```
print("Implementation of KNN algo....\n")
data=pd.read_csv("E:\\jupyter\\dataset\\MinScale\\minscaleTrain.csv")
data=data.sample(frac=1)
print(data)
x=data.drop(columns=['Label'])
y=data['Label']
x_train=pd.read_csv("E:\\jupyter\\dataset\\knn\\x_train.csv")
x_test=pd.read_csv("E:\\jupyter\\dataset\\knn\\x_test.csv")
y_train=pd.read_csv("E:\jupyter\dataset\knn\y_train.csv")
y_test=pd.read_csv("E:\jupyter\dataset\knn\y_test.csv")

knn=KNeighborsClassifier(n_neighbors=1,p=2,algorithm="ball_tree")
start_time=time.time()
knn.fit(x_train,y_train["Label"].values.ravel())
execution_time=time.time()-start_time
knn=KNeighborsClassifier(n_neighbors=1,p=2,algorithm="ball_tree")
knn.fit(x_train,y_train["Label"].values.ravel())
prediction=knn.predict(x_test)
m=classification_report(y_test["Label"].values.ravel(),prediction,digits=6)
print(m);                    print("Confusion Matrix\n")
cm = confusion_matrix(y_test["Label"].values.ravel(), prediction)
print(cm)
cohen_kappa=cohen_kappa_score(y_test["Label"].values.ravel(),prediction)
matthew_corr=matthews_corrcoef(y_test["Label"].values.ravel(),prediction)

print("\nKappa Score");                print(cohen_kappa)
print("Matthew_correlation");          print(matthew_corr)
print("Execution Time:",execution_time,"Seconds")
```

KNN Classifier - Output

	Unnamed: 0	dpkts	doctets	input	output	srcport	dstport
36391	3406	0.000153	0.000011	0.000000	0.000031	0.882980	0.006904
36176	1532	0.000305	0.000012	0.000031	0.000000	0.007263	0.936959
784	54581	0.000611	0.000098	0.073908	0.073908	0.728942	0.000951
12593	3380	0.003054	0.000606	0.000000	0.000031	0.985704	0.006904
34158	10063	0.000000	0.000002	0.000000	0.000031	0.673547	0.000508
...	...	...	...	...	...	...	...
17446	11458	0.000458	0.000019	0.000031	0.000000	0.007263	0.850958
27488	32571	0.000611	0.000086	0.073908	0.073908	0.569166	0.006904
3312	32734	0.000611	0.000054	0.073908	0.073908	0.007263	0.537917
9272	40097	0.000611	0.000081	0.073908	0.073908	0.007263	0.842397
22321	49097	0.000611	0.000069	0.073908	0.073908	0.001312	0.545329

KNN Classifier - Output

```
        prot  tos  tcp_flags  Label
36391  0.3125  0.0   0.774194      0
36176  0.3125  0.0   0.548387      0
784    0.3125  0.0   0.870968      1
12593  0.3125  0.0   0.870968      0
34158  1.0000  0.0   0.000000      0
...       ...  ...        ...    ...
17446  0.3125  0.0   0.548387      0
27488  0.3125  0.0   0.870968      1
3312   0.3125  0.0   0.870968      1
9272   0.3125  0.0   0.870968      1
22321  0.3125  0.0   0.870968      1

[57229 rows x 11 columns]
```

```
[57229 rows x 11 columns]
              precision    recall  f1-score   support

           0   0.725443  0.722933  0.724186      5782
           1   0.718156  0.720692  0.719422      5664

    accuracy                       0.721824     11446
   macro avg   0.721799  0.721813  0.721804     11446
weighted avg   0.721837  0.721824  0.721828     11446

Confusion Matrix

[[4180 1602]
 [1582 4082]]

Kappa Score
0.44360934763967064
Matthew_correlation
0.4436120568841371
Execution Time: 0.06882309913635254 Seconds
```

Voting Classifier

```
print("Voting Classifier Algo...\n")
data=pd.read_csv("E:\\jupyter\\dataset\\MinScale\\robustScaleTrain.csv")
data=data.sample(frac=1)
print(data);            x=data.drop(columns=['Label'])
y=data['Label']
x_train=pd.read_csv("E:\\jupyter\\dataset\\VC\\x_train.csv")
x_test=pd.read_csv("E:\\jupyter\\dataset\\VC\\x_test.csv")
y_train=pd.read_csv("E:\\jupyter\\dataset\\VC\\y_train.csv")
y_test=pd.read_csv("E:\\jupyter\\dataset\\VC\\y_test.csv")

robust=VotingClassifier(estimators=[("Logistic_Regression_Classifier",LogisticRegression(dual=False)),("KNeighbors_Classifier",KNeighborsClassifier(n_neighbors=5)),("SGD_Classifier",SGDClassifier()),("LinearSVC",LinearSVC(dual=False)),("Random_Forest_Classifier",RandomForestClassifier(n_estimators=100))],voting="hard")
```

```
start_time=time.time()
robust.fit(x_train,y_train["Label"].values.ravel())
execution_time=time.time()-start_time
prediction=robust.predict(x_test)
m=classification_report(y_test["Label"].values.ravel(),prediction,digits=6,zero_division=1)
print(m);            print("Confusion Matrix\n")
cm = confusion_matrix(y_test["Label"].values.ravel(), prediction)
print(cm)
cohen_kappa=cohen_kappa_score(y_test["Label"].values.ravel(),prediction)
matthew_corr=matthews_corrcoef(y_test["Label"].values.ravel(),prediction)
print("\nKappa Score");            print(cohen_kappa)
print("Matthew_correlation");      print(matthew_corr)
print("Execution Time:",execution_time,"Seconds")
```

Voting Classifier - Output

	Unnamed: 0	dpkts	doctets	input	output	srcport	dstport
36391	3406	0.000153	0.000011	0.000000	0.000031	0.882980	0.006904
36176	1532	0.000305	0.000012	0.000031	0.000000	0.007263	0.936959
784	54581	0.000611	0.000098	0.073908	0.073908	0.728942	0.000951
12593	3380	0.003054	0.000606	0.000000	0.000031	0.985704	0.006904
34158	10063	0.000000	0.000002	0.000000	0.000031	0.673547	0.000508
...	...	...	...	...	...	...	...
17446	11458	0.000458	0.000019	0.000031	0.000000	0.007263	0.850958
27488	32571	0.000611	0.000086	0.073908	0.073908	0.569166	0.006904
3312	32734	0.000611	0.000054	0.073908	0.073908	0.007263	0.537917
9272	40097	0.000611	0.000081	0.073908	0.073908	0.007263	0.842397
22321	49097	0.000611	0.000069	0.073908	0.073908	0.001312	0.545329

Voting Classifier - Output

```
         prot  tos  tcp_flags  Label
36391  0.3125  0.0   0.774194      0
36176  0.3125  0.0   0.548387      0
784    0.3125  0.0   0.870968      1
12593  0.3125  0.0   0.870968      0
34158  1.0000  0.0   0.000000      0
...       ...  ...        ...    ...
17446  0.3125  0.0   0.548387      0
27488  0.3125  0.0   0.870968      1
3312   0.3125  0.0   0.870968      1
9272   0.3125  0.0   0.870968      1
22321  0.3125  0.0   0.870968      1

[57229 rows x 11 columns]
```

```
[57229 rows x 11 columns]
              precision    recall  f1-score   support

           0   0.798318  0.875177  0.834983      5640
           1   0.866236  0.785222  0.823742      5806

    accuracy                       0.829547     11446
   macro avg   0.832277  0.830200  0.829362     11446
weighted avg   0.832769  0.829547  0.829281     11446

Confusion Matrix

[[4936  704]
 [1247 4559]]

Kappa Score
0.6594918121970941
Matthew_correlation
0.6624734656580583
Execution Time: 107.90777206420898 Seconds
```

Perceptron + SGD

```
print("Perceptron+SGD....")

data=pd.read_csv("E:\\jupyter\\dataset\\MinScale\\robustscaleTrain.csv")
data=data.sample(frac=1)
print(data)
x=data.drop(columns=['Label'])
y=data['Label']

x_train=pd.read_csv("E:\\jupyter\\dataset\\perceptron+SGD\\x_train.csv")
x_test=pd.read_csv("E:\\jupyter\\dataset\\perceptron+SGD\\x_test.csv")
y_train=pd.read_csv("E:\\jupyter\\dataset\\perceptron+SGD\\y_train.csv")
y_test=pd.read_csv("E:\\jupyter\\dataset\\perceptron+SGD\\y_test.csv")

sgd=SGDClassifier(loss="perceptron",n_iter_no_change=9,penalty=None,
learning_rate="optimal",early_stopping=False,max_iter=1400,shuffle=True)
start_time=time.time()
sgd.fit(x_train,y_train["Label"].values.ravel())
execution_time=time.time()-start_time
prediction=sgd.predict(x_test)
m=classification_report(y_test["Label"].values.ravel(),prediction,digits=
6,zero_division=1)
print(m)
cm=confusion_matrix(y_test["Label"].values.ravel(),prediction)
print(cm)
cohen_kappa=cohen_kappa_score(y_test["Label"].values.ravel(),predicti
on)
matthew_corr=matthews_corrcoef(y_test["Label"].values.ravel(),predicti
on)

print("\nKappa Score");                    print(cohen_kappa)
print("Matthew_correlation");              print(matthew_corr)
print("Execution Time:",execution_time,"Seconds")
```

Perceptron + SGD

	Unnamed: 0	dpkts	doctets	input	output	srcport	dstport
40961	36183	0.0	-0.026087	0.999587	1.000000	0.225957	-0.003049
48490	50720	0.0	0.088406	0.999587	1.000000	-0.692558	0.762704
26372	3621	-4.0	-0.820290	0.000000	-0.000413	-0.693130	1.150758
37369	22625	-3.0	-0.772464	0.000000	-0.000413	-0.684876	0.838253
36882	44908	0.0	-0.117391	0.999587	1.000000	-0.692558	1.020687
...	...	...	...	...	...	...	...
45721	14247	67.0	5.500000	-0.000413	0.000000	0.555537	-0.003049
39848	4429	7.0	1.105797	-0.000413	0.000000	0.433627	-0.003049
26845	19692	18.0	2.215942	-0.000413	0.000000	0.022350	-0.003049
29118	53597	0.0	0.256522	0.999587	1.000000	-0.692558	0.886021
25977	12774	0.0	-0.284058	0.000000	-0.000413	-0.684876	1.049610

Perceptron + SGD

	prot	tos	tcp_flags	Label
40961	0.0	0.0	0.000000	1
48490	0.0	0.0	0.000000	1
26372	11.0	0.0	-9.000000	0
37369	0.0	0.0	-3.666667	0
36882	0.0	0.0	0.000000	1
...	...	...	...	...
45721	0.0	0.0	1.333333	0
39848	0.0	0.0	1.333333	0
26845	0.0	0.0	1.333333	0
29118	0.0	0.0	0.000000	1
25977	0.0	0.0	-1.000000	0

[57229 rows x 11 columns]

```
              precision    recall  f1-score   support

           0   0.999199  0.874715  0.932823      5707
           1   0.889147  0.999303  0.941012      5739

    accuracy                       0.937183     11446
   macro avg   0.944173  0.937009  0.936917     11446
weighted avg   0.944019  0.937183  0.936929     11446

[[4992  715]
 [   4 5735]]

Kappa Score
0.8743219568971681
Matthew_correlation
0.8811533377750003
Execution Time: 4.104285001754761 Seconds
```

LSVC

```
print("Linear Support Vector Classification\n")
data=pd.read_csv("E:\\jupyter\\dataset\\MinScale\\robustscaleTrain.csv")
data=data.sample(frac=1)
print(data)
x=data.drop(columns=['Label'])
y=data['Label']

x_train=pd.read_csv("E:\\jupyter\\dataset\\LSVC\\x_train.csv")
x_test=pd.read_csv("E:\\jupyter\\dataset\\LSVC\\x_test.csv")
y_train=pd.read_csv("E:\\jupyter\\dataset\\LSVC\\y_train.csv")
y_test=pd.read_csv("E:\\jupyter\\dataset\\LSVC\\y_test.csv")

lsvc=LinearSVC(dual=True,max_iter=1800,loss="squared_hinge")
start_time=time.time()
lsvc.fit(x_train,y_train["Label"].values.ravel())
execution_time=time.time()-start_time
prediction=lsvc.predict(x_test)
m=classification_report(y_test["Label"].values.ravel(),prediction,digits=6,zero_division=1)
print(m)
print("Confusion Matrix\n")
cm = confusion_matrix(y_test["Label"].values.ravel(), prediction)
print(cm)

cohen_kappa=cohen_kappa_score(y_test["Label"].values.ravel(),prediction)
matthew_corr=matthews_corrcoef(y_test["Label"].values.ravel(),prediction)
print("\nKappa Score");                print(cohen_kappa)
print("Matthew_correlation");          print(matthew_corr)
print("Execution Time:",execution_time,"Seconds")
```

LSVC

	Unnamed: 0	dpkts	doctets	input	output	srcport	dstport
37158	41753	0.0	0.001449	0.999587	1.000000	0.124238	-0.010735
13167	22192	0.0	-0.234783	0.000000	-0.000413	-0.684876	0.972792
13558	56164	0.0	0.165217	0.999587	1.000000	-0.684876	0.954370
47068	49505	0.0	0.089855	0.999587	1.000000	-0.684876	1.013022
37531	16574	70.0	5.743478	-0.000413	0.000000	0.380545	-0.003049
...	...	...	...	...	...	...	...
3615	50789	0.0	0.004348	0.999587	1.000000	0.083686	-0.010735
55406	7474	-4.0	-0.533333	0.000000	-0.000413	-0.693130	1.109744
15286	48850	0.0	0.069565	0.999587	1.000000	-0.692558	0.681820
32833	39816	0.0	0.050725	0.999587	1.000000	0.164409	-0.010735
40050	5620	-4.0	-0.676812	0.000000	-0.000413	-0.693130	1.083849

LSVC

	prot	tos	tcp_flags	Label
37158	0.0	0.0	0.000000	1
13167	0.0	0.0	-1.000000	0
13558	0.0	0.0	0.000000	1
47068	0.0	0.0	0.000000	1
37531	0.0	0.0	1.333333	0
...	...	...	...	...
3615	0.0	0.0	0.000000	1
55406	11.0	0.0	-9.000000	0
15286	0.0	0.0	0.000000	1
32833	0.0	0.0	0.000000	1
40050	11.0	0.0	-9.000000	0

[57229 rows x 11 columns]

	precision	recall	f1-score	support
0	0.767331	0.870238	0.815551	5749
1	0.848559	0.733720	0.786972	5697
accuracy			0.802289	11446
macro avg	0.807945	0.801979	0.801262	11446
weighted avg	0.807760	0.802289	0.801326	11446

```
Confusion Matrix

[[5003  746]
 [1517 4180]]

Kappa Score
0.6043276845199257
Matthew_correlation
0.6098947004152991
Execution Time: 12.499801397323608 Seconds
```

Logistic Regression Classifier

```
print("Logistic Regression Classifier....")
data=pd.read_csv("E:\\jupyter\\dataset\\MinScale\\robustscaleTrain.csv")
data=data.sample(frac=1)
print(data)
x=data.drop(columns=['Label'])
y=data['Label']

x_train=pd.read_csv("E:\\jupyter\\dataset\\LR\\x_train.csv")
x_test=pd.read_csv("E:\\jupyter\\dataset\\LR\\x_test.csv")
y_train=pd.read_csv("E:\\jupyter\\dataset\\LR\\y_train.csv")
y_test=pd.read_csv("E:\\jupyter\\dataset\\LR\\y_test.csv")

lr=LogisticRegression(dual=False,max_iter=150,intercept_scaling=1.0)
start_time=time.time()
lr.fit(x_train,y_train["Label"].values.ravel())
execution_time=time.time()-start_time
prediction=lr.predict(x_test)
m=classification_report(y_test["Label"].values.ravel(),prediction,digits=6,zero_division=1)
print(m)
cm=confusion_matrix(y_test["Label"].values.ravel(),prediction)
print(cm)

cohen_kappa=cohen_kappa_score(y_test["Label"].values.ravel(),prediction)
matthew_corr=matthews_corrcoef(y_test["Label"].values.ravel(),prediction)

print("\nKappa Score");                print(cohen_kappa)
print("Matthew_correlation");          print(matthew_corr)
print("Execution Time:",execution_time,"Seconds")
```

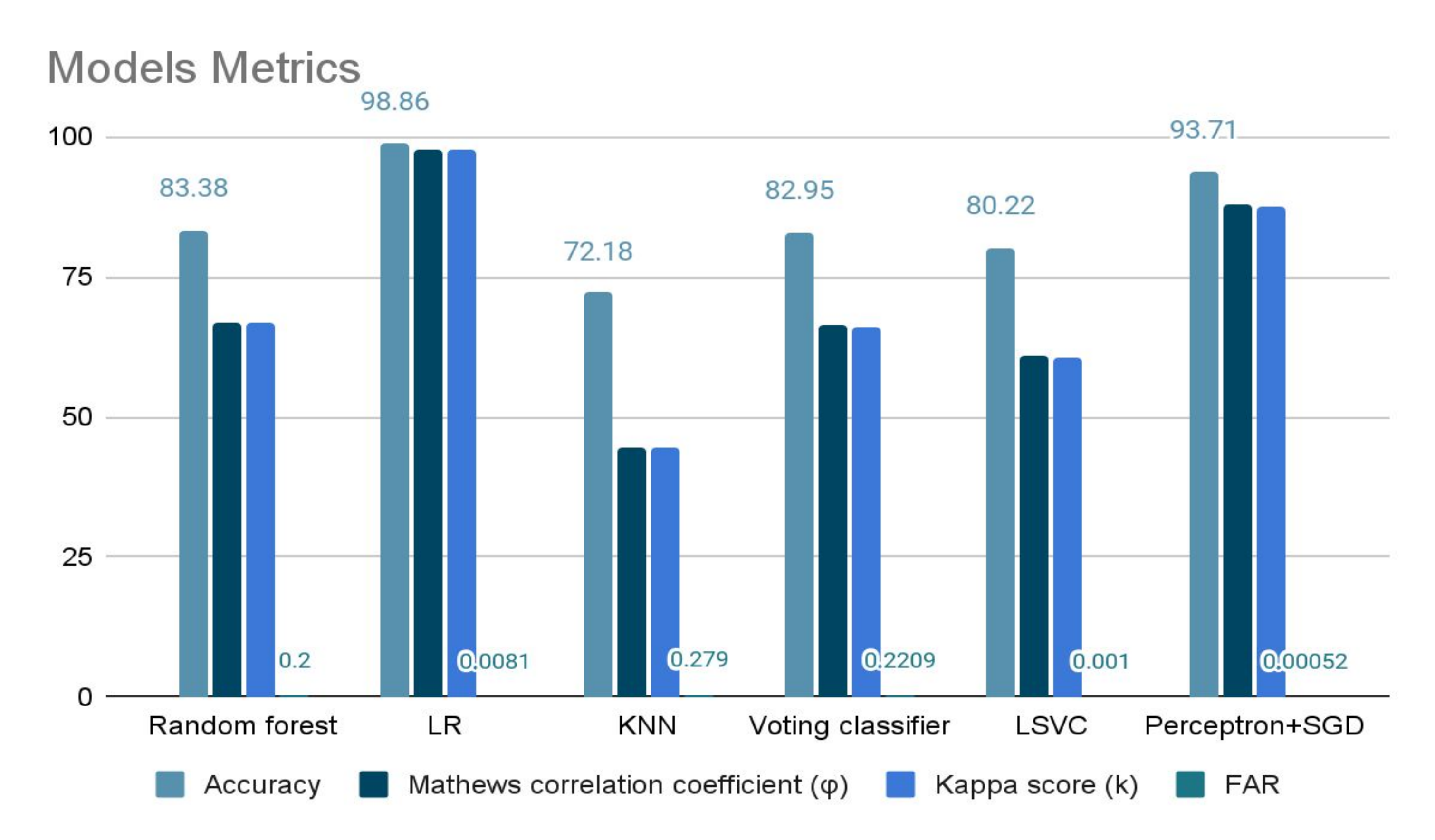
Logistic Regression Classifier

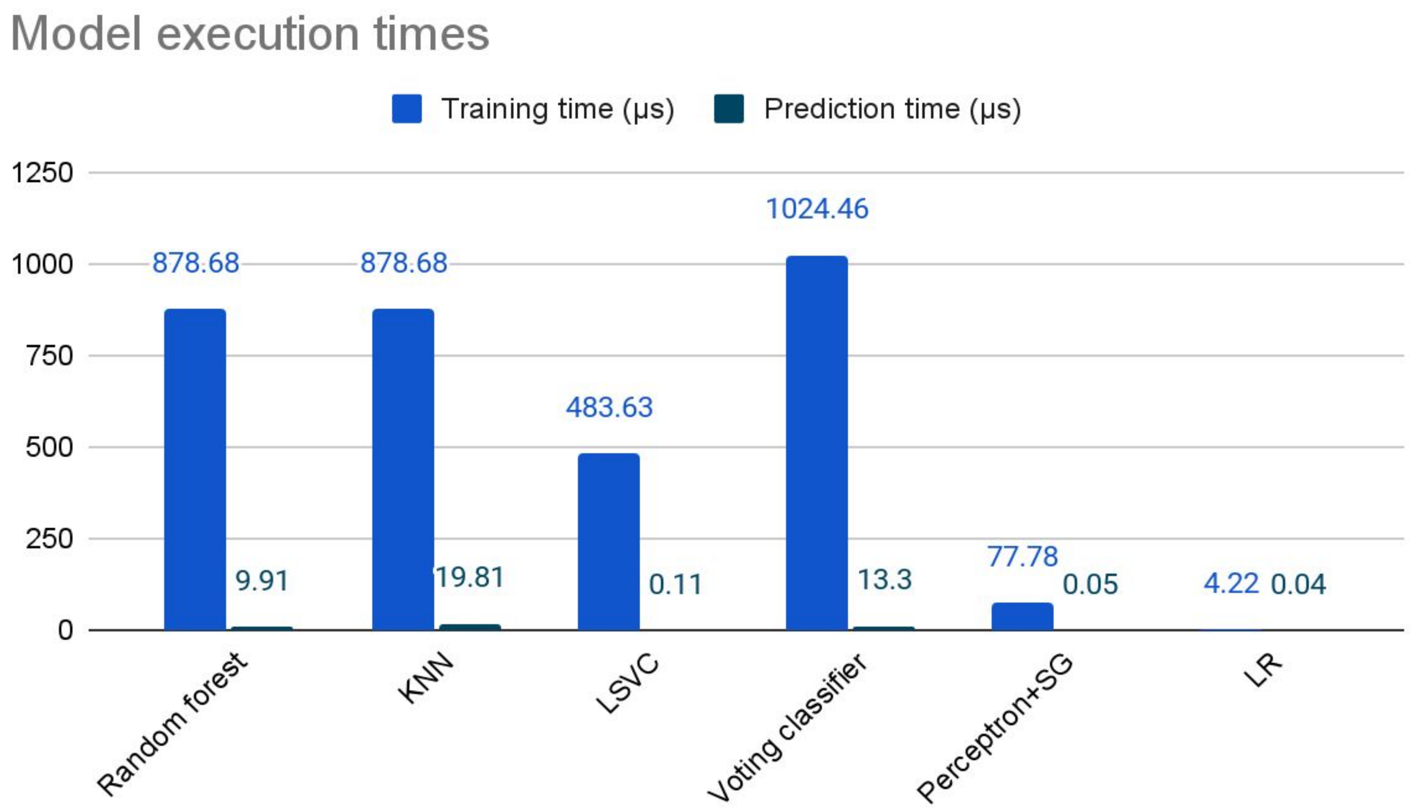
```
Logistic Regression Classifier....
       Unnamed: 0  dpkts   doctets     input    output   srcport   dstport  \
45174       50271    0.0 -0.026087  0.999587  1.000000  0.061590 -0.010735
38555       53294    0.0  0.394203  0.999587  1.000000 -0.692558  1.054607
10775       55551    0.0  0.240580  0.999587  1.000000 -0.692558  0.970759
16233        5553   -4.0 -0.826087 -0.000413  0.000000  0.340861 -0.011307
4152        36863    0.0  0.233333  0.999587  1.000000 -0.684876  0.756437
...           ...    ...       ...       ...       ...       ...       ...
20879        6896    3.0  3.984058  0.000000 -0.000413 -0.684876  0.699394
42012       42780    0.0  0.036232  0.999587  1.000000  0.504656 -0.010735
7487         8957    9.0  1.218841 -0.000413  0.000000  0.360015  0.000000
42582       19075   -2.0  0.147826  0.000000 -0.000413 -0.684876  0.989858
34861        8494   -4.0 -0.757971  0.000000 -0.000413 -0.693130  0.965508
```

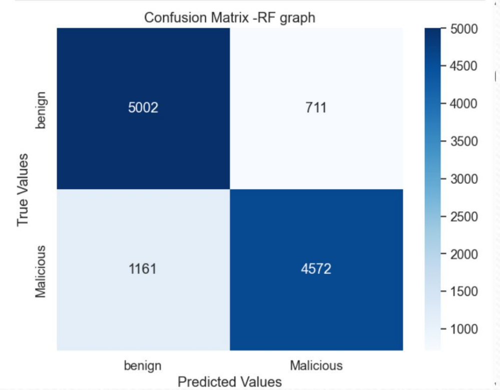
Logistic Regression Classifier

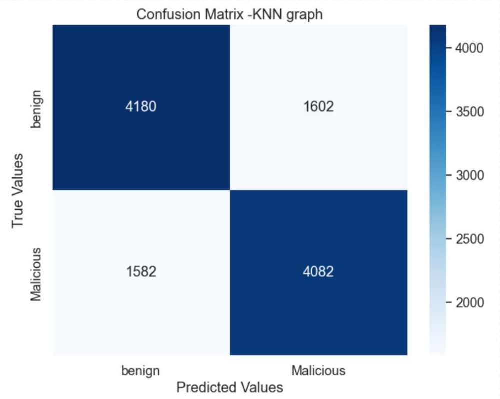
```
       prot  tos  tcp_flags  Label
45174   0.0  0.0        0.0      1
38555   0.0  0.0        0.0      1
10775   0.0  0.0        0.0      1
16233  11.0  0.0       -9.0      0
4152    0.0  0.0        0.0      1
...     ...  ...        ...    ...
20879   0.0  0.0        0.0      0
42012   0.0  0.0        0.0      1
7487    0.0  0.0        0.0      0
42582   0.0  0.0       -1.0      0
34861  11.0  0.0       -9.0      0

[57229 rows x 11 columns]
```

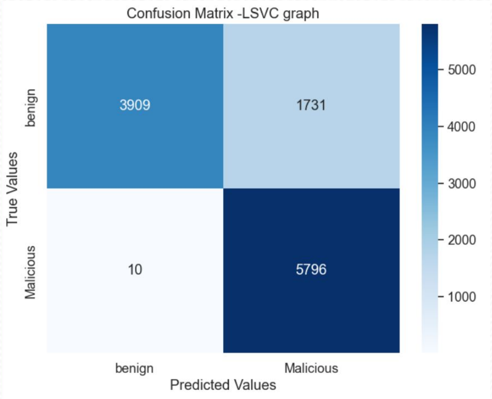
```
              precision    recall  f1-score   support

           0   0.991638  0.985328  0.988473      5657
           1   0.985751  0.991881  0.988807      5789

    accuracy                       0.988642     11446
   macro avg   0.988695  0.988605  0.988640     11446
weighted avg   0.988661  0.988642  0.988642     11446

[[5574   83]
 [  47 5742]]

Kappa Score
0.9772799710450105
Matthew_correlation
0.9772993146740303
Execution Time: 0.39945459365844727 Seconds
```

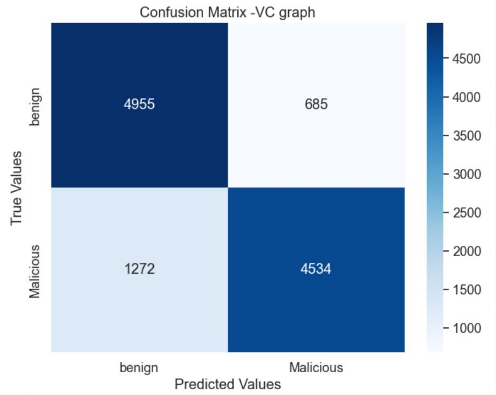

Models Metrics
100
75
50
25
0
83.38
98.86
72.18
82.95
80.22
93.71
0.2
0.0081
0.279
0.2209
0.001
0.00052
Random forest
LR
KNN
Voting classifier
LSVC
Perceptron+SGD
Accuracy
Mathews correlation coefficient (φ)
Kappa score (k)
FAR

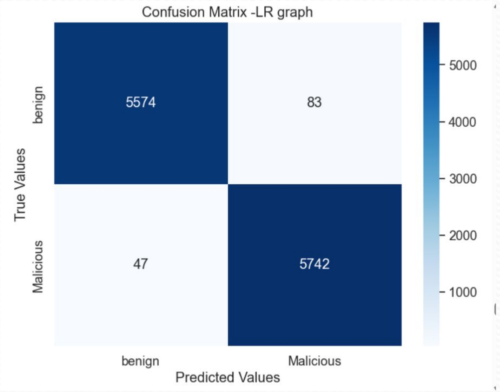

Model execution times
Training time (µs)
Prediction time (µs)
1250
1000
750
500
250
0
878.68
9.91
878.68
19.81
483.63
0.11
1024.46
13.3
77.78
0.05
4.22
0.04
Random forest
KNN
LSVC
Voting classifier
Perceptron+SG
LR

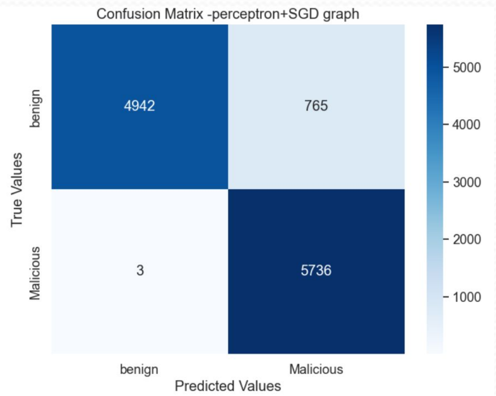

Confusion Matrix -RF graph
True Values
benign
Malicious
5002
711
1161
4572
benign
Malicious
Predicted Values
5000
4500
4000
3500
3000
2500
2000
1500
1000


Confusion Matrix -KNN graph
True Values
benign
Malicious
4180
1602
1582
4082
benign
Malicious
Predicted Values
4000
3500
3000
2500
2000


Confusion Matrix -LSVC graph
True Values
benign
Malicious
3909
1731
10
5796
benign
Malicious
Predicted Values
5000
4000
3000
2000
1000


Confusion Matrix -VC graph
True Values
benign
Malicious
4955
685
1272
4534
benign
Malicious
Predicted Values
4500
4000
3500
3000
2500
2000
1500
1000


Confusion Matrix -LR graph
True Values
benign
Malicious
5574
83
47
5742
benign
Malicious
Predicted Values
5000
4000
3000
2000
1000


Confusion Matrix -perceptron+SGD graph
True Values
benign
Malicious
4942
765
3
5736
benign
Malicious
Predicted Values
5000
4000
3000
2000
1000

Conclusion

- The objective of this paper is to detect SQL injection attacks (SQLIA) in network flow data by training and testing several machine learning models.
- The results show that the Logistic Regression and Perceptron with Stochastic Gradient Descent models achieve the best performance.
- The authors conclude that detecting SQLIA attacks in networks is possible using NetFlow as a flow-based protocol.

References

- Campazas-Vega, A., Crespo-Martínez, I. S., 2022. SQL Injection Attack Netflow (D1-D2). :Online; accessed July 26, 2022.
- Deriba, F. G., SALAU, A. O., Mohammed, S. H., Kassa, T. M., Demilie, W. B., 2022.Development of a compressive framework using machine learning approaches for SQL injection attacks. Przeglad Elektrotechniczny
- IPT-netflow, 2022. Ipt-netflow: netflow iptables module for linux kernel. https://github.com/aabc/ipt-netflow (accessed July 28, 2022).
- Junjin, M., 2009. An approach for SQL injection vulnerability detection. In: 2009 Sixth International Conference on Information Technology: New Generations,pp. 1411–1414. doi: 10.1109/ITNG.2009.34
- Kemp, C., Calvert, C., Khoshgoftaar, T., 2018. Utilizing netflow data to detect slow read attacks. In: 2018 IEEE Iternational Conference on Information Reuse and Integration (IRI). IEEE, pp. 108–116 .
- A. Campazas-Vega, I.S. Crespo-Martínez, Á.M. Guerrero-Higueras, C. Fernández-Llamas "Flow-data gathering using netflow sensors for fitting malicious-traffic detection models Sensors", 20 (24) (2020), p. 7294

Thank You