

Intrusion Detection

Erwin Erikson

Faculty of Information Technology
Institut Teknologi Batam
Batam, Indonesia
1822003@student.iteba.ac.id

Muhammad Al Imron

Faculty of Information Technology
Institut Teknologi Batam
Batam, Indonesia
1822007@student.iteba.ac.id

Farhan Ghulam Hadi Saputra

Faculty of Information Technology
Institut Teknologi Batam
Batam, Indonesia
1822014@student.iteba.ac.id

Abstract—Meningkatnya perkembangan internet tidak terlepas dari serangan seperti malware infection. Intrusion Detection System (IDS) adalah masalah nonlinier, rumit dan berhubungan dengan data lalu lintas jaringan. IDS mencoba untuk mengidentifikasi dan memberi tahu aktivitas pengguna sebagai anomali normal. Untuk mendeteksi berbagai serangan jaringan dapat dilatih dengan melakukan percobaan menggunakan dataset NSL-KDD [?] dengan beberapa algoritma yaitu Random Forest, K-Neighbors [?], SVM dan Ensemble Learning.

Keywords—intrusion detection, Random Forest, K-Neighbors, SVM, Ensemble Learning, NSL-KDD dataset.

I. PENDAHULUAN

Meningkatnya pengguna internet dari tahun ke tahun tidak terlepas dari serangan yang timbul dari teknologi jaringan seperti serangan *malware infection*. Oleh karena itu, diperlukan keamanan dalam sistem komputer untuk mencegah dari serangan.

IDS (*Intrusion Detection System*) merupakan sebuah aplikasi yang mampu mencatat kegiatan dalam suatu jaringan dan menganalisa paket-paket yang dikirim melalui lalu lintas jaringan secara *realtime*. Tujuan dari sistem ini yaitu mengawasi jika terjadi penetrasi ke dalam sistem, mengawasi *traffic* yang terjadi pada jaringan, mendeteksi *anomaly* terjadinya penyimpangan dari sistem yang normal atau tingkah laku user.

Dalam melakukan deteksi serangan, dapat digunakan beberapa algoritma yaitu Random Forest, K-Neighbors, SVM dan Ensemble Learning. Dan tujuan dari penelitian ini adalah untuk membandingkan performa dari masing-masing algoritma.

II. PENJELASAN TEORI

A. Random Forest

Hutan acak (*Random Forest*) adalah kumpulan pohon keputusan yang digunakan untuk meningkatkan akurasi, biasanya dilatih dengan metode "*bagging*". Ide umum dari metode *bagging* adalah bahwa kombinasi model pembelajaran meningkatkan hasil secara keseluruhan.

Keuntungan *Random Forest* adalah sebagai berikut [?]:

- 1) Hutan yang dihasilkan dapat disimpan untuk referensi di masa mendatang.
- 2) Hutan acak mengatasi masalah penyesuaian.
- 3) Dalam akurasi RF dan kepentingan variabel secara otomatis dihasilkan.

B. K-Nearest Neighbors

K-nearest neighbors (knn) adalah algoritma yang berfungsi untuk melakukan klasifikasi suatu data berdasarkan data pembelajaran (*train data sets*), yang diambil dari k tetangga terdekatnya (*nearest neighbors*), dengan k merupakan banyaknya tetangga terdekat. Beberapa formula yang digunakan adalah:

• Euclidean Distance

Untuk mendefinisikan jarak antara dua titik yaitu titik pada data training (x) dan titik pada data testing (y), maka digunakan rumus *Euclidean* [?], yaitu:

$$d(x, y) = \sqrt{\sum_{i=1}^n (xi - yi)^2} \quad (1)$$

dimana:

d = jarak antara 2 titik

x = data uji

y = data latih

i = merepresentasikan nilai atribut

n = merupakan dimensi atribut.

• City Block Distance

City Block Distance umumnya dihitung antara 2 koordinat objek yang berpasangan. Ini adalah penjumlahan dari perbedaan absolut antara 2 koordinat. *City Block Distance* 2-titik a dan b dengan dimensi k dihitung secara matematis menggunakan rumus berikut ini:

$$d_{ij} = \sum_{i=1}^k |a_i - b_i| \quad (2)$$

• Manhattan Distance

Manhattan Distance merupakan salah satu pengukuran yang paling banyak digunakan meliputi penggantian perbedaan kuadrat dengan menjumlahkan perbedaan *absolute* dari variabel-variabel. Fungsi ini hanya akan menjumlahkan selisih nilai x dan y dari dua buah titik.

• Minkowski Distance

Minkowski Distance adalah metrik dalam ruang vektor bernorma yang dapat dianggap sebagai generalisasi dari kedua jarak *Euclidean* dan jarak *Manhattan*. Jarak

Minkowski antara dua variabel X dan Y didefinisikan sebagai:

$$d = \left(\sum_{i=1}^n |X_i - Y_i|^p \right)^{1/p} \quad (3)$$

Kasus di mana $p = 1$ setara dengan jarak *Manhattan* dan kasus di mana $p = 2$ setara dengan jarak *Euclidean*.

C. SVM

Teori SVM berasal dari statistik dan prinsip dasar SVM adalah menemukan *hyperplane* linier yang optimal dalam ruang fitur yang secara maksimal memisahkan dua kelas target [?].

Dalam kaitannya dengan fungsi kernel, fungsi diskriminan mengambil bentuk berikut:

$$f(x) = \sum_i^n \alpha_i k(x, x_i) + b \quad (4)$$

Dalam pekerjaan ini, kernel *Gaussian* telah digunakan untuk membangun pengklasifikasi SVM.

Gaussian kernel:

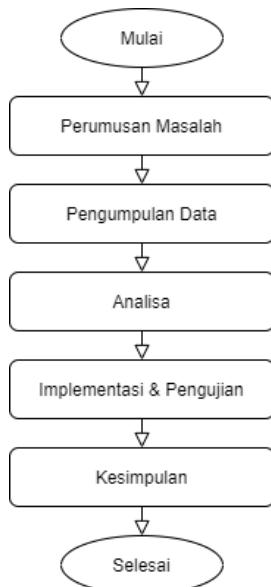
$$K(x_i, x_j) = \exp \left(-\frac{\|x_i - x_j\|^2}{2\sigma^2} \right) \quad (5)$$

dimana σ adalah lebar fungsi.

Fungsi kernel dan parameternya harus dipilih untuk membangun pengklasifikasi SVM. Melalih SVM menemukan *hyperplane* margin besar, yaitu menetapkan parameter α .

III. METODOLOGI

Metodologi adalah tahapan yang akan dilakukan dalam melakukan penelitian agar dapat memenuhi tujuan sesuai dengan yang diharapkan. Tahapan penelitian yang akan dilakukan dapat dilihat pada "Gambar. 1".



Gambar 1: Tahapan Penelitian

Tahapan pada penelitian ini dapat dijelaskan sebagai berikut:

Perumusan Masalah

Merupakan tahap awal dari metodologi penelitian. Rumusan masalah di dalam penelitian yakni bagaimana mengklasifikasi data serangan Intrusion Detection System (IDS)

Pengumpulan Data

Pengumpulan data yang dilakukan dengan membaca dan mempelajari penelitian sebelumnya yang berhubungan dengan IDS.

Analisa

Pada tahap ini adalah menganalisa data yaitu data latih yang digunakan untuk standarisasi melakukan pengujian, dan data uji yang digunakan untuk mengetahui penilaian yang dihasilkan dari data latih.

Tahap ini juga menganalisa metode yang digunakan dalam penelitian yang berkaitan dengan sistem yang digunakan.

Implementasi dan Pengujian

Proses implementasi adalah merealisasikan aplikasi IDS sesuai dengan dengan bahasa pemrograman yang digunakan yaitu Phyton menggunakan JupyterLab.

Tahap pengujian adalah tahap yang dilakukan untuk mengevaluasi masing-masing metode yang digunakan dalam penelitian dengan tujuan untuk mengetahui perbandingan performa dari setiap algoritma.

Kesimpulan

Merupakan tahap penentuan kesimpulan terhadap hasil pengujian yang telah dilakukan.

IV. HASIL DAN PEMBAHASAN

Untuk melakukan pelatihan/pengujian digunakan dataset NSL-KDD dimana NSL_KDD_Train sebagai data latih dan NSL_KDD_Test sebagai data uji seperti pada "Gambar. 2".

```

File Edit View Run Kernel Tabs Settings Help
NID.ipynb
[1]: import pandas as pd
import numpy as np
import sys
import sklearn
import io
import random

[2]: train_url = 'https://raw.githubusercontent.com/mertteroglu/NSL-KDD-Network-Intrusion-Detection/master/NSL_KDD_Train.csv'
test_url = 'https://raw.githubusercontent.com/mertteroglu/NSL-KDD-Network-Intrusion-Detection/master/NSL_KDD_Test.csv'
  
```

Gambar 2: Data Latih dan Data Uji

Pada tahap data preprocessing menggunakan One-Hot-Encoding untuk mengonversi semua properti kategorikal menjadi properti biner. Untuk mengonversi setiap kategori menjadi angka, properti harus dikonversi terlebih dahulu dengan LabelEncoder seperti pada "Gambar. 3", "Gambar. 4", dan "Gambar. 5".

```

[8]: from sklearn.preprocessing import LabelEncoder
categorical_columns=['protocol_type', 'service', 'flag']

df_categorical_values = df[categorical_columns]
testdf_categorical_values = df_test[categorical_columns]

df_categorical_values.head()

[8]:
  protocol_type service flag
  0      tcp  ftp_data SF
  1      udp     other SF
  2      tcp  private SO
  3      tcp     http SF
  4      tcp     http SF

```

Gambar 3: Menyisipkan fitur kategoris

```

[10]: df_categorical_values_enc=df_categorical_values.apply(LabelEncoder().fit_transform)
print(df_categorical_values.head())
print("-----")
print(df_categorical_values_enc.head())

# test set
testdf_categorical_values_enc=testdf_categorical_values.apply(LabelEncoder().fit_transform)

  protocol_type service flag
  0      1  ftp_data SF
  1      2     other SF
  2      3  private SO
  3      4     http SF
  4      5     http SF
  -----
  protocol_type service flag
  0      1      20   9
  1      2      44   9
  2      1      49   5
  3      1      24   9
  4      1      24   9

```

Gambar 4: Mengubah fitur kategoris menjadi angka

```

[11]: enc = OneHotEncoder(categories='auto')
df_categorical_values_enc = enc.fit_transform(df_categorical_values[['service','proto']])
df_cat_data = pd.DataFrame(df_categorical_values_enc.toarray(),columns=enc.columns)

# test set
testdf_cat_data = pd.get_dummies(testdf_categorical_values[['service','proto']],columns=testdf.columns)

df_cat_data.head()

[11]:
Protocol_type_1 Protocol_type_2 Protocol_type_3 service_ICMP service_XMAS service_ZRQ service_RST service_SYN service_RST_SYN service_SYN_RST ... Reg.REJ Reg.RS
  0      0.0      1.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
  1      0.0      0.0      1.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
  2      0.0      1.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
  3      0.0      1.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
  4      0.0      1.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
  5 rows x 84 columns

```

Gambar 5: One-Hot-Encoding

Selanjutnya menambahkan kolom yang hilang dalam set pengujian seperti pada “Gambar 6”.

```

[12]: trainservice=df['service'].tolist()
testservice= df_test['service'].tolist()
difference=list(set(trainservice) - set(testservice))
string = 'service_'
difference=[string + x for x in difference]

[12]: ['service_urh_i',
       'service_http_8001',
       'service_red_i',
       'service_harvest',
       'service_aol',
       'service_http_2784']

[13]: for col in difference:
    testdf_cat_data[col] = 0

print(df_cat_data.shape)
print(testdf_cat_data.shape)

(125973, 84)
(22544, 84)

```

Gambar 6: Menambahkan kolom yang hilang

Lalu menambahkan kolom numerik baru ke dataframe

utama seperti pada “Gambar. 7”.

```

[14]: newdf=df.join(df_cat_data)
newdf.drop('flag', axis=1, inplace=True)
newdf.drop('protocol_type', axis=1, inplace=True)
newdf.drop('service', axis=1, inplace=True)

# test data
newdf_test=df_test.join(testdf_cat_data)
newdf_test.drop('flag', axis=1, inplace=True)
newdf_test.drop('protocol_type', axis=1, inplace=True)
newdf_test.drop('service', axis=1, inplace=True)

print(newdf.shape)
print(newdf_test.shape)

(125973, 123)
(22544, 123)

```

Gambar 7: Menambahkan kolom numerik baru

Kemudian membagi dataset untuk setiap kategori serangan yaitu 0 = Normal, 1 = DoS, 2 = Probe, 3 = R2L, 4 = U2R seperti pada “Gambar. 8”.

```

[15]: labeldf=df[df['label']==1]
labeldf_test=newdf_test['label']

# change the label column
labeldf.replace({'normal': 0, 'neptune': 1, 'back': 1, 'land': 1, 'pod': 1, 'nefertiti': 1, 'smurf': 1, 'teardrop': 1, 'malignbyte': 1, 'apache2': 1, 'pervade': 1, 'buffer_overflow': 2, 'ftp_write': 2, 'imap': 2, 'guess_passwd': 2, 'http://': 2, 'multihop': 2, 'nmap': 2, 'perl': 2, 'rootkit': 2, 'xterm': 2}, inplace=True)
labeldf.replace({'normal': 0, 'neptune': 1, 'back': 1, 'land': 1, 'pod': 1, 'nefertiti': 1, 'smurf': 1, 'teardrop': 1, 'malignbyte': 1, 'apache2': 1, 'pervade': 1, 'buffer_overflow': 2, 'ftp_write': 2, 'imap': 2, 'guess_passwd': 2, 'http://': 2, 'multihop': 2, 'nmap': 2, 'perl': 2, 'rootkit': 2, 'xterm': 2}, inplace=True)
labeldf.replace({'normal': 0, 'neptune': 1, 'back': 1, 'land': 1, 'pod': 1, 'nefertiti': 1, 'smurf': 1, 'teardrop': 1, 'malignbyte': 1, 'apache2': 1, 'pervade': 1, 'buffer_overflow': 2, 'ftp_write': 2, 'imap': 2, 'guess_passwd': 2, 'http://': 2, 'multihop': 2, 'nmap': 2, 'perl': 2, 'rootkit': 2, 'xterm': 2}, inplace=True)

# put the non-label column back
labeldf['label']=labeldf['label'].values
labeldf_test['label']=labeldf_test['label'].values

```

Gambar 8: Membagi dataset untuk kategori serangan

Tahap selanjutnya melakukan penskalaan fitur dengan memisahkan kerangka data menjadi X dan Y dimana X Properties, variabel hasil Y seperti pada “Gambar. 9”.

```

File Edit View Run Kernel Tabs Settings Help
NID.ipynb NID.ipynb
+ X □ ▶ ▷ C ▶ Code
X_DoS = DoS_df.drop('label',1)
Y_DoS = DoS_df.label

X_Probe = Probe_df.drop('label',1)
Y_Probe = Probe_df.label

X_R2L = R2L_df.drop('label',1)
Y_R2L = R2L_df.label

# test set
X_DoS_test = DoS_df_test.drop('label',1)
Y_DoS_test = DoS_df_test.label

X_Probe_test = Probe_df_test.drop('label',1)
Y_Probe_test = Probe_df_test.label

X_R2L_test = R2L_df_test.drop('label',1)
Y_R2L_test = R2L_df_test.label

X_U2R_test = U2R_df_test.drop('label',1)
Y_U2R_test = U2R_df_test.label

```

Gambar 9: Penskalaan fitur

Berikutnya tahap pemilihan 13 fitur terbaik (sebagai grup) menggunakan *Recursive Feature Elimination* (RFE) seperti pada “Gambar. 10”.

```

[ ]: from sklearn.feature_selection import RFE
rfe = RFE(estimator=clf, n_features_to_select=13, step=1)

rfe.fit(X_DoS, Y_DoS.astype(int))
X_rfDoS=rfe.transform(X_DoS)
true=rfe.support_
rfecolindex_DoS=[i for i, x in enumerate(true) if x]
rfecolname_DoS=list(colNames[i] for i in rfecolindex_DoS)

[ ]: rfe.fit(X_Probe, Y_Probe.astype(int))
X_rfProbe=rfe.transform(X_Probe)
true=rfe.support_
rfecolindex_Probe=[i for i, x in enumerate(true) if x]
rfecolname_Probe=list(colNames[i] for i in rfecolindex_Probe)

[ ]: rfe.fit(X_R2L, Y_R2L.astype(int))
X_rfR2L=rfe.transform(X_R2L)
true=rfe.support_
rfecolindex_R2L=[i for i, x in enumerate(true) if x]
rfecolname_R2L=list(colNames[i] for i in rfecolindex_R2L)

[ ]: rfe.fit(X_U2R, Y_U2R.astype(int))
X_rfU2R=rfe.transform(X_U2R)
true=rfe.support_
rfecolindex_U2R=[i for i, x in enumerate(true) if x]
rfecolname_U2R=list(colNames[i] for i in rfecolindex_U2R)

```

Gambar 10: Pemilihan fitur

Fitur yang dipilih untuk beberapa serangan oleh RFE dapat dilihat pada “Gambar. 11”.

```

[18]: print("Features selected for DoS:",rfecolname_DoS)
print("Features selected for Probe:",rfecolname_Probe)
print("Features selected for R2L:",rfecolname_R2L)
print("Features selected for U2R:",rfecolname_U2R)

Features selected for DoS: ['src_bytes', 'dst_bytes', 'urion_fragment', 'count', 'srv_count', 'srv_serror_rate', 'same_srv_rate', 'diff_srv_rate', 't_Host_dif_src_rate', 'dst_host_error_rate', 'service_ec2', 'flag_SF']

Features selected for Probe: ['src_bytes', 'dst_bytes', 'count', 'dst_host_count', 'dst_host_srv_count', 'dst_host_same_srv_rate', 'dst_host_diff_srv_rate', 'dst_host_same_src_port_rate', 'dst_host_srv_diff_rate', 'dst_host_ip_probes', 'service_ec2', 'service_dhl', 'proto']

Features selected for R2L: ['duration', 'src_bytes', 'dst_bytes', 'host', 'num_failed_logins', 'is_guest_login', 'dst_host_count', 'dst_host_src_count', 'dst_host_srv_count', 'dst_host_diff_src_rate', 'dst_host_same_src_port_rate', 'dst_host_srv_diff_rate', 'service_ftp_data']

Features selected for U2R: ['duration', 'src_bytes', 'dst_bytes', 'host', 'num_compressed', 'root_shell', 'num_file_creations', 'count', 'dst_host_count', 'dst_host_src_count', 'dst_host_srv_count', 'dst_host_diff_src_rate', 'dst_host_same_src_port_rate']

```

Gambar 11: Fitur Serangan oleh RFE

Selanjutnya dilakukan pengujian menggunakan algoritma *Random Forest*. Model pengklasifikasi disimpan dalam variabel *clf* seperti pada “Gambar. 12”.

```

[26]: from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=10,n_jobs=2)

# all features
clf_DoS=RandomForestClassifier(n_estimators=10,n_jobs=2)
clf_Probe=RandomForestClassifier(n_estimators=10,n_jobs=2)
clf_R2L=RandomForestClassifier(n_estimators=10,n_jobs=2)
clf_U2R=RandomForestClassifier(n_estimators=10,n_jobs=2)
clf_DoS.fit(X_DoS, Y_DoS.astype(int))
clf_Probe.fit(X_Probe, Y_Probe.astype(int))
clf_R2L.fit(X_R2L, Y_R2L.astype(int))
clf_U2R.fit(X_U2R, Y_U2R.astype(int))

[26]: RandomForestClassifier(n_estimators=10, n_jobs=2)

```

Gambar 12: Pengujian Algoritma Random Forest

Hasil evaluasi kinerja model atau algoritma *Random Forest* untuk semua fitur dengan Cross Validation.

Untuk DoS dapat dilihat pada “Gambar. 13”.

```

[34]: from sklearn.model_selection import cross_val_score
from sklearn import metrics
accuracy = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='precision')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='recall')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='f1')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

Accuracy: 0.99795 (+/- 0.00023)
Precision: 0.99696 (+/- 0.00210)
Recall: 0.99651 (+/- 0.00322)
F-measure: 0.99765 (+/- 0.00219)

```

Gambar 13: Cross Validation DoS All Fitur

Untuk *Probe* dapat dilihat pada “Gambar. 14”.

```

[35]: accuracy = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

Accuracy: 0.99847 (+/- 0.00213)
Precision: 0.99875 (+/- 0.00511)
Recall: 0.99262 (+/- 0.00549)
F-measure: 0.99469 (+/- 0.00512)

```

Gambar 14: Cross Validation Probe All Fitur

Untuk *U2R* dapat dilihat pada “Gambar. 15”.

```

[36]: accuracy = cross_val_score(clf_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

Accuracy: 0.99775 (+/- 0.00255)
Precision: 0.99864 (+/- 0.12803)
Recall: 0.82014 (+/- 0.12368)
F-measure: 0.87775 (+/- 0.12470)

```

Gambar 15: Cross Validation U2R All Fitur

Untuk *R2L* dapat dilihat pada “Gambar. 16”.

```

[37]: accuracy = cross_val_score(clf_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

Accuracy: 0.99547 (+/- 0.00513)
Precision: 0.97295 (+/- 0.01596)
Recall: 0.98844 (+/- 0.01355)
F-measure: 0.97295 (+/- 0.00642)

```

Gambar 16: Cross Validation R2L All Fitur

Hasil evaluasi kinerja model atau algoritma *Random Forest* untuk 13 fitur dengan Cross Validation.

Untuk DoS dapat dilihat pada “Gambar. 17”.

```

[43]: accuracy = cross_val_score(clf_rfDoS, X_DoS_test2, Y_DoS_test2, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_rfDoS, X_DoS_test2, Y_DoS_test2, cv=10, scoring='precision')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_rfDoS, X_DoS_test2, Y_DoS_test2, cv=10, scoring='recall')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_rfDoS, X_DoS_test2, Y_DoS_test2, cv=10, scoring='f1')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

Accuracy: 0.99796 (+/- 0.00216)
Precision: 0.99839 (+/- 0.00201)
Recall: 0.99651 (+/- 0.00552)
F-measure: 0.99718 (+/- 0.00305)

```

Gambar 17: Cross Validation DoS 13 Fitur

Untuk *Probe* dapat dilihat pada “Gambar. 18”.

```
[44]: accuracy = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

Accuracy: 0.99382 (+/- 0.000385)
Precision: 0.98973 (+/- 0.00764)
Recall: 0.98668 (+/- 0.01083)
F-measure: 0.98758 (+/- 0.00914)
```

Gambar 18: Cross Validation Probe 13 Fitur

Untuk R2L dapat dilihat pada “Gambar. 19”.

```
[45]: accuracy = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

Accuracy: 0.98756 (+/- 0.000710)
Precision: 0.98720 (+/- 0.00923)
Recall: 0.98525 (+/- 0.01522)
F-measure: 0.98688 (+/- 0.012125)
```

Gambar 19: Cross Validation R2L 13 Fitur

Untuk U2R dapat dilihat pada “Gambar. 20”.

```
[46]: accuracy = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

Accuracy: 0.99093 (+/- 0.000259)
Precision: 0.98526 (+/- 0.00233)
Recall: 0.88183 (+/- 0.17841)
F-measure: 0.98644 (+/- 0.09956)
```

Gambar 20: Cross Validation U2R 13 Fitur

Selanjutnya dilakukan pengujian menggunakan algoritma *K-Neighbors*. Model pengklasifikasi disimpan dalam variabel *clf* seperti pada “Gambar. 21”.

```
[47]: from sklearn.neighbors import KNeighborsClassifier
clf_KNN_Pos=KNeighborsClassifier()
clf_KNN_Probe=KNeighborsClassifier()
clf_KNN_R2L=KNeighborsClassifier()
clf_KNN_U2R=KNeighborsClassifier()

clf_KNN_Pos.fit(X_DoS, Y_DoS.astype(int))
clf_KNN_Probe.fit(X_Probe, Y_Probe.astype(int))
clf_KNN_R2L.fit(X_R2L, Y_R2L.astype(int))
clf_KNN_U2R.fit(X_U2R, Y_U2R.astype(int))

KNeighborsClassifier()
```

Gambar 21: Pengujian Algoritma K-Neighbors

Hasil evaluasi kinerja model atau algoritma *K-Neighbors* dengan Cross Validation.

Untuk DoS dapat dilihat pada “Gambar. 22”.

```
[52]: from sklearn.model_selection import cross_val_score
from sklearn import metrics
accuracy = cross_val_score(clf_KNN_Pos, X_DoS_test, Y_DoS_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_KNN_Pos, X_DoS_test, Y_DoS_test, cv=10, scoring='precision')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_KNN_Pos, X_DoS_test, Y_DoS_test, cv=10, scoring='recall')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_KNN_Pos, X_DoS_test, Y_DoS_test, cv=10, scoring='f1')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

Accuracy: 0.99915 (+/- 0.000278)
Precision: 0.99868 (+/- 0.000383)
Recall: 0.99868 (+/- 0.000344)
F-measure: 0.99872 (+/- 0.000320)
```

Gambar 22: Cross Validation DoS K-Neighbors

Untuk *Probe* dapat dilihat pada “Gambar. 23”.

```
[53]: accuracy = cross_val_score(clf_KNN_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_KNN_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_KNN_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_KNN_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

Accuracy: 0.99978 (+/- 0.000045)
Precision: 0.99965 (+/- 0.000044)
Recall: 0.99965 (+/- 0.000045)
F-measure: 0.99972 (+/- 0.000045)
```

Gambar 23: Cross Validation Probe K-Neighbors

Untuk R2L dapat dilihat pada “Gambar. 24”.

```
[54]: accuracy = cross_val_score(clf_KNN_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_KNN_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_KNN_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_KNN_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

Accuracy: 0.98709 (+/- 0.000395)
Precision: 0.98500 (+/- 0.000375)
Recall: 0.98548 (+/- 0.01401)
F-measure: 0.98534 (+/- 0.01070)
```

Gambar 24: Cross Validation R2L K-Neighbors

Untuk U2R dapat dilihat pada “Gambar. 25”.

```
[55]: accuracy = cross_val_score(clf_KNN_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_KNN_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_KNN_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_KNN_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

Accuracy: 0.99703 (+/- 0.000262)
Precision: 0.99513 (+/- 0.00479)
Recall: 0.85073 (+/- 0.17639)
F-measure: 0.87831 (+/- 0.11390)
```

Gambar 25: Cross Validation U2R K-Neighbors

Selanjutnya dilakukan pengujian menggunakan algoritma SVM. Model pengklasifikasi disimpan dalam variabel *clf* seperti pada “Gambar. 26”.

```
[56]: from sklearn.svm import SVC

clf_SVM_DoS=SVC(kernel='linear', C=1.0, random_state=0)
clf_SVM_Probe=SVC(kernel='linear', C=1.0, random_state=0)
clf_SVM_R2L=SVC(kernel='linear', C=1.0, random_state=0)
clf_SVM_U2R=SVC(kernel='linear', C=1.0, random_state=0)

clf_SVM_DoS.fit(X_DoS, Y_DoS.astype(int))
clf_SVM_Probe.fit(X_Probe, Y_Probe.astype(int))
clf_SVM_R2L.fit(X_R2L, Y_R2L.astype(int))
clf_SVM_U2R.fit(X_U2R, Y_U2R.astype(int))

SVC(kernel='linear', random_state=0)
```

Gambar 26: Pengujian Algoritma SVM

Hasil evaluasi kinerja model atau algoritma SVM dengan *Cross Validation*.

Untuk DoS dapat dilihat pada “Gambar. 27”.

```
[61]: from sklearn.model_selection import cross_val_score
from sklearn import metrics
accuracy = cross_val_score(clf_SVM_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_SVM_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='precision')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_SVM_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='recall')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_SVM_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='f1')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

Accuracy: 0.99371 (+/- 0.00375)
Precision: 0.99107 (+/- 0.00785)
Recall: 0.99450 (+/- 0.00388)
F-measure: 0.99278 (+/- 0.00428)
```

Gambar 27: Cross Validation DoS SVM

Untuk *Probe* dapat dilihat pada “Gambar. 28”.

```
[62]: accuracy = cross_val_score(clf_SVM_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_SVM_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_SVM_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_SVM_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

Accuracy: 0.98450 (+/- 0.00526)
Precision: 0.96907 (+/- 0.01031)
Recall: 0.98365 (+/- 0.00866)
F-measure: 0.97613 (+/- 0.00800)
```

Gambar 28: Cross Validation Probe SVM

Untuk R2L dapat dilihat pada “Gambar. 29”.

```
[63]: accuracy = cross_val_score(clf_SVM_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_SVM_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_SVM_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_SVM_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

Accuracy: 0.98793 (+/- 0.00738)
Precision: 0.94854 (+/- 0.00994)
Recall: 0.98264 (+/- 0.01388)
F-measure: 0.95520 (+/- 0.01048)
```

Gambar 29: Cross Validation R2L SVM

Untuk U2R dapat dilihat pada “Gambar. 30”.

```
[64]: accuracy = cross_val_score(clf_SVM_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_SVM_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_SVM_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_SVM_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

Accuracy: 0.99032 (+/- 0.00390)
Precision: 0.91056 (+/- 0.17934)
Recall: 0.82909 (+/- 0.21833)
F-measure: 0.84869 (+/- 0.16029)
```

Gambar 30: Cross Validation U2R SVM

Kemudian dilakukan pengujian dengan metode *Ensemble Learning* menggunakan algoritma *Random Forest*, *K-Neighbors*, dan SVM. Model pengklasifikasi disimpan dalam variabel clf seperti pada “Gambar. 31”.

```
[65]: from sklearn.ensemble import VotingClassifier
clf_voting_DoS = VotingClassifier(estimators=[('rf', clf_DoS), ('knn', clf_XNN_DoS), ('svm', clf_SVM_DoS)], voting='hard')
clf_voting_Probe = VotingClassifier(estimators=[('rf', clf_Probe), ('knn', clf_XNN_Probe), ('svm', clf_SVM_Probe)], voting='hard')
clf_voting_R2L = VotingClassifier(estimators=[('rf', clf_R2L), ('knn', clf_XNN_R2L), ('svm', clf_SVM_R2L)], voting='hard')
clf_voting_U2R = VotingClassifier(estimators=[('rf', clf_U2R), ('knn', clf_XNN_U2R), ('svm', clf_SVM_U2R)], voting='hard')

clf_voting_DoS.fit(X_DoS, Y_DoS.astype(int))
clf_voting_Probe.fit(X_Probe, Y_Probe.astype(int))
clf_voting_R2L.fit(X_R2L, Y_R2L.astype(int))
clf_voting_U2R.fit(X_U2R, Y_U2R.astype(int))

[66]: VotingClassifier(estimators=[('rf', RandomForestClassifier(n_estimators=10, n_jobs=-1)), ('knn', KNeighborsClassifier()), ('svm', SVC(kernel='linear', random_state=0))])
```

Gambar 31: Pengujian Algoritma Ensemble Learning

Hasil evaluasi kinerja model atau algoritma *Ensemble Learning* dengan *Cross Validation*.

Untuk DoS dapat dilihat pada “Gambar. 32”.

```
[70]: from sklearn.model_selection import cross_val_score
from sklearn import metrics
accuracy = cross_val_score(clf_voting_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_voting_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='precision')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_voting_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='recall')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_voting_DoS, X_DoS_test, Y_DoS_test, cv=10, scoring='f1')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

Accuracy: 0.99808 (+/- 0.00209)
Precision: 0.99852 (+/- 0.00253)
Recall: 0.99718 (+/- 0.00327)
F-measure: 0.99777 (+/- 0.00324)
```

Gambar 32: Cross Validation DoS Ensemble Learning

Untuk *Probe* dapat dilihat pada “Gambar. 33”.

```
[71]: accuracy = cross_val_score(clf_voting_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_voting_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_voting_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_voting_Probe, X_Probe_test, Y_Probe_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

Accuracy: 0.99275 (+/- 0.00382)
Precision: 0.99745 (+/- 0.00762)
Recall: 0.98952 (+/- 0.00708)
F-measure: 0.98841 (+/- 0.00542)
```

Gambar 33: Cross Validation Probe Ensemble Learning

Untuk R2L dapat dilihat pada “Gambar. 34”.

```
[72]: accuracy = cross_val_score(clf_voting_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_voting_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_voting_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_voting_R2L, X_R2L_test, Y_R2L_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

Accuracy: 0.97158 (+/- 0.00546)
Precision: 0.95838 (+/- 0.00901)
Recall: 0.96409 (+/- 0.01397)
F-measure: 0.96079 (+/- 0.00645)
```

Gambar 34: Cross Validation R2L Ensemble Learning

Untuk U2R dapat dilihat pada “Gambar. 35”.

```
[73]: accuracy = cross_val_score(clf_voting_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_voting_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_voting_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_voting_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))

Accuracy: 0.99744 (+/- 0.00278)
Precision: 0.94270 (+/- 0.12370)
Recall: 0.88758 (+/- 0.15174)
F-measure: 0.91119 (+/- 0.09351)
```

Gambar 35: Cross Validation U2R Ensemble Learning

Dari hasil evaluasi kinerja dari masing-masing model atau algoritma, dapat dilihat pada tabel-tabel berikut:

Pengujian menggunakan algoritma *Random Forest* untuk semua fitur dapat dilihat pada “Tabel. 1”.

Tabel I: Algoritma Random Forest untuk semua Fitur

	Accuracy	Precision	Recall	F-measure
DoS	0.99796	0.99906	0.99651	0.99765
Probe	0.99670	0.99675	0.99262	0.99469
R2L	0.99775	0.94964	0.82014	0.87775
U2R	0.98047	0.97293	0.96844	0.97290

Pengujian menggunakan algoritma *Random Forest* untuk 13 fitur dapat dilihat pada “Tabel. 2”.

Tabel II: Algoritma Random Forest untuk 13 Fitur

	Accuracy	Precision	Recall	F-measure
DoS	0.99796	0.99839	0.99651	0.99718
Probe	0.99382	0.98973	0.98668	0.98758
R2L	0.97856	0.97280	0.96525	0.96838
U2R	0.99693	0.96256	0.83183	0.90644

Pengujian menggunakan algoritma *K-Neighbors* dapat dilihat pada “Tabel. 3”.

Tabel III: Algoritma K-Neighbors

	Accuracy	Precision	Recall	F-measure
DoS	0.99715	0.99678	0.99665	0.99672
Probe	0.99077	0.98606	0.98508	0.98553
R2L	0.96705	0.95265	0.95439	0.95344
U2R	0.99703	0.93143	0.85073	0.87831

Pengujian menggunakan algoritma SVM dapat dilihat pada “Tabel. 4”.

Tabel IV: Algoritma SVM

	Accuracy	Precision	Recall	F-measure
DoS	0.99371	0.99107	0.99450	0.99278
Probe	0.98450	0.96907	0.98365	0.97613
R2L	0.96793	0.94854	0.96264	0.95529
U2R	0.99632	0.91056	0.82909	0.84869

Pengujian menggunakan algoritma *Ensemble Learning* dapat dilihat pada “Tabel. 5”.

Tabel V: Algoritma Ensemble Learning

	Accuracy	Precision	Recall	F-measure
DoS	0.99808	0.99852	0.99718	0.99772
Probe	0.99275	0.98765	0.98953	0.98841
R2L	0.97158	0.95838	0.96409	0.96079
U2R	0.99744	0.94270	0.88758	0.91119

V. KESIMPULAN

Dalam penelitian ini, kami membandingkan beberapa model untuk sistem deteksi trusi menggunakan *Random Forest*, *K-Neighbors*, *Support Vector Machine*, dan *Ensemble Learning* dengan ketiga model diatas. Performa keempat pendekatan ini telah diamati berdasarkan *accuracy*, *precision*, *recall*, dan *f-measure* (F_1 -score).

Dari hasil pengujian dari masing-masing algoritma yang ada pada tabel, menunjukkan kemampuan klasifikasi algoritma *Ensemble Learning* lebih tinggi tingkat akurasi dan ketepatan.