Imports & Configs

In [1]:
```
#uncomment the next line to download all the modules used in this project
#it only needs to run once per system used
#%pip install numpy pandas seaborn matplotlib optuna scikit-learn xgboost catboost lightgbm sklearn_relief
```

In [2]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from pandas.api.types import is_numeric_dtype
import warnings
import optuna
import sklearn_relief as sr
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.tree  import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier, AdaBoostClassifier, VotingClassifier, GradientB
from sklearn.svm import SVC, LinearSVC
from sklearn.datasets import make_classification
from sklearn.naive_bayes import BernoulliNB
from lightgbm import LGBMClassifier
from sklearn.feature_selection import RFE
import itertools
from catboost import CatBoostClassifier
from xgboost import XGBClassifier
from tabulate import tabulate
import os
warnings.filterwarnings('ignore')
optuna.logging.set_verbosity(optuna.logging.WARNING)
```

Data Preprocessing & EDA

In [3]:
```
for dirname, _, filenames in os.walk('.\kaggle\input'):
    for filename in filenames:
```

```
        print(os.path.join(dirname, filename))
        if 'Train' in filename:
            train=pd.read_csv(os.path.join(dirname, filename))
        if 'Test' in filename:
            test=pd.read_csv(os.path.join(dirname, filename))
train
```

```
.\kaggle\input\Test_data.csv
.\kaggle\input\Train_data.csv
```

Out[3]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_sar
0	0	tcp	ftp_data	SF	491	0	0	0	0	0	...	25	
1	0	udp	other	SF	146	0	0	0	0	0	...	1	
2	0	tcp	private	S0	0	0	0	0	0	0	...	26	
3	0	tcp	http	SF	232	8153	0	0	0	0	...	255	
4	0	tcp	http	SF	199	420	0	0	0	0	...	255	
...	...	...	...	...	...	...	...	...	...	...	...	...	
25187	0	tcp	exec	RSTO	0	0	0	0	0	0	...	7	
25188	0	tcp	ftp_data	SF	334	0	0	0	0	0	...	39	
25189	0	tcp	private	REJ	0	0	0	0	0	0	...	13	
25190	0	tcp	nnsp	S0	0	0	0	0	0	0	...	20	
25191	0	tcp	finger	S0	0	0	0	0	0	0	...	49	

25192 rows × 42 columns

In [4]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25192 entries, 0 to 25191
Data columns (total 42 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   duration                     25192 non-null  int64  
 1   protocol_type                25192 non-null  object 
 2   service                      25192 non-null  object 
 3   flag                         25192 non-null  object 
 4   src_bytes                    25192 non-null  int64  
 5   dst_bytes                    25192 non-null  int64  
 6   land                         25192 non-null  int64  
 7   wrong_fragment               25192 non-null  int64  
 8   urgent                       25192 non-null  int64  
 9   hot                          25192 non-null  int64  
 10  num_failed_logins            25192 non-null  int64  
 11  logged_in                    25192 non-null  int64  
 12  num_compromised              25192 non-null  int64  
 13  root_shell                   25192 non-null  int64  
 14  su_attempted                 25192 non-null  int64  
 15  num_root                     25192 non-null  int64  
 16  num_file_creations           25192 non-null  int64  
 17  num_shells                   25192 non-null  int64  
 18  num_access_files             25192 non-null  int64  
 19  num_outbound_cmds            25192 non-null  int64  
 20  is_host_login                25192 non-null  int64  
 21  is_guest_login               25192 non-null  int64  
 22  count                        25192 non-null  int64  
 23  srv_count                    25192 non-null  int64  
 24  serror_rate                  25192 non-null  float64
 25  srv_serror_rate              25192 non-null  float64
 26  rerror_rate                  25192 non-null  float64
 27  srv_rerror_rate              25192 non-null  float64
 28  same_srv_rate                25192 non-null  float64
 29  diff_srv_rate                25192 non-null  float64
 30  srv_diff_host_rate           25192 non-null  float64
 31  dst_host_count               25192 non-null  int64  
 32  dst_host_srv_count           25192 non-null  int64  
 33  dst_host_same_srv_rate       25192 non-null  float64
 34  dst_host_diff_srv_rate       25192 non-null  float64
 35  dst_host_same_src_port_rate  25192 non-null  float64
 36  dst_host_srv_diff_host_rate  25192 non-null  float64
 37  dst_host_serror_rate         25192 non-null  float64
 38  dst_host_srv_serror_rate     25192 non-null  float64
 39  dst_host_rerror_rate         25192 non-null  float64
```

```
 40  dst_host_srv_rerror_rate     25192 non-null  float64
 41  class                        25192 non-null  object
dtypes: float64(15), int64(23), object(4)
memory usage: 8.1+ MB
```

In [5]:
```
train.head()
```

Out[5]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv
0	0	tcp	ftp_data	SF	491	0	0	0	0	0	...	25	
1	0	udp	other	SF	146	0	0	0	0	0	...	1	
2	0	tcp	private	S0	0	0	0	0	0	0	...	26	
3	0	tcp	http	SF	232	8153	0	0	0	0	...	255	
4	0	tcp	http	SF	199	420	0	0	0	0	...	255	

5 rows × 42 columns

In [6]:
```
train.describe()
```

Out[6]:

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in
count	25192.000000	2.519200e+04	2.519200e+04	25192.000000	25192.000000	25192.00000	25192.000000	25192.000000	25192.000000
mean	305.054104	2.433063e+04	3.491847e+03	0.000079	0.023738	0.00004	0.198039	0.001191	0.394768
std	2686.555640	2.410805e+06	8.883072e+04	0.008910	0.260221	0.00630	2.154202	0.045418	0.488811
min	0.000000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000
25%	0.000000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000
50%	0.000000	4.400000e+01	0.000000e+00	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000
75%	0.000000	2.790000e+02	5.302500e+02	0.000000	0.000000	0.00000	0.000000	0.000000	1.000000
max	42862.000000	3.817091e+08	5.151385e+06	1.000000	3.000000	1.00000	77.000000	4.000000	1.000000

8 rows × 38 columns

In [7]:
```
train.describe(include='object')
```

Out[7]:

	protocol_type	service	flag	class
count	25192	25192	25192	25192
unique	3	66	11	2
top	tcp	http	SF	normal
freq	20526	8003	14973	13449

Missing Data

In [8]:
```
total = train.shape[0]
missing_columns = [col for col in train.columns if train[col].isnull().sum() > 0]
for col in missing_columns:
    null_count = train[col].isnull().sum()
    per = (null_count/total) * 100
    print(f"{col}: {null_count} ({round(per, 3)}%)")
```

No missing values

Duplicates

In [9]:
```
if train.duplicated().sum():
    print(f"Number of duplicate rows: {train.duplicated().sum()}")
else:
    print(f"There are no duplicates within the dataset")
```

```
There are no duplicates within the dataset
```

Outliers

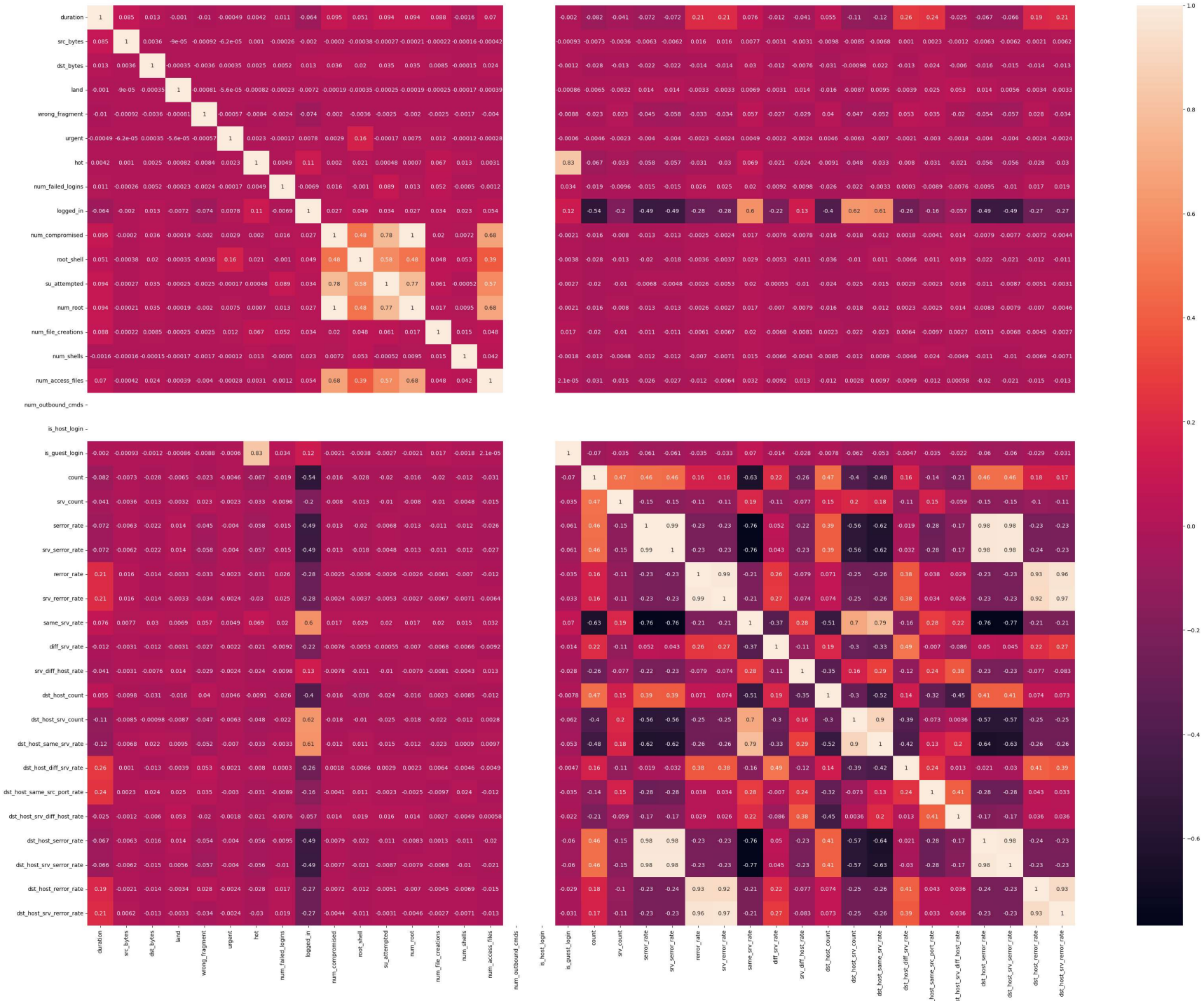
In [10]:
```
# for col in df:
#     if col != 'class' and is_numeric_dtype(df[col]):
#         fig, ax = plt.subplots(2, 1, figsize=(12, 8))
#         g1 = sns.boxplot(x = df[col], ax=ax[0])
```

```
#         g2 = sns.scatterplot(data=df, x=df[col],y=df['class'], ax=ax[1])
#         plt.show()
```

No outliers

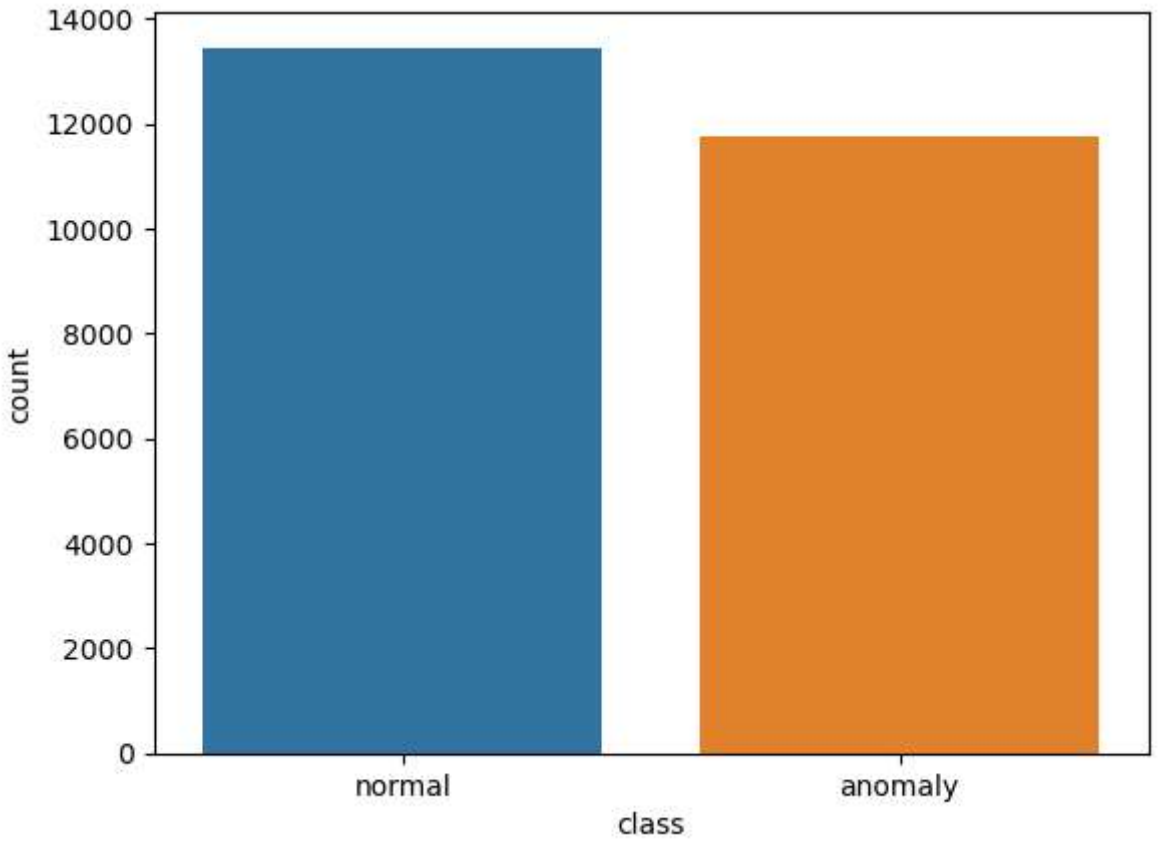
In [11]:
```
plt.figure(figsize=(40,30))
sns.heatmap(train.corr(numeric_only=True), annot=True)

# import plotly.express as px
# fig = px.imshow(df.corr(), text_auto=True, aspect="auto")
# fig.show()
```

Out[11]:
```
<Axes: >
```


duration
src_bytes
dst_bytes
land
wrong_fragment
urgent
hot
num_failed_logins
logged_in
num_compromised
root_shell
su_attempted
num_root
num_file_creations
num_shells
num_access_files
num_outbound_cmds
is_host_login
is_guest_login
count
srv_count
serror_rate
srv_serror_rate
rerror_rate
srv_rerror_rate
same_srv_rate
diff_srv_rate
srv_diff_host_rate
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_host_rate
dst_host_serror_rate
dst_host_srv_serror_rate
dst_host_rerror_rate
dst_host_srv_rerror_rate
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

In [12]:
```
sns.countplot(x=train['class'])
```

Out[12]:
```
<Axes: xlabel='class', ylabel='count'>
```



Label Encoding

In [13]:
```
def le(df):
    for col in df.columns:
        if df[col].dtype == 'object':
                label_encoder = LabelEncoder()
                df[col] = label_encoder.fit_transform(df[col])

le(train)
le(test)
```

In [14]:
```
train.drop(['num_outbound_cmds'], axis=1, inplace=True)
test.drop(['num_outbound_cmds'], axis=1, inplace=True)
train.head()
```

Out[14]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_
0	0	1	19	9	491	0	0	0	0	0	...	25	
1	0	2	41	9	146	0	0	0	0	0	...	1	
2	0	1	46	5	0	0	0	0	0	0	...	26	
3	0	1	22	9	232	8153	0	0	0	0	...	255	
4	0	1	22	9	199	420	0	0	0	0	...	255	

5 rows × 41 columns

Feature selection

In [15]:
```
X_train = train.drop(['class'], axis=1)
Y_train = train['class']
```

In [16]:
```
rfc = RandomForestClassifier()

rfe = RFE(rfc, n_features_to_select=10)
rfe = rfe.fit(X_train, Y_train)

feature_map = [(i, v) for i, v in itertools.zip_longest(rfe.get_support(), X_train.columns)]
selected_features = [v for i, v in feature_map if i==True]

selected_features
```

Out[16]:
```
['protocol_type',
 'flag',
 'src_bytes',
 'dst_bytes',
 'count',
 'same_srv_rate',
 'diff_srv_rate',
 'dst_host_srv_count',
 'dst_host_same_srv_rate',
 'dst_host_same_src_port_rate']
```

In [17]:
```
X_train = X_train[selected_features]
```

Split and scale data

In [18]:
```
scale = StandardScaler()
X_train = scale.fit_transform(X_train)
test = scale.fit_transform(test)
```

In [19]:
```
x_train, x_test, y_train, y_test = train_test_split(X_train, Y_train, train_size=0.70, random_state=2)
```

K Nearest Neighbors (KNN) classification model

In [20]:
```
def objective(trial):
    n_neighbors = trial.suggest_int('KNN_n_neighbors', 2, 16, log=False)
    classifier_obj = KNeighborsClassifier(n_neighbors=n_neighbors)
    classifier_obj.fit(x_train, y_train)
    accuracy = classifier_obj.score(x_test, y_test)
    return accuracy
```

In [21]:
```
study_KNN = optuna.create_study(direction='maximize')
study_KNN.optimize(objective, n_trials=1)
print(study_KNN.best_trial)
```

```
FrozenTrial(number=0, state=1, values=[0.9798888594866367], datetime_start=datetime.datetime(2023, 9, 7, 10, 32, 34, 505126), datetime_complete=datetime.datetime(2023, 9, 7, 10, 32, 36, 35104), params={'KNN_n_neighbors': 13}, user_attrs={}, system_attrs={}, intermediate_values={}, distributions={'KNN_n_neighbors': IntDistribution(high=16, log=False, low=2, step=1)}, trial_id=0, value=None)
```

In [22]:
```
KNN_model = KNeighborsClassifier(n_neighbors=study_KNN.best_trial.params['KNN_n_neighbors'])
KNN_model.fit(x_train, y_train)

KNN_train, KNN_test = KNN_model.score(x_train, y_train), KNN_model.score(x_test, y_test)

print(f"Train Score: {KNN_train}")
print(f"Test Score: {KNN_test}")
```

```
Train Score: 0.9799251446070092
Test Score: 0.9798888594866367
```

Logistic Regression Model

In [23]:
```
lg_model = LogisticRegression(random_state = 42)
lg_model.fit(x_train, y_train)
```

Out[23]:
```
LogisticRegression
LogisticRegression(random_state=42)
```

In [24]:
```
lg_train, lg_test = lg_model.score(x_train , y_train), lg_model.score(x_test , y_test)

print(f"Training Score: {lg_train}")
print(f"Test Score: {lg_test}")
```

```
Training Score: 0.9418169445389588
Test Score: 0.938872717650172
```

Decision Tree Classifier

In [25]:
```
def objective(trial):
    dt_max_depth = trial.suggest_int('dt_max_depth', 2, 32, log=False)
    dt_max_features = trial.suggest_int('dt_max_features', 2, 10, log=False)
    classifier_obj = DecisionTreeClassifier(max_features = dt_max_features, max_depth = dt_max_depth)
    classifier_obj.fit(x_train, y_train)
    accuracy = classifier_obj.score(x_test, y_test)
    return accuracy
```

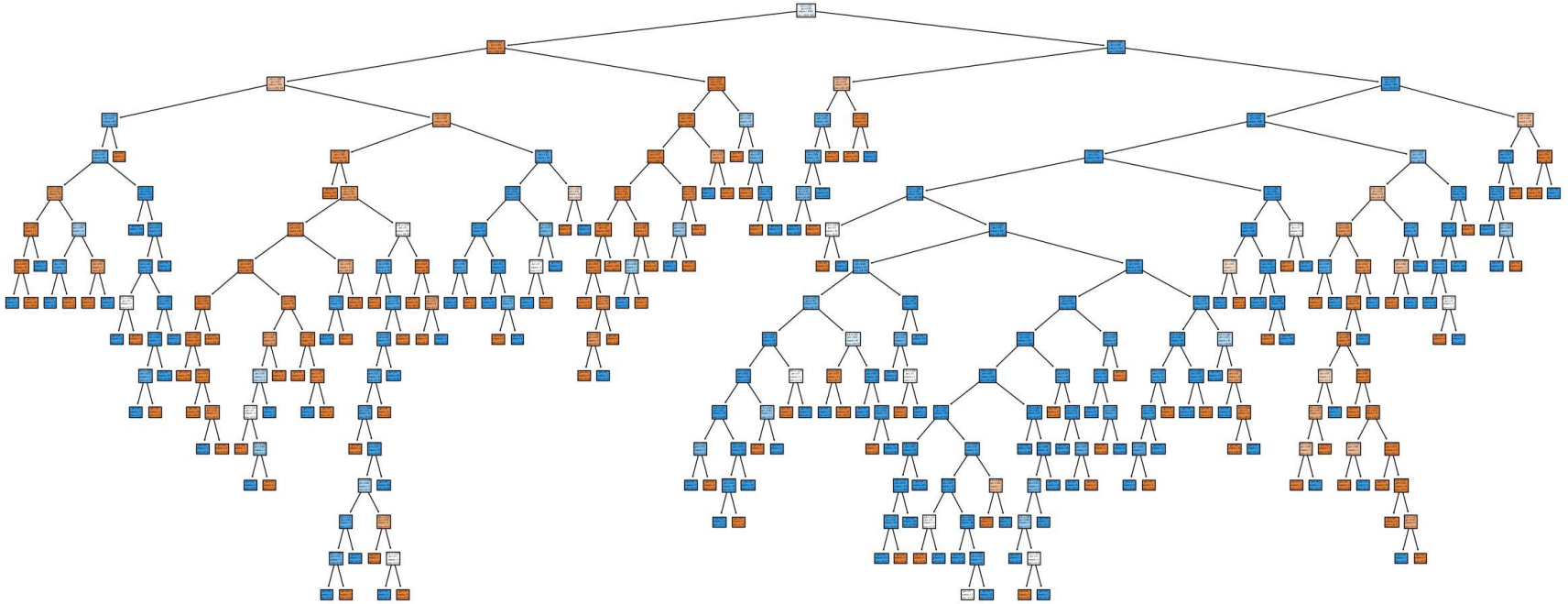
In [26]:
```
study_dt = optuna.create_study(direction='maximize')
study_dt.optimize(objective, n_trials=30)
print(study_dt.best_trial)
```

```
FrozenTrial(number=13, state=1, values=[0.9953691452765282], datetime_start=datetime.datetime(2023, 9, 7, 10, 32, 40, 4
28189), datetime_complete=datetime.datetime(2023, 9, 7, 10, 32, 40, 475082), params={'dt_max_depth': 16, 'dt_max_featur
es': 6}, user_attrs={}, system_attrs={}, intermediate_values={}, distributions={'dt_max_depth': IntDistribution(high=3
2, log=False, low=2, step=1), 'dt_max_features': IntDistribution(high=10, log=False, low=2, step=1)}, trial_id=13, valu
e=None)
```

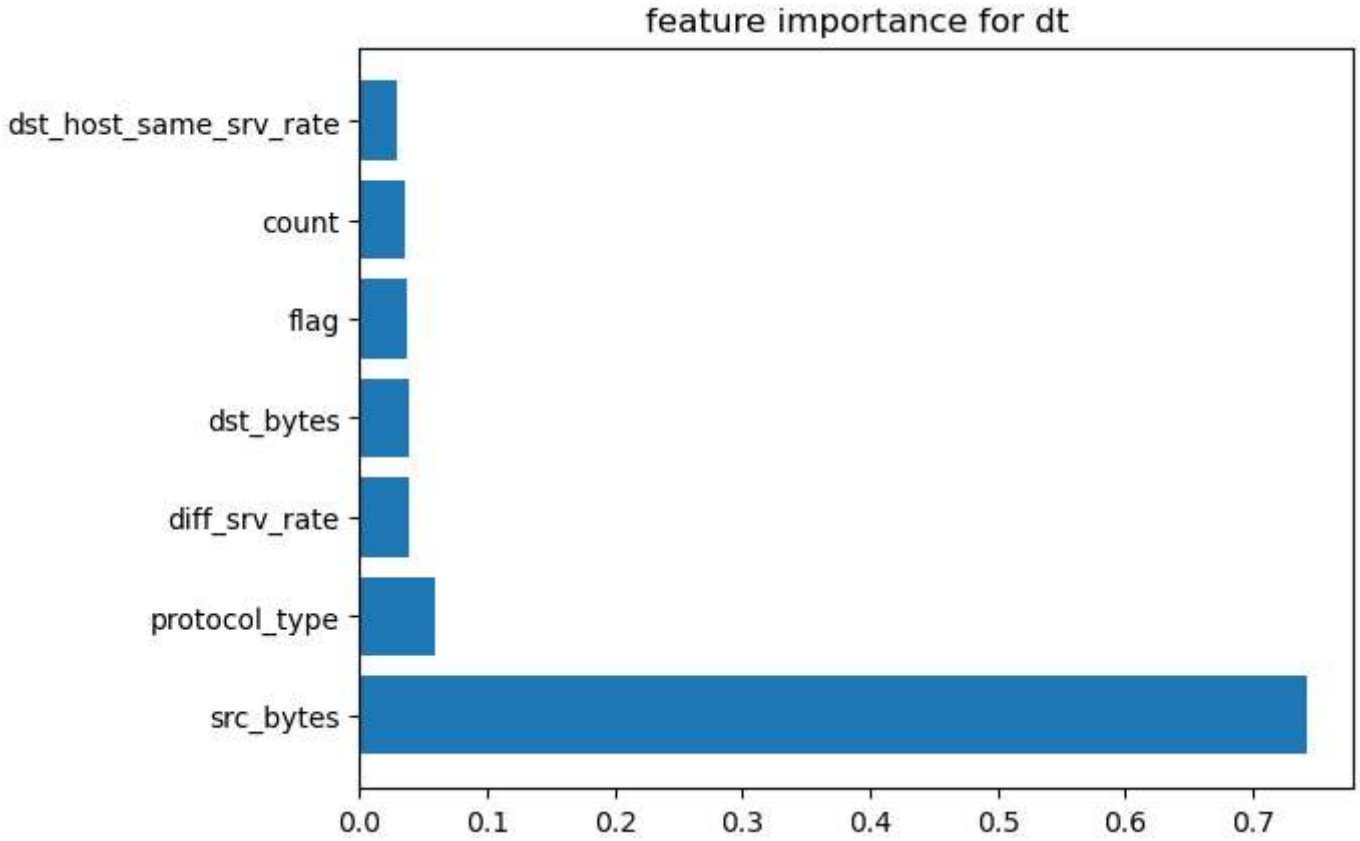
In [27]:
```
dt = DecisionTreeClassifier(max_features = study_dt.best_trial.params['dt_max_features'], max_depth = study_dt.best_tri
dt.fit(x_train, y_train)

dt_train, dt_test = dt.score(x_train, y_train), dt.score(x_test, y_test)

print(f"Train Score: {dt_train}")
print(f"Test Score: {dt_test}")
```

```
Train Score: 0.9999432913689463
Test Score: 0.9916644614977508
```

In [28]:
```
fig = plt.figure(figsize = (30,12))
tree.plot_tree(dt, filled=True);
plt.show()
```



In [29]:
```
from matplotlib import pyplot as plt

def f_importance(coef, names, top=-1):
    imp = coef
    imp, names = zip(*sorted(list(zip(imp, names))))

    # Show all features
    if top == -1:
        top = len(names)

    plt.barh(range(top), imp[::-1][0:top], align='center')
    plt.yticks(range(top), names[::-1][0:top])
    plt.title('feature importance for dt')
    plt.show()

# whatever your features are called
features_names = selected_features

# Specify your top n features you want to visualize.
# You can also discard the abs() function
# if you are interested in negative contribution of features
f_importance(abs(dt.feature_importances_), features_names, top=7)
```



Random Forest Classifier

In [30]:
```
def objective(trial):
    rf_max_depth = trial.suggest_int('rf_max_depth', 2, 32, log=False)
    rf_max_features = trial.suggest_int('rf_max_features', 2, 10, log=False)
    rf_n_estimators = trial.suggest_int('rf_n_estimators', 3, 20, log=False)
    classifier_obj = RandomForestClassifier(max_features = rf_max_features, max_depth = rf_max_depth, n_estimators = rf
    classifier_obj.fit(x_train, y_train)
    accuracy = classifier_obj.score(x_test, y_test)
    return accuracy
```

In [31]:
```
study_rf = optuna.create_study(direction='maximize')
study_rf.optimize(objective, n_trials=30)
print(study_rf.best_trial)
```

```
FrozenTrial(number=15, state=1, values=[0.9957660756813972], datetime_start=datetime.datetime(2023, 9, 7, 10, 33, 3, 16
516), datetime_complete=datetime.datetime(2023, 9, 7, 10, 33, 3, 579014), params={'rf_max_depth': 17, 'rf_max_feature
s': 7, 'rf_n_estimators': 13}, user_attrs={}, system_attrs={}, intermediate_values={}, distributions={'rf_max_depth': I
ntDistribution(high=32, log=False, low=2, step=1), 'rf_max_features': IntDistribution(high=10, log=False, low=2, step=
1), 'rf_n_estimators': IntDistribution(high=20, log=False, low=3, step=1)}, trial_id=15, value=None)
```

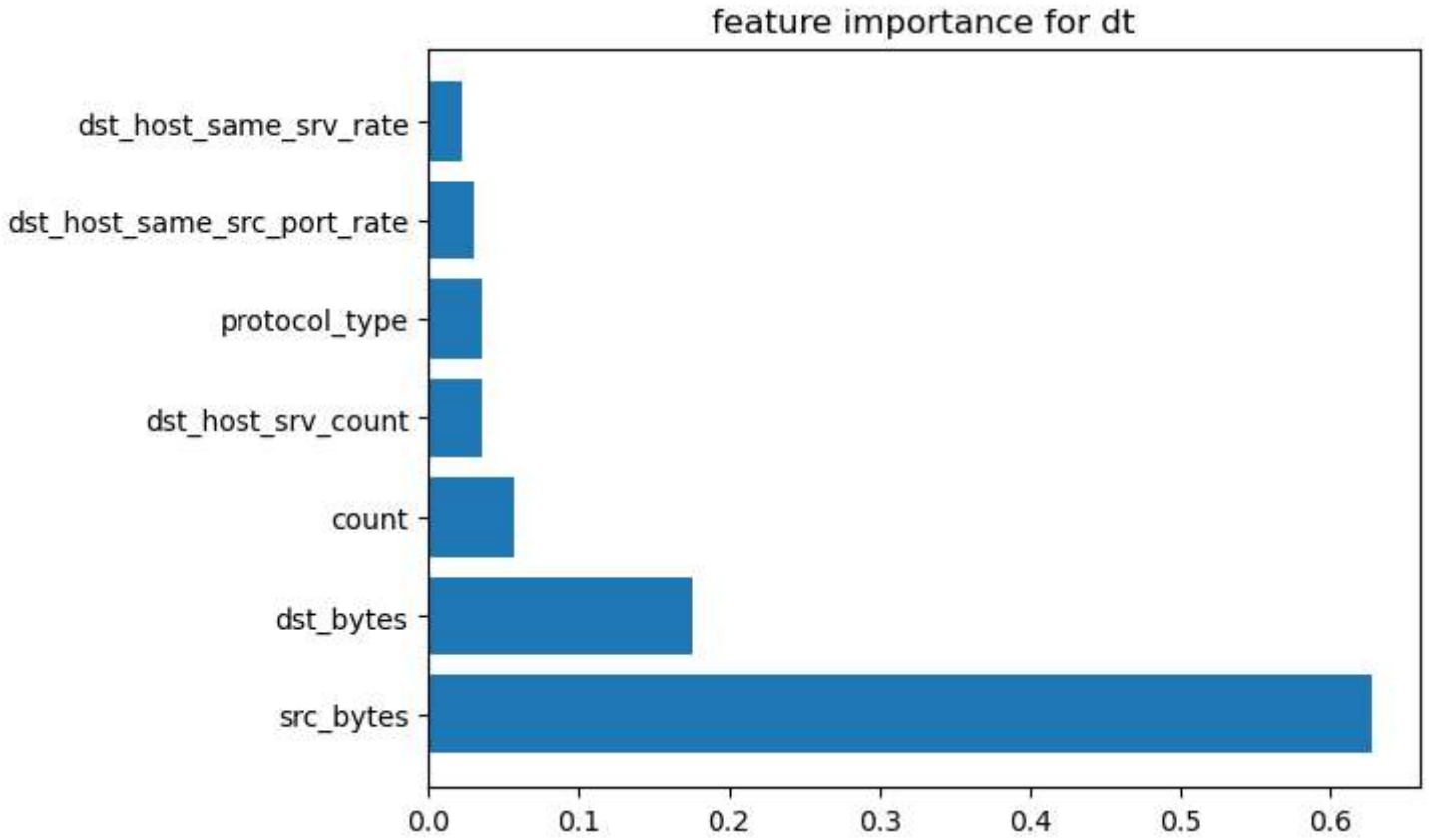
In [32]:

```
rf = RandomForestClassifier(max_features = study_rf.best_trial.params['rf_max_features'], max_depth = study_rf.best_tri
rf.fit(x_train, y_train)

rf_train, rf_test = rf.score(x_train, y_train), rf.score(x_test, y_test)

print(f"Train Score: {rf_train}")
print(f"Test Score: {rf_test}")
```

```
Train Score: 0.9997164568447318
Test Score: 0.9952368351415718
```

In [33]:

```
from matplotlib import pyplot as plt

def f_importance(coef, names, top=-1):
    imp = coef
    imp, names = zip(*sorted(list(zip(imp, names))))

    # Show all features
    if top == -1:
        top = len(names)

    plt.barh(range(top), imp[::-1][0:top], align='center')
    plt.yticks(range(top), names[::-1][0:top])
    plt.title('feature importance for dt')
    plt.show()

# whatever your features are called
features_names = selected_features

# Specify your top n features you want to visualize.
# You can also discard the abs() function
# if you are interested in negative contribution of features
f_importance(abs(rf.feature_importances_), features_names, top=7)
```



SKLearn Gradient Boosting Model

In [34]:
```
SKGB = GradientBoostingClassifier(random_state=42)
SKGB.fit(x_train, y_train)
```

Out[34]:
```
GradientBoostingClassifier(random_state=42)
```

In [35]:
```
SKGB_train, SKGB_test = SKGB.score(x_train , y_train), SKGB.score(x_test , y_test)

print(f"Training Score: {SKGB_train}")
print(f"Test Score: {SKGB_test}")
```

```
Training Score: 0.9952364749914937
Test Score: 0.9920613919026198
```

XGBoost Gradient Boosting Model

In [36]:
```
xgb_model = XGBClassifier(objective="binary:logistic", random_state=42)
xgb_model.fit(x_train, y_train)
```

Out[36]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=42, ...)
```

In [37]:
```
xgb_train, xgb_test = xgb_model.score(x_train , y_train), xgb_model.score(x_test , y_test)

print(f"Training Score: {xgb_train}")
print(f"Test Score: {xgb_test}")
```

```
Training Score: 0.9998865827378927
Test Score: 0.9956337655464409
```

Light Gradient Boosting Model

In [38]:
```
lgb_model = LGBMClassifier(random_state=42)
lgb_model.fit(x_train, y_train)
```

```
[LightGBM] [Info] Number of positive: 9389, number of negative: 8245
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead of testing was 0.001686 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1339
[LightGBM] [Info] Number of data points in the train set: 17634, number of used features: 10
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.532437 -> initscore=0.129932
[LightGBM] [Info] Start training from score 0.129932
```

Out[38]:

LGBMClassifier

LGBMClassifier(random_state=42)

In [39]:
```
lgb_train, lgb_test = lgb_model.score(x_train , y_train), lgb_model.score(x_test , y_test)

print(f"Training Score: {lgb_train}")
print(f"Test Score: {lgb_test}")
```

```
Training Score: 0.9999432913689463
Test Score: 0.9956337655464409
```

SKLearn AdaBoost Model

In [40]:
```
ab_model = AdaBoostClassifier(random_state=42)
```

In [41]:
```
ab_model.fit(x_train, y_train)
```

Out[41]:

AdaBoostClassifier

AdaBoostClassifier(random_state=42)

In [42]:
```
ab_train, ab_test = ab_model.score(x_train , y_train), ab_model.score(x_test , y_test)

print(f"Training Score: {ab_train}")
print(f"Test Score: {ab_test}")
```

```
Training Score: 0.981399569014404
Test Score: 0.9817412013760254
```

CatBoost Classifier Model

In [43]:
```
cb_model = CatBoostClassifier(verbose=0)
```

In [44]:
```
cb_model.fit(x_train, y_train)
```

Out[44]:
```
<catboost.core.CatBoostClassifier at 0x21a11a84d10>
```

In [45]:
```
cb_train, cb_test = cb_model.score(x_train , y_train), cb_model.score(x_test , y_test)

print(f"Training Score: {cb_train}")
print(f"Test Score: {cb_test}")
```

```
Training Score: 0.9984688669615516
Test Score: 0.9940460439269648
```

Naive Baye Model

In [46]:
```
BNB_model = BernoulliNB()
BNB_model.fit(x_train, y_train)
```

Out[46]:
```
BernoulliNB
BernoulliNB()
```

In [47]:
```
BNB_train, BNB_test = BNB_model.score(x_train , y_train), BNB_model.score(x_test , y_test)

print(f"Training Score: {BNB_train}")
print(f"Test Score: {BNB_test}")
```

```
Training Score: 0.8937847340365204
Test Score: 0.8948134427097115
```

Voting Model

In [48]:
```
v_clf = VotingClassifier(estimators=[('KNeighborsClassifier', KNN_model), ("XGBClassifier", xgb_model), ("RandomForestC
```

In [49]:
```
v_clf.fit(x_train, y_train)
```

```
[LightGBM] [Info] Number of positive: 9389, number of negative: 8245
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead of testing was 0.001093 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1339
[LightGBM] [Info] Number of data points in the train set: 17634, number of used features: 10
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.532437 -> initscore=0.129932
[LightGBM] [Info] Start training from score 0.129932
```

Out[49]:

In [50]:
```
voting_train, voting_test = v_clf.score(x_train , y_train), v_clf.score(x_test , y_test)

print(f"Training Score: {voting_train}")
print(f"Test Score: {voting_test}")
```

```
Training Score: 0.9999432913689463
Test Score: 0.9955014554114845
```

Bagging classifier

In [51]:
```
baggin = BaggingClassifier(estimator=SVC(), n_estimators=10, random_state=0)
baggin.fit(x_train , y_train)
bag_train, bag_test = baggin.score(x_train , y_train), baggin.score(x_test , y_test)

print(f"Training Score: {bag_train}")
print(f"Test Score: {bag_test}")
```

```
Training Score: 0.967449245775207
Test Score: 0.9655993649113522
```

RReliefF classifier

In [52]:
```
#baggin = BaggingClassifier(estimator=SVC(), n_estimators=10, random_state=0)
#baggin.fit(x_train , y_train)
```

```
#bag_train, bag_test = baggin.score(x_train , y_train), baggin.score(x_test , y_test)

#r = sr.RReliefF(n_features = 20)
#print(r.fit_transform(x_train,y_train))

#print(f"Training Score: {bag_train}")
#print(f"Test Score: {bag_test}")
```

In [53]:
```
data = [["*KNN", KNN_train, KNN_test],
        ["*Logistic Regression", lg_train, lg_test],
        ["Decision Tree", dt_train, dt_test],
        ["Random Forest", rf_train, rf_test],
        ["GBM", SKGB_train, SKGB_test],
        ["XGBM", xgb_train, xgb_test],
        ["*Adaboost", ab_train, ab_test],
        ["light GBM", lgb_train, lgb_test],
        ["CatBoost", cb_train, cb_test],
        ["*Naive Baye Model", BNB_train, BNB_test],
        ["*Voting", voting_train, voting_test],
        ["*Baggings", bag_train, bag_test]]

col_names = ["Model", "Train Score", "Test Score"]
print(tabulate(data, headers=col_names, tablefmt="fancy_grid"))
print('SVM Model takes a bit of time to run')
print('please wait while it runs')
```

Model	Train Score	Test Score
*KNN	0.979925	0.979889
*Logistic Regression	0.941817	0.938873
Decision Tree	0.999943	0.991664
Random Forest	0.999716	0.995237
GBM	0.995236	0.992061
XGBM	0.999887	0.995634
*Adaboost	0.9814	0.981741
light GBM	0.999943	0.995634
CatBoost	0.998469	0.994046
*Naive Baye Model	0.893785	0.894813
*Voting	0.999943	0.995501
*Baggings	0.967449	0.965599

SVM Model takes a bit of time to run
please wait while it runs

SVM Model

In [60]:
```
def objective(trial):
    kernel = trial.suggest_categorical('kernel', ['linear', 'rbf', 'poly', 'linearSVC'])
    c = trial.suggest_float('c', 0.02, 1.0, step=0.02)
    if kernel in ['linear', 'rbf']:
        classifier_obj = SVC(kernel=kernel, C=c).fit(x_train, y_train)
    elif kernel == 'linearSVC':
        classifier_obj = LinearSVC(C=c).fit(x_train, y_train)
    elif kernel == 'poly':
        degree = trial.suggest_int('degree', 2, 10)
        classifier_obj = SVC(kernel=kernel, C=c, degree=degree).fit(x_train, y_train)
```

```
    accuracy = classifier_obj.score(x_test, y_test)
    return accuracy
```

In [61]:
```
study_svm = optuna.create_study(direction='maximize')
study_svm.optimize(objective, n_trials=3)
print(study_svm.best_trial)
```

```
FrozenTrial(number=1, state=1, values=[0.9585869277586663], datetime_start=datetime.datetime(2023, 9, 7, 10, 36, 55, 43
7611), datetime_complete=datetime.datetime(2023, 9, 7, 10, 37, 1, 333421), params={'kernel': 'rbf', 'c': 0.06}, user_at
trs={}, system_attrs={}, intermediate_values={}, distributions={'kernel': CategoricalDistribution(choices=('linear', 'r
bf', 'poly', 'linearSVC')), 'c': FloatDistribution(high=1.0, log=False, low=0.02, step=0.02)}, trial_id=1, value=None)
```

In [62]:
```
if study_svm.best_trial.params['kernel'] in ['linear', 'rbf']:
    SVM_model = SVC(kernel=study_svm.best_trial.params['kernel'], C=study_svm.best_trial.params['c'])
elif kernel == 'linearSVC':
    SVM_model = LinearSVC(C=study_svm.best_trial.params['c'])
elif kernel == 'poly':
    SVM_model = SVC(kernel=study_svm.best_trial.params['kernel'], C=study_svm.best_trial.params['c'], degree=study_svm.

SVM_model.fit(x_train, y_train)
```

Out[62]:

▾ SVC

SVC(C=0.06)

In [63]:
```
SVM_train, SVM_test = SVM_model.score(x_train , y_train), SVM_model.score(x_test , y_test)

print(f"Training Score: {SVM_train}")
print(f"Test Score: {SVM_test}")
```

```
Training Score: 0.9547465124191902
Test Score: 0.9585869277586663
```

Summary

In [64]:
```
data = [["*KNN", KNN_train, KNN_test],
        ["*Logistic Regression", lg_train, lg_test],
        ["Decision Tree", dt_train, dt_test],
        ["Random Forest", rf_train, rf_test],
        ["GBM", SKGB_train, SKGB_test],
        ["XGBM", xgb_train, xgb_test],
        ["*Adaboost", ab_train, ab_test],
```

```
        ["light GBM", lgb_train, lgb_test],
        ["CatBoost", cb_train, cb_test],
        ["*Naive Baye Model", BNB_train, BNB_test],
        ["*Voting", voting_train, voting_test],
        ["*Baggings", bag_train, bag_test],
        ["*SVM", SVM_train, SVM_test]]

col_names = ["Model", "Train Score", "Test Score"]
print(tabulate(data, headers=col_names, tablefmt="fancy_grid"))
```

Model	Train Score	Test Score
*KNN	0.979925	0.979889
*Logistic Regression	0.941817	0.938873
Decision Tree	0.999943	0.991664
Random Forest	0.999716	0.995237
GBM	0.995236	0.992061
XGBM	0.999887	0.995634
*Adaboost	0.9814	0.981741
light GBM	0.999943	0.995634
CatBoost	0.998469	0.994046
*Naive Baye Model	0.893785	0.894813
*Voting	0.999943	0.995501
*Baggings	0.967449	0.965599
*SVM	0.954747	0.958587