Task 1: Load the KDD99Cup data and prepare to train DOS vs non-DOS

We have downloaded the the KDD99Cup data and have it in the local directory as "kddcup.data". We have also created a file called "kddcup.headers" in the local directory, which is a space delimited list of the header names for the features. We load the data into a DataFrame. We create a target array which is 1 for DOS attacks and 0 for all other records.

Since SVM is very computationally expensive, we reduce our data set to 100,000 DOS records and 100,000 non-DOS records. We then use OneHotEncoder to encode the categorical columns, and finally scale our columns using MinMaxScaler.

In [1]:
```
# Load kddcup data. This was downloaded and placed in the same directo
# For header names, use file we created in the local directory called
import pandas as pd
import numpy as np
import timeit

df = pd.read_csv('kddcup.data')
headers = np.genfromtxt('kddcup.headers', dtype=str, delimiter=" ")
```

In [2]:

```
# Print the shape of the original data set, for reference
print(f"shape of data set: {df.shape}")

# Set the feature set (X) to columns 0-40 of the data set. Set the hea
# file we imported above.
X = df.iloc[:, 0:41]
X.columns = headers
print(f"shape of X: {X.shape}")

# Create an ndarray "y_raw" with the target labels from column 41 of t
# will encode it to create our target array y.
y_raw = df.iloc[:, 41].values
print(f"shape of y_raw: {y_raw.shape}")
print()

#Print the number of records of each class
from collections import Counter
class_counts = Counter(y_raw)
print("Count of records of each type:")
for c in class_counts:
    print(f"{c}: {class_counts[c]}")
```

```
shape of data set: (4898430, 42)
shape of X: (4898430, 41)
shape of y_raw: (4898430,)

Count of records of each type:
normal.: 972780
buffer_overflow.: 30
loadmodule.: 9
perl.: 3
neptune.: 1072017
smurf.: 2807886
guess_passwd.: 53
pod.: 264
teardrop.: 979
portsweep.: 10413
ipsweep.: 12481
land.: 21
ftp_write.: 8
back.: 2203
imap.: 12
satan.: 15892
phf.: 4
nmap.: 2316
multihop.: 7
warezmaster.: 20
warezclient.: 1020
spy.: 2
rootkit.: 10
```

In [3]:
```
# Look at the occurence of the different categories in the categorical
print(Counter(X['protocol_type']))
print(Counter(X['service']))
print(Counter(X['flag']))
```

```
Counter({'icmp': 2833545, 'tcp': 1870597, 'udp': 194288})
Counter({'ecr_i': 2811660, 'private': 1100831, 'http': 623090, 'smtp'
: 96554, 'other': 72653, 'domain_u': 57782, 'ftp_data': 40697, 'eco_i
': 16338, 'finger': 6891, 'urp_i': 5378, 'ftp': 5214, 'telnet': 4277,
'ntp_u': 3833, 'auth': 3382, 'pop_3': 1981, 'time': 1579, 'domain': 1
113, 'Z39_50': 1078, 'gopher': 1077, 'mtp': 1076, 'ssh': 1075, 'whois
': 1073, 'remote_job': 1073, 'rje': 1070, 'link': 1069, 'imap4': 1069
, 'ctf': 1068, 'name': 1067, 'supdup': 1060, 'echo': 1059, 'discard':
1059, 'nntp': 1059, 'uucp_path': 1057, 'netstat': 1056, 'daytime': 10
56, 'systat': 1056, 'sunrpc': 1056, 'netbios_ssn': 1055, 'pop_2': 105
5, 'netbios_ns': 1054, 'vmnet': 1053, 'netbios_dgm': 1052, 'sql_net':
1052, 'iso_tsap': 1052, 'shell': 1051, 'csnet_ns': 1051, 'klogin': 10
50, 'hostnames': 1050, 'bgp': 1047, 'login': 1045, 'exec': 1045, 'pri
nter': 1045, 'http_443': 1044, 'efs': 1042, 'uucp': 1041, 'ldap': 104
1, 'kshell': 1040, 'nnsp': 1038, 'courier': 1021, 'IRC': 521, 'urh_i'
: 148, 'X11': 135, 'tim_i': 12, 'red_i': 9, 'pm_dump': 5, 'tftp_u': 3
, 'harvest': 2, 'aol': 2, 'http_8001': 2, 'http_2784': 1})
Counter({'SF': 3744327, 'S0': 869829, 'REJ': 268874, 'RSTR': 8094, 'R
STO': 5344, 'SH': 1040, 'S1': 532, 'S2': 161, 'RSTOS0': 122, 'OTH': 5
7, 'S3': 50})
```

In [4]:
```
# Define a function for converting target labels to 1 (DOS) or 0 (non-
# to encode the target array.
dosAttackTypes = ["back.", "land.", "neptune.", "pod.", "smurf.", "tea

def DOSEncode(x):
    if (x in dosAttackTypes):
        return 1
    else:
        return 0
```

In [5]:
```
# Use the Dataframe function "applymap" to apply our encoding functior
# labels into 1's and 0's.
labels=pd.DataFrame(data=y_raw)
labels2 = labels.applymap(DOSEncode)
y = labels2[0].values

# Print out the counts of the encoded values.
print(Counter(y))
```

```
Counter({1: 3883370, 0: 1015060})
```

In [6]:
```
# Use OneHotEncoder on protocol_type
from sklearn.preprocessing import OneHotEncoder

# Create a dataframe of encoded data from "protocol_type"
enc = OneHotEncoder(sparse=False)
X2 = pd.DataFrame(enc.fit_transform(X[['protocol_type']]))
X2.columns = enc.get_feature_names_out(['protocol_type'])

# Drop the original "protocol_type" column, and merge in the new encod
X3 = X.copy()
X3.drop(['protocol_type'], axis=1, inplace=True)
X4 = pd.concat([X3, X2], axis=1)

# Print the resultant shape
print(X4.shape)
```

```
(4898430, 43)
```

In [7]:
```
# Use OneHotEncoder on service

# Create a dataframe of encoded data from "service"
X5 = pd.DataFrame(enc.fit_transform(X4[['service']]))
X5.columns = enc.get_feature_names_out(['service'])

# Drop the original "service" column, and merge in the new encoded col
X6 = X4.copy()
X6.drop(['service'], axis=1, inplace=True)
X7 = pd.concat([X6, X5 ], axis=1)

# Print the resultant shape
print(X7.shape)
```

```
(4898430, 112)
```

In [8]:
```
# Use OneHotEncoder on flag

# Create a dataframe of encoded data from "flag"
X8 = pd.DataFrame(enc.fit_transform(X7[['flag']]))
X8.columns = enc.get_feature_names_out(['flag'])

# Drop the original "flag" column, and merge in the new encoded column
X9 = X7.copy()
X9.drop(['flag'], axis=1, inplace=True)
X10 = pd.concat([X9, X8], axis=1)

# Print the resultant shape
print(X10.shape)
```

```
(4898430, 122)
```

In [9]:
```
# The dataset is unbalanced, we have many more failure cases than succ
# balance the number of records of each type.
from collections import Counter
from imblearn.under_sampling import RandomUnderSampler

# Print the shape of the original data set
print('Original dataset shape %s' % Counter(y))

# Use RandomUnderSampler, to take 100,000 DOS records and 100,000 non-
# Had to limit to only that number of samples, or else SVM was not con
rus = RandomUnderSampler(random_state=0, sampling_strategy={1:200000,
X11, y1 = rus.fit_resample(X10, y)

# Print shape of resampled data set, and count of label values in the
print(f'Resampled dataset shape: {X11.shape}')
print(f'Count of label values: {Counter(y1)}')
```

```
Original dataset shape Counter({1: 3883370, 0: 1015060})
Resampled dataset shape: (400000, 122)
Count of label values: Counter({0: 200000, 1: 200000})
```

In [10]:
```
# Store final feature set into X_encoded
X_encoded = X11

# View dataframe
X_encoded
```

Out[10]:

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins
0	0	253	2174	0	0	0	0	0
1	1	1564	673	0	0	0	0	0
2	0	349	22780	0	0	0	0	0
3	0	311	1310	0	0	0	0	0
4	0	215	955	0	0	0	0	0
...	...	...	...	...	...	...	...	...
399995	0	1032	0	0	0	0	0	0
399996	0	1032	0	0	0	0	0	0
399997	0	1032	0	0	0	0	0	0
399998	0	1032	0	0	0	0	0	0
399999	0	0	0	0	0	0	0	0

400000 rows × 122 columns

In [11]:
```
# Standardize the feature set data to help with training.
from sklearn import preprocessing

# Use MinMaxScaler to scale the data set
scaler = preprocessing.MinMaxScaler().fit(X_encoded)
scaledArray = scaler.transform(X_encoded)
X_scaled = pd.DataFrame(data=scaledArray, columns=X_encoded.columns)

# View dataframe
X_scaled.describe()
```

Out[11]:

	duration	src_bytes	dst_bytes	land	wrong_fragment	ur
count	400000.000000	4.000000e+05	400000.000000	400000.000000	400000.000000	400000.00
mean	0.002748	7.650367e-06	0.000141	0.000008	0.000118	0.00
std	0.026535	1.594802e-03	0.003057	0.002739	0.010619	0.00
min	0.000000	0.000000e+00	0.000000	0.000000	0.000000	0.00
25%	0.000000	4.832533e-07	0.000000	0.000000	0.000000	0.00
50%	0.000000	1.399133e-06	0.000000	0.000000	0.000000	0.00
75%	0.000000	4.749690e-06	0.000033	0.000000	0.000000	0.00
max	1.000000	1.000000e+00	1.000000	1.000000	1.000000	1.00

8 rows × 122 columns

Split data into Test and Train data sets

In [12]:
```
# Split the data into a training set (80%) and testing set (20%).
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y1, test
print(f"training set shape: {X_train.shape}")
print(f"testing set shape: {X_test.shape}")
```

```
training set shape: (320000, 122)
testing set shape: (80000, 122)
```

Task 2: Run the SVM Model with 4 Different Kernels, and compare the results

Train and test with "rbf" kernel

In [13]:
```
# Create RBF classifier and train with the training set.
from sklearn.svm import SVC
import time

svm1 = SVC(kernel='rbf', random_state=0)
print(svm1.get_params())

# Time how long it takes to train.
start = time.time()
svm1.fit(X_train, y_train)
end = time.time()
svm1_train_time = end - start

print(f"secs to fit: {svm1_train_time:.2f}")
print(f"iterations required: {svm1.n_iter_}")
```

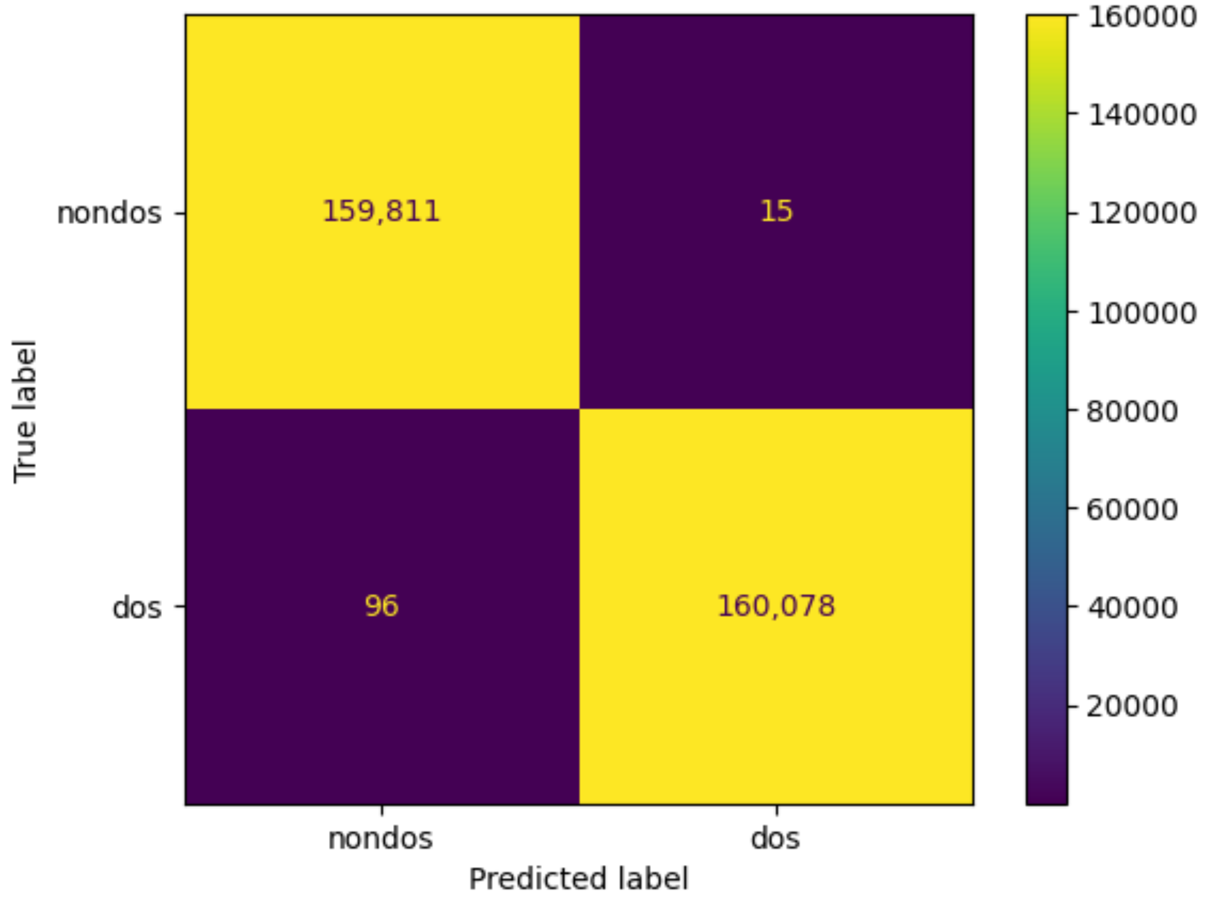
```
{'C': 1.0, 'break_ties': False, 'cache_size': 200, 'class_weight': No
ne, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree': 3, 'gam
ma': 'scale', 'kernel': 'rbf', 'max_iter': -1, 'probability': False,
'random_state': 0, 'shrinking': True, 'tol': 0.001, 'verbose': False}
secs to fit: 18.15
iterations required: [826]
```

In [14]:
```
# Predict the training set. Time how long it takes to predict.
start = time.time()
svm1_train_predict = svm1.predict(X_train)
end = time.time()
svm1_pred_time = end - start

print(f"secs to predict (training set): {svm1_pred_time:.2f}")
```

```
secs to predict (training set): 19.79
```

In [15]:
```
# Print classification report and ConfusionMatrix for training data se
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

target_labels = [0, 1]
target_names = ['nondos', 'dos']
print(classification_report(y_train, svm1_train_predict, labels=target

ConfusionMatrixDisplay.from_predictions(y_train, svm1_train_predict, v
plt.show()
```

```
              precision    recall  f1-score   support

      nondos     0.9994    0.9999    0.9997    159826
         dos     0.9999    0.9994    0.9997    160174

    accuracy                         0.9997    320000
   macro avg     0.9997    0.9997    0.9997    320000
weighted avg     0.9997    0.9997    0.9997    320000
```

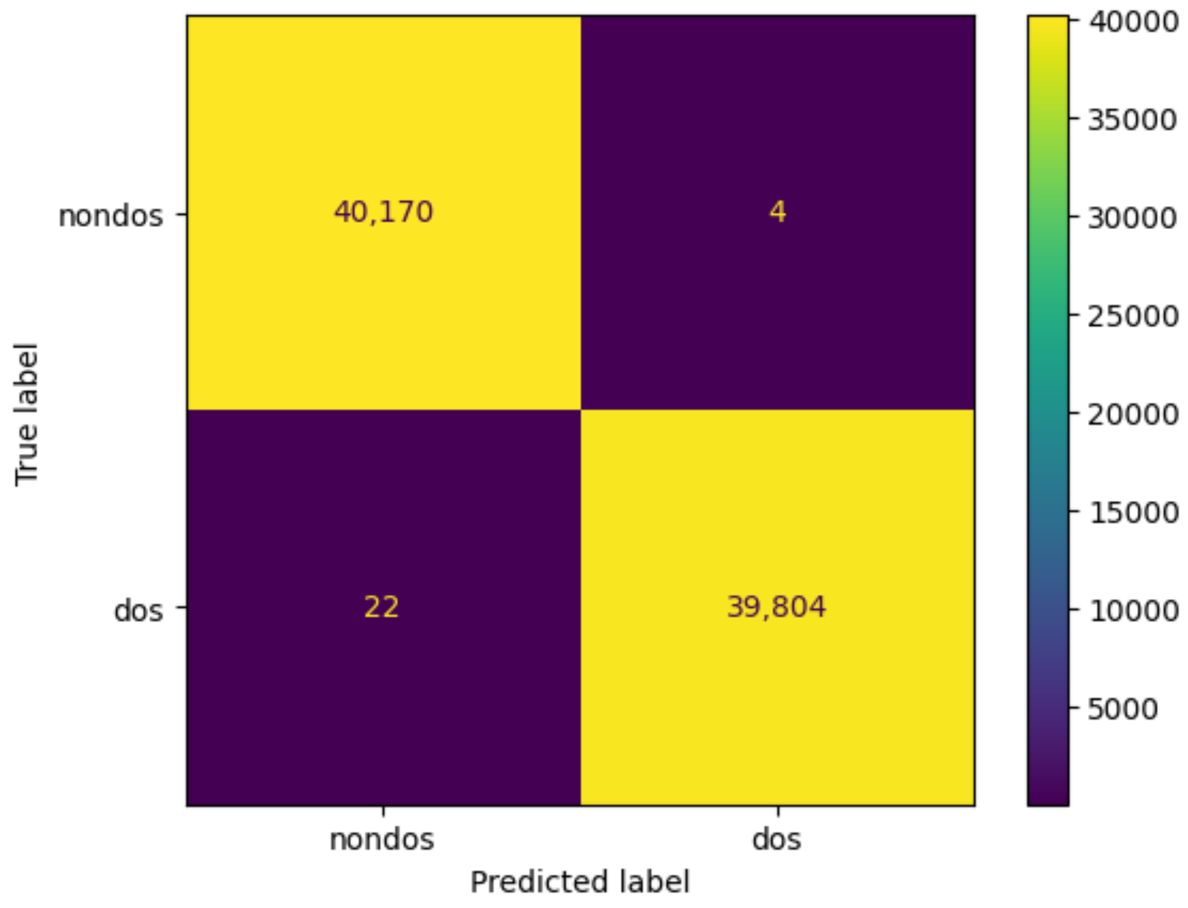


In [16]:
```
# Predict the test set. Time how long it takes to predict.
start = time.time()
svm1_test_predict = svm1.predict(X_test)
end = time.time()
svm1_test_time = end - start

print(f"secs to predict (test set): {svm1_test_time:.2f}")
```

```
secs to predict (test set): 4.93
```

In [17]:
```
# Print classification report and ConfusionMatrix for test data set
from sklearn.metrics import classification_report

target_labels = [0, 1]
target_names = ['nondos', 'dos']
print(classification_report(y_test, svm1_test_predict, labels=target_l

ConfusionMatrixDisplay.from_predictions(y_test, svm1_test_predict, val
plt.show()
```

```
              precision    recall  f1-score   support

      nondos     0.9995    0.9999    0.9997     40174
         dos     0.9999    0.9994    0.9997     39826

    accuracy                         0.9997     80000
   macro avg     0.9997    0.9997    0.9997     80000
weighted avg     0.9997    0.9997    0.9997     80000
```



Train and test with "linear" kernel

In [18]:
```
# Create Linear classifier and train with the training set.
from sklearn.svm import SVC
import time

svm2 = SVC(kernel='linear', random_state=0)
print(svm2.get_params())

# Time how long it takes to train.
start = time.time()
svm2.fit(X_train, y_train)
end = time.time()
svm2_train_time = end - start

print(f"secs to fit: {svm2_train_time:.2f}")
print(f"iterations required: {svm2.n_iter_}")
```

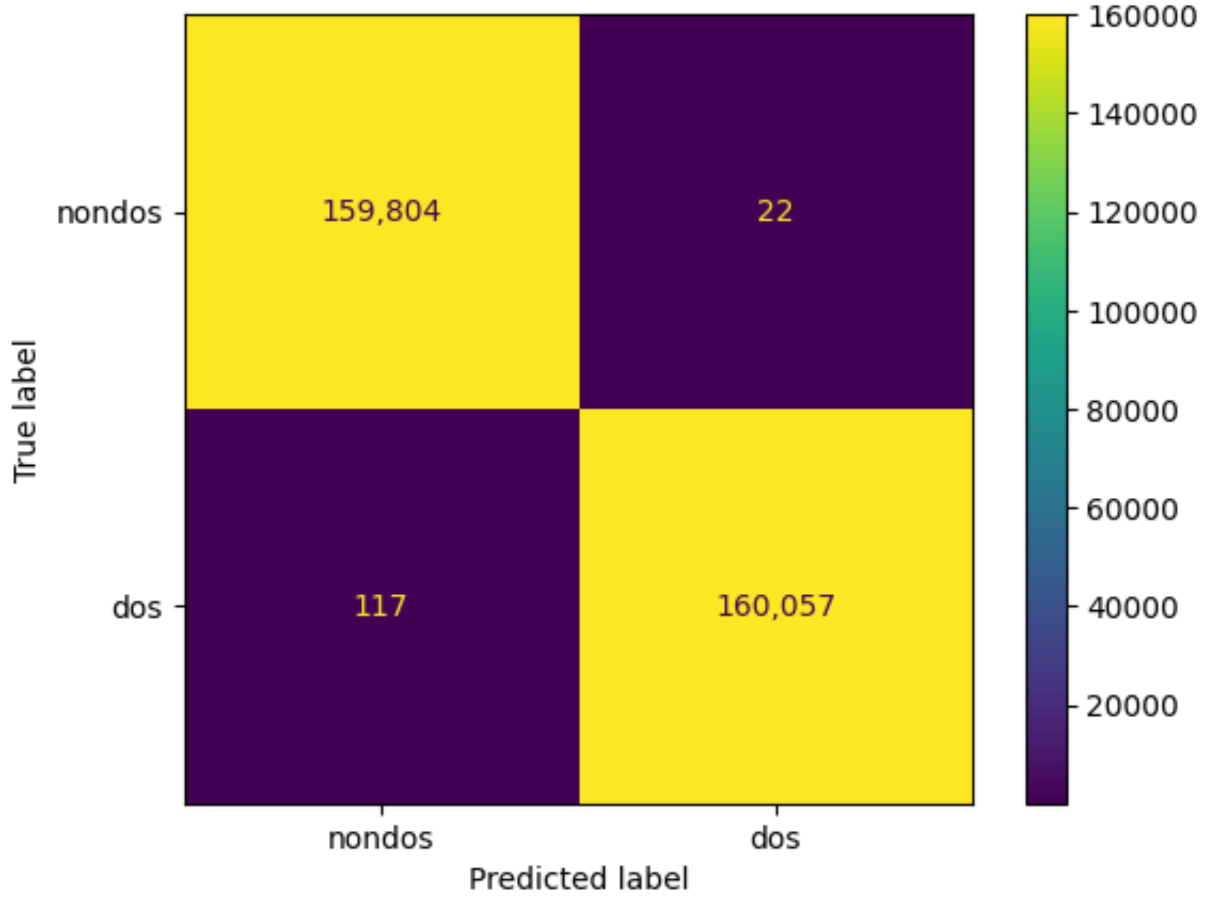
```
{'C': 1.0, 'break_ties': False, 'cache_size': 200, 'class_weight': No
ne, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree': 3, 'gam
ma': 'scale', 'kernel': 'linear', 'max_iter': -1, 'probability': Fals
e, 'random_state': 0, 'shrinking': True, 'tol': 0.001, 'verbose': Fal
se}
secs to fit: 19.83
iterations required: [9633]
```

In [19]:
```
# Predict the training set. Time how long it takes to predict.
start = time.time()
svm2_train_predict = svm2.predict(X_train)
end = time.time()
svm2_pred_time = end - start

print(f"secs to predict (training set): {svm2_pred_time:.2f}")
```

```
secs to predict (training set): 9.11
```

In [20]:
```
# Print classification report and ConfusionMatrix for training data se
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

target_labels = [0, 1]
target_names = ['nondos', 'dos']
print(classification_report(y_train, svm2_train_predict, labels=target

ConfusionMatrixDisplay.from_predictions(y_train, svm2_train_predict, v
plt.show()
```

```
              precision    recall  f1-score   support

      nondos     0.9993    0.9999    0.9996    159826
         dos     0.9999    0.9993    0.9996    160174

    accuracy                         0.9996    320000
   macro avg     0.9996    0.9996    0.9996    320000
weighted avg     0.9996    0.9996    0.9996    320000
```

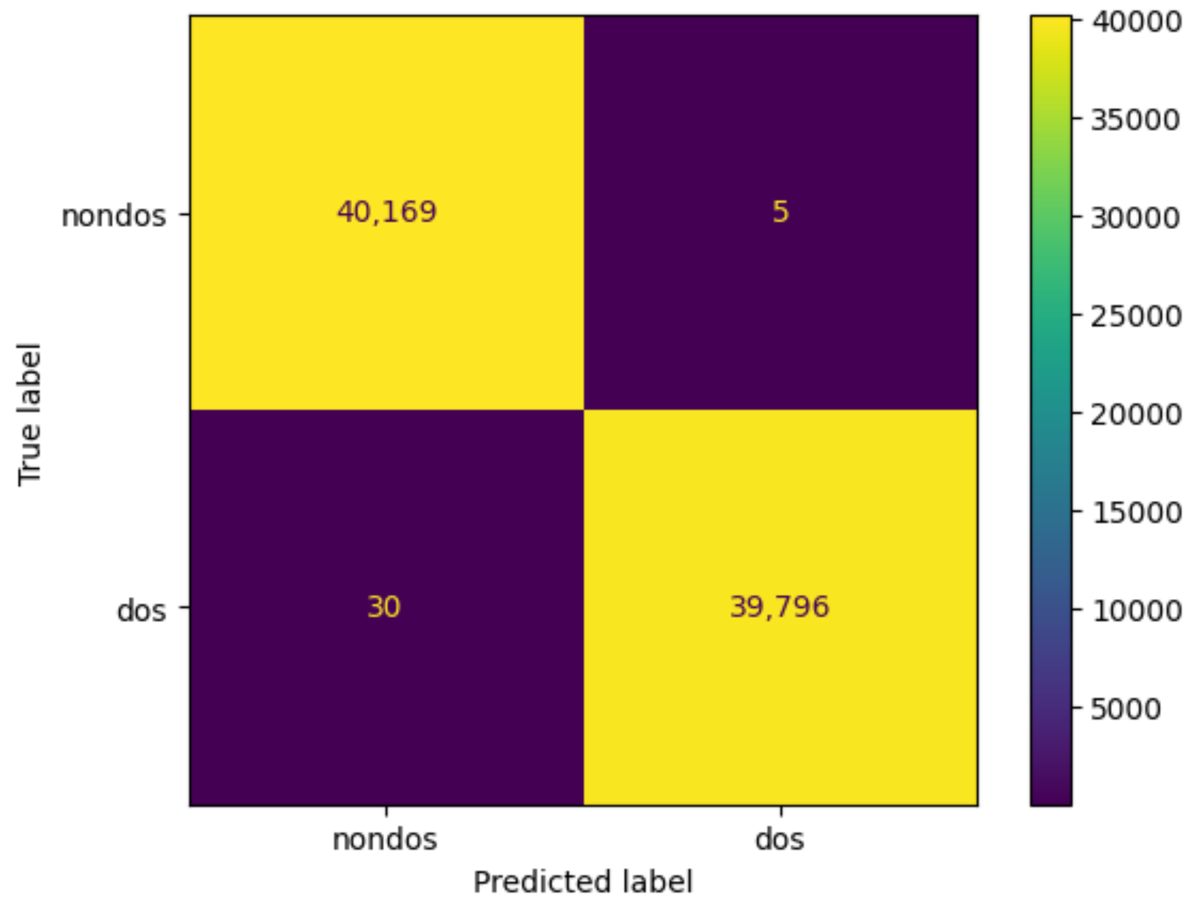


In [21]:
```
# Predict the test set. Time how long it takes to predict.
start = time.time()
svm2_test_predict = svm2.predict(X_test)
end = time.time()
svm2_test_time = end - start

print(f"secs to predict (test set): {svm2_test_time:.2f}")
```

```
secs to predict (test set): 2.14
```

In [22]:
```
# Print classification report and ConfusionMatrix for test data set
from sklearn.metrics import classification_report

target_labels = [0, 1]
target_names = ['nondos', 'dos']
print(classification_report(y_test, svm2_test_predict, labels=target_l

ConfusionMatrixDisplay.from_predictions(y_test, svm2_test_predict, val
plt.show()
```

```
              precision    recall  f1-score   support

      nondos     0.9993    0.9999    0.9996     40174
         dos     0.9999    0.9992    0.9996     39826

    accuracy                         0.9996     80000
   macro avg     0.9996    0.9996    0.9996     80000
weighted avg     0.9996    0.9996    0.9996     80000
```



Train and test with "poly" kernel

In [23]:
```
# Create Poly classifier and train with the training set.
from sklearn.svm import SVC
import time

svm3 = SVC(kernel='poly', random_state=0)
print(svm3.get_params())

# Time how long it takes to train.
start = time.time()
svm3.fit(X_train, y_train)
end = time.time()
svm3_train_time = end - start

print(f"secs to fit: {svm3_train_time:.2f}")
print(f"iterations required: {svm3.n_iter_}")
```

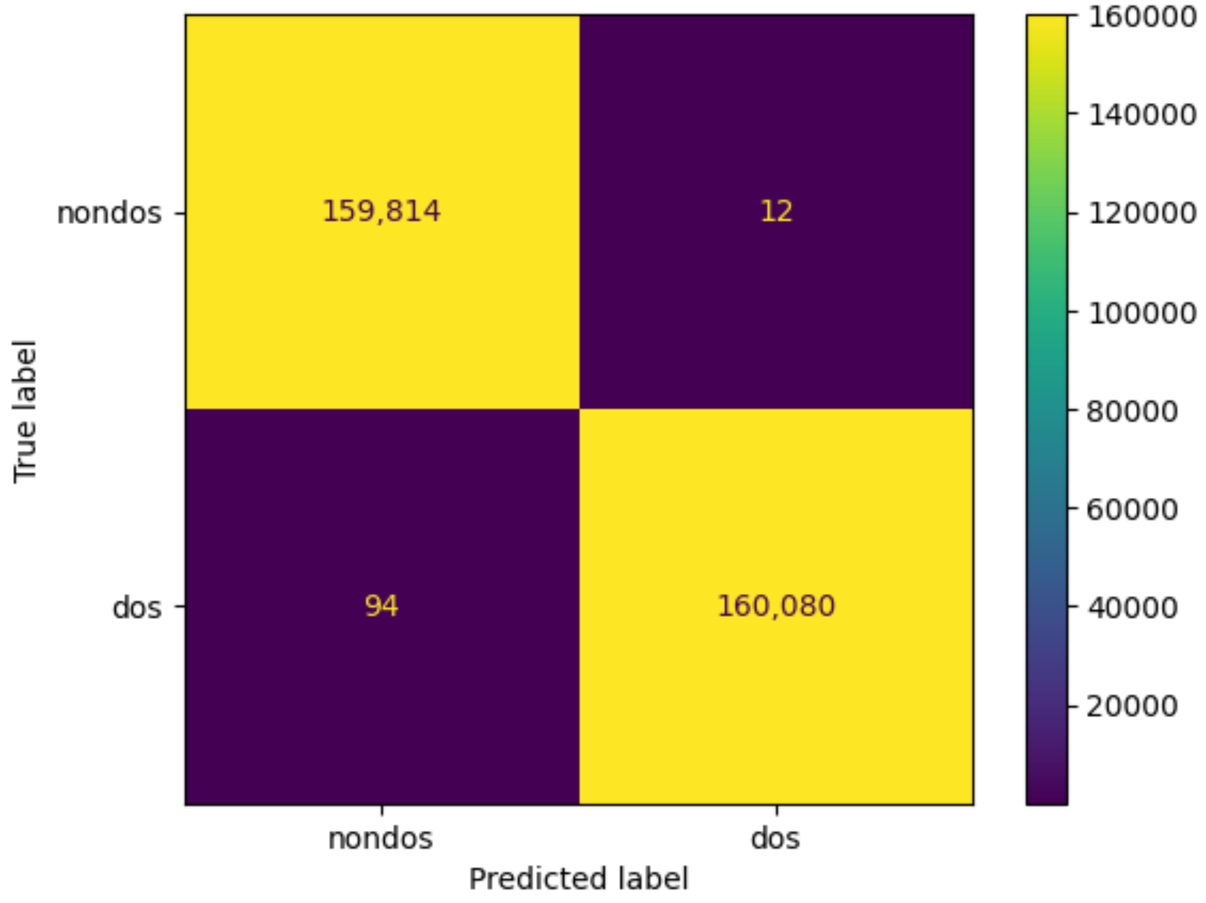
```
{'C': 1.0, 'break_ties': False, 'cache_size': 200, 'class_weight': No
ne, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree': 3, 'gam
ma': 'scale', 'kernel': 'poly', 'max_iter': -1, 'probability': False,
'random_state': 0, 'shrinking': True, 'tol': 0.001, 'verbose': False}
secs to fit: 15.20
iterations required: [947]
```

In [24]:
```
# Predict the training set. Time how long it takes to predict.
start = time.time()
svm3_train_predict = svm3.predict(X_train)
end = time.time()
svm3_pred_time = end - start

print(f"secs to predict (training set): {svm3_pred_time:.2f}")
```

```
secs to predict (training set): 6.47
```

In [25]:
```
# Print classification report and ConfusionMatrix for training data se
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

target_labels = [0, 1]
target_names = ['nondos', 'dos']
print(classification_report(y_train, svm3_train_predict, labels=target

ConfusionMatrixDisplay.from_predictions(y_train, svm3_train_predict, v
plt.show()
```

```
              precision    recall  f1-score   support

      nondos     0.9994    0.9999    0.9997    159826
         dos     0.9999    0.9994    0.9997    160174

    accuracy                         0.9997    320000
   macro avg     0.9997    0.9997    0.9997    320000
weighted avg     0.9997    0.9997    0.9997    320000
```

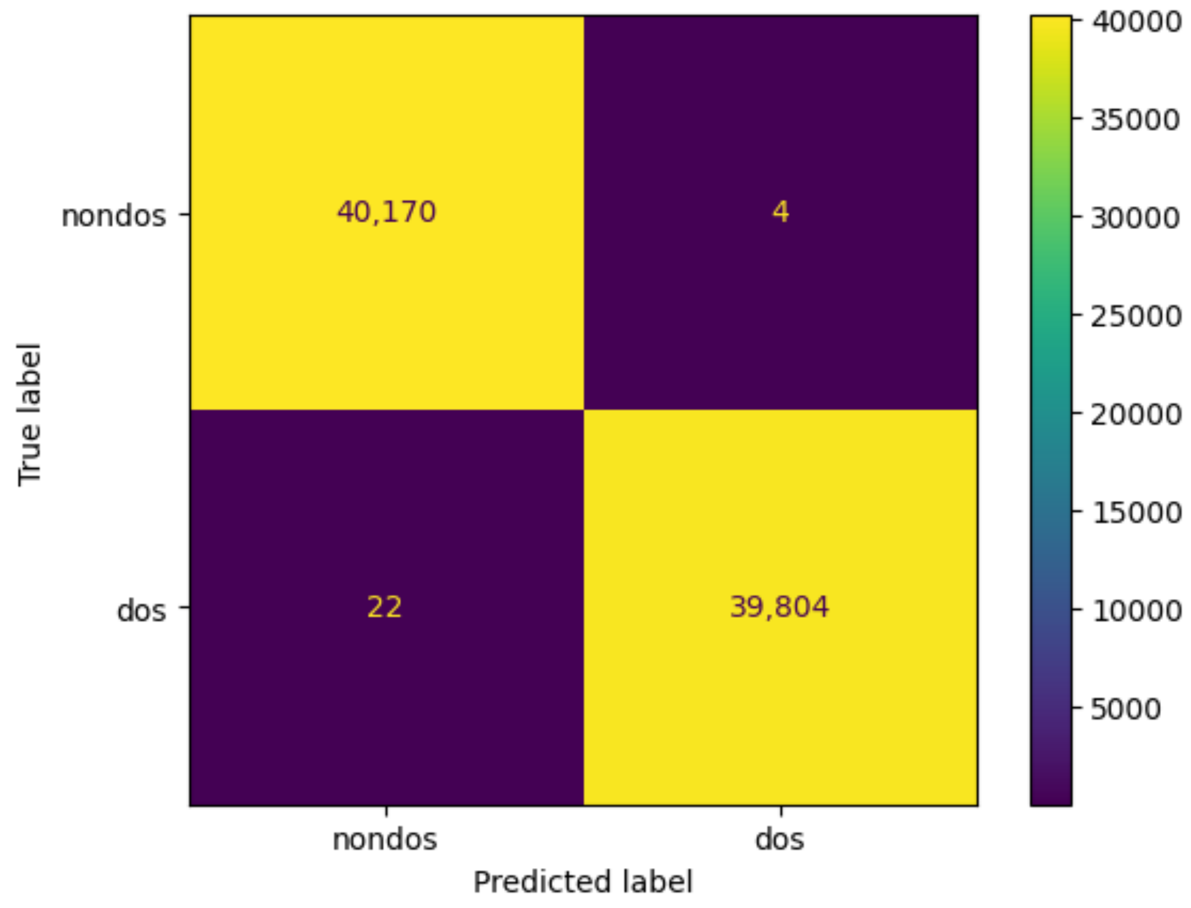


In [26]:
```
# Predict the test set. Time how long it takes to predict.
start = time.time()
svm3_test_predict = svm3.predict(X_test)
end = time.time()
svm3_test_time = end - start

print(f"secs to predict (test set): {svm3_test_time:.2f}")
```

```
secs to predict (test set): 2.00
```

In [27]:
```
# Print classification report and ConfusionMatrix for test data set
from sklearn.metrics import classification_report

target_labels = [0, 1]
target_names = ['nondos', 'dos']
print(classification_report(y_test, svm3_test_predict, labels=target_l

ConfusionMatrixDisplay.from_predictions(y_test, svm3_test_predict, val
plt.show()
```

```
              precision    recall  f1-score   support

      nondos     0.9995    0.9999    0.9997     40174
         dos     0.9999    0.9994    0.9997     39826

    accuracy                         0.9997     80000
   macro avg     0.9997    0.9997    0.9997     80000
weighted avg     0.9997    0.9997    0.9997     80000
```



Train and test with "sigmoid" kernel

In [28]:
```
# Create Sigmoid classifier and train with the training set.
from sklearn.svm import SVC
import time

svm4 = SVC(kernel='sigmoid', random_state=0)
print(svm4.get_params())

# Time how long it takes to train.
start = time.time()
svm4.fit(X_train, y_train)
end = time.time()
svm4_train_time = end - start

print(f"secs to fit: {svm4_train_time:.2f}")
print(f"iterations required: {svm4.n_iter_}")
```

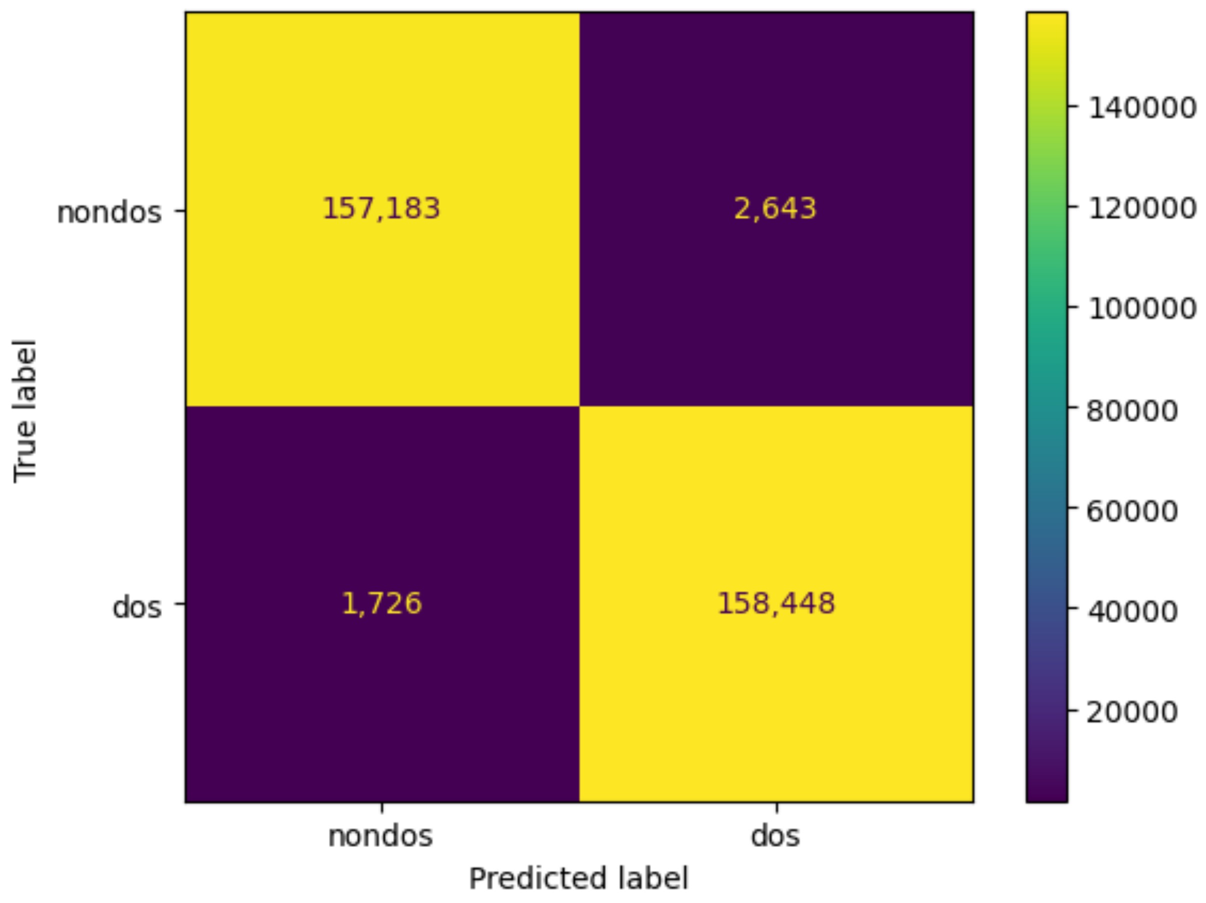
```
{'C': 1.0, 'break_ties': False, 'cache_size': 200, 'class_weight': No
ne, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree': 3, 'gam
ma': 'scale', 'kernel': 'sigmoid', 'max_iter': -1, 'probability': Fal
se, 'random_state': 0, 'shrinking': True, 'tol': 0.001, 'verbose': Fa
lse}
secs to fit: 522.40
iterations required: [4383]
```

In [29]:
```
# Predict the training set. Time how long it takes to predict.
start = time.time()
svm4_train_predict = svm4.predict(X_train)
end = time.time()
svm4_pred_time = end - start

print(f"secs to predict (training set): {svm4_pred_time:.2f}")
```

```
secs to predict (training set): 88.59
```

In [30]:
```
# Print classification report and ConfusionMatrix for training data se
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

target_labels = [0, 1]
target_names = ['nondos', 'dos']
print(classification_report(y_train, svm4_train_predict, labels=target

ConfusionMatrixDisplay.from_predictions(y_train, svm4_train_predict, v
plt.show()
```

```
              precision    recall  f1-score   support

      nondos     0.9891    0.9835    0.9863    159826
         dos     0.9836    0.9892    0.9864    160174

    accuracy                         0.9863    320000
   macro avg     0.9864    0.9863    0.9863    320000
weighted avg     0.9864    0.9863    0.9863    320000
```

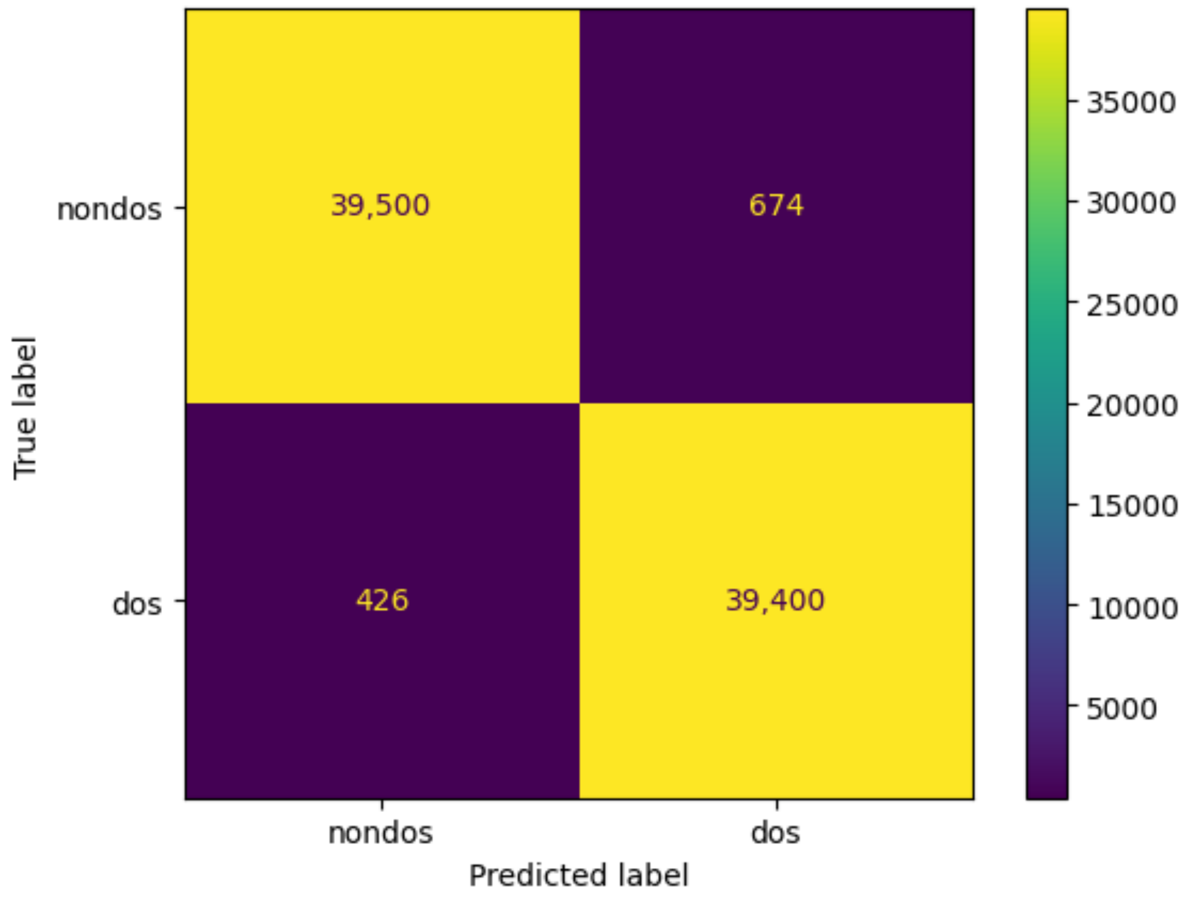


In [31]:
```
# Predict the test set. Time how long it takes to predict.
start = time.time()
svm4_test_predict = svm4.predict(X_test)
end = time.time()
svm4_test_time = end - start

print(f"secs to predict (test set): {svm4_test_time:.2f}")
```

```
secs to predict (test set): 29.48
```

In [32]:
```
# Print classification report and ConfusionMatrix for test data set
from sklearn.metrics import classification_report

target_labels = [0, 1]
target_names = ['nondos', 'dos']
print(classification_report(y_test, svm4_test_predict, labels=target_l

ConfusionMatrixDisplay.from_predictions(y_test, svm4_test_predict, val
plt.show()
```

```
              precision    recall  f1-score   support

      nondos     0.9893    0.9832    0.9863     40174
         dos     0.9832    0.9893    0.9862     39826

    accuracy                         0.9862     80000
   macro avg     0.9863    0.9863    0.9862     80000
weighted avg     0.9863    0.9862    0.9863     80000
```



Compare Kernels

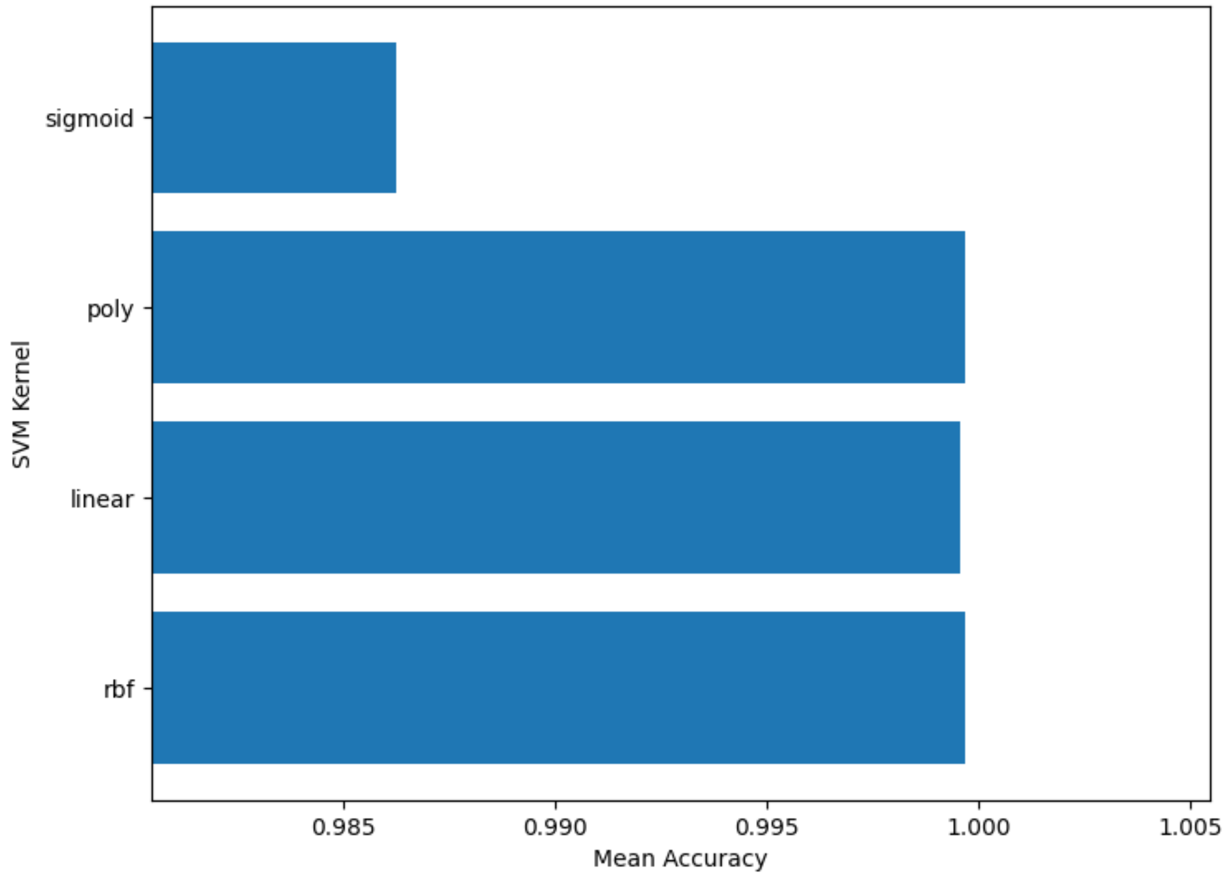
Below we compare the results of the different kernels. We see that Poly and RBF had the best accuracy scores, with linear slightly lower and sigmoid much lower. Sigmoid took much more time to train than the other two, with the other three much faster but Poly having the fastest time. Linear required a lot of iterations to train, but the time needed was close to that required by Sigmoid and Poly. Poly had the fastest testing time, with linear close behind, RBF behind that, and Sigmoid again much slower than the rest.

On this data set, Poly performed the best; it was the fastest for training and testing, and tied with RBF for the best testing accuracy. Linear and RBF were both close behind, with linear a little less accurate, and RBF a little slower. Sigmoid was by far the slowest and also the least accurate.

In [55]:
```
# Generate scores array for all the different SVM kernels
scores = []
scores.append(svm1.score(X_test, y_test))
scores.append(svm2.score(X_test, y_test))
scores.append(svm3.score(X_test, y_test))
scores.append(svm4.score(X_test, y_test))
scores
```

Out[55]:
```
[0.999675, 0.9995625, 0.999675, 0.98625]
```

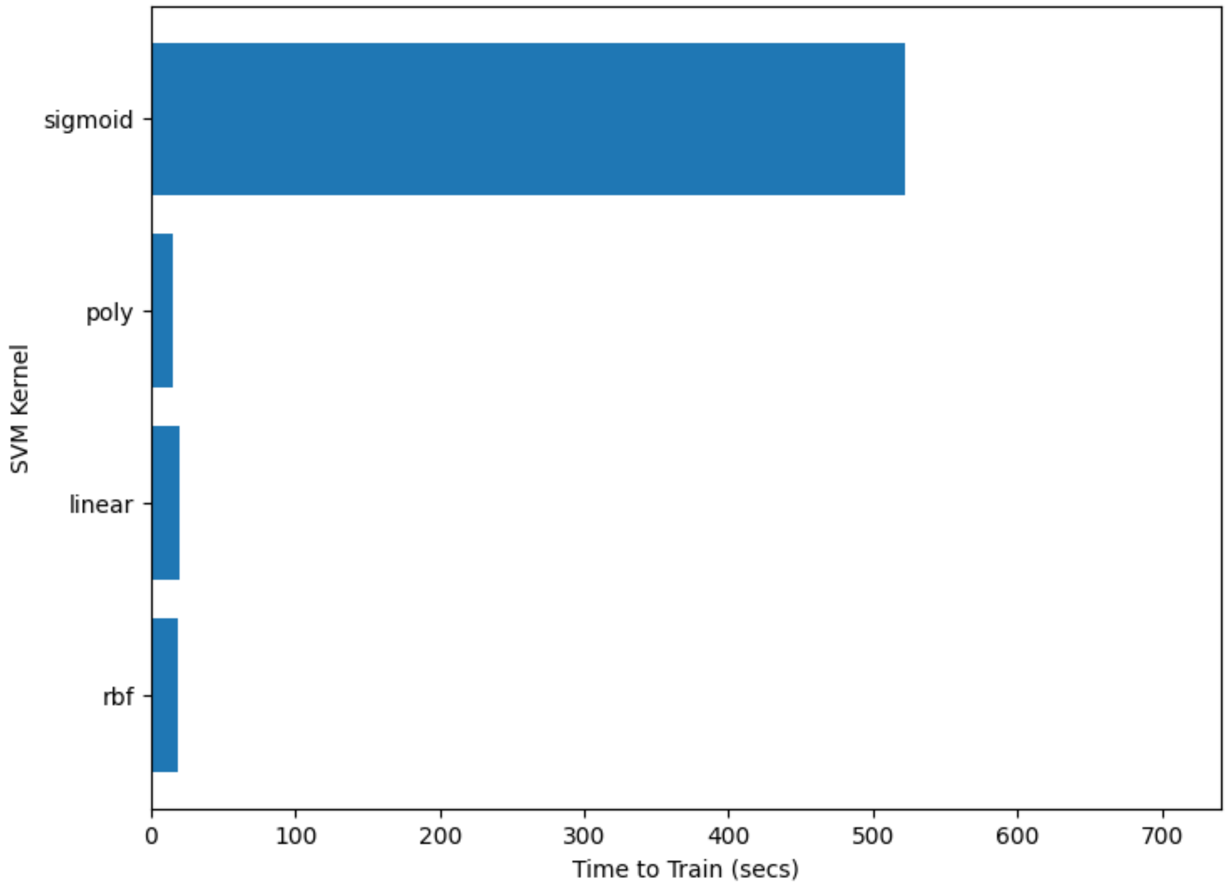
In [34]:
```
# Plot the scores (mean accuracy) for each SVM kernel type
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
kernels = ['rbf', 'linear', 'poly', 'sigmoid']
ax.barh(kernels,scores)

max_xlim = max(scores) + np.std(scores)
min_xlim = min(scores) - np.std(scores)
plt.xlim(min_xlim, max_xlim)
plt.xlabel('Mean Accuracy')
plt.ylabel('SVM Kernel')
plt.show()
```



In [56]:
```
# Generate train_times array
train_times = []
train_times.append(svm1_train_time)
train_times.append(svm2_train_time)
train_times.append(svm3_train_time)
train_times.append(svm4_train_time)
train_times
```

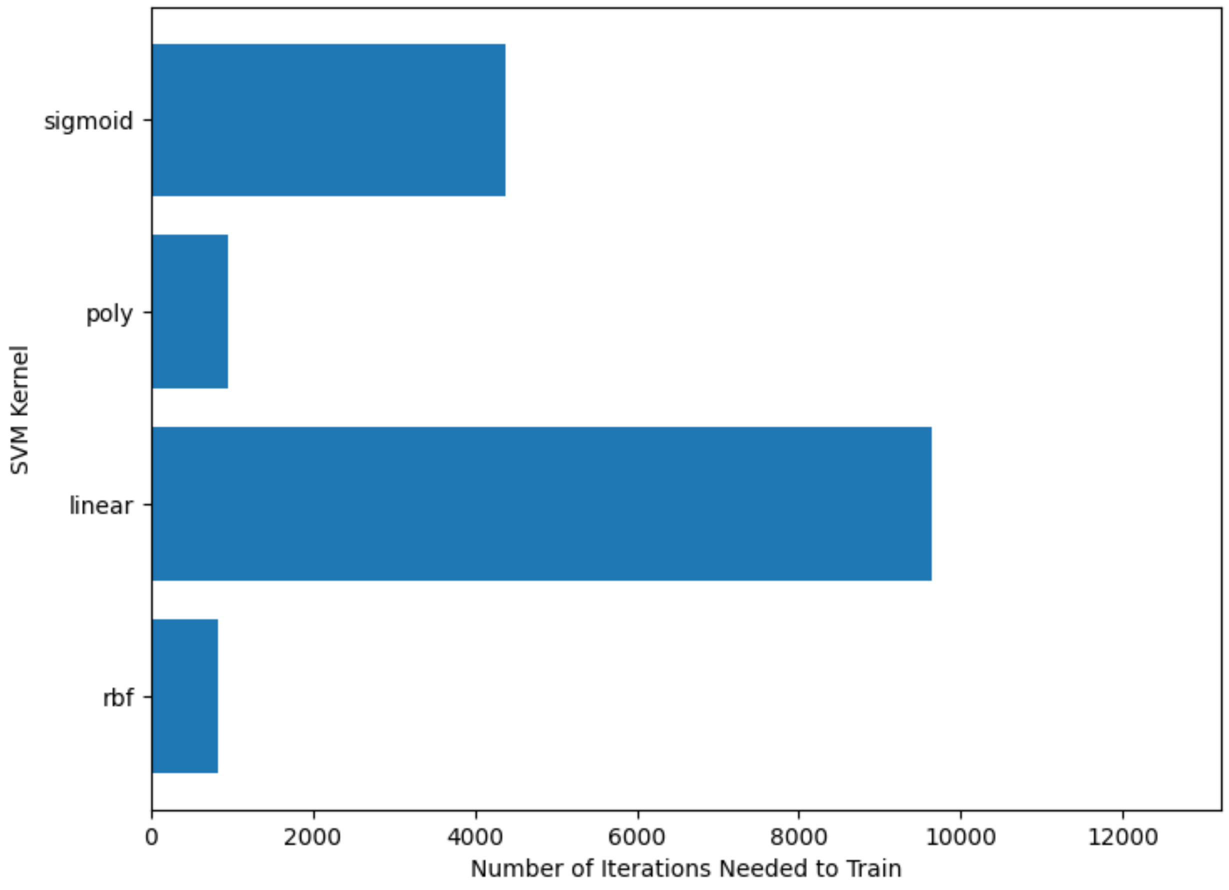
Out[56]:
```
[18.149382829666138, 19.83376717567444, 15.19580078125, 522.404981851
5778]
```

In [36]:
```
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
kernels = ['rbf', 'linear', 'poly', 'sigmoid']
ax.barh(kernels,train_times)

max_xlim = max(train_times) + np.std(train_times)
min_xlim = 0
plt.xlim(min_xlim, max_xlim)
plt.xlabel('Time to Train (secs)')
plt.ylabel('SVM Kernel')
plt.show()
```



In [37]:
```
# Generate number of iterations array
num_iter = []
num_iter.append(svm1.n_iter_[0])
num_iter.append(svm2.n_iter_[0])
num_iter.append(svm3.n_iter_[0])
num_iter.append(svm4.n_iter_[0])
```

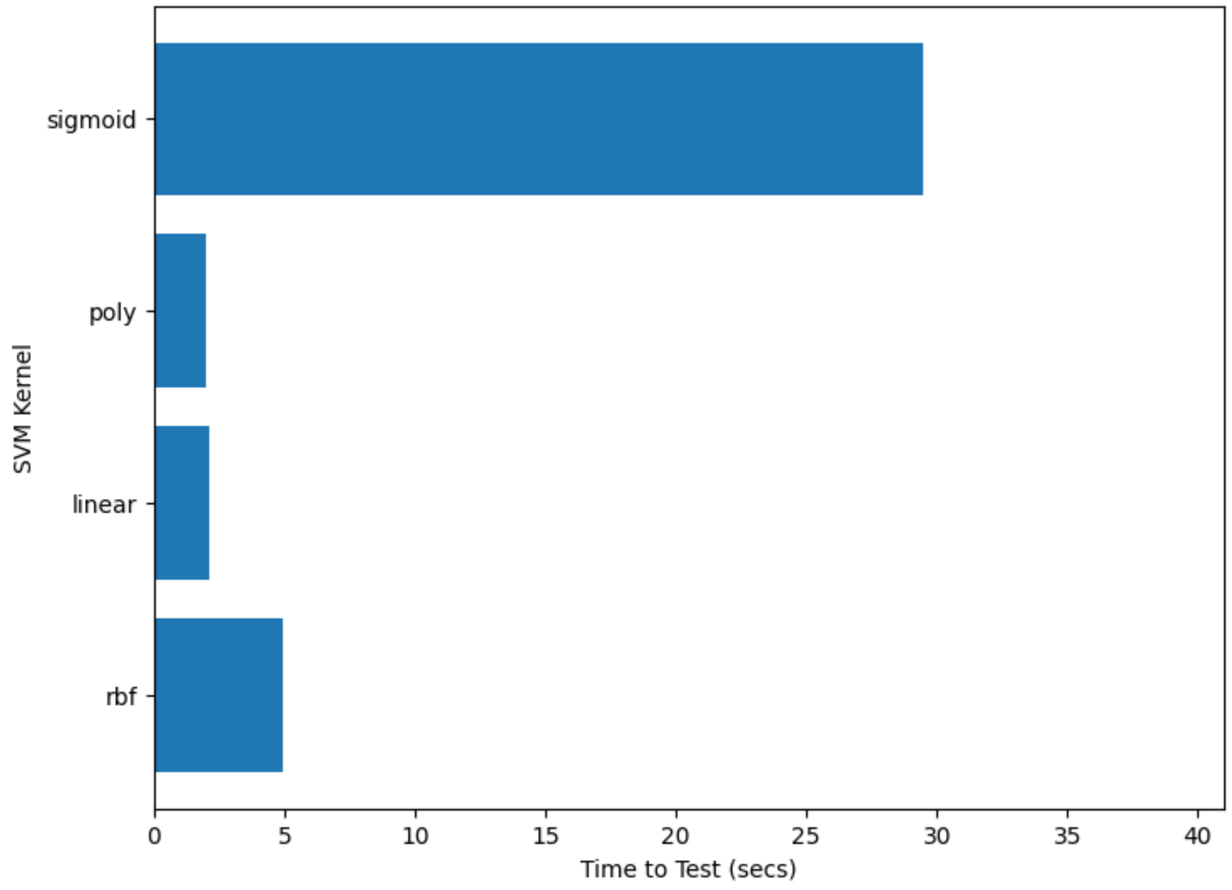
In [38]:
```
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
kernels = ['rbf', 'linear', 'poly', 'sigmoid']
ax.barh(kernels,num_iter)

max_xlim = max(num_iter) + np.std(num_iter)
min_xlim = 0
plt.xlim(min_xlim, max_xlim)
plt.xlabel('Number of Iterations Needed to Train')
plt.ylabel('SVM Kernel')
plt.show()
```



In [57]:
```
# Generate test_times array
test_times = []
test_times.append(svm1_test_time)
test_times.append(svm2_test_time)
test_times.append(svm3_test_time)
test_times.append(svm4_test_time)
test_times
```

Out[57]:
```
[4.932121753692627, 2.1427078247070312, 1.9960789680480957, 29.478855
13305664]
```

In [40]:
```
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
kernels = ['rbf', 'linear', 'poly', 'sigmoid']
ax.barh(kernels,test_times)

max_xlim = max(test_times) + np.std(test_times)
min_xlim = 0
plt.xlim(min_xlim, max_xlim)
plt.xlabel('Time to Test (secs)')
plt.ylabel('SVM Kernel')
plt.show()
```



Task 3: Select 2 Features. Train and test the Linear and RBF kernels against the 2 feature set, and visualize the decision bounary

Select Top 2 Features via Feature Selection

In [41]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2, f_classif, mutual_info_cla

print("selecting best via chi2:")
selector = SelectKBest(chi2, k=2).fit(X_scaled, y1)
print(selector.get_feature_names_out())
```

```
selecting best via chi2:
['count' 'service_ecr_i']
```

Find important features using LogisticRegression

In [42]:
```
# Train the LogisticRegression model with the training data.
# After trial and error, 300 is roughly the minimum number of iteratio
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(random_state=0, max_iter=5000)
print(lr.get_params())
lr.fit(X_train, y_train)
print(f"iterations required: {lr.n_iter_}")
```

```
{'C': 1.0, 'class_weight': None, 'dual': False, 'fit_intercept': True
, 'intercept_scaling': 1, 'l1_ratio': None, 'max_iter': 5000, 'multi_
class': 'auto', 'n_jobs': None, 'penalty': 'l2', 'random_state': 0, '
solver': 'lbfgs', 'tol': 0.0001, 'verbose': 0, 'warm_start': False}
iterations required: [122]
```

In [63]:
```
# Print number of each feature with its coeffiecient, in sorted order
numFeatures = lr.coef_[0].size
sorted_coef = np.empty(shape = (numFeatures,2), dtype=object)
for i in range(numFeatures):
    sorted_coef[i, 0] = i
    sorted_coef[i, 1] = lr.coef_[0][i]

sorted_coef = sorted_coef[sorted_coef[:,1].argsort()]

print("Most positively correlated features:")
for rank in range (121,111,-1):
    colnum, coef = sorted_coef[rank]
    print(f"{X_scaled.columns[colnum]} ({colnum}): {coef:6.2f}")

print()

print("Most negatively correlated features:")
for rank in range (0,10):
    colnum, coef = sorted_coef[rank]
    print(f"{X_scaled.columns[colnum]} ({colnum}): {coef:6.2f}")
```

```
Most positively correlated features:
wrong_fragment (4):   8.89
srv_count (20):   7.21
dst_host_serror_rate (34):   5.84
service_ecr_i (56):   5.84
hot (6):   4.39
dst_host_count (28):   4.33
count (19):   3.92
dst_host_rerror_rate (36):   3.91
protocol_type_icmp (38):   3.24
flag_S0 (116):   2.65

Most negatively correlated features:
diff_srv_rate (26):  -7.34
dst_host_diff_srv_rate (31):  -5.88
dst_host_same_src_port_rate (32):  -3.59
same_srv_rate (25):  -3.48
protocol_type_udp (40):  -2.96
service_smtp (95):  -2.32
service_urp_i (106):  -2.31
rerror_rate (23):  -1.73
service_other (85):  -1.71
service_domain_u (53):  -1.64
```

Train using 2 features

We decide to use "count" and "srv_count" for our features. "srv_count" has the second largest absolute value coefficient from our LogisticRegression model, and "count" is picked as one of the two most important features by the SelectKBest class. More importantly, those columns have a more even data spread than most columns, and so they give us a more interesting data visualization than the alternatives.

In [45]:
```
# Pull out the feature set and result set from the data
print(f"shape of data set: {df.shape}")
#for i in range(X_scaled.columns.size):
#    print(f"{i}: {X_scaled.columns[i]}")

# Include only count and srv_count in the new data set (X20)
#X20 = X.iloc[:, [23, 35]]
#X20 = X.iloc[:, [4, 5]]
#X20 = X_encoded.iloc[:, [1, 2]] # src_bytes, dst_bytes
X20 = X_scaled.iloc[:, [19, 20]] # count, srv_count

print(f"shape of X: {X20.shape}")
print(X20)
```

```
shape of data set: (4898430, 42)
shape of X: (400000, 2)
           count  srv_count
0       0.001957   0.001957
1       0.003914   0.003914
2       0.023483   0.054795
3       0.007828   0.007828
4       0.031311   0.031311
...          ...        ...
399995  1.000000   1.000000
399996  1.000000   1.000000
399997  1.000000   1.000000
399998  1.000000   1.000000
399999  0.270059   0.029354

[400000 rows x 2 columns]
```

In [46]:
```
# Reduce to only 50 positive and 50 negative records, to make visualiz
# Our new data set we call X21.
from collections import Counter
from imblearn.under_sampling import RandomUnderSampler

print('Original dataset shape %s' % Counter(y1))
rus = RandomUnderSampler(random_state=0, sampling_strategy={1:50, 0:50
X21, y21 = rus.fit_resample(X20, y1)
print('Resampled dataset shape %s' % Counter(y21))
print(X21.shape)
X21.describe()
```

```
Original dataset shape Counter({0: 200000, 1: 200000})
Resampled dataset shape Counter({0: 50, 1: 50})
(100, 2)
```

Out[46]:

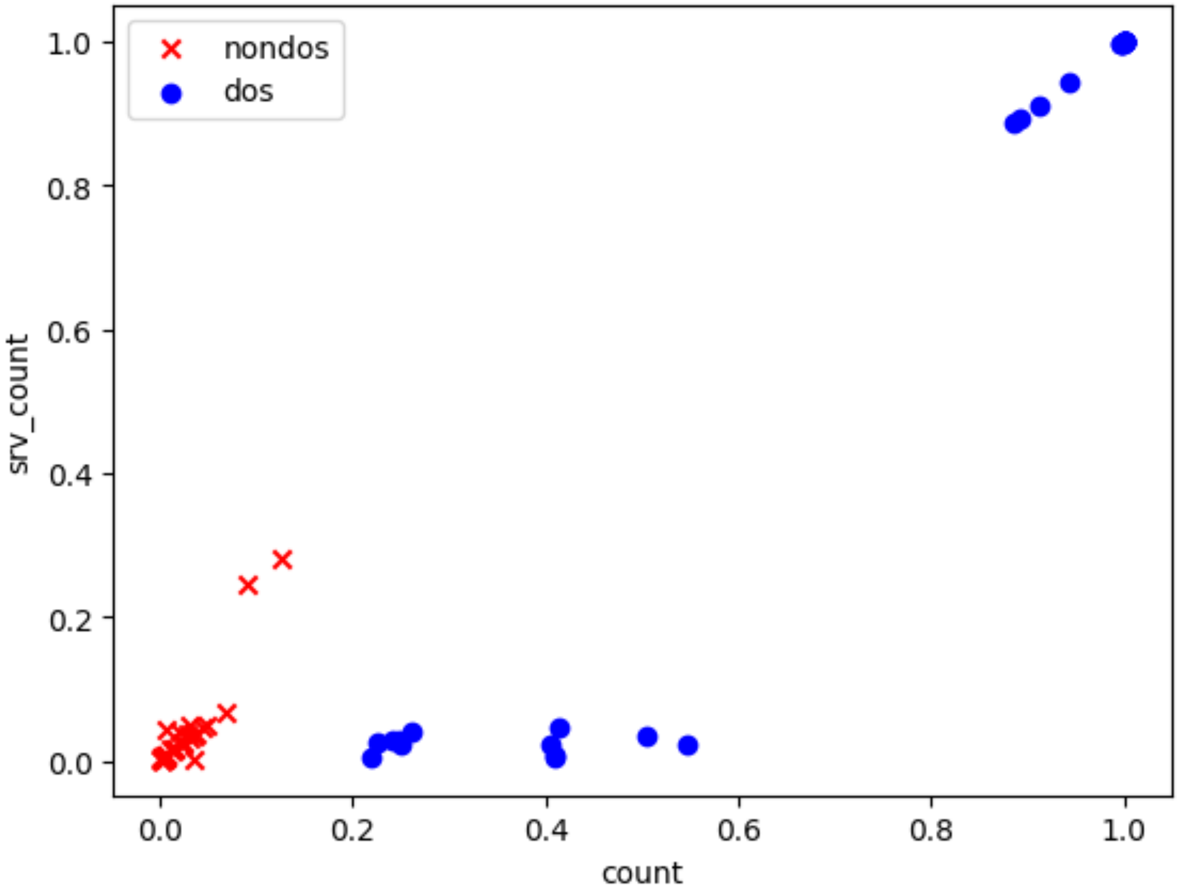
	count	srv_count
count	100.000000	100.000000
mean	0.425890	0.391213
std	0.457937	0.472982
min	0.001957	0.001957
25%	0.005382	0.005871
50%	0.172211	0.036204
75%	1.000000	1.000000
max	1.000000	1.000000

In [47]:
```
# Our plotting mechanisms want ndarray, so convert from Dataframe to n
X22 = X21.values
y22 = y21

print(X22.shape)
print(y22.shape)
```

```
(100, 2)
(100,)
```

In [48]:
```python
# Create arrays for plotting the data set
pos_feat1 = []
pos_feat2 = []
neg_feat1 = []
neg_feat2 = []

for i in range(y22.size):
    if y22[i] == 1:
        pos_feat1.append(X22[i][0])
        pos_feat2.append(X22[i][1])
    else:
        neg_feat1.append(X22[i][0])
        neg_feat2.append(X22[i][1])
```

In [49]:
```
# Plot the data set, using Red X's for non-DOS and Blue Circles for DO
%matplotlib inline
import matplotlib.pyplot as plt

# plot data
plt.scatter(neg_feat1, neg_feat2,
            color='red', marker='x', label='nondos')
plt.scatter(pos_feat1, pos_feat2,
            color='blue', marker='o', label='dos')

plt.xlabel(X21.columns[0])
plt.ylabel(X21.columns[1])
plt.legend(loc='upper left')

plt.show()
```



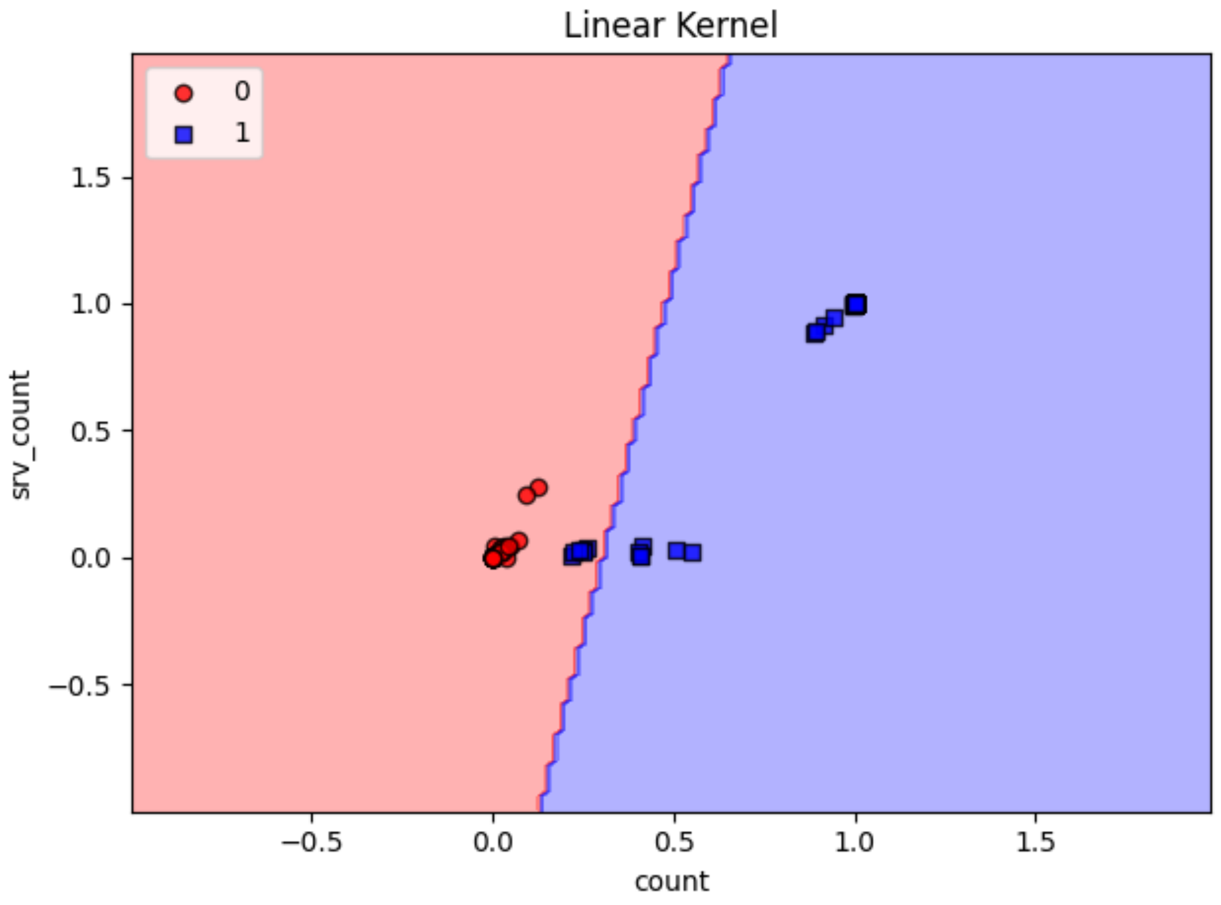
Train against linear kernel

In [50]:
```
from sklearn.svm import SVC

svm5 = SVC(kernel='linear', random_state=0)
print(svm5.get_params())
svm5.fit(X22, y22)

print(f"iterations required: {svm5.n_iter_}")
```

```
{'C': 1.0, 'break_ties': False, 'cache_size': 200, 'class_weight': No
ne, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree': 3, 'gam
ma': 'scale', 'kernel': 'linear', 'max_iter': -1, 'probability': Fals
e, 'random_state': 0, 'shrinking': True, 'tol': 0.001, 'verbose': Fal
se}
iterations required: [13]
```

In [51]:
```
# Define the plot_decision_regions function, which is copied from Week
from matplotlib.colors import ListedColormap

def plot_decision_regions(X, y, classifier, resolution=0.02):

    # setup marker generator and color map
    markers = ('o', 's', 'x', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                           np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    # plot class samples
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0],
                    y=X[y == cl, 1],
                    alpha=0.8,
                    c=colors[idx],
                    marker=markers[idx],
                    label=cl,
                    edgecolor='black')
```

In [52]:
```
# Plot the decision region generated by the Linear SVM kernel
plot_decision_regions(X22, y22, classifier=svm5)
plt.title('Linear Kernel')
plt.xlabel(X21.columns[0])
plt.ylabel(X21.columns[1])
plt.legend(loc='upper left')
plt.tight_layout()
plt.show()
```

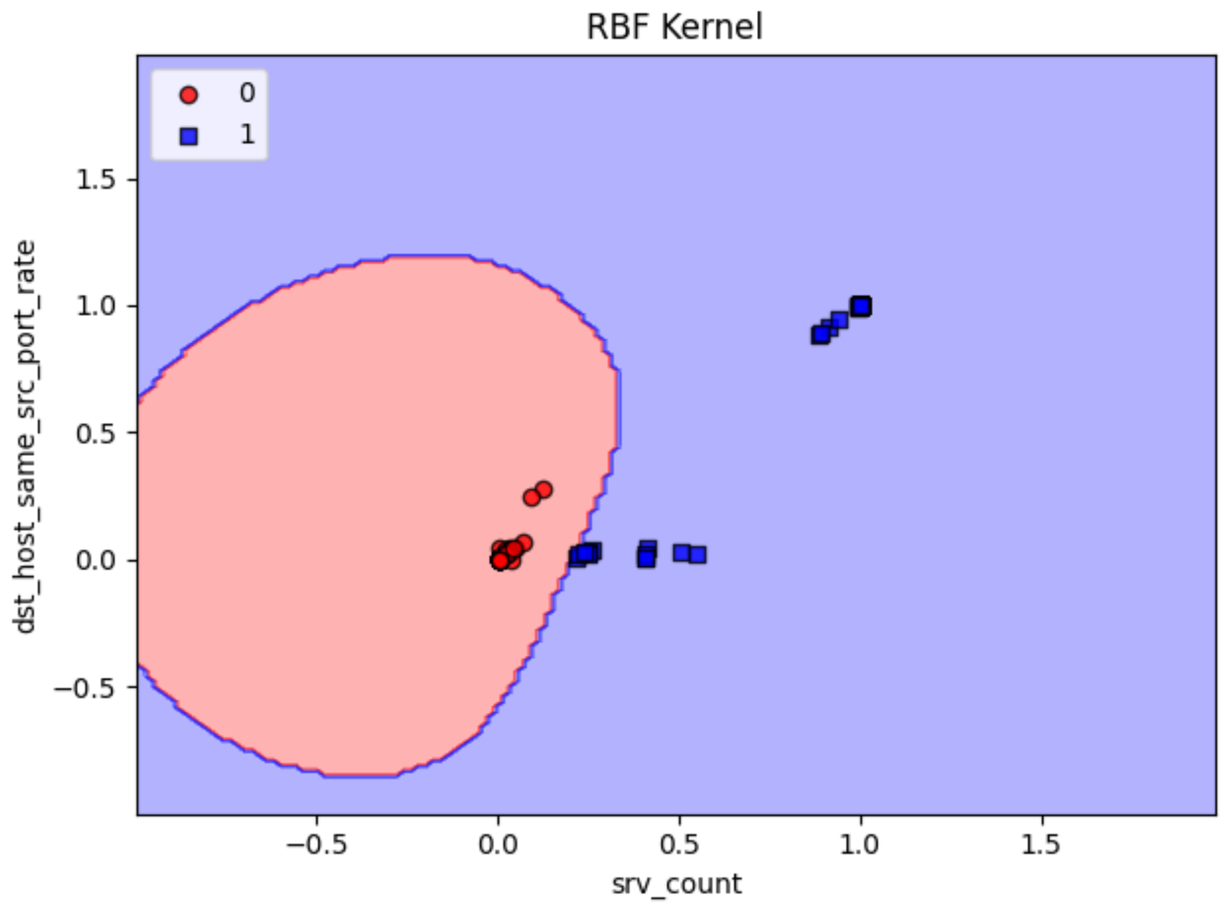


Train against RBF kernel

In [53]:
```
from sklearn.svm import SVC

svm6 = SVC(kernel='rbf', random_state=0)
print(svm6.get_params())
svm6.fit(X22, y22)

print(f"iterations required: {svm6.n_iter_}")
```

```
{'C': 1.0, 'break_ties': False, 'cache_size': 200, 'class_weight': No
ne, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree': 3, 'gam
ma': 'scale', 'kernel': 'rbf', 'max_iter': -1, 'probability': False,
'random_state': 0, 'shrinking': True, 'tol': 0.001, 'verbose': False}
iterations required: [14]
```

In [54]:
```
# Plot the decision region generated by the RBF SVM Kernel
plot_decision_regions(X22, y22, classifier=svm6)
plt.title('RBF Kernel')
plt.xlabel('srv_count')
plt.ylabel('dst_host_same_src_port_rate')
plt.legend(loc='upper left')
plt.tight_layout()
plt.show()
```



Discuss our observations

As indicated by the name, the linear kernel created a straight-line boundary for the decision regions. The decision boundary does not work very well for this very small, 2 feature data set. The RBF kernel, on the other hand, drew a shape for the decision boundary that looks like a slightly distorted circle. The boundary is very interesting and captures the data sets much better than the linear kernel did.

In [ ]:

```
```