NSL-KDD

1 Mounting Google Drive

[ ]:

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

2 Imports

[ ]:

```
import pandas as pd
import os
import numpy as np
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler
from sklearn.decomposition import PCA
import joblib
import matplotlib.pyplot as plt
```

3 Reading Data

[ ]:

```
DATA_DIR = '/content/drive/My Drive/Colab Notebooks/Intrusion Detection/data/
  ↪NSL-KDD'
```

3.1 Training Data

[ ]:

```
train_data = pd.read_csv(os.path.join(DATA_DIR, 'KDDTrain+.txt'), header=None)
```

[ ]:

```
train_data.head()
```

3.2 Testing Data

[ ]:
```
test_data = pd.read_csv(os.path.join(DATA_DIR, 'KDDTest+.txt'), header=None)
```

[ ]:
```
test_data.head()
```

4 Splitting Data Into Numeric and Nominal Features

4.1 Dropping Redundant Columns

[ ]:
```
def remove_redundant_attributes(train_data, test_data):
    drop_cols = []
    for i in range(41):
        if train_data.loc[:, i].min() == train_data.loc[:, i].max():
            drop_cols.append(i)

    train_data_dropped = train_data.drop(drop_cols, axis=1)
    test_data_dropped = test_data.drop(drop_cols, axis=1)

    return train_data_dropped, test_data_dropped, drop_cols
```

[ ]:
```
train_data_dropped, test_data_dropped, dropped_cols =␣
 ↪remove_redundant_attributes(
    train_data, test_data
)
```

[ ]:
```
dropped_cols
```

[ ]:
```
train_data_dropped.head()
```

[ ]:
```
test_data_dropped.head()
```

4.2 Numeric Features

[ ]:
```
numeric_features = np.asarray(
    pd.concat(
        [pd.DataFrame(train_data_dropped.loc[:,0]), train_data_dropped.loc[:, 4:
 ↪40]],
        axis=1
    )
)
```

4.2.1 MinMaxScaler

[ ]:

```
min_max_scaler_numeric = MinMaxScaler()
min_max_scaler_numeric.fit(numeric_features)
```

4.2.2 Extraction

[ ]:

```
def extract_numeric_features(data, min_max_scaler):
    numeric_features = np.asarray(
        pd.concat(
            [pd.DataFrame(data.loc[:,0]), data.loc[:, 4:40]],
            axis=1
        )
    )

    numeric_features_scaled = min_max_scaler.transform(numeric_features)
    numeric_features_final = numeric_features_scaled.astype('float64')

    return np.asarray(numeric_features_final)
```

[ ]:

```
numeric_features_train = extract_numeric_features(train_data_dropped,␣
 ↪min_max_scaler_numeric)
```

[ ]:

```
numeric_features_train.shape
```

[ ]:

```
numeric_features_test = extract_numeric_features(test_data_dropped,␣
 ↪min_max_scaler_numeric)
```

[ ]:

```
numeric_features_test.shape
```

4.3 Nominal Features

[ ]:

```
nominal_features = np.asarray(train_data.loc[:, 1:3])
```

4.3.1 One Hot Encoder

[ ]:

```
one_hot_encoder = OneHotEncoder(sparse=False)
one_hot_encoder.fit(nominal_features)
```

[ ]:

```
list(map(len, one_hot_encoder.categories_))
```

4.3.2 Extraction

[ ]:
```
def extract_nominal_features(data, one_hot_encoder):
    nominal_features = np.asarray(data.loc[:, 1:3])

    nominal_features_one_hot = one_hot_encoder.transform(nominal_features)
    nominal_features_final = nominal_features_one_hot.astype('float64')

    return nominal_features_final
```

[ ]:
```
nominal_features_train = extract_nominal_features(train_data, one_hot_encoder)
```

[ ]:
```
nominal_features_train.shape
```

[ ]:
```
nominal_features_test = extract_nominal_features(test_data, one_hot_encoder)
```

[ ]:
```
nominal_features_test.shape
```

5 Final Features

5.1 Training

[ ]:
```
final_features_train = np.concatenate([numeric_features_train,␣
 ↪nominal_features_train], axis=1)
```

[ ]:
```
final_features_train.shape
```

5.2 Testing

[ ]:
```
final_features_test = np.concatenate([numeric_features_test,␣
 ↪nominal_features_test], axis=1)
```

[ ]:
```
final_features_test.shape
```

5.3 PCA

[ ]:
```
pca = PCA()
pca.fit(final_features_train)
```

[ ]:
```
def get_components(pca, threshold):
    if threshold >= 1:
        threshold /= 100
    ratio_sum = 0
    i = 0
    for ratio in pca.explained_variance_ratio_:
        i += 1
        ratio_sum += ratio
        if ratio_sum >= threshold:
            return i, ratio_sum

    return None, None
```

[ ]:
```
get_components(pca, threshold=0.99)
```

[ ]:
```
variance_ratios = [0]
curr_sum = 0
for ratio in pca.explained_variance_ratio_:
    curr_sum += ratio
    variance_ratios.append(curr_sum)

len(variance_ratios)
```

[ ]:
```
np.linspace(10, 120, 12, dtype='int64') - 1
```

[ ]:
```
plt.figure(figsize=(8,8))
plt.plot(
    np.arange(122),
    variance_ratios,
    'k--',
    markevery=20,
    marker='D'
)

plt.grid()

plt.xlabel('Retained Components')
plt.ylabel('Retained Variance Ratio')

plt.savefig('Principal Component Analysis.png')
```

5.3.1 67 Components

[ ]:
```
pca_67dims = PCA(n_components=67)
pca_67dims.fit(final_features_train)
```

[ ]:

```
train_features_pca = pca_67dims.transform(final_features_train)
test_features_pca = pca_67dims.transform(final_features_test)
```

5.4 MinMaxScaling

[ ]:

```
min_max_scaler_pca = MinMaxScaler()
min_max_scaler_pca.fit(train_features_pca)
```

[ ]:

```
train_features_scaled = min_max_scaler_pca.transform(train_features_pca)
test_features_scaled = min_max_scaler_pca.transform(test_features_pca)
```

[ ]:

```
train_features_scaled.shape
```

[ ]:

```
test_features_scaled.shape
```

6 Processing Labels

[ ]:

```
def combine_classes(labels, combine_dict):
    labels_multiclass = np.zeros((labels.shape[0], 5))
    for i in range(labels.shape[0]):
        labels_multiclass[i, combine_dict[labels[i]]] = 1

    labels_binary = np.zeros((labels.shape[0], 2))
    for i in range(labels.shape[0]):
        if labels[i] == 'normal':
            labels_binary[i, 0] = 1
        else:
            labels_binary[i, 1] = 1

    return pd.DataFrame(labels_multiclass).astype('int64'), pd.
 ↪DataFrame(labels_binary).astype('int64')
```

[ ]:

```
def extract_labels(data):
    labels = np.asarray(data.loc[:, 41])
    combine_dict = {
        'normal': 0,
        'neptune': 1,
        'warezclient': 3,
        'ipsweep': 2,
        'portsweep': 2,
        'teardrop': 1,
        'nmap': 2,
        'satan': 2,
        'smurf': 1,
        'pod': 1,
        'back': 1,
        'guess_passwd': 3,
        'ftp_write': 3,
        'multihop': 3,
        'rootkit': 4,
        'buffer_overflow': 4,
        'imap': 3,
        'warezmaster': 3,
        'phf': 3,
        'land': 1,
        'loadmodule': 4,
        'spy': 3,
        'perl': 4,
        'saint': 2,
        'mscan': 2,
        'apache2': 1,
        'snmpgetattack': 3,
        'processtable': 1,
        'httptunnel': 3,
        'ps': 4,
        'snmpguess': 3,
        'named': 3,
        'sendmail': 3,
        'xterm': 3,
        'worm': 1,
        'xlock': 3,
        'xsnoop': 3,
        'sqlattack': 4,
        'udpstorm': 1,
        'mailbomb': 3
    }

    final_labels, final_binary_labels = combine_classes(labels, combine_dict)

    return final_labels, final_binary_labels
```

[ ]:

```
train_labels_multiclass, train_labels_binary = extract_labels(train_data)
```

[ ]:

```
train_labels_multiclass.shape, train_labels_binary.shape
```

[ ]:

```
test_labels_multiclass, test_labels_binary = extract_labels(test_data)
```

[ ]:

```
test_labels_multiclass.shape, test_labels_binary.shape
```

7 Saving Final CSVs

```
[ ]: def save_csv(data, name):
         if isinstance(data, np.ndarray):
             data = pd.DataFrame(data=data[0:, 0:])

         data.to_csv(name, header=False, index=False)
```

7.1 Training

```
[ ]: save_csv(train_features_scaled, 'train_features.csv')
```

```
[ ]: save_csv(train_labels_multiclass, 'train_labels_multiclass.csv')
```

```
[ ]: save_csv(train_labels_binary, 'train_labels_binary.csv')
```

7.2 Testing

```
[ ]: save_csv(test_features_scaled, 'test_features.csv')
```

```
[ ]: save_csv(test_labels_multiclass, 'test_labels_multiclass.csv')
```

```
[ ]: save_csv(test_labels_binary, 'test_labels_binary.csv')
```

8 Saving Preprocessing Models

```
[ ]: joblib.dump(min_max_scaler_numeric, 'min_max_scaler_numeric.joblib')
```

```
[ ]: joblib.dump(min_max_scaler_pca, 'min_max_scaler_pca.joblib')
```

```
[ ]: joblib.dump(one_hot_encoder, 'one_hot_encoder.joblib')
```

```
[ ]: joblib.dump(pca_67dims, 'pca_67dims.joblib')
```

```
[ ]:
```