# Assignment 2 - Naive Bayes and Logistic Regression

## Dataset: Network Attack - NB15

The raw network packets of the **UNSW-NB 15 dataset** was created by the IXIA PerfectStorm tool in the Cyber Range Lab of the Australian Centre for Cyber Security (ACCS) for generating a hybrid of real modern normal activities and synthetic contemporary attack behaviours.

Tcpdump tool is used to capture the raw traffic (e.g., Pcap files). This dataset has nine types of attacks, namely, Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode and Worms. The Argus, Bro-IDS tools are used and twelve algorithms are developed to generate totally 49 features with the class label.

- Dataset Link
- Description of all the 49 features

### Install required libraries

```
1
```

```
1 import pandas
2 import numpy
```

### Q1. EDA and basic data pre-processing and preparation (4*0.5 = 2M)

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

```
Mounted at /content/drive
```

```
1 dataset = pandas.read_csv("/content/drive/MyDrive/bits-pillani/PG_AIML/Cassification/assignement2/Network_attacks_UNSW_N
```

```
1 dataset["attack_cat"].unique()
```

```
array(['Normal', 'Backdoor', 'Analysis', 'Fuzzers', 'Shellcode',
       'Reconnaissance', 'Exploits', 'DoS', 'Worms', 'Generic'],
      dtype=object)
```

### a. Null/Outlier treatment

```
1 clean_dataset = dataset.dropna()
2
```

### b. Remove non-important features

```
1 from sklearn.feature_selection import SelectKBest, mutual_info_classif
```

```
1 clean_dataset = clean_dataset.drop(columns = ["id", "state"])
```

### c. Split training and testing data set

```
1 from sklearn.model_selection import train_test_split
2 X = clean_dataset.drop(columns = ["attack_cat"])
3 y = clean_dataset[["attack_cat"]]
4
5 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
6
7
```

### d. Standardise/Normalise the variables whenever required

```
from sklearn.preprocessing import StandardScaler

def std_numeric_columns(df_) :
    df = df_.copy()
    numeric_columns = df.select_dtypes(include=['int64', 'float64']).columns

    scaler = StandardScaler()
    df[numeric_columns] = scaler.fit_transform(df[numeric_columns])
    return df, scaler

def std_numeric_cols_test(scaler, df_):
    numeric_columns = df_.select_dtypes(include=['int64', 'float64']).columns
    df = df_
    df[numeric_columns] = std_scaler.transform(df[numeric_columns])
    return df
```

```
def concat(df1_, df2):
    df1 = df1_.reset_index().drop(columns= ["index"])
    for col in  df2.columns:
        df1[col] = df2[col]
    return df1
```

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

def label_encoder(df_, column):
    df = df_.copy()
    label_encoder = LabelEncoder()
    df[f"{column}"] = label_encoder.fit_transform(df[column])
    # df = df.drop(columns = [column])
    return df, label_encoder

def encode_text_to_onehot(df_, column):
    df = df_.copy()
    hot_encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
    onehot_encoded = hot_encoder.fit_transform(df[[column]])
    onehot_encoded_df = pandas.DataFrame(onehot_encoded, columns=hot_encoder.get_feature_names_out([column]))
    df = concat(df, onehot_encoded_df)
    df = df.drop(columns = [column])
    return df, hot_encoder, onehot_encoded_df

def test_encode_text_to_onehot(df_, column, hot_encoder):
    df = df_
    onehot_encoded = hot_encoder.transform(df[[column]])
    onehot_encoded_df = pandas.DataFrame(onehot_encoded, columns=hot_encoder.get_feature_names_out([column]))
    df = concat(df, onehot_encoded_df)
    df = df.drop(columns = [column])
    return df, onehot_encoded_df

enc_X_test =  X_test
enc_X_train = X_train

# enc_y_train, target_label_encoder = encode_text_to_onehot(y_train, "attack_cat")
# enc_y_test = target_label_encoder.transfrorm(y_test)
# enc_y_train

# enc_X_train, proto_hot_encoder, tys = encode_text_to_onehot(X_train, "proto")
# enc_X_test["proto"] = proto_hot_encoder.transform(enc_X_test[["proto"]])
# enc_X_test, _ = test_encode_text_to_onehot(X_test, "proto", proto_hot_encoder)
enc_X_train, proto_label_encoder = label_encoder(enc_X_train, "proto")
enc_X_test["proto"] = proto_label_encoder.transform(enc_X_test["proto"])

# enc_X_train, service_hot_encoder, tys = encode_text_to_onehot(enc_X_train, "service")
# enc_X_test["service"] = service_hot_encoder.transform(enc_X_test[["service"]])
# enc_X_test, _ = test_encode_text_to_onehot(enc_X_test, "service", service_hot_encoder)
enc_X_train, service_label_encoder = label_encoder(enc_X_train, "service")
enc_X_test["service"] = service_label_encoder.transform(enc_X_test["service"])


# enc_X_train, state_hot_encoder = encode_text_to_onehot(enc_X_train, "state")
# enc_X_test["state"] = state_hot_encoder.transform(enc_X_test[["state"]])
```

```
enc_X_train, std_scaler = std_numeric_columns(enc_X_train)
enc_X_test = std_numeric_cols_test(std_scaler, enc_X_test)
```

```
enc_X_train.dropna()
```

| | dur | proto | service | spkts | dpkts | sbytes | dbytes | rate | sttl | dttl | ... | ct_dst_ltm | ct_s |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **96203** | -0.129900 | 0.153088 | 1.464739 | -0.076656 | -0.100649 | -0.045975 | -0.097123 | -0.577575 | -1.140489 | 1.560782 | ... | -0.395935 | |
| **58960** | -0.209992 | 0.465716 | -0.701255 | -0.137573 | -0.174221 | -0.049529 | -0.105296 | 0.097298 | 0.724029 | -0.720255 | ... | -0.395935 | |
| **65069** | -0.067947 | 0.153088 | -0.701255 | -0.015740 | -0.063864 | -0.014404 | -0.100511 | -0.577583 | 0.724029 | 1.560782 | ... | -0.644264 | |
| **64133** | -0.054916 | 0.153088 | 1.464739 | -0.076656 | -0.082256 | -0.046166 | -0.075515 | -0.577660 | -1.140489 | 1.560782 | ... | -0.644264 | |
| **111445** | -0.209992 | -2.124633 | -0.701255 | -0.137573 | -0.174221 | -0.049529 | -0.105296 | 0.637355 | 0.724029 | -0.720255 | ... | -0.395935 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **119879** | -0.209992 | 0.421055 | 0.165143 | -0.137573 | -0.174221 | -0.050044 | -0.105296 | 0.637355 | 0.724029 | -0.720255 | ... | 3.204835 | |
| **103694** | 0.477718 | 0.153088 | 0.598341 | 0.029947 | 0.028101 | -0.043115 | -0.093764 | -0.577713 | -1.140489 | 1.560782 | ... | -0.520100 | |
| **131932** | -0.209992 | 0.421055 | 0.165143 | -0.137573 | -0.174221 | -0.050044 | -0.105296 | 0.181682 | 0.724029 | -0.720255 | ... | 1.218203 | |
| **146867** | -0.209992 | 0.421055 | 0.165143 | -0.137573 | -0.174221 | -0.050044 | -0.105296 | 0.941138 | 0.724029 | -0.720255 | ... | 3.701493 | |
| **121958** | -0.209992 | 0.421055 | 0.165143 | -0.137573 | -0.174221 | -0.050044 | -0.105296 | 0.097298 | 0.724029 | -0.720255 | ... | 1.218203 | |

140272 rows × 41 columns

## 2. Train a classification model to classify the 9 different attacks (2 + 2 = 4M)

(Use scikit-learn library)

+ Code + Text

### a. Naive-Bayes

```
from sklearn.naive_bayes import GaussianNB

def train_NB(X_train, y_train):
    model = GaussianNB()
    model.fit(X_train, y_train)
    return model
```

```
NB_model = train_NB(enc_X_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was p
  y = column_or_1d(y, warn=True)
```

### b. Logistic Regression

```
from sklearn.linear_model import LogisticRegression

def train_LR(X_train, y_train):
    model = LogisticRegression()
    model.fit(X_train, y_train)
    return model
```

```

LR_model = train_LR(enc_X_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was p
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to conve
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

## 3. Compute accuracy and F1 scores for both types of models

[1+1 = 2M]

```

```

```
from sklearn.metrics import accuracy_score, f1_score

def compute_accuarcy_and_f1_score_for_model(model, X_test):
    test_pred = model.predict(X_test)
    a_score = accuracy_score(y_test, test_pred)
    f_score = f1_score(y_test, test_pred, average='weighted')
    return a_score, f_score

```

```
nb_accuracy_score, nb_f1_score = compute_accuarcy_and_f1_score_for_model(NB_model, enc_X_test)
lr_accuracy_score, lr_f1_score = compute_accuarcy_and_f1_score_for_model(LR_model, enc_X_test)
```

```
print(f"Naive Bayes \nAccuracy Score : {nb_accuracy_score}\nF1 Score : {nb_f1_score}\n\n")
print(f"Logistic Regression \nAccuracy Score : {lr_accuracy_score}\nF1 Score : {lr_f1_score}\n\n")
```

```
Naive Bayes
Accuracy Score : 0.4052011748267701
F1 Score : 0.4603245520552212


Logistic Regression
Accuracy Score : 0.7607858792665887
F1 Score : 0.7418402246262925
```

## 4. Apply feature reduction techniques

[1M]

```
from sklearn.decomposition import PCA
# Performing PCA
pca = PCA(n_components=30) # 10 components
pca_X_train = pca.fit_transform(enc_X_train)
pca_X_test = pca.transform(enc_X_test)

```

```

```

## 5. Train model on the reduced feature subset using NB and Logistic regression

[1+1 = 2M]

### a. Naive Bayes

```
pca_nb_model = train_NB(pca_X_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was p
  y = column_or_1d(y, warn=True)
```

### b. Logistic Regression

```
pca_lr_model = train_LR(pca_X_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was p
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to conve
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
pca_nb_accuracy_score, pca_nb_f1_score = compute_accuarcy_and_f1_score_for_model(pca_nb_model, pca_X_test)
pca_lr_accuracy_score, pca_lr_f1_score = compute_accuarcy_and_f1_score_for_model(pca_lr_model, pca_X_test)
```

```

```

```
print(f"Naive Bayes \nAccuracy Score : {pca_nb_accuracy_score}\nF1 Score : {pca_nb_f1_score}\n\n")
print(f"Logistic Regression \nAccuracy Score : {pca_lr_accuracy_score}\nF1 Score : {pca_lr_f1_score}\n\n")
```

```
Naive Bayes
Accuracy Score : 0.5066297869913599
F1 Score : 0.5691762439511556


Logistic Regression
Accuracy Score : 0.7500641592289486
F1 Score : 0.7296824554941133
```

## 6. Comment on performance of these approaches (NB and Logistic Regression) on reduced dataset (after applying PCA/feature reduction)

[1M]

Read the dataset and removed the null valuesm non row with NA values. Removed the state column. Splited the data in train and test.

Trianed and tested the data on both Naive Bayes and Logistic regression, Logistic Regression performed very well compare to NB.

After perfoming PCA, to 30 features there was slight increase in performance of both models, this might due to PCA compacting the importat features .

```

```