Lab 2a: Network Traffic Analysis with Unsupervised ML

By: Malachi Jones, PhD

OUTLINE

- Objectives
- Lab 2a.1: Approximate Agglomerative Clustering KDD
- References

LAB 2A OBJECTIVES

Lab 2a Objectives

- After this lab, students should be able to
  - Understand the process of pre-processing raw data (e.g. network traffic) such that it is suitable for ingesting into an unsupervised ML algorithm
  - Utilize libraries and packages such as numpy and panda to facilitate the pre-processing of the data

LAB 2A OVERVIEW

Lab 2a Overview

- This lab will consist of the following sections:
  - Lab 2a.1: Approximate Agglomerative Clustering of Network Traffic Data (i.e. KDD)

LAB 2A.1: APPROXIMATE AGGLOMERATIVE CLUSTERING OF KDD

Lab 2a.1 Approx. Agglom. Clustering of KDD

- Lab 2a.1 Objectives
  1. Remove non-continuous features from the DataFrame of the KDD Dataset

Data Frame

```python
def load_dataset_from_file(self):

    if self._is_data_set_loaded:
        return

    train_df = pd.read_csv(self._kdd_training_file_path, names=header_names)

    # Aggregates the attack types into categories
    train_df['attack_category'] = train_df['attack_type'].map(lambda x: attack_mapping[x])

    self._train_Y = train_df['attack_category']

    # Remove the attack type labels
    train_x_raw = train_df.drop(['attack_type', 'attack_category'], axis=1)

    # Remove the features from set that are not continuous
    raise Exception("Need to create a training data set with only continuous features")
    train_x_continuous = None # Training data set with only continuous features


    # Standardize the data
    standard_scaler = StandardScaler().fit(train_x_continuous)
    self._train_x_continuous_std = standard_scaler.transform(train_x_continuous)

    self._pca = PCA(n_components=2)
    self._train_x_pca = self._pca.fit_transform(self._train_x_continuous_std)


    self._cluster_node_list = [ClusterNode(data_point) for data_point in self._train_x_continuous_std]
```

Create DataFrame with only continuous features

Lab 2a.1 Implementing Adaline w/ SGD

- Lab 2a.1 Objectives
  2. Implement a Euclidean Distance for the KDD Dataset

```
class ClusterNode(object):

    def __init__(self, data_point, cluster_id=None, nearest_prototype_cluster_node=None):...

    @staticmethod
    def distance(node_a, node_b):

        node_a_data_point = node_a.data_point
        node_b_data_point = node_b.data_point

        raise Exception("@todo: Implement a Euclidean distance function for kdd data set")

    @staticmethod
    def similarity(node_a, no    b):...

    @
    d

    @assigned_cluster.setter
    def assigned_cluster(self, value):...

    @property
    def cluster_id(self):...

    @property
    def is_prototype(self):...

    @property
    def data_point(self):
        return self._data_point
```

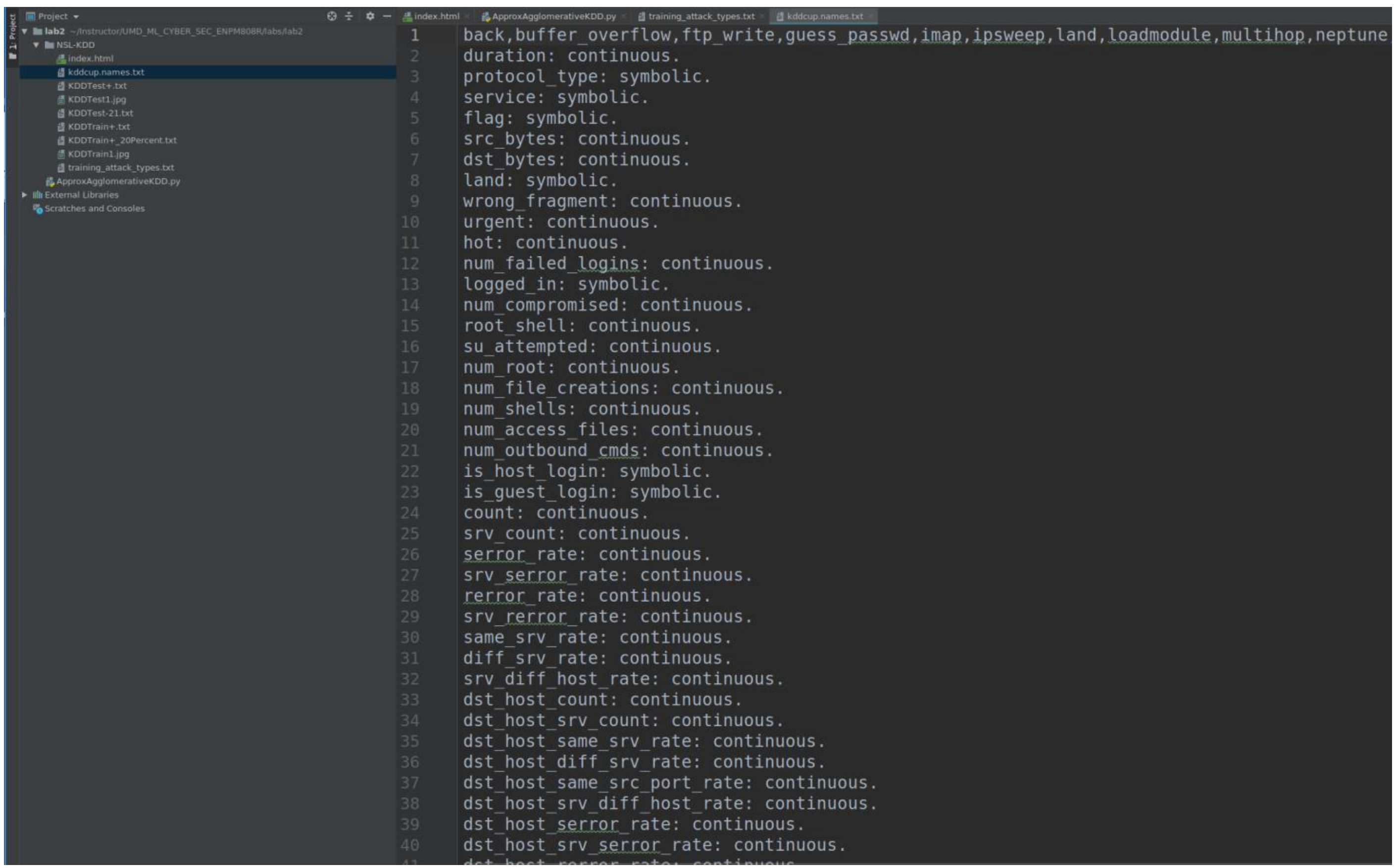
Implement Distance Function

Lab 2a.1 Implementing Adaline w/ SGD

- The relevant files for Lab 2a.1 (included in folder Lab2) consists of the following:
  - ApproxAgglomerativeKDD.py
  - KDDTrain+_20Percent.txt
  - Kddcup.names.txt
  - Training_Attack_types.txt

Lab 2a.1 Approx. Agglom. Clustering of KDD

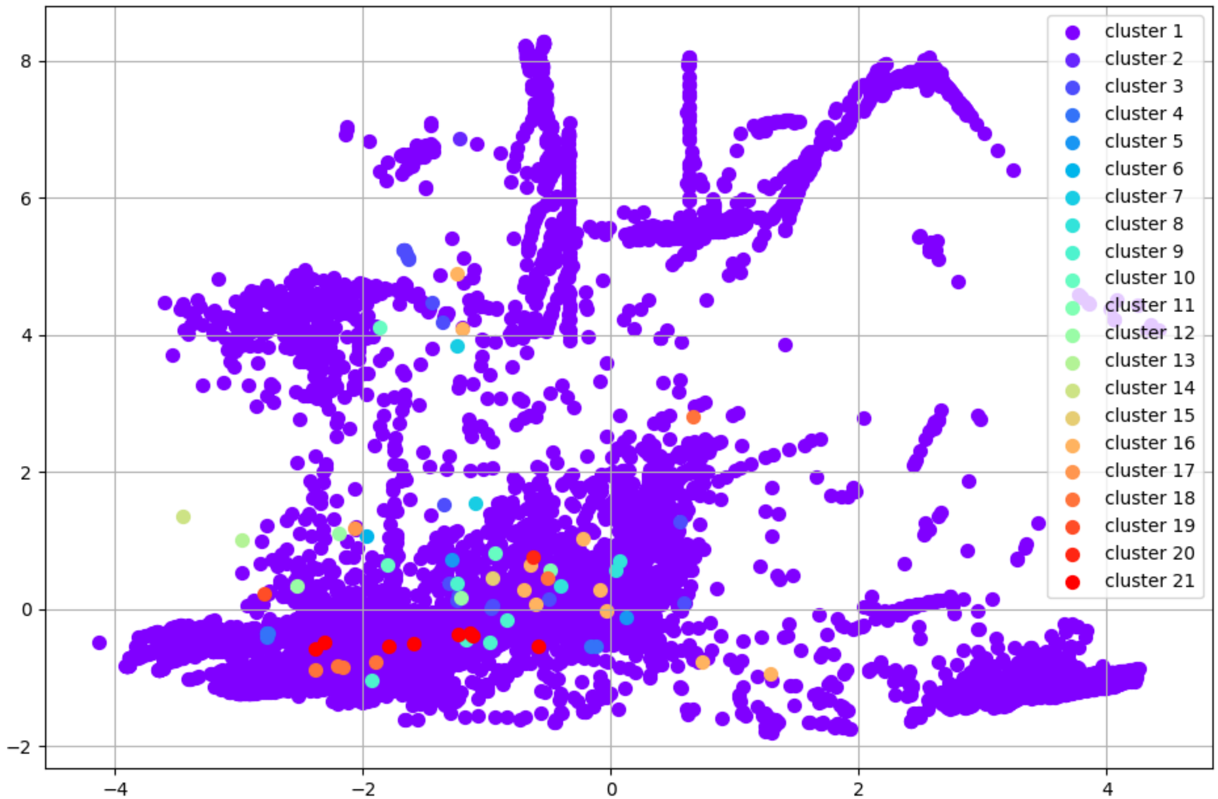
- **Hint #1**
  - `Kddcup.names.txt` contains each feature along with its respective type



```
back,buffer_overflow,ftp_write,guess_passwd,imap,ipsweep,land,loadmodule,multihop,neptune
duration: continuous.
protocol_type: symbolic.
service: symbolic.
flag: symbolic.
src_bytes: continuous.
dst_bytes: continuous.
land: symbolic.
wrong_fragment: continuous.
urgent: continuous.
hot: continuous.
num_failed_logins: continuous.
logged_in: symbolic.
num_compromised: continuous.
root_shell: continuous.
su_attempted: continuous.
num_root: continuous.
num_file_creations: continuous.
num_shells: continuous.
num_access_files: continuous.
num_outbound_cmds: continuous.
is_host_login: symbolic.
is_guest_login: symbolic.
count: continuous.
srv_count: continuous.
serror_rate: continuous.
srv_serror_rate: continuous.
rerror_rate: continuous.
srv_rerror_rate: continuous.
same_srv_rate: continuous.
diff_srv_rate: continuous.
srv_diff_host_rate: continuous.
dst_host_count: continuous.
dst_host_srv_count: continuous.
dst_host_same_srv_rate: continuous.
dst_host_diff_srv_rate: continuous.
dst_host_same_src_port_rate: continuous.
dst_host_srv_diff_host_rate: continuous.
dst_host_serror_rate: continuous.
dst_host_srv_serror_rate: continuous.
```

Lab 2a.1 Approx. Agglom. Clustering of KDD

- **Hint #2**
  - The resulting plot of the cluster should look similar to the following:



Lab 2a.1 Approx. Agglom. Clustering of KDD

- **Hint #3**
  - The below illustrates how the attack_type and attack_category features can be dropped:

```
# Remove the attack type labels
train_x_raw = train_df.drop(['attack_type', 'attack_category'], axis=1)
```

Lab 2a.1 Approx. Agglom. Clustering of KDD

- **Submission**
  - You will submit a folder called lab_2a with the following contents
    i. ApproxAgglomerativeClusteringKDD.py, which has your modifications that satisfy the lab 2a.1 objectives

  - **Note:** *Please do not submit any additional artifacts as they will not be evaluated*

References

1. Rieck, K., Trinius, P., Willems, C., & Holz, T. (2011). Automatic analysis of malware behavior using machine learning. *Journal of Computer Security*, *19*(4), 639-668.