EE 418 CRYPTOGRAPHY AND NETWORK SECURITY

Project 2: Cyber Threat Detection Using Machine Learning

Mike Warren, Hugo Baldner
0323799 , 1471121

Task 1: Implement the Python Notebook and Read the Texts:

Mike did the code. One minor edit Mike did was in task 7, He edited some of the provided code to shrink the legend and put it in the bottom left to make the plot more readable (the legend was going over the data before making the change). In addition, one of the plots provided separately as a .png has edited resolution and background transparency, as the values we were told to implement didn't make for the easiest visualization. Meanwhile Hugo did the reading and answering of the text questions and the compilation of our work. While all this code will be turned in separately in its whole here we will place the snippets that were made by our group:

In [11]:
```
data = dict()
for label in data_labels:

    ###################################################################
    ###### Start code here ######
    data[label] = np.reshape(imported_data[label], (len(imported_data[label]),
 len(imported_data[label][0])))
    ###### End code here ######
    ###################################################################

    print('{}: {}'.format(label, data[label].shape))
```

In [16]:
```
vali_x = np.array([], dtype=float).reshape(0, 33292)  # num_features = 33292
vali_y = np.array([], dtype=float).reshape(0, 1)

for label in data_labels:
    class_ID = data_labels[label]

    ###################################################################
    ##### Start code here #####
    total_num = data[label].shape[0]  # Count the total number of examples

    validation_num = int(total_num * validation_ratio) # Compute the number of
 validation examples

    start_idx = int(total_num * training_ratio + 1) # Start index of the valid
ation set
    end_idx = start_idx + validation_num # End index of the validation set

    validation_data = data[label][start_idx:end_idx, :] # Obtain the validatio
n data
    validation_labels = np.ones((validation_num, 1)) * class_ID # Create label
s for the validation data

    # Concatenate the validation samples and labels for each dataset
    vali_x = np.concatenate((vali_x, validation_data))
    vali_y = np.concatenate((vali_y, validation_labels))
    ##### End code here #####
    ###################################################################

    print('Validation set: {} examples for Class {} ({})'.format(validation_nu
m, class_ID, label))

print('Total number of validation examples: {}'.format(vali_x.shape[0]))
```

In [18]:
```
test_x = np.array([], dtype=float).reshape(0, 33292)  # num_features = 33292
test_y = np.array([], dtype=float).reshape(0, 1)

for label in data_labels:
    class_ID = data_labels[label]

    ###################################################################
    ###### Start code here ######
    total_num = data[label].shape[0]  # Count the total number of examples

    test_num = int(total_num * testing_ratio) # Compute the number of test exa
mples

    start_idx = int(total_num - test_num) # Start index of the test set
    end_idx = start_idx + test_num # End index of the test set

    test_data = data[label][start_idx:end_idx, :] # Obtain the validation data
    test_labels = np.ones((test_num, 1)) * class_ID # Create labels for the va
lidation data

    # Concatenate the validation samples and labels for each dataset
    test_x = np.concatenate((test_x, test_data))
    test_y = np.concatenate((test_y, test_labels))
    ###### End code here ######
    ###################################################################

    print('Testing set: {} examples for Class {} ({})'.format(test_num, class_
ID, label))

print('Total number of testing examples: {}'.format(test_x.shape[0]))
```

In [42]:
```
train_x_copy = copy.deepcopy(train_x)

#####################################################################
##### Start code here. #####
# Step 1: Prepare the transformer
transformer = KernelPCA(n_components=3, kernel="rbf", fit_inverse_transform=Tr
ue, gamma=10)
# Step 2: Fit the model to train_x_copy and transform train_x_copy.
train_x_pca = transformer.fit_transform(train_x_copy)
##### End code here. #####
#####################################################################
```

In [45]:
```
train_x_pca_2d = None

#####################################################################
##### Start code here #####
transformer = KernelPCA(n_components=2, kernel="rbf", fit_inverse_transform=Tr
ue, gamma=10)
train_x_pca_2d = transformer.fit_transform(train_x_copy)
##### End code here #####
#####################################################################
```

In [48]:
```
# Generate a figure
fig = plt.figure(figsize=(13,8))
plt.title("Kernel PCA 2D Visualization", fontsize=16)

# List of colors
colors = ['r', 'b', 'g', 'y', 'c', 'm', 'k', 'brown']

# Plotting data points in the 2D plot with labels.
#####################################################################
##### Start code here #####
idx = 0
label_idx = 0

for label in data_labels:
    plt.scatter(train_x_pca_2d[idx : idx + train_size[label], 0], \
                train_x_pca_2d[idx : idx + train_size[label], 1], \
                c = colors[label_idx], \
                label = label)
    idx = idx + train_size[label]
    label_idx = label_idx + 1

##### End code here #####
#####################################################################

# Generate labels.
plt.xlabel('X-Axis')
plt.ylabel('Y-Axis')

# Generate the legend.
plt.legend(prop={'size': 16})

# Show the plot
plt.show()
```

In [50]:
```
# We need to also project the testing data onto the same 2D plane.
test_x_pca_2d = transformer.transform(copy.deepcopy(test_x))

#####################################################################
##### Start code here #####
knn_clf.predict(test_x_pca_2d)
##### End code here #####
#####################################################################
```

In [54]:
```
plt.figure(figsize=(13,8))

cmap_custom = ListedColormap(['r', 'b', 'g', 'y', 'c', 'm'])

########### Plot decision bounaries ###########
################################################################
##### Start code here #####
h= 0.05
alp =0.05

train_x_min, train_x_max = train_x_pca_2d[:, 0].min(), train_x_pca_2d[:, 0].ma
x()
train_y_min, train_y_max = train_x_pca_2d[:, 1].min(), train_x_pca_2d[:, 1].ma
x()
xx, yy = np.meshgrid(np.arange(train_x_min, train_x_max, h),
                     np.arange(train_y_min, train_y_max, h))
z = knn_c1f.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
z = z.reshape(xx.shape)
plt.figure()
plt.pcolormesh(xx, yy, z, cmap=cmap_custom, alpha=alp)
plt.title("k-NN Decision Boundaries h=%.3f" %h + " "+ "alpha=%.3f" %alp)
##### End code here #####
################################################################

########### Plot training data points ###########
idx = 0
label_idx = 0
for label in data_labels:
    plt.scatter(train_x_pca_2d[idx : idx + train_size[label], 0], \
                train_x_pca_2d[idx : idx + train_size[label], 1], \
                c = colors[label_idx], \
                label = 'train - ' + label)
    idx = idx + train_size[label]
    label_idx = label_idx + 1

########### Plot testing data points ###########
idx = 0
label_idx = 0
for label in data_labels:
    plt.scatter(test_x_pca_2d[idx : idx + test_size[label], 0], \
                test_x_pca_2d[idx : idx + test_size[label], 1], \
                c = colors[label_idx], \
                label = 'test - ' + label, edgecolor='k')
    idx = idx + test_size[label]
    label_idx = label_idx + 1

plt.legend(loc=3, fontsize=6)
plt.show()
```

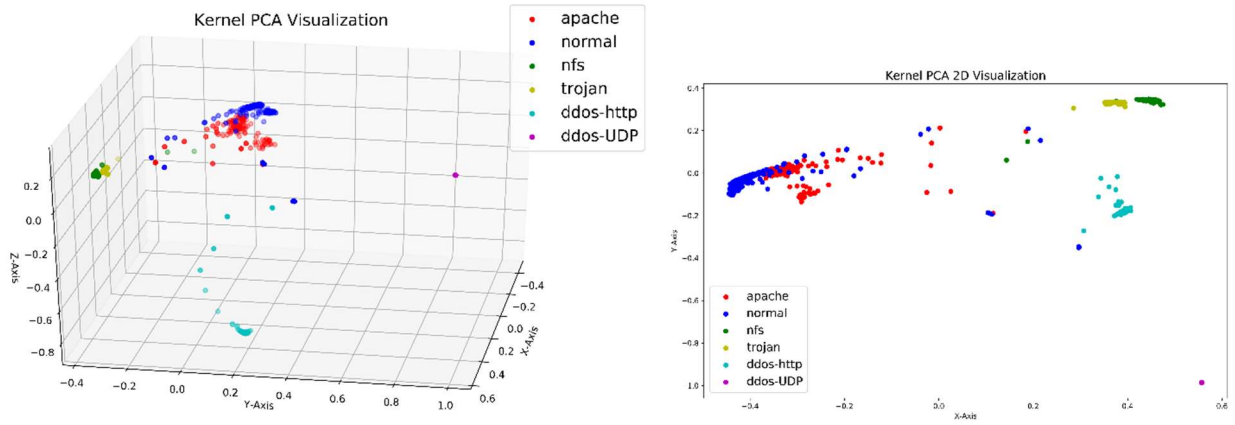
In [58]:
```
num_features = train_x.shape[1]
num_classes = train_y_one_hot.shape[1]

model = None
################################################################
##### Start code here #####
np.random.seed(5)
dnn = Sequential()
dnn.add(Dense(64, input_dim=num_features, activation='relu'))
dnn.add(Dense(16, activation='relu'))
dnn.add(Dense(num_classes, activation='softmax'))
dnn.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accur
acy'])
##### End code here
################################################################
```

In [59]:
```
history = None

################################################################
##### Start code here #####
history = dnn.fit(train_x, train_y_one_hot, validation_data=(vali_x, vali_y_on
e_hot),epochs=30, batch_size=10)

##### End code here #####
################################################################
```

In [63]:
```
training_acc = history.history['acc']
validation_acc = history.history['val_acc']

################################################################
##### Start code here #####
plt.plot(training_acc, label="Training")
plt.plot(validation_acc, label="Validation")
plt.title("DNN Model Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
##### End code here #####
################################################################
```

In [64]:
```
################################################################
##### Start code here #####
test_results = dnn.evaluate(test_x, test_y_one_hot)
print("The testing accuracy is %.3f" %(test_results[1]*100) +"%")
##### End code here #####
################################################################
```

Task 2: Plot accumulated offset curves as function of the elapsed time for the three messages:



- Comparison between the attack data against the normal data:

  - What attacks are significantly different from the normal data?

NFS, Trojan, and ddos-http/UDP are all very different from normal data as can be seen by their clusters being far away from the normal clusters.

  - Which attack is the closest to the normal data?

By far the apache attack is the closest to the normal data this can be seen by the fact that it's data points are nestled into the normal points.

- Comparison among different attacks:

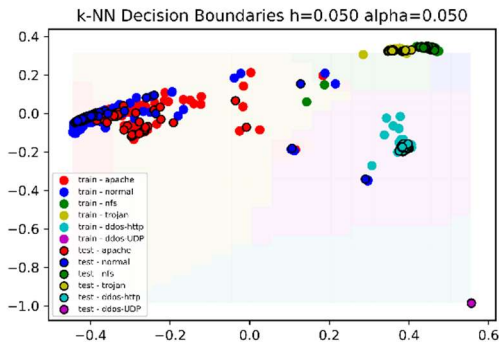
  - Which attacks are very similar to each other?

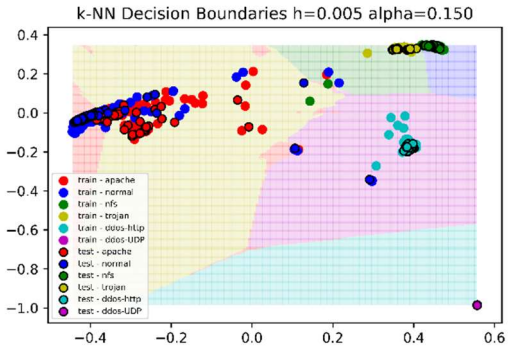
The NFS and Trojan attacks are very similar to each other. They are not nearly as related as the apache attack is to the normal data but they are close neighbors.

  - Which attacks are very different from others?

The most different attack is the ddos-UDP method. It is in another area of its own that can only really be seen in the 3D visualization. The only one that even compares is the ddos-http method which is also far away from the others but some of its further points are rather closer to other attack's points.

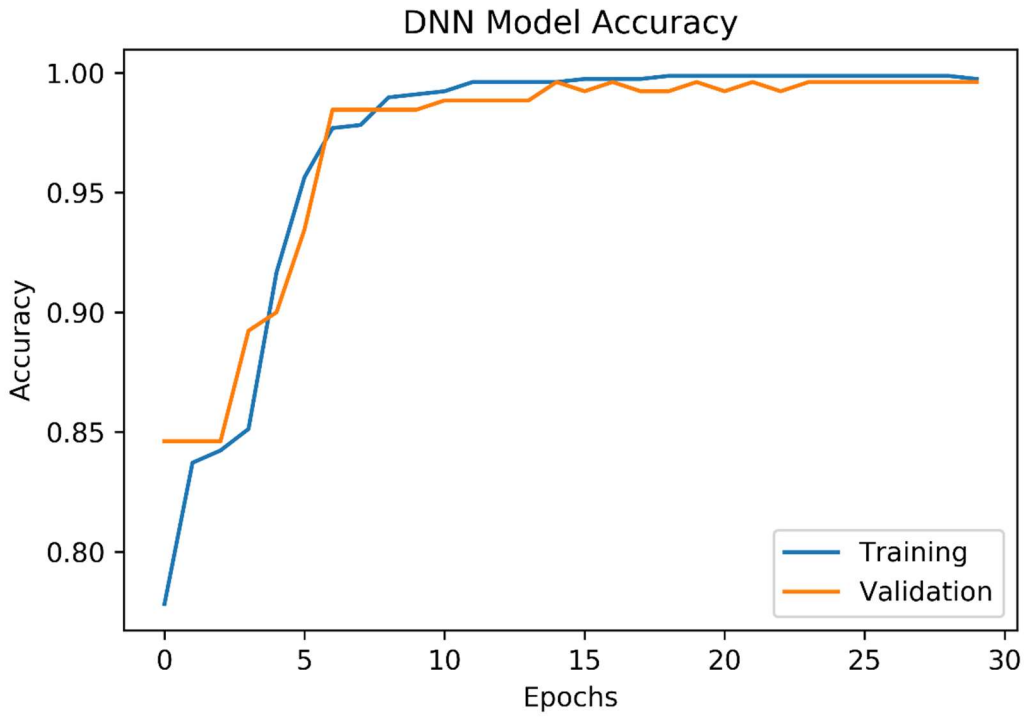
Task 7: Plot the decision boundaries of the above kNN classifier.





The classification accuracy is 94.63% This is very good considering how rudimentary this algorithm is since it doesn't use much learning.

Task 8: Implement a DNN using Keras; Plot both training and validation accuracies.



The testing accuracy is 99.615%. We see here the accuracy of the DNN model. It already starts at a very good amount at 76.9% Accuracy but as more and more epochs are passed the accuracy gets to almost perfect levels.

- For the kNN classifier, there are several design parameters such as `n neighbors` and `weights` that we can choose. Please describe how to leverage the validation set to choose the best design parameters.

We want to specifically choose the variables with that amongst the data has the highest variance. This way we end up with parameters that dictate the points without delving into the minor differences amongst similar points. This helps us avoid overfitting.

- For DNN, what is the difference between different activation functions, specifically, linear, sigmoid, ReLU, vs. softmax ?

The activation function decides the firing rate of a computer chip. Using these a learning network can be established and allow for the solving of more and more complex problems. To use fewer nodes however more and more complex and nonlinear functions are needed ergo why nonlinear functions are used.

- In ML, what is overfitting and underfitting?

Overfitting is when a prediction method is too trained on a specific data set. It will often have an amazing level of prediction on that data but fall flat when asked to predict other sets of data.

Underfitting is when a prediction method is incapable of being trained by a data set. This often happens when trying to oversimplify your prediction method and not using enough parameters.