Network Intrusion Detection using Big Data Analytics in Python

Jacob Wilson
Eric Turner
Chris Pitsilides
Nathan Apperson
Jake Day
Zane Densborn
Indiana University - Purdue University - Indianapolis

Abstract

This paper overviews a similar subset of the data from the 1999 KDD intrusion detection contest. Our team used feature selection on the most prominent features to help in the classification of each record as either a normal network request or a malignant network request. We used a decision tree classification model to determine the classifications of each request. Using this model and the subset of data we were provided, we were able to train a model to 100% accuracy in classifying the network records correctly. The majority of the network requests we found to be in the normal range and the rest were broken into the following attack types: smurf, guess password, ftp write and spy.

Introduction and Feature Selection

The focus of this task was to look into the data provided, and determine which features showed hints of attacked connections or normal connections. The data received showed too much random information that wouldn’t prove an attack, it would more prove normal connections than an attack or any difference. The main focus was for there to be noticable abnormalities in the data that would prove that an attack has occured.

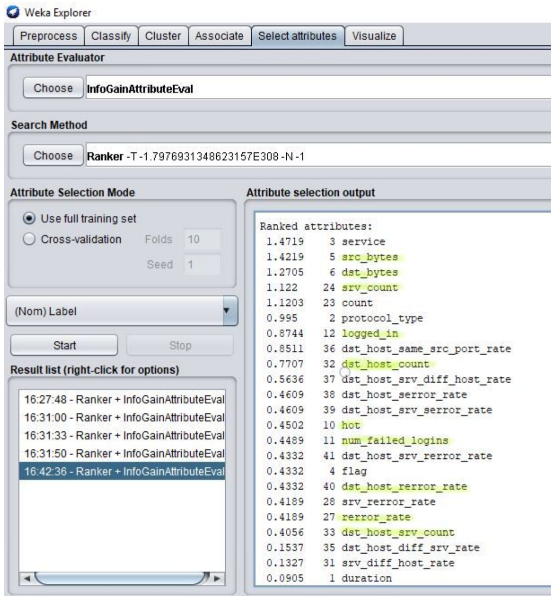
The features that we selected were src_bytes, dst_bytes, hot, num_failed_login, logged_in, srv_count, rerror_rate, dst_host_count, dst_host_srv_count, and dst_host_rerror_rate. These were the features that showed a pattern when an attack was run against the system. Not just an attack was proven but suspicious activity in general. The fields selected above were for the Big Data Management group to use to further detect issues in the data. The IT Risk Assessment team found that adding the service and label columns helped to differentiate between the attacks and normal connection traffic.

The Risk Assessment team found that the service column was helpful due to the connection type that was displayed or used. The reason for this is telnet isn’t going to be a good service to use compared to tcp or ftp. The service column gave a further idea when looking at the label, and src bytes columns proved smurf attacks and other normal data transference. The label column represented what was going on between different services as well. For example; all telnet services had guess_password outcomes, this sparked the interest of the Risk Assessment team. It was more of a surprise when the telnet through a spy label to further understand that telnet was a service to look into. The Risk Assessment team discovered a red flag on the majority of the ecr_i connections because each of them had a smurf attack attempted on the connection. The Big Data Management team helped to support some of the Risk Assessment teams findings.

The Big Team found that src_bytes helped to represent that a smurf attack throughs as 1032 byte payload from the source but the dst_bytes showed a 0 byte return. For the hot column the teams realized that it only showed on telent services. "Hot" also showed up on basic ftp connections. The groups figured that each of the hot connections had to do with failed logins on telnet connections. Due to the next important column being num_failed_login. This helped represent how many failed attempts occurred during each connection. While comparing the num_failed_login and logged_in columns this showed that there was a large amounts of attempts on telnet connections without continuing the login process. This could have been by mistake or the attacker was trying brute force attacks to gain access. There was also an increase in http service logins which could prove normal connection status.

The next important column would have to be the srv_count, this column represents the traffic among the server and the connection destination. This showed a unique byte increase when the smurf attacks started occurring on the ecr_i connections. This column could also support the normal traffic byte rate. Another column that could prove the smurf DDoS attacks were the dst_host_count and dst_host_srv_count. These columns show that there is a increase in server pings as it goes from host to server and floods the connections with traffic. The next column with significance is the rerror_rate. This column is used with the service column and label column to represent failed guess_passwds, on telnet transmissions. The last column that could be used to describe telnet traffic has to do with the dst_host_rerror_rate. This columns shows the percentage of failure on each telnet transmission sequence from the hosts machine.

The last thing that each group noticed was that there was a gradual increase from the start of the data into a large error rate from the telnet connections. The Big Data team also ran an information gain attribute selection in WEKA to determine what the best attributes for selection would be based off the entropy of the attributes. We used this information to influence our decision in the attribute selections that we used for the actual project. Below, in Figure 1, is a graphic depiction of the features that WEKA suggested that we used, based off entropy, with the highlights of the actual attributes that we selected.



Attribute Evaluator: InfoGainAttributeEval

Search Method: Ranker -T -1.7976931348623157E308 -N -1

Attribute Selection Mode: [✓ Use full training set] Cross-validation Folds 10 Seed 1

(Nom) Label

Result list (right-click for options)

```
16:27:48 - Ranker + InfoGainAttributeEval
16:31:00 - Ranker + InfoGainAttributeEval
16:31:33 - Ranker + InfoGainAttributeEval
16:31:50 - Ranker + InfoGainAttributeEval
16:42:36 - Ranker + InfoGainAttributeEval
```

Attribute selection output

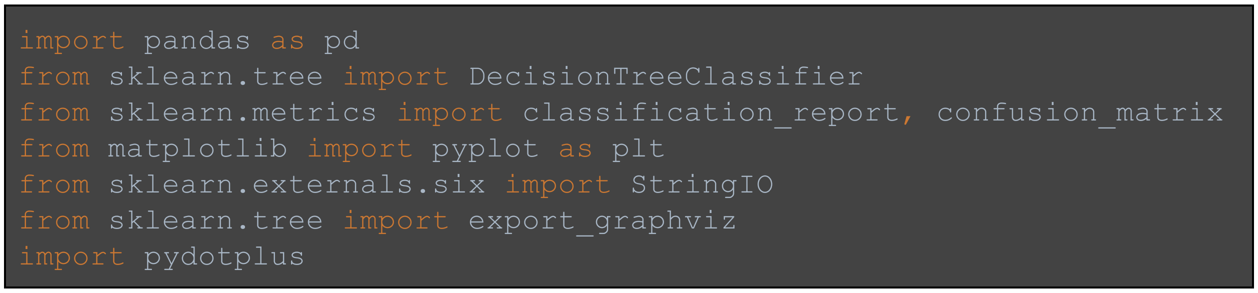
```
Ranked attributes:
 1.4719    3 service
 1.4219    5 src_bytes
 1.2705    6 dst_bytes
 1.122    24 srv_count
 1.1203   23 count
 0.995     2 protocol_type
 0.8744   12 logged_in
 0.8511   36 dst_host_same_src_port_rate
 0.7707   32 dst_host_count
 0.5636   37 dst_host_srv_diff_host_rate
 0.4609   38 dst_host_serror_rate
 0.4609   39 dst_host_srv_serror_rate
 0.4502   10 hot
 0.4489   11 num_failed_logins
 0.4332   41 dst_host_srv_rerror_rate
 0.4332    4 flag
 0.4332   40 dst_host_rerror_rate
 0.4189   28 srv_rerror_rate
 0.4189   27 rerror_rate
 0.4056   33 dst_host_srv_count
 0.1537   35 dst_host_diff_srv_rate
 0.1327   31 srv_diff_host_rate
 0.0905    1 duration
```

Figure 1: Weka Attribute Selection with our attributes highlighted

Our group decided on 10 features and our features were within the top 25 features in the information gain selection. We could also use a model to predict service as a category as well in a separate form of research if we wanted to for future studies. At this time, the service attribute is just used as a reference for the Risk Management team to help determine the malicious intent by seeing the exact service used in the event of an attack for trends and information to lock down those specific services in the future.

Methodology

Using the two CSV files we were provided, testing and training data, we decided to build a classification predictive model in Python. In order to begin in the classification, a list of libraries needed to be imported for use in the classification and data pre-processing. The libraries that were involved in the Python code are listed below in Figure 1.



```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
from matplotlib import pyplot as plt
from sklearn.externals.six import StringIO
from sklearn.tree import export_graphviz
import pydotplus
```

Figure 1: Imported Libraries

A further analysis of the specific uses of these libraries will be discussed in the following paragraphs. This section, the methodology, will discuss, in a step-by-step form, the exact process that was undertaken, including the relevant sections of Python code for reproduction purposes in any further or future cases of testing.

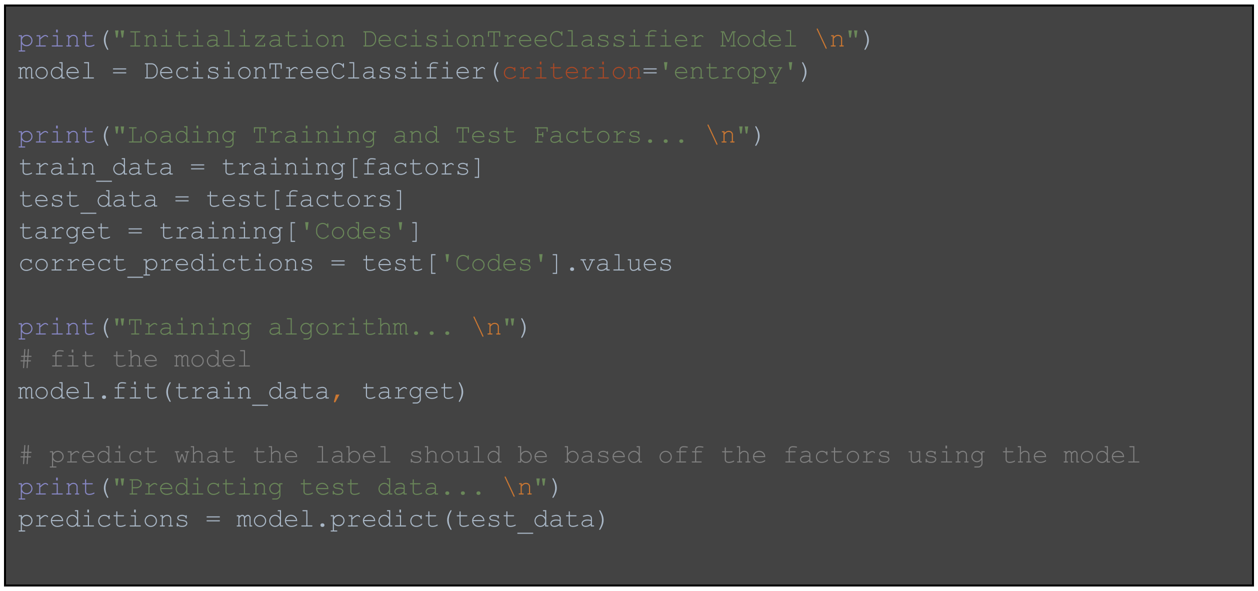
We begin by using a subset of the data acquired by Lincoln Labs in the late 1990's that has been pre-distributed into a testing and training set in an approximate 10/90 percent split, respectively (*Luo, 2018*). We selected the features that were aforementioned in the previous section to classify the data based on the categorical Label column, which has five unique categories: normal, smurf, guess_password, ftp_write, and spy.

In our Python code we create an array of the features that we discussed in the feature selection section to easily pass to the model that we will be creating later on. Then, we will need to do some data pre-processing because the libraries that we will be using in sklearn can only predict numerical data not categorical data. The array of factors and the conversion to numerical data is presented below in Figure 2.

```
# input the factors that will be used for the model
factors = ['src_bytes', 'dst_bytes', 'srv_count', 'logged_in', 'dst_host_count',
'hot', 'num_failed_logins', 'dst_host_rerror_rate', 'rerror_rate',
'dst_host_srv_count']

# convert categories into num codes on both training and test sets
# the integers will correspond with the below categories array [Source 3]
#                0              1            2          3          4
categories = ['normal.', 'guess_passwd.', 'smurf.', 'spy.', 'ftp_write.']

print("Converting Categorical Data Types to Numeric... \n")
training["Codes"] =
training["Label"].astype(pd.api.types.CategoricalDtype(categories=categories)).cat.cod
es
test["Codes"] =
test["Label"].astype(pd.api.types.CategoricalDtype(categories=categories)).cat.codes
```

Figure 2: Setting up factors and categorical data pre-processing

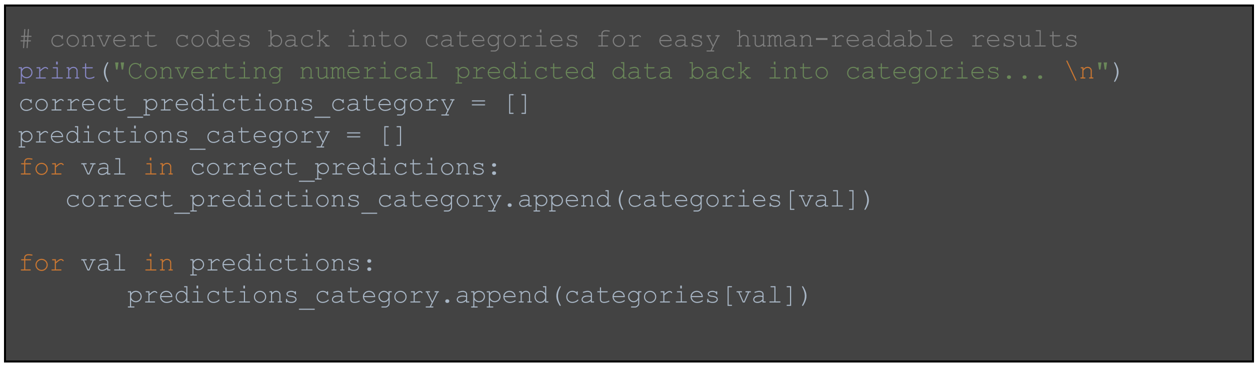
The model is now ready to be initialized, trained and tested. We decided to use a decision tree classifier here due to the many features we have selected all have an impact on what the outcome of the attack can be, and a decision tree is the best way to view how the variations in the factors can create a branch to a different outcome. Python code for this section was derived from an article in Stackabuse (*Robinson, 2018*). As well as the scikit learn online manual for the DecisionTreeClassifier() (*scikit-learn*). We first initialize the DecisionTreeClassifier and assign it to a variable for further use, by default the classifier uses the gini impurity to calculate the decision tree but we passed a argument to switch the calculation to entropy. New variables are created for the training and testing data which uses the subset of our imported data, that uses ONLY the factors that we have deemed relevant for the classification. Our target variable is initialized and this is the numerical conversion of the categorical Label column, the field in which we are trying to predict. We also grab the values of the correct testing numerical codes to compare to the predicted values once the model has been trained. The model gets fitted with the now initialized training data and the target that it needs to predict. Then finally gets ran with the test data (WITHOUT the already-known label column included) to test the accuracy of the model. The code for the initialization, training and prediction is in Figure 3, below.



```
print("Initialization DecisionTreeClassifier Model \n")
model = DecisionTreeClassifier(criterion='entropy')

print("Loading Training and Test Factors... \n")
train_data = training[factors]
test_data = test[factors]
target = training['Codes']
correct_predictions = test['Codes'].values

print("Training algorithm... \n")
# fit the model
model.fit(train_data, target)

# predict what the label should be based off the factors using the model
print("Predicting test data... \n")
predictions = model.predict(test_data)
```

Figure 3: Initialization, Training and Prediction of the Model

Although not necessarily needed, for ease-of-consumption in reading the visualizations, we converted the numerical codes back into the categories, seen in Figure 4, below.



```
# convert codes back into categories for easy human-readable results
print("Converting numerical predicted data back into categories... \n")
correct_predictions_category = []
predictions_category = []
for val in correct_predictions:
   correct_predictions_category.append(categories[val])

for val in predictions:
       predictions_category.append(categories[val])
```

Figure 4: Conversion from numerical codes back into categorical labels

The full source code is available in Figure 9 at the end of the report which includes the code which created the visualizations that are interpreted in the results.

Results

Our model displayed a 100% accuracy from the features that we selected, which in-turn created a confusion matrix with all of the correct categories and a 100% accurate classification report, as seen in Figure 5, below.

```
=============================

Accuracy Score: 1.0

Confusion Matrix:
   a    b   c   d  e
[[ 24   0   0   0  0]   a - normal.
 [  0   20  0   0  0]   b - guess-passwd.
 [  0    0  13  0  0]   c - smurf.
 [  0    0   0   1  0]   d - spy.
 [  0    0   0   0  3]]  e - ftp_write.

Classification Report:
              precision     recall     f1-score     support

          0        1.00       1.00         1.00          24
          1        1.00       1.00         1.00          20
          2        1.00       1.00         1.00          13
          3        1.00       1.00         1.00           1
          4        1.00       1.00         1.00           3

avg / total        1.00       1.00         1.00          61

=============================
```

Figure 5: Output of Accuracy, Confusion Matrix and Classification Report [Formatted for user readability]

A scatter chart was generated using PyPlot that provides a visualization of the model accuracy as well, as seen in Figure 6, below.

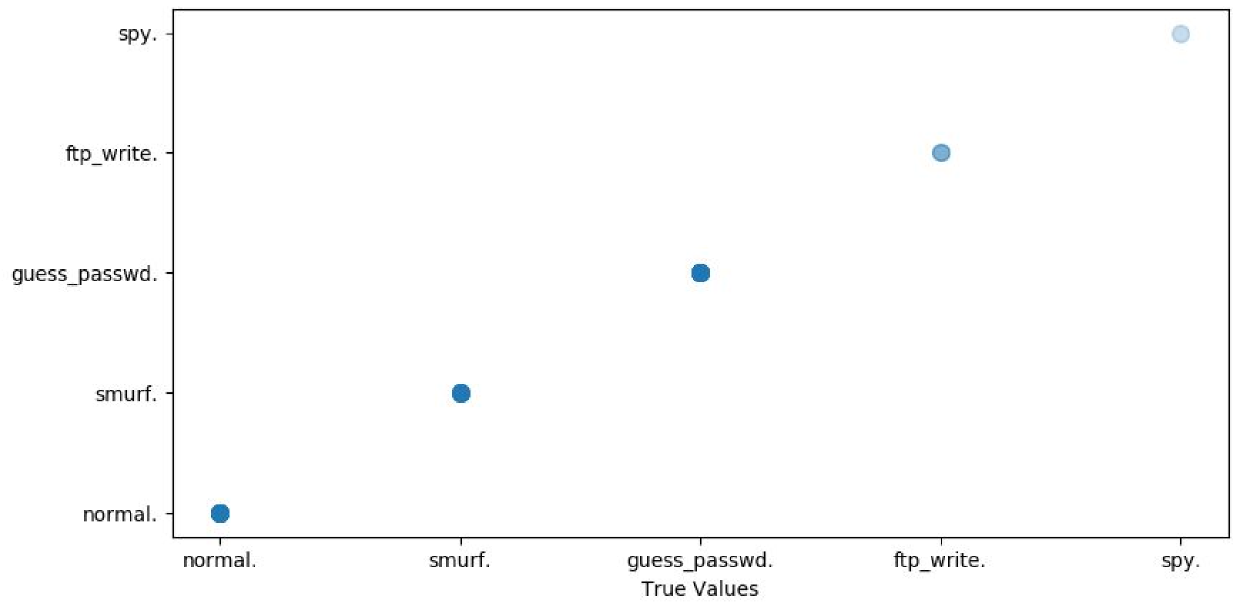


Figure 6: Scatter Chart of Model Accuracy

The scatter chart has an alpha value set which helps display the amount of overlapping that is happening here. There were a total count of 24 ‘normal.’ records of activity, and a combined 37 records that have attacks in them. Figure 7, below, breaks out the total correctly predicted normal and malicious attacks.

Type	**Records**
Normal Traffic	24
Smurf Attack	13
Guess Password	20
FTP Write	3
Spy	1

Figure 7: Breakdown of the Correctly Predicted (and Total Number) of Categories in Testing Data

There is also one final output we have, a decision tree, that is very informative towards the accuracy of our data as well, in Figure 8 on the next page. The decision tree helps us understand the path the model took in determining the classification of the network traffic. Our tree uses binary splits, opposed to multiway splits. To fully understand the tree, we need to fully break down each element of the tree. Each node — colored rectangle — has a number of descriptors about that node. The first line is the determinant for the node, for example in the first node the model is checking if the srv_count factor is less than or equal to 175. If the srv_count is less than or equal to 175, True, then it follows the left path to continue breaking down to determine which classification the record falls under. If the srv_count is greater than 175, false, then it follows the right path which classifies the network activity as a smurf attack. The decision tree follows this pattern, true being on the left and false being on the right, for the entirety. The entropy is the calculation that we forced the classifier to use. Entropy is a mathematical equation that has several complicated definitions, a fantastic simplified definition given by a great blog post from Benjamin Ricaud describes entropy as "an indicator of how messy your data is"(*Ricaud, 2017*). The model always will choose the factor with the highest entropy in its decision for classifying the next node. The samples and value of the nodes go hand in hand for classification. The samples are the total number of records that are classified under that node. The values breaks down that sample into what category each network traffic record falls under. For example, in the first node we have a total of 531 samples (records). This is broken down in the values field, the array from left to right is the amount of records classified as normal, guess_passwd, smurf, spy, and ftp_write respectively. So for the first node we can see 232 of the 531 records are normal and so on and so forth. While trickling down the decision tree, the model breaks down how the remaining samples fit into each node until it finalizes on the end classifications. Finally the class attribute is the most important attribute for human readability, along with the color coding of each node, we are able to distinguish visually the decisions from the model.

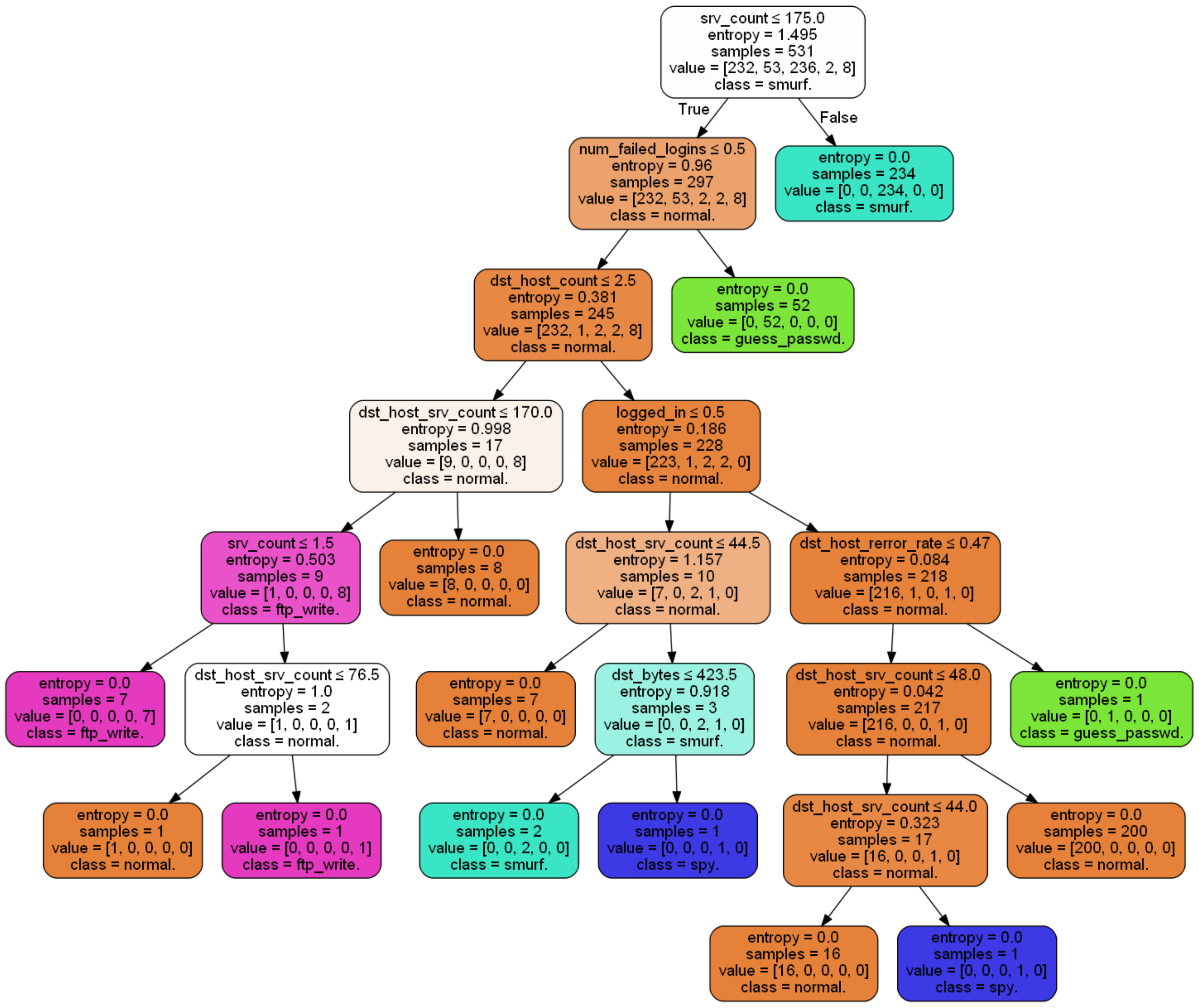


Figure 8: Decision Tree of Model

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
from matplotlib import pyplot as plt
from sklearn.externals.six import StringIO
from sklearn.tree import export_graphviz
import pydotplus
import collections

# Sources
# [1] https://secdevops.ai/learning-packet-analysis-with-data-science-5356a3340d4e -  Original Guide
# [2] https://stackoverflow.com/questions/45681387/predict-test-data-using-model-based-on-training-data-set -  Using two files
for training and testing data
# [3] https://stackoverflow.com/questions/34007308/linear-regression-analysis-with-string-categorical-features-variables -
Converting Categorical into Numerical Values
# [4] https://stackabuse.com/decision-trees-in-python-with-scikit-learn/ - Decision Trees in Python
# [5] https://medium.com/@rnbrown/creating-and-visualizing-decision-trees-with-python-f8e8fa394176 - Visualizing Decision Trees
# [6] http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html - Learning bout the Classifier
# [7] http://intelligentonlinetools.com/blog/2018/02/10/how-to-create-data-visualization-for-association-rules-in-data-mining/
- Usage of a applying a random number to prevent scatter plot overlapping
# [8] https://stackoverflow.com/questions/42891148/changing-colors-for-decision-tree-plot-created-using-export-graphviz -
Changing colors of decision tree

# Load training and test data from csv [Source 2]
print("Importing .CSVs... \n")
training = pd.read_csv('kddcup.data_subset_training.csv', header=0, sep=',')
test = pd.read_csv('kddcup.data_subset_test.csv', header=0, sep=',')

# input the factors that will be used for the model
factors = ['src_bytes', 'dst_bytes', 'srv_count', 'logged_in', 'dst_host_count', 'hot', 'num_failed_logins',
'dst_host_rerror_rate', 'rerror_rate', 'dst_host_srv_count']

# convert categories into num codes on both training and test sets
# the integers will correspond with the below categories array [Source 3]
#                0              1            2         3          4
categories = ['normal.', 'guess_passwd.', 'smurf.', 'spy.', 'ftp_write.']

print("Converting Categorical Data Types to Numeric... \n")
training["Codes"] = training["Label"].astype(pd.api.types.CategoricalDtype(categories=categories)).cat.codes
test["Codes"] = test["Label"].astype(pd.api.types.CategoricalDtype(categories=categories)).cat.codes

# setup the model
# used a decision tree classifier because the many factors can have an effect on the path taken to classify correctly [Source
4, 6]
print("Initialization DecisionTreeClassifier Model \n")
model = DecisionTreeClassifier(criterion='entropy')

# train and test the data against the target [Source 2]
print("Loading Training and Test Factors... \n")
train_data = training[factors]
test_data = test[factors]
target = training['Codes']
correct_predictions = test['Codes'].values

print("Training algorithm... \n")
# fit the model
model.fit(train_data, target)

# predict what the label should be based off the factors using the model
print("Predicting test data... \n")
predictions = model.predict(test_data)

# convert the integers back into categorical labels for easy human-readable results
print("Converting numerical predicted data back into categories... \n")
correct_predictions_category = []
predictions_category = []
for val in correct_predictions:
   correct_predictions_category.append(categories[val])

for val in predictions:
       predictions_category.append(categories[val])

print("Generating Dataframe of Correct vs Predicted Test Data... \n")
# Print out a table of the correct vs the predicted side by side
pred = pd.DataFrame({'Correct': correct_predictions_category, 'Predicted': predictions_category})
# print(pred) # Predictions
print("Exporting dataframe to CSV... \n")
# Export the table to a CSV in this working directory to see full results
pred.to_csv('model_results.csv', index=False)

print("Generating Statistics on Model... \n")
```

```python
print("=============================\n")
# Calculate the accuracy score (data predicted vs actual known values)
print("Accuracy Score:", model.score(test_data, correct_predictions))

print("Confusion Matrix: ")
print(confusion_matrix(predictions, correct_predictions))

print("Classification Report: ")
print(classification_report(predictions, correct_predictions))
print("=============================\n")

print("Creating Scatter Plot of Model Accuracy...\n")

# Show the predictions vs actual known test label in a visualization
plt.scatter(correct_predictions_category, predictions_category, s=70, alpha=0.25)

plt.xlabel("True Values")
plt.ylabel("Predictions")
plt.show()

print("Creating Decision Tree..\n")
# Display Decision Tree [Source 5]
dot_data = StringIO()
export_graphviz(model, out_file=dot_data,
               filled=True, rounded=True,
               special_characters=True, feature_names=factors, class_names=categories)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())

print("Exporting Decision Tree to File...\n")
graph.write_png('tree.png')
print("Analysis Complete\n")
```

Figure 9: Full Python Source Code

Conclusion

The model that we trained was able to, with 100 percent accuracy, classify the correct network requests into normal or different categories of network attacks. However, a 100 percent accurate model usually can be the case of not enough data, according to a question post on the website ResearchGate (*ResearchGate, n.d.*). Therefore, we believe having a larger subset, or possibly the entire set of data, would be better to fully train and test our model. We could then try using cross-validation to fully ensure our classification model is as accurate as possible. But with only just over 500 records of data, this paper and the model we trained was accurate based on the dataset we were using.

References

Luo, X. (2018, October 10). *Project 1 Description* [DOCX]. Indianapolis: IU Canvas.

ResearchGate. (n.d.). Is it possible that accuracy for classification of testing data set is 1.00? Retrieved November 5, 2018, from https://www.researchgate.net/post/Is_it_possible_that_accuracy_for_classification_of_testing_data_set_is_100

Ricaud, B. (2017, August 27). A simple explanation of entropy in decision trees. Retrieved October 26, 2018, from https://bricaud.github.io/personal-blog/entropy-in-decision-trees/

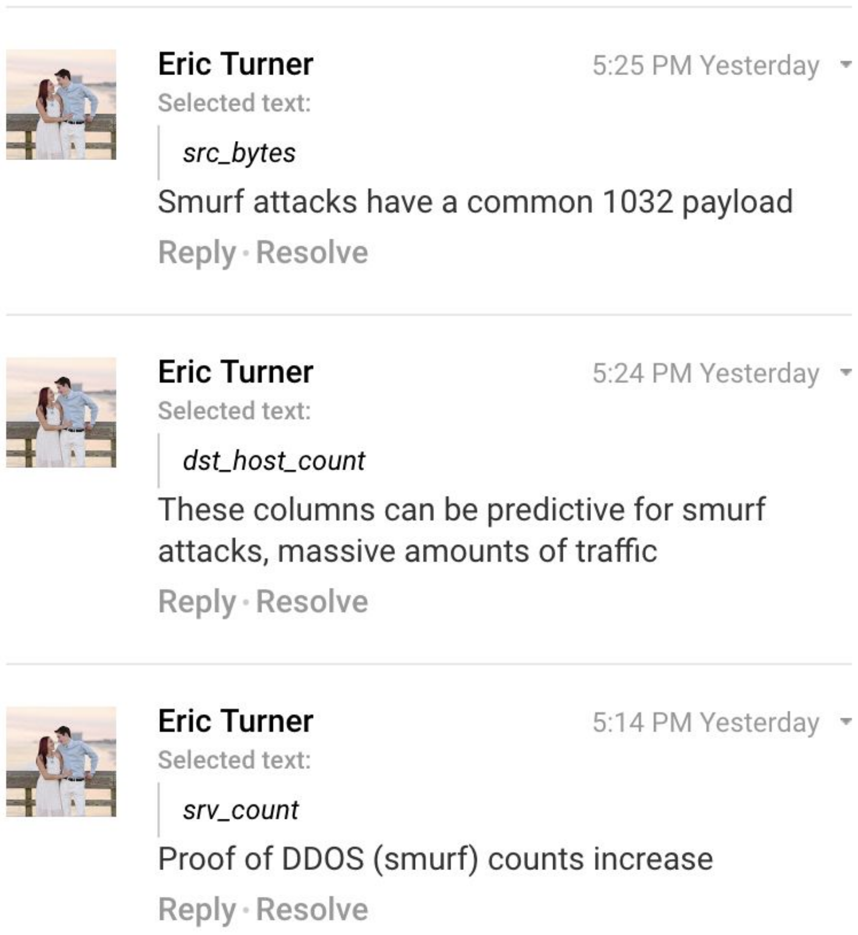
Robinson, S. (2018, February 28). Decision Trees in Python with Scikit-Learn. Retrieved October 17, 2018, from https://stackabuse.com/decision-trees-in-python-with-scikit-learn/

Scikit-learn. (n.d.). Sklearn.tree.DecisionTreeClassifier¶. Retrieved October 17, 2018, from http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

Appendix:

Figure 1, a screenshot from the WEKA program is part of the determination in our attribute selection.

Much of the selection for attributes was done in class on 17 October using the following Google Sheet. In this Google Sheet, we went column by column to determine which features we wanted, the header of these columns includes notes about each decision. (A picture of some of the comments is below, the rest can be viewed on the actual google sheet)


Eric Turner 5:25 PM Yesterday

Selected text:

src_bytes

Smurf attacks have a common 1032 payload

Reply · Resolve

Eric Turner 5:24 PM Yesterday

Selected text:

dst_host_count

These columns can be predictive for smurf attacks, massive amounts of traffic

Reply · Resolve

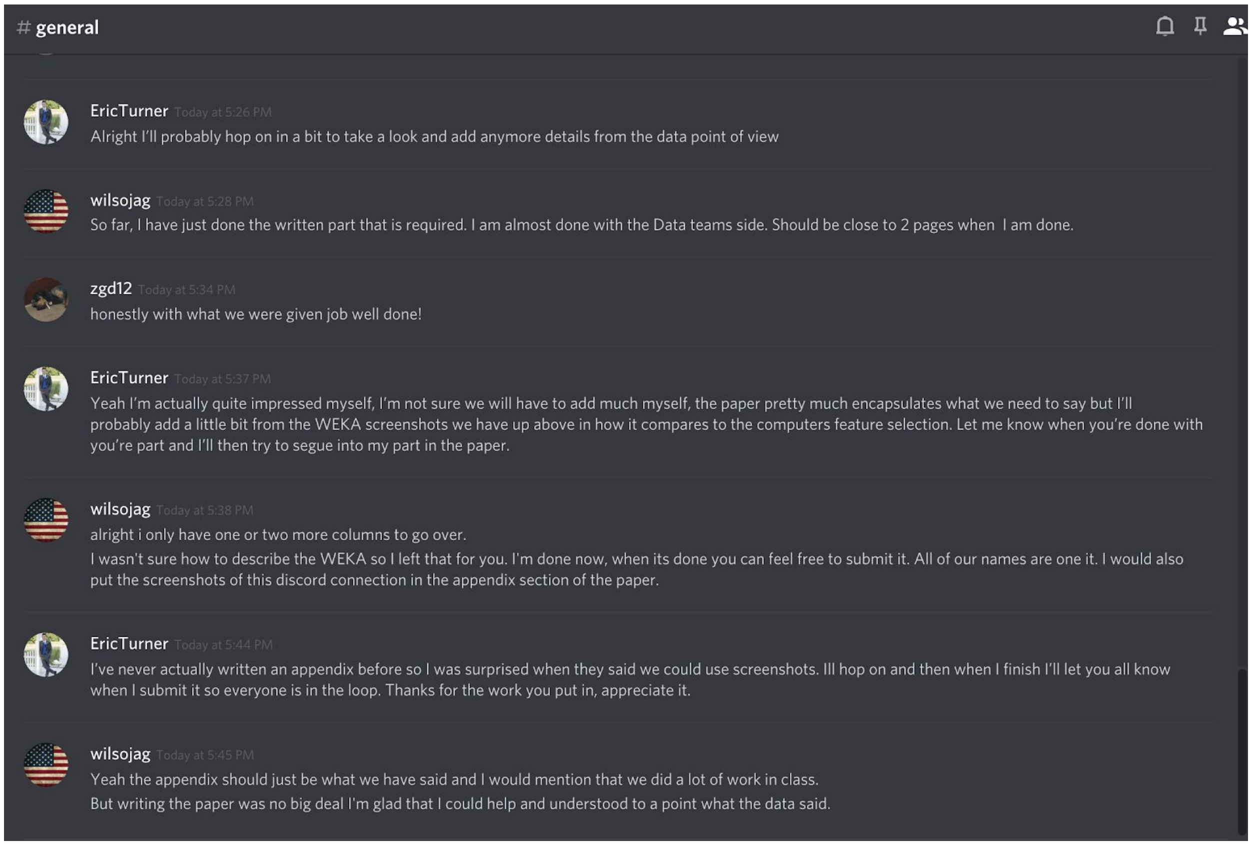
Eric Turner 5:14 PM Yesterday

Selected text:

srv_count

Proof of DDOS (smurf) counts increase

Reply · Resolve



EricTurner Today at 5:26 PM
Alright I'll probably hop on in a bit to take a look and add anymore details from the data point of view

wilsojag Today at 5:28 PM
So far, I have just done the written part that is required. I am almost done with the Data teams side. Should be close to 2 pages when I am done.

zgd12 Today at 5:34 PM
honestly with what we were given job well done!

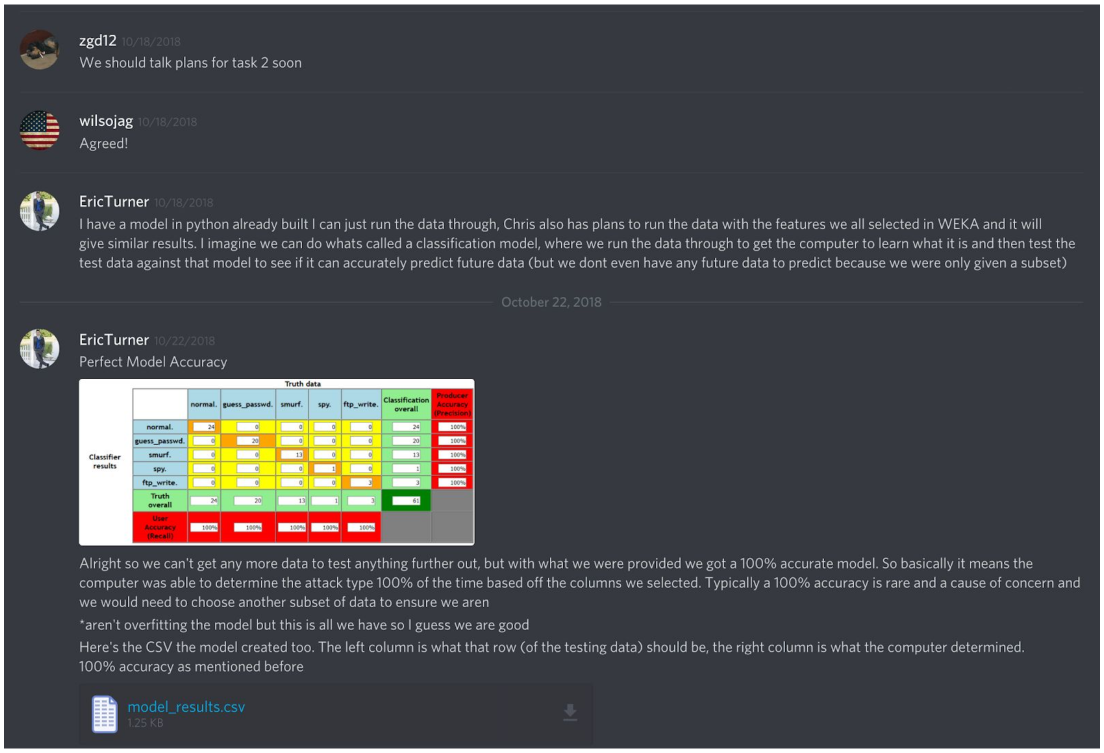
EricTurner Today at 5:37 PM
Yeah I'm actually quite impressed myself, I'm not sure we will have to add much myself, the paper pretty much encapsulates what we need to say but I'll probably add a little bit from the WEKA screenshots we have up above in how it compares to the computers feature selection. Let me know when you're done with you're part and I'll then try to segue into my part in the paper.

wilsojag Today at 5:38 PM
alright i only have one or two more columns to go over.
I wasn't sure how to describe the WEKA so I left that for you. I'm done now, when its done you can feel free to submit it. All of our names are one it. I would also put the screenshots of this discord connection in the appendix section of the paper.

EricTurner Today at 5:44 PM
I've never actually written an appendix before so I was surprised when they said we could use screenshots. Ill hop on and then when I finish I'll let you all know when I submit it so everyone is in the loop. Thanks for the work you put in, appreciate it.

wilsojag Today at 5:45 PM
Yeah the appendix should just be what we have said and I would mention that we did a lot of work in class.
But writing the paper was no big deal I'm glad that I could help and understood to a point what the data said.

Discussion of the feature selection and introduction



zgd12 10/18/2018
We should talk plans for task 2 soon

wilsojag 10/18/2018
Agreed!

EricTurner 10/18/2018
I have a model in python already built I can just run the data through, Chris also has plans to run the data with the features we all selected in WEKA and it will give similar results. I imagine we can do whats called a classification model, where we run the data through to get the computer to learn what it is and then test the test data against that model to see if it can accurately predict future data (but we dont even have any future data to predict because we were only given a subset)

October 22, 2018

EricTurner 10/22/2018
Perfect Model Accuracy

		Truth data						
		normal.	guess_passwd.	smurf.	spy.	ftp_write.	Classification overall	Producer Accuracy (Precision)
Classifier results	normal.	24	0	0	0	0	24	100%
	guess_passwd.	0	20	0	0	0	20	100%
	smurf.	0	0	13	0	0	13	100%
	spy.	0	0	0	1	0	1	100%
	ftp_write.	0	0	0	0	3	3	100%
	Truth overall	24	20	13	1	3	61	
	User Accuracy (Recall)	100%	100%	100%	100%	100%		

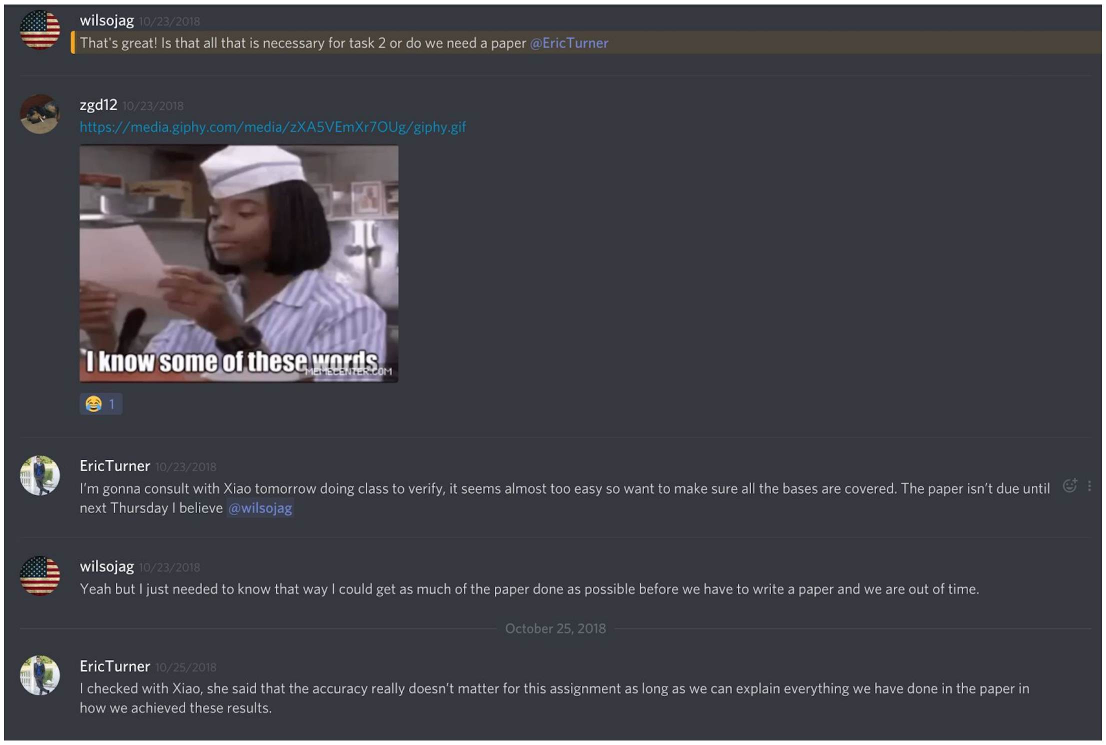
Alright so we can't get any more data to test anything further out, but with what we were provided we got a 100% accurate model. So basically it means the computer was able to determine the attack type 100% of the time based off the columns we selected. Typically a 100% accuracy is rare and a cause of concern and we would need to choose another subset of data to ensure we aren

*aren't overfitting the model but this is all we have so I guess we are good

Here's the CSV the model created too. The left column is what that row (of the testing data) should be, the right column is what the computer determined. 100% accuracy as mentioned before

model_results.csv
1.25 KB

Discussion on the beginning of the methodology



wilsojag 10/23/2018
That's great! Is that all that is necessary for task 2 or do we need a paper @EricTurner

zgd12 10/23/2018
https://media.giphy.com/media/zXA5VEmXr7OUg/giphy.gif

😂 1

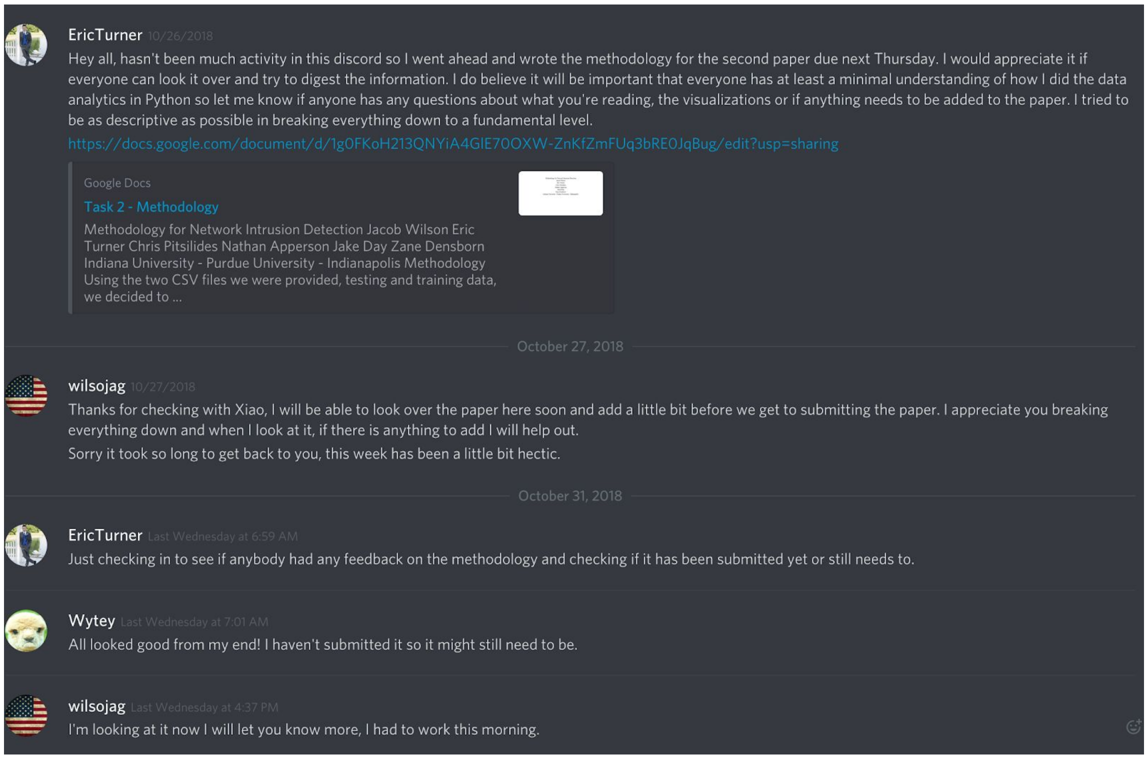
EricTurner 10/23/2018
I'm gonna consult with Xiao tomorrow doing class to verify, it seems almost too easy so want to make sure all the bases are covered. The paper isn't due until next Thursday I believe @wilsojag

wilsojag 10/23/2018
Yeah but I just needed to know that way I could get as much of the paper done as possible before we have to write a paper and we are out of time.

October 25, 2018

EricTurner 10/25/2018
I checked with Xiao, she said that the accuracy really doesn't matter for this assignment as long as we can explain everything we have done in the paper in how we achieved these results.

Further discussion on the scope of the methodology requirements



EricTurner 10/26/2018
Hey all, hasn't been much activity in this discord so I went ahead and wrote the methodology for the second paper due next Thursday. I would appreciate it if everyone can look it over and try to digest the information. I do believe it will be important that everyone has at least a minimal understanding of how I did the data analytics in Python so let me know if anyone has any questions about what you're reading, the visualizations or if anything needs to be added to the paper. I tried to be as descriptive as possible in breaking everything down to a fundamental level.
https://docs.google.com/document/d/1g0FKoH213QNYiA4GlE70OXW-ZnKfZmFUq3bRE0JqBug/edit?usp=sharing

Google Docs
Task 2 - Methodology
Methodology for Network Intrusion Detection Jacob Wilson Eric Turner Chris Pitsilides Nathan Apperson Jake Day Zane Densborn Indiana University - Purdue University - Indianapolis Methodology Using the two CSV files we were provided, testing and training data, we decided to ...

October 27, 2018

wilsojag 10/27/2018
Thanks for checking with Xiao, I will be able to look over the paper here soon and add a little bit before we get to submitting the paper. I appreciate you breaking everything down and when I look at it, if there is anything to add I will help out.
Sorry it took so long to get back to you, this week has been a little bit hectic.

October 31, 2018

EricTurner Last Wednesday at 6:59 AM
Just checking in to see if anybody had any feedback on the methodology and checking if it has been submitted yet or still needs to.

Wytey Last Wednesday at 7:01 AM
All looked good from my end! I haven't submitted it so it might still need to be.

wilsojag Last Wednesday at 4:37 PM
I'm looking at it now I will let you know more, I had to work this morning.

Discussion of the completion and proofreading over the methodology