**Project Report/Presentation Outline**

# Abstract

* + A
  + A

# Introduction (describing the dataset as well)

## Machine Learning Overview

* + - What is ML
    - History
      * <https://www.dataversity.net/a-brief-history-of-machine-learning/>
      * [Early History of Machine Learning](https://www.sciencedirect.com/science/article/pii/S2405896320325027)

## Topic Introduction (SPAM vs HAM)

* + - first iteration of Machine learning in Cyber security was a spam filter for email – evolution from then until now
      * A
  + Cyber Security
  + Objective [segue]

# Related Work

* + A
  + A

# Methods

## Data Configuration

### Import and review dataset

* + - * Pandas was utilized to import the data set
      * The data set was reviewed for any issues or null values
      * Once the data set was reviewed for issues it was then analyzed for various properties such as Count, Unique, Top, and Frequeancy values throughout the dataset.
        + Count – Count/Occurances of each feature
        + Unique – The number of possible unique observations
        + Top – The most frequent value
        + Freq – The frequency of the top value

### Initial Data Analysis

The features of this dataset are 'Class' and 'sms', where 'Class' indicates whether the message is spam or a valid sms message, ham and 'sms' contains the corresponding message.

The count values above shows us there are 5572 non-null data enteries in each feature. As each feature contains the same count value, we can conclude there are no missing data points that we need to trim.

The unique value of 2 under the Class feature verifies all messages are either spam or ham, and contain no erroneous values. Since the sms feature contains a unique value of 5169, which is less than 5572, we can assume that some messages are identical.

The top and freq values under Class show us that most messages are categorized as ham, with 4825 occurrences. We can therefore determine there are 747 remaining messages categorized as spam. The top and freq values under sms confirm the previous hypothesis that some messages are identical; we can see that the most frequent message, occurring 30 times, contains the text "Sorry, I'll call later"

Using this information, we can identify the format of our data, determine its completeness, and verify the values contained are expected.

### Modify data

* + - * Additional metadata was added to the data which allows for a numerical representation/discrete value for classification for Spam == 1 or Ham == 0 this column is referred to as Class\_num.
      * Additional metadata was added to the data which allow for message length attribute to help assist the classification of the data. This column is referred to as sms\_len.

### Graph data

* + - * Leveraging seaborn and matplotlib the data is then plotted to look for patterns using the message length for a visual of the data. The plot show us that spam messages are typically longer than ham messages while ham messages lengths come in various smaller sizes.

### Detailed Analysis – Complete data set

* + - * Data set includes Ham (0) and Spam (1) combined into Class\_num as previous stated.
      * A Class\_num mean of 0.134 means that 13.4% of our data is spam
        + Inversely, 86.8% is Ham
      * SMS messages average 80.48 characters
      * The shortest message length is 2 characters
      * The longest message length is 910 characters

### Detailed Analysis - Ham Data Set

* + - * This data set include Ham (0) only
      * There are 4825 ham messages
      * Since Ham is 0 all other stats in Class\_num will == 0
      * Ham messages average 71.48 characters
      * The shortest ham message length is 2 characters
      * The longest ham message length is 910 characters

### Detailed Analysis – Spam Data Set

* + - * This data set include Spam (1) only
      * There are 747 spam messages
      * Since Spam is 1 all other stats in Class\_num will == 1
        + Except standard deviation, as there is no deviation between 1 and 1
      * Spam messages average 138.67 characters
      * The shortest spam message length is 13 characters
      * The longest spam message length is 223 characters

### Prepare data using Natural Language Processing (NLP)

#### NLP Summary and Overview

##### NLP

Natural language processing (NLP) is a subfield of computer science and artificial intelligence that focuses on the interaction between computers and human language. NLP aims to enable computers to process, understand, and generate natural language text or speech in the same way that humans do. This involves a wide range of tasks, including sentiment analysis, language translation, speech recognition, text summarization, and more.

NLP techniques are commonly used in spam versus ham classification to automatically identify and filter unwanted messages in email or text communication. In this context, "ham" refers to legitimate messages, while "spam" refers to unwanted messages that are typically sent in large volumes with the intention of advertising or scamming the recipient.

NLP techniques can be used to analyze the content and structure of sms messages, and identify certain characteristics that are common to spam messages. For example, spam messages may contain unusual words or phrases, excessive use of capitalization or punctuation, or certain types of attachments or links. By contrast, ham messages tend to be more structured and contain typical language patterns.

One common NLP technique for spam versus ham classification is machine learning, where a model is trained on a large dataset of labeled email messages to learn patterns and relationships between features and labels. The model can then be used to classify new, unseen messages as either ham or spam with a high degree of accuracy.

Another NLP technique for spam versus ham classification is rule-based systems, where a set of rules are defined to detect certain characteristics of spam messages. These rules may be based on specific keywords or phrases, the presence of certain types of attachments or links, or other features of the message.

Overall, NLP techniques are essential for accurately identifying and filtering unwanted spam messages, and can save time and improve productivity for individuals and organizations that rely on email or text communication.

##### Stopwords

Stopwords are common words that are typically removed from text data during natural language processing (NLP) tasks, including spam versus ham classification. These words include common prepositions, conjunctions, and other frequently occurring words that do not carry much meaning on their own.

In the context of spam versus ham classification, stopwords can play a role in distinguishing between spam and ham messages. Spam messages often contain a higher percentage of stopwords than ham messages, since spammers may use these words to try to evade detection by spam filters or to make their messages more difficult to categorize.

However, simply removing stopwords is not always an effective approach to spam versus ham classification, since some stopwords may actually be useful in identifying certain types of messages. For example, the presence of certain stopwords like "bank" or "credit" may be indicative of a legitimate financial message, while the presence of other stopwords like "earn" or "guaranteed" may be indicative of a spam message.

Therefore, in spam versus ham classification, it is important to use a more nuanced approach to stopwords, and to consider other features of the message as well, such as the presence of certain keywords, the structure of the message, or the sender's email address. By combining these different features, NLP techniques can be used to accurately identify and filter spam messages while minimizing false positives and false negatives.

#### NLP in Code

##### Create function to cleanup data

A list of stopwords and common abberviations relevant to sms data was created to help cleanup message data.

##### Code Breakdown for Process\_Msg function

nopunc = [char for char in sms if char not in string.punctuation]

" for every character in the message,   
if the character is not in the list of punctuation,   
save that char into the list 'nopunc' "

removes punctuation

Essentially, nopunc is the same as sms, just without the punctuation

nopunc = ' '.join([word for word in nopunc.split() if word.lower() not in STOPWORDS])

" for every word in the 'nopunc' list,   
if the word [changed to lowercase] is not in 'STOPWORDS',   
save it into the list 'nopunc'"

removes Stopwords

##### Data Modification

Create column for the clean messages results from the Process\_Msg function.

Review the data for an issues or anomalies.

##### Data extraction – Ham messages

For each ham message we converted all text into lowercase and split words into a list which we call ‘ham\_words’

##### Data extraction – Spam messages

For each spam message we converted all text into lowercase and split words into a list which we call ‘spam\_words’

##### Create Frequency Table – Ham messages

The ham\_words list was fed into a word frequency counter function in which the program will iterate through each word and provide a count of that occurance of word in the data set.

This is a good place to check for additional stopwords.  
For example, 2 of the top 3 most common words here are 'U' and '2' - these would be great additions to the stopword list.

If unsure about adding a specific word to the stopwords list, ask if the word adds any context - if not, it would likely work well as a stopword.

##### Create Frequency Table – Spam messages

The spam\_words list was fed into a word frequency counter function in which the program will iterate through each word and provide a count of that occurrences of that word in the data set.

This is yet another chance to capture additional stopwords to add to our list here we have ‘call’ and ‘free’ as the top two occurrences in our spam data set.

kc note: These are great examples of words that, although top occurrences, should not be included as stop words as they provide context. For example, many spam messages will request a victim ‘call’ a number, or attempt to offer ‘free’ items.

### Data Standardizing and Scaling

* + - * Scaling or standardizing the data any further was not necessary since we are dealing with discrete values 0,1.

## Pre-Training Setup

### Define Variables

* + - * Dependent variable ‘y’ was created to host the class discrete value from Class\_num
      * Independent variable ‘X’ was created to host the string value from sms\_clean variable
      * Observation was conducted on the independent and dependent varibles to ensure data alignment.
        + X: 5572 observations, 1 feature
        + y: 5572 observations, 1 feature
      * Since Training and Testing observations match, and features are the expected value, we may proceed with splitting the training and testing data.

### Split Training and Testing Data

* + - * The data was then split and trained into the model.
      * Split data into 75% for training and 25% for testing
      * *X\_train: 4179 observations, 1 feature*
      * *X\_test: 1393 observations, 1 feature*
      * *y\_train: 4179 observations, 1 feature*
      * *y\_test: 1393 observations, 1 feature*
      * *Since Training and Testing observations match, and features are the expected value (missing means 1), the training and testing data was split correctly.*
    - Transform and fit data
      * *The data was then converted to matrix of token counts*
      * *The data was then fit and transformed into trainng and test data.*
      * *Error checks conducted to ensure the vectorizer was completed successfully.*
      * *The data transformation was successful, as both X\_train and X\_test produced the same output for columns [rows x columns]*
        + *X\_test\_dtm*

*1393 x* ***8011***

* + - * + *X\_train\_dtm*

*4197 x* ***8011***

## Algorithms

### K-Nearest Neighbors (KNN)

#### Introduction

##### How it works (Summary)

K-Nearest Neighbors (KNN) is a simple machine learning algorithm used for classification and regression. It works by finding the K data points in the training dataset that are closest to a given data point, and using their labels to predict the label of the new data point.

##### How it works (Steps)

Load the dataset: First, we load the dataset that we want to classify. The dataset consists of examples with known labels.

Choose a value for K: We need to choose a value for K, which represents the number of nearest neighbors that we want to consider.

Calculate distances: Next, we calculate the distance between the new data point and all other data points in the dataset. The most common distance measure used is the Euclidean distance, which is the straight-line distance between two points in a Euclidean space.

Find the K nearest neighbors: We then find the K data points in the training dataset that are closest to the new data point based on the calculated distances.

Predict the label: Finally, we use the labels of the K nearest neighbors to predict the label of the new data point. For classification problems, we typically use majority voting to determine the predicted label. This means that we count the number of neighbors that belong to each class, and predict the class with the highest count.

##### Why it was selected as a model (refer to pros below)

##### Pros

Simple and intuitive, making it easy to understand and implement.

Non-parametric, meaning it does not assume any specific distribution of the data.

Can handle both classification and regression tasks.

Can capture complex nonlinear relationships between features.

Can be easily adapted to handle imbalanced classes or rare events.

##### Cons

* + - * + Computationally intensive, especially for large datasets, as it requires calculating the distances between each point.
        + Sensitive to the choice of distance metric and the number of neighbors chosen.
        + Requires a large amount of memory to store the training data.
        + Can be affected by the curse of dimensionality, where the performance degrades as the number of dimensions increase.
        + Can be affected by noisy or irrelevant features.

#### Steps/ Process Used (in our dataset)

#### Results

### Decision Tree

#### Introduction

##### How it works (Summary)

Decision Trees are a popular machine learning algorithm used for both classification and regression problems. They work by recursively splitting the dataset into subsets based on the most informative feature, until a stopping criterion is met. The result is a tree-like model that can be used to make predictions on new data.

##### How it works (Steps)

1. Load the dataset: First, we load the dataset that we want to classify. The dataset consists of examples with known labels.
2. Choose a root node: We choose a root node for the decision tree based on the feature that is the most informative for distinguishing between the classes.
3. Split the data: We split the dataset into subsets based on the chosen feature. Each subset contains examples with similar values for the chosen feature.
4. Choose the next node: We then choose the next node in the tree by selecting the feature that is most informative for distinguishing between the classes within each subset.
5. Repeat: We repeat the process of splitting the dataset and choosing the most informative feature until a stopping criterion is met. For example, we may stop when all examples in a subset belong to the same class, or when we reach a maximum depth for the tree.
6. Make predictions: Finally, we use the decision tree to make predictions on new data by traversing the tree based on the values of the features in the new data, until we reach a leaf node that corresponds to a class label.

##### Why it was selected as a model (refer to pros below)

##### Pros

Easy to understand and interpret, as the decision process is represented graphically.

Can handle both categorical and numerical data.

Can handle missing data by placing them in a separate branch.

Can be used for both classification and regression tasks.

Can capture nonlinear relationships between features.

##### Cons

Prone to overfitting, especially if the tree is too deep or if there are too many features.

Can be sensitive to small variations in the data, leading to different tree structures.

May not perform well with imbalanced classes or rare events.

Can be biased towards features with many levels or values.

#### Steps/ Process Used (in our dataset)

#### Results

### Naive Bayes

#### Introduction

##### How it works (Summary)

Naive Bayes is a simple and effective machine learning algorithm used for classification problems. It works by calculating the probability of a new data point belonging to each class based on the probabilities of its features given the class, and then selecting the class with the highest probability.

Naive Bayes is a statistical classification algorithm that is based on the Bayes' theorem of conditional probability. In classification, the goal is to assign a class label to a given data instance based on its features. Naive Bayes assumes that the features are conditionally independent given the class label, which means that the presence or absence of one feature does not affect the presence or absence of any other feature.

Naive Bayes works by first estimating the prior probability of each class label, which is the probability of the class label occurring in the dataset without considering any of the features. Then, for each feature, the algorithm calculates the conditional probability of that feature given each class label. This is done by estimating the probability distribution of each feature for each class label.

Once the prior and conditional probabilities have been estimated, the algorithm can use Bayes' theorem to calculate the posterior probability of each class label given the features of a new data instance. The class label with the highest posterior probability is then assigned to the new instance. Naive Bayes is called "naive" because it makes the simplifying assumption of feature independence, which is often not true in practice. Despite this simplification, Naive Bayes can perform surprisingly well on a wide range of classification tasks, especially when the number of features is large relative to the amount of training data available. Naive Bayes is also computationally efficient and can be trained quickly even on large datasets.

Multinomial Naive Bayes is a variant of the Naive Bayes algorithm that is commonly used for text classification tasks where the features represent word frequencies. In this variant, the algorithm models the probability distribution of the counts of each word in each class, which is often referred to as the multinomial distribution. Multinomial Naive Bayes assumes that the features are discrete counts, which represent the number of times each word occurs in a document. It also assumes that the frequency of each word is conditionally independent given the class label. To classify a new document, Multinomial Naive Bayes first calculates the prior probability of each class label based on the training data. Then, for each word in the document, the algorithm calculates the conditional probability of that word given each class label. This is done by estimating the probability distribution of the counts of each word for each class label.

Finally, the algorithm uses Bayes' theorem to calculate the posterior probability of each class label given the counts of the words in the new document. The class label with the highest posterior probability is then assigned to the new document. Multinomial Naive Bayes is a popular algorithm for text classification because it can handle large vocabularies and sparse data efficiently. However, it is not well-suited for tasks where the features are continuous or where there is strong dependence among the features.

##### How it works (Steps)

Load the dataset: First, we load the dataset that we want to classify. The dataset consists of examples with known labels.

Calculate class probabilities: We calculate the prior probability of each class by counting the number of examples in each class and dividing by the total number of examples.

Calculate feature probabilities: For each feature in the dataset, we calculate the probability of the feature given each class by counting the number of examples in the class that have that feature and dividing by the total number of examples in the class.

Calculate joint probabilities: We calculate the joint probability of each data point belonging to each class by multiplying the prior probability of the class by the probabilities of its features given the class.

Select the class with the highest probability: Finally, we select the class with the highest joint probability as the predicted class for the new data point.

##### Why it was selected as a model

Naive Bayes is a widely used classification algorithm that is simple, fast, and accurate. One of the advantages of using Naive Bayes for spam filtering is that it can effectively handle high-dimensional data with a large number of features. In the case of spam vs ham classification, the features might include the presence or absence of certain words, the frequency of specific characters or patterns, or other characteristics of the email message.

Naive Bayes is well-suited for this task because it assumes that the features are conditionally independent given the class label, which is a reasonable assumption for many spam filtering applications. This means that the algorithm can quickly and accurately identify spam messages based on their unique characteristics, without being affected by the presence or absence of other features.

Another advantage of Naive Bayes is its simplicity and ease of implementation. The algorithm only requires a small amount of training data to accurately classify new instances, and it can be trained quickly even on large datasets. This makes it an attractive choice for real-world applications where computational resources are limited or where data is constantly changing.

However, there are also some limitations to using Naive Bayes for spam filtering. One potential issue is that it may be susceptible to overfitting if the training data is not representative of the overall population of messages. In addition, Naive Bayes assumes that the features are independent, which is not always true in practice. Finally, the algorithm may not be effective in identifying new types of spam messages that have not been seen before, as it relies on patterns in the training data to make predictions.

##### Pros

* + - * + Fast and efficient, making it suitable for real-time applications.
        + Requires a small amount of training data to estimate the parameters necessary for classification.
        + Handles high-dimensional data well, making it useful for text classification and spam filtering.
        + Performs well when the assumption of independence between features is not violated.

##### Cons

* + - * + Assumes that all features are independent, which may not be true in some cases.
        + Can suffer from the problem of zero probability, where a feature has never been observed in a particular class, resulting in a probability of zero.
        + May not perform well with imbalanced classes or rare events.
        + May not work well with noisy data.

#### Steps/ Process Used (in our dataset)

##### Create Naïve Bayes Model

Using sklearn we selected the Multinomial naïve bayes model

The instance was created and stored in the variable ‘nb’

Model was then fit with the training data

#### Results

##### Make Predictions

The variable y\_pred\_NB was created to store the predictive data testing set in which we return the first 15 occurrences.

##### Model Metrics

Compare First 15 Values

Predictions: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]

True Data: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]

Confusion Matrix

|  |  |  |  |
| --- | --- | --- | --- |
| [[ | 1200 | 7 | ] |
| [ | 14 | 172 | ]] |

Accuracy Score

0.9849246231155779

##### Metrics Analysis

Confusion Matrix

|  |  |  |  |
| --- | --- | --- | --- |
| [[ | 1200 Predicted HAM Correctly | 7 Predicted SPAM incorrectly, was actually HAM | ] |
| [ | 14 Predicted HAM incorrectly,  was actually SPAM | 172 Predicted SPAM Correctly | ]] |

Model Analysis

As we can see, the Naive Bayes method is quite successful at correctly identifying Spam and Ham, achieving an accuracy score of 98.49%. This is shown first by displaying the first 15 values - the fifteenth message (at array location 14) is correctly identified as spam. Furthermore, the Confusion Matrix above correctly identifies 172 instances of Spam and 1200 instances of Ham. Since the model is not perfect, there were also 7 instances of Ham incorrectly identified as Spam, and 14 instances of Spam incorrectly identified as Ham. A snippet of each of these are shown below.

# Results [review of all methods]

* + performance
  + precision
  + accuracy

# Conclusion

* + Algorithms
    - KNN
      * In summary, KNN is a simple and intuitive algorithm that uses distances between data points to make predictions. It is often used in problems where there is a clear separation between classes, but can be less effective when classes overlap or when the number of features is large. It is also important to choose an appropriate value for K and to normalize the data to ensure that features are on the same scale.
    - Naïve Bayes
      * Naive Bayes is "naive" because it makes a simplifying assumption that the features are independent of each other given the class, which may not be true in practice. Despite this assumption, Naive Bayes can be very effective in practice, particularly for text classification problems such as spam filtering or sentiment analysis. It is also computationally efficient and requires only a small amount of training data.
    - Decision Trees
      * In summary, decision trees are a simple and interpretable algorithm that works by recursively splitting the data based on the most informative feature. They are often used in problems where there are complex decision boundaries, but can be prone to overfitting and are sensitive to noisy data. It is also important to tune the parameters of the decision tree, such as the maximum depth and minimum samples per leaf, to prevent overfitting.
  + Cyber Security

# References

* + A
  + A