Classification of Classic Cipher Methods Using Machine Learning

CS 271 Project Report

Presented to

Dr. Mark Stamp

Department of Computer Science

San Jose State University

In Partial Fulfillment

Of the Requirements for the Class

CS 271

By

Lilou Sicard-Noel

And

Juhi Malkani

November 2023

ABSTRACT

Classification of Classic Cipher Methods Using Machine Learning

By Lilou Sicard-Noel and Juhi Malkani

Multitudes of different machine learning and deep learning techniques have been successful at distinguishing various label data. Examples of popular learning techniques include K-Nearest Neighbors, SVM, CNN, and Naive Bayes. In this study, we applied the above-mentioned techniques to classify classic cipher methods. The encryption methods used are Simple Substitution, One-time pad, and Double-Transposition. The machines were tested with three different features: the letter count, the bigram distribution, and both features simultaneously. The result of the study demonstrates that the best feature to classify the encryption method based on the cipher is the letter count.

Table of Contents

1. Introduction

The principle of Information Security is founded on cryptography. Cryptography or "secret codes" provide confidentiality and integrity [1]. Cryptanalysis, the art of breaking code, has been practiced since the creation of cryptography. Examples of cryptosystems include simple substitution, double transposition, and One-time pad.

Each generation of ciphertext creates a variation of letter sequences. This study aims to find if a general pattern exists in cipher encrypted with the same method. More specifically, the research focuses on classic encryption techniques. Those are simple substitution, double transposition, and One-time pad. Research has been done in the past to classify classic ciphers by applying machine learning techniques [2, 3]. However, those studies do not compare traditional and deep learning methods as their main point. One new method used in this paper and not seen in the study above is Naive Bayes.

This research is limited to finding which machine learning method produces the best cipher classification results. There is also a side research as to which features among the one extracted produce the best results (an individual feature or a combination of the features). The approach chosen is to let the highest result win. This method is used for testing the techniques' parameters and determining the final result.

2. Background

In this section, we introduce the various Machine Learning algorithms that appear in this paper.

2.1. K-Nearest Neighbors

The K-Nearest Neighbors algorithm is a machine learning algorithm known for its simplicity and effectiveness in classification and regression problems. KNN operates on the rules of proximity, i.e., it identifies the majority or average of the k-nearest data points for a given input in the feature space. It has a wide range of applications because of its intuitiveness and straightforward implementation. KNN's performance depends on the choice of k and distance metric, which gives it the ability to adapt to various scenarios and datasets. Despite its simplicity, KNN is highly robust, even in situations where the decision boundaries are nonlinear and complex. KNN is a highly valuable machine learning algorithm for its ease of use and ease of interpretation.

2.2. Support Vector Machine

Support Vector Machines (SVM) are used for regression and classification. It works by finding an optimal hyperplane that separates the data into different classes. SVM aims to maximize the margin between the data points of distinct classes. This helps in better classification of testing data. SVM can handle both linear and non-linear classification problems, which is done using kernel functions. Even with high-dimensional datasets, SVM is resilient to overfitting. SVM has a wide range of applications, including bioinformatics, image recognition, text classification, and a wide variety of machine learning challenges, as it handles all of them efficiently.

2.3. Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a deep learning model that is designed to deal with image classification efficiently. The typical architecture of a CNN includes convolutional layers that apply filters to detect features, pooling layers that reduce the spatial size of the representation, and fully connected layers that perform classification. This structure enables CNNs to handle high-dimensional data efficiently, making them a powerful tool for image classification, object detection, and similar tasks in computer vision.

2.4. Naive Bayes

The Naïve Bayes classifier is a popular supervised machine learning algorithm used for classification tasks such as text classification. It belongs to the family of generative learning algorithms, which means that it models the distribution of inputs for a given class or category. The algorithm is based on applying Bayes' theorem with the 'naive' assumption of conditional independence between every pair of features given the value of the class variable. This simplicity allows for efficient computation and robust performance with small datasets and high-dimensional feature spaces. Naive Bayes classifiers work well in scenarios where the independence assumption holds.

3. Methodology

3.1. Dataset

For this research, experiments using different lengths of ciphertext are conducted. Using ChatGPT, texts of 100, 200, and 300 character lengths were generated. All these texts are meaningful sentences. Each sentence was then passed to every encryption method, i.e., One-time pad, double transposition cipher, and substitution cipher. Each of these encrypted texts was then stored in the data file with the key of the method that was used to encrypt them. This was repeated about 250-300 times to give us a text file of about 300 lines fed to all the machine learning algorithms split into training and testing data.

3.2. Feature Extraction

The Feature Extraction methods presented here were inspired by various sources [1, 3, 4]. Since the number of characters varies by cipher, we decided to use two main features: the letter frequency and the sets of bigrams present in the cipher text. Each feature was normalized so that the length of the cipher does not impact the result in any significant way. All those extractions were performed in Python using the NumPy library [5].

3.2.1. Letter Frequency

The letter frequency in a ciphertext refers to the count of each letter's occurrence within that encrypted text. This frequency is a crucial element in cryptanalysis, as it can provide insights into the nature of the encryption algorithm and potentially help decrypt the message. This representation allows for easier comparison of the ciphertext, which can be instrumental in identifying patterns or anomalies that might suggest a decryption method.

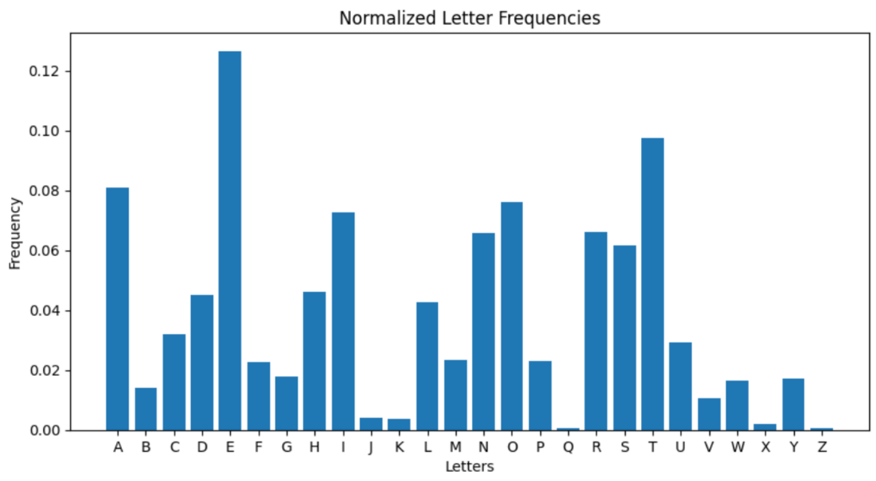


Figure 1: Letter Frequency for the first 10,000 characters of Brown Corpus

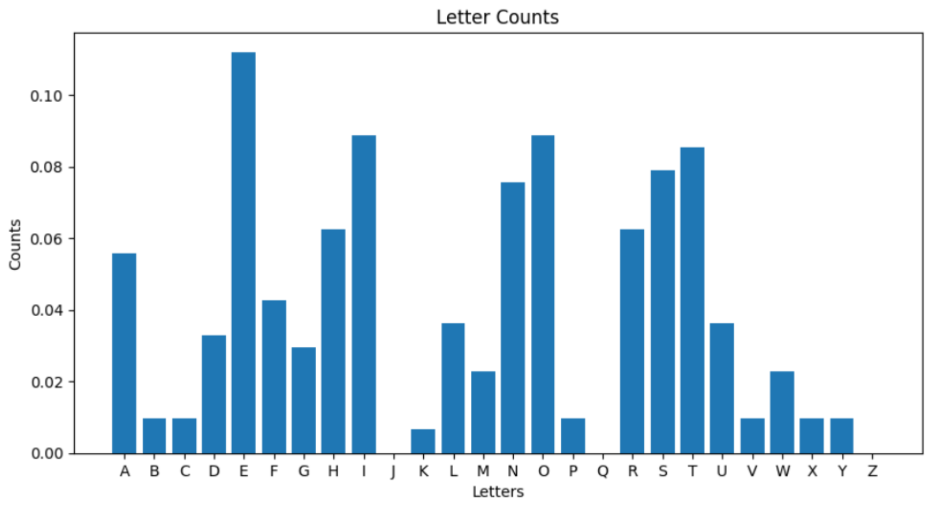


Figure 2: Letter Frequency for a cipher encrypted with double transposition method

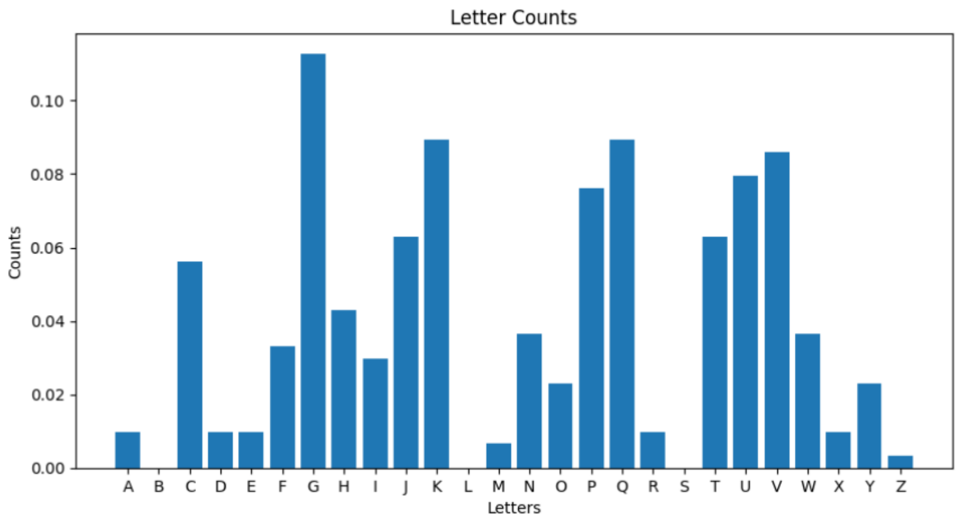


Figure 3: Letter Frequency for a cipher encrypted with the simple-substitution method

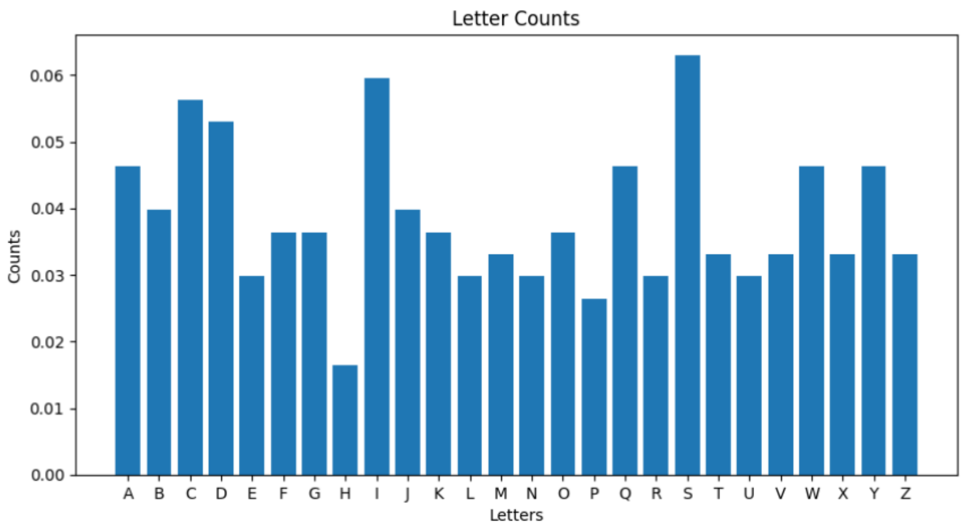


Figure 4: Letter Frequency for a cipher encrypted with the One-Time pad method

3.2.2. Bigram

The frequency of bigram sequences in a ciphertext refers to the count of the occurrence of each pair of adjacent letters within that encrypted text. Bigrams are two-letter combinations, like 'TH,' 'EN,' or 'RE.' Analyzing the frequency of these bigrams is a significant aspect of cryptanalysis, especially when trying to break more complex ciphers. Such patterns that might not be apparent when looking at the raw data can provide valuable insights for decrypting or understanding the structure of the ciphertext.

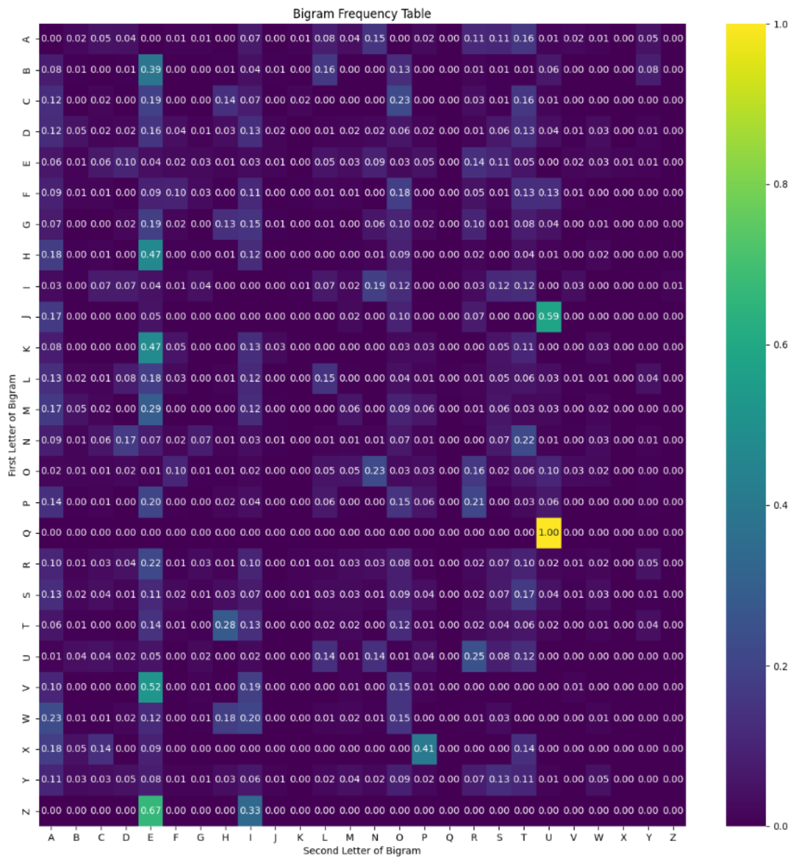


Figure 5: Bigram table for the first 10,000 characters of Brown Corpus

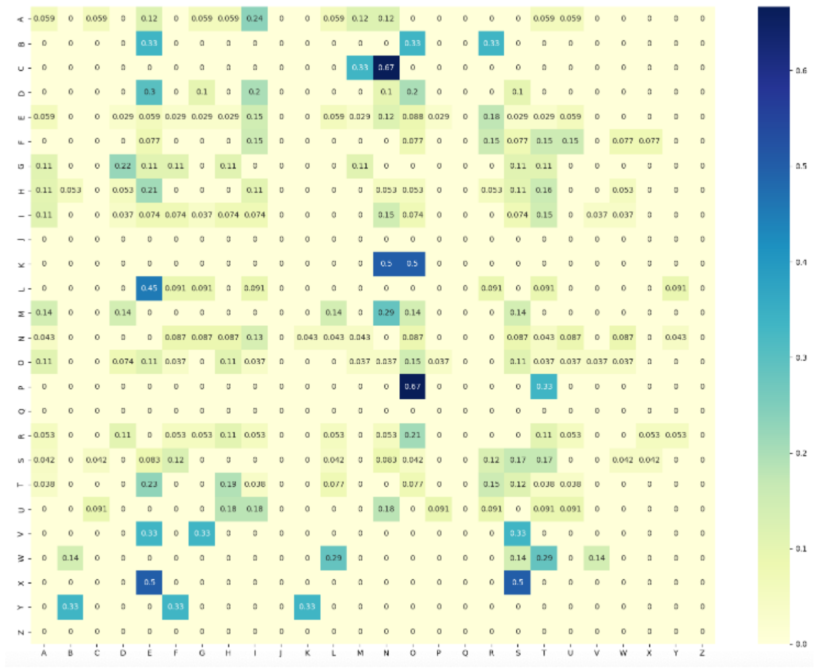


Figure 6: Bigram table for a cipher encrypted with the double transposition method

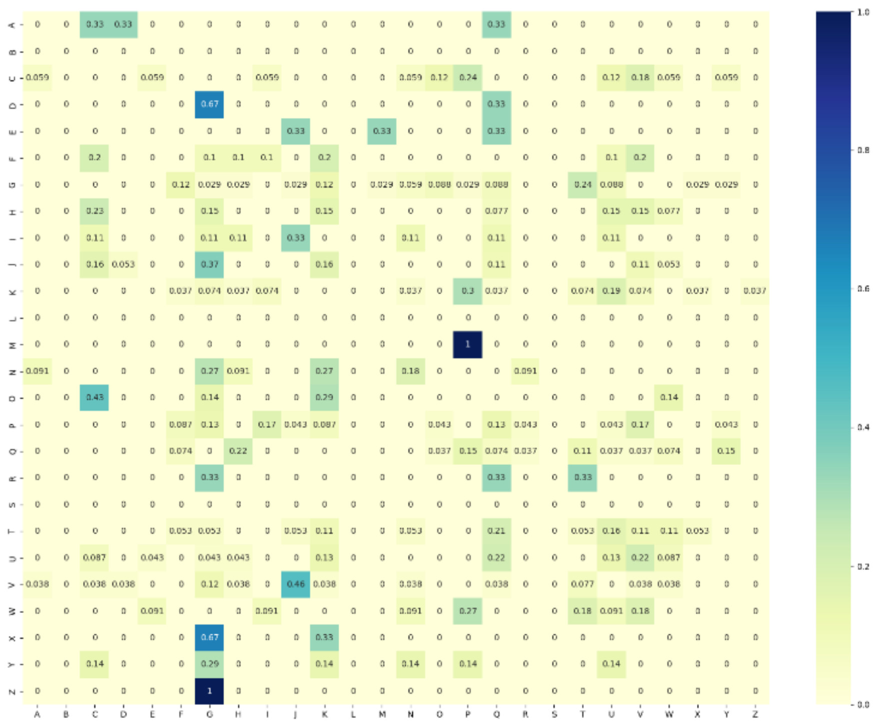


Figure 7: Bigram table for a cipher encrypted with the simple-substitution method

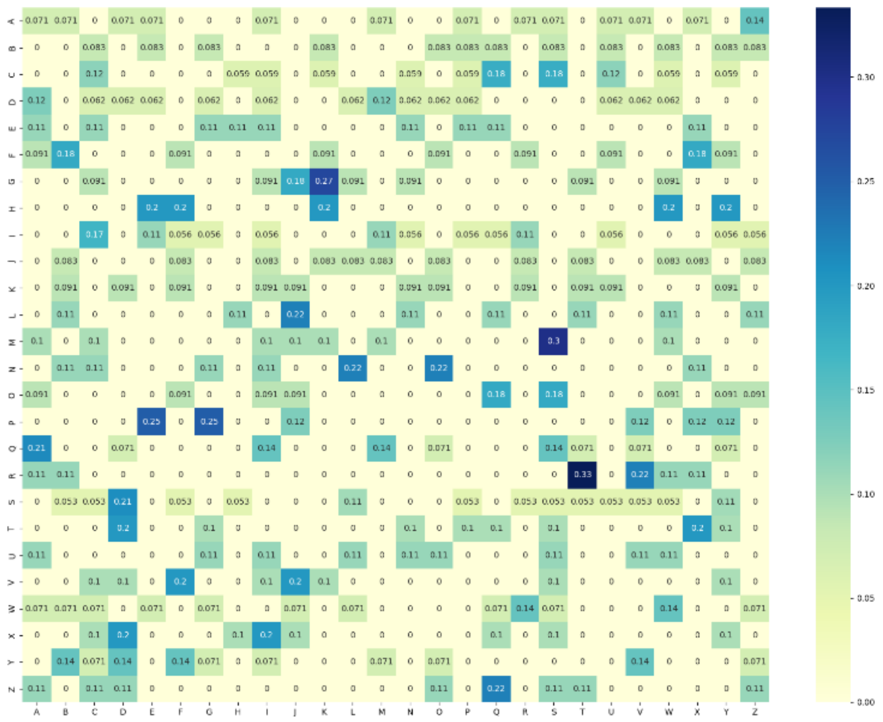


Figure 8: Bigram table for a cipher encrypted with the One-Time pad method

4. Experiment and Result

4.1. K-Nearest Neighbors Training

We used three different KNNs in this study. They were generated using either or both features listed above. The three different KNNs used in this research have the same hyperparameters since they perform the best under the same parameters. Every model is trained under the same split i.e., 75% of the data for the training and 25% of data for the testing purposes. The KNN program is built with the Scikit-learn library in Python [6]. The summary of the hyperparameters tested and used is presented in the table below.

Hyperparameter	Values
n_neighbors	**3**,4,5
Distance (cross-validation)	Euclidean, **Manhattan**, Minkowski

Figure 9: Hyperparameters tested and used for the KNN

4.1.1. Results

KNN gives the best result when the model is built on the letter-count feature for classification with an accuracy of about 90%. With only bigrams, it provides the worst accuracy in the classification of ciphers. As expected, the one-time pad and the simple substitution cipher are the most confused since they create a bigger change in the bigram and letter variation. Interestingly, KNN works the best with neighbors equal to 3 when the number of classes to be classified in is 3 as well.

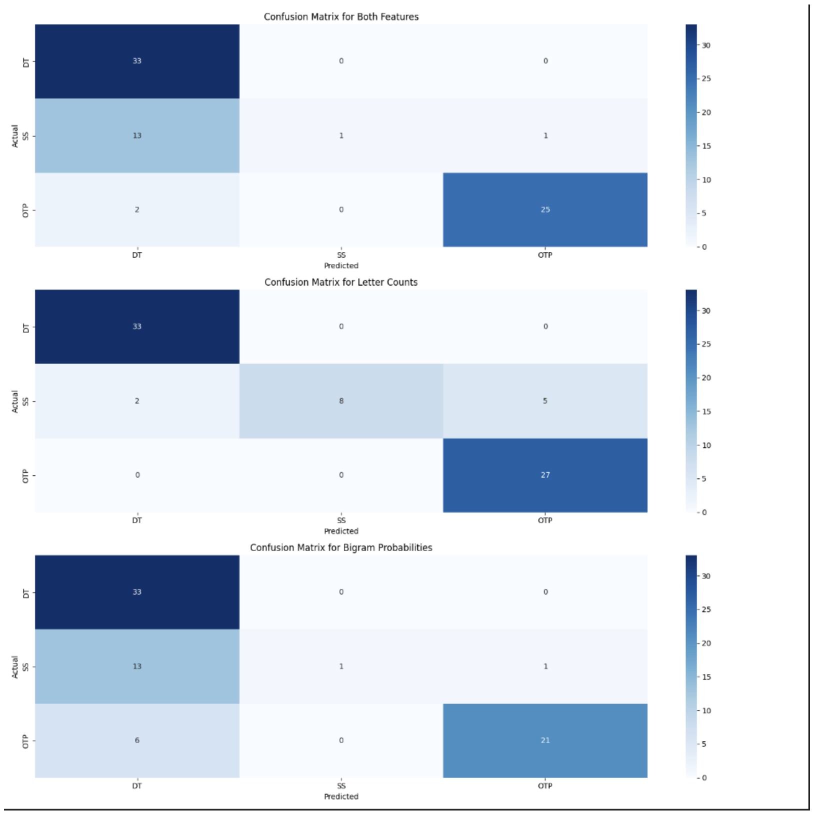
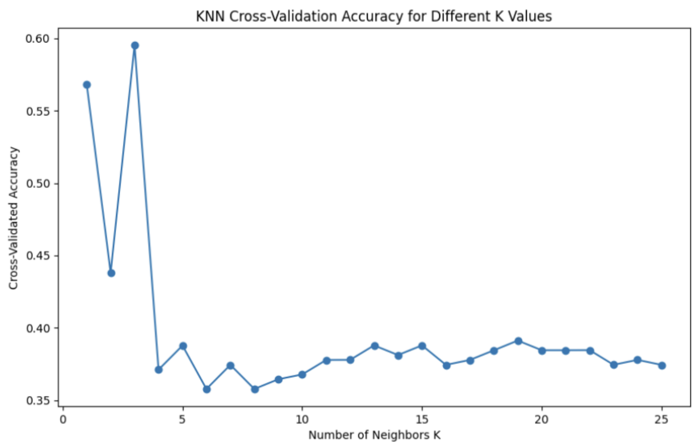
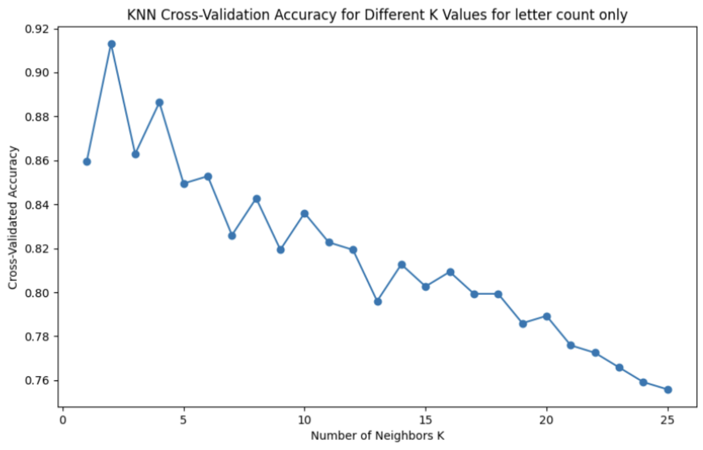


Figure 10: Confusion Matrix for all three KNN





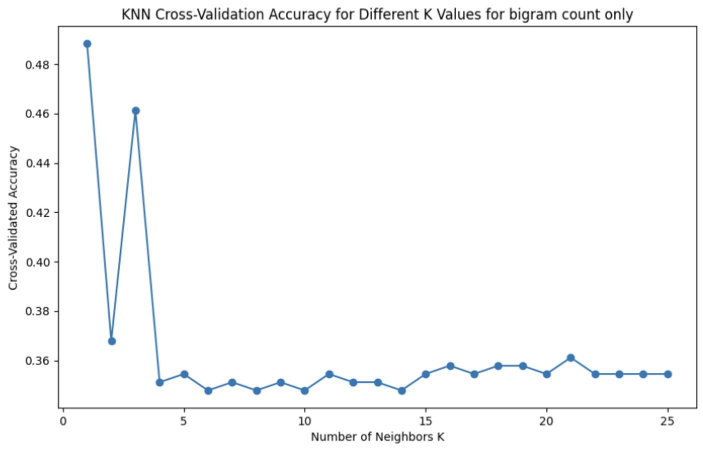


Figure 11: Graph of the result for all three K-Nearest Neighbors over K neighbors

4.2. SVM training

The three different SVMs used in this study had the same hyperparameters since they all produced a maximum result under the same parameters. Each model is trained and tested using an 80-20 split. That is, 80% of the data is used for training, while the remaining 20% of the data is used for testing.

A summary of the hyperparameters tested and used is presented in the table below. Each SVM operates either one or both of the features in the training and testing process. The SVM program is in Python and built with the Scikit-learn library [6].

Hyperparameter	Values
C	0.5, 1, **5**, 10
Kernel	Linear, Poly, **RBF**, sigmoid
Degree	**2**, 3, 4, 5

Figure 12: Hyperparameters tested and used for SVM

4.2.1. Results

The SVM code produced a very steady result throughout the experiment. Interestingly, it performs better when only the letter counts feature is used in the classification. The SVM that uses only letter count is one of the highest-performing machines studied. As expected, the one-time pad and the simple substitution cipher are the most confused since they create a bigger change in the bigram and letter variation.

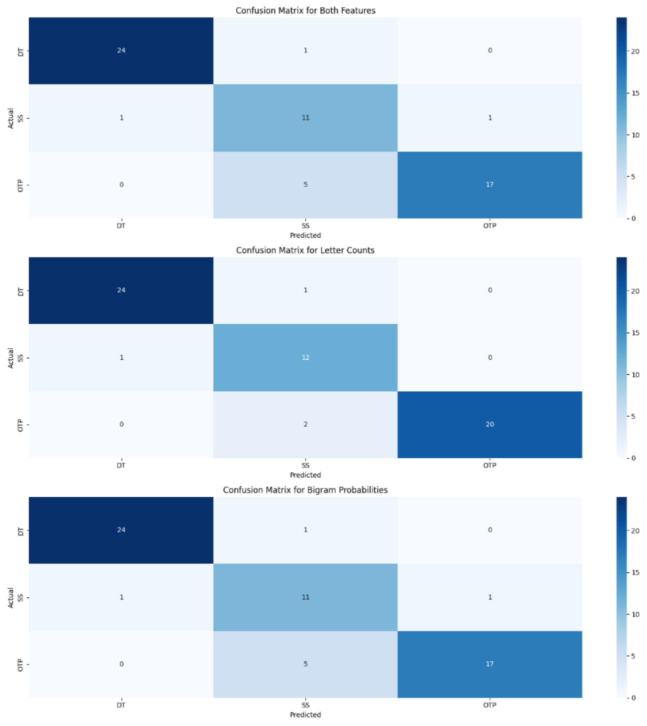


Figure 13: Confusion Matrix for all three SVM

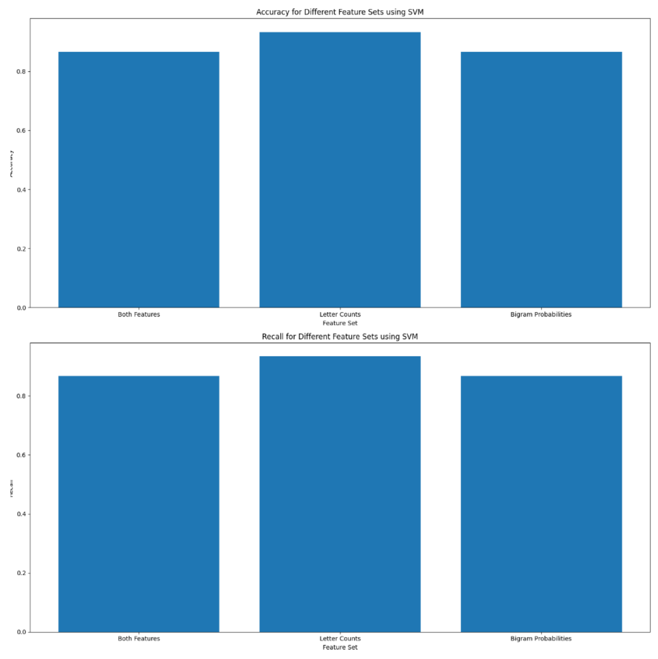


Figure 14: Histogram of Accuracy and Recall for all three SVM

4.3. CNN training

The research conducted only involves one CNN training. Our CNN uses both features at the same time. The CNN is trained and tested using a 75-25 split over 100 epochs. The architecture of the CNN is defined using the Sequential model from the Keras library. The model consists of a 1D convolutional layer with 32 filters and a kernel size of 2, followed by a max pooling layer with a pool size 2. The output from the max pooling layer is then flattened and passed through a dense layer with 64 nodes. The final layer is another dense layer with a number of nodes equal to the number of cipher types. It uses a softmax activation function to output a probability distribution over the cipher types. The CNN program is in Python and built with the NumPy, Scikit-learn, and TensorFlow libraries [5, 6, 7].

Hyperparameter	Values
Learning Rate	0.0001, **0.001**, 0.01
Optimizer	**Adam**, RMSProp, Adagrad, Adadelta, Nadam, Ftrl

Figure 15: The hyperparameters tested and used in the CNN

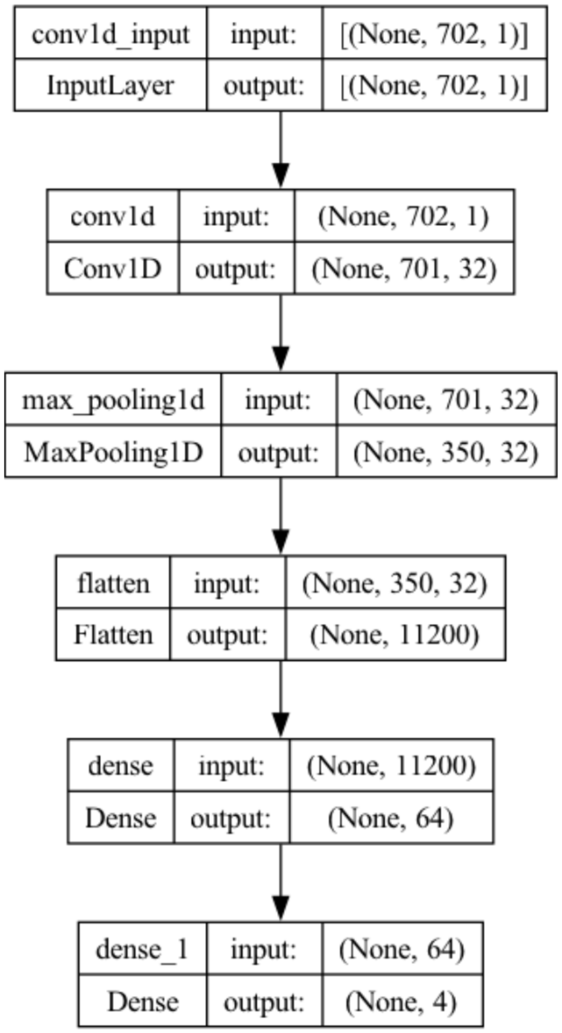

conv1d_input | input: | [(None, 702, 1)]
InputLayer | output: | [(None, 702, 1)]
conv1d | input: | (None, 702, 1)
Conv1D | output: | (None, 701, 32)
max_pooling1d | input: | (None, 701, 32)
MaxPooling1D | output: | (None, 350, 32)
flatten | input: | (None, 350, 32)
Flatten | output: | (None, 11200)
dense | input: | (None, 11200)
Dense | output: | (None, 64)
dense_1 | input: | (None, 64)
Dense | output: | (None, 4)

Figure 16: The layers of the CNN

4.3.1. Results

The CNN is the only method that produced consistent and high results with both features employed at the same time. It is the best method to identify the Double Transposition cipher in this study. However, the validation accuracy reaches its limit very early in the training process. 100 epochs create a high loss for a limited increase in accuracy.

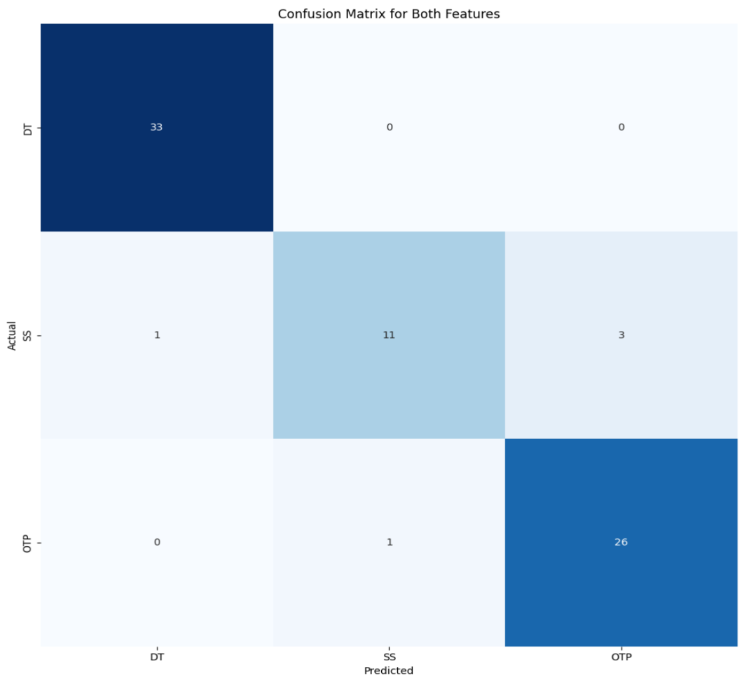


Figure 17: The CNN Confusion Matrix

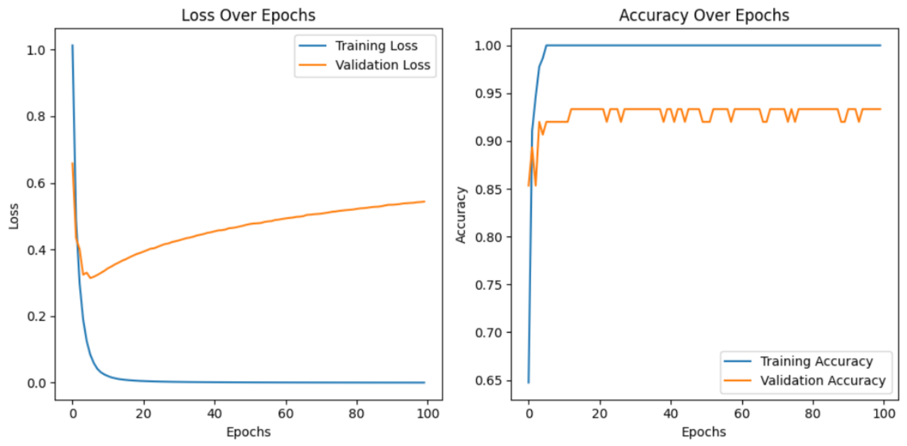


Figure 18: Graph of the loss and accuracy of the CNN

4.4. Naive Bayes training

Naive Bayes has been used in this study in a similar method to the one used for the SVM. Three different machines, all with the same hyperparameters, were trained with either one or both features. Something interesting to note is that the Priors parameter did not affect the accuracy of any of the machines. All machines are trained and tested using a 75-25 split. The Naive Bayes machines were coded in Python using the Scikit-learn library [6].

Hyperparameter	Values
Priors	Default values
Var- Smoothing	0.01, **0.1**, 1

Figure 19: Hyperparameter tested and used for Naive Bayes

4.4.1. Results

The Naive Bayes results are the most dispersed. The accuracy result varies from the overall best of 95% accuracy to the lowest deep-learning accuracy of 81%. Once again, the machine using only the letter count as a feature produced the best results. This could indicate that the feature has better results overall. Interestingly, it performs worst at identifying Double-Transposition ciphers when only the letter count is involved. The nature of the Double-Transposition ciphers makes the letter count in those ciphers equal to the plaintext one. A deeper dive into the mechanism of Naive Bayes could be interesting to perform to see if the explanation for this observation is identifiable.

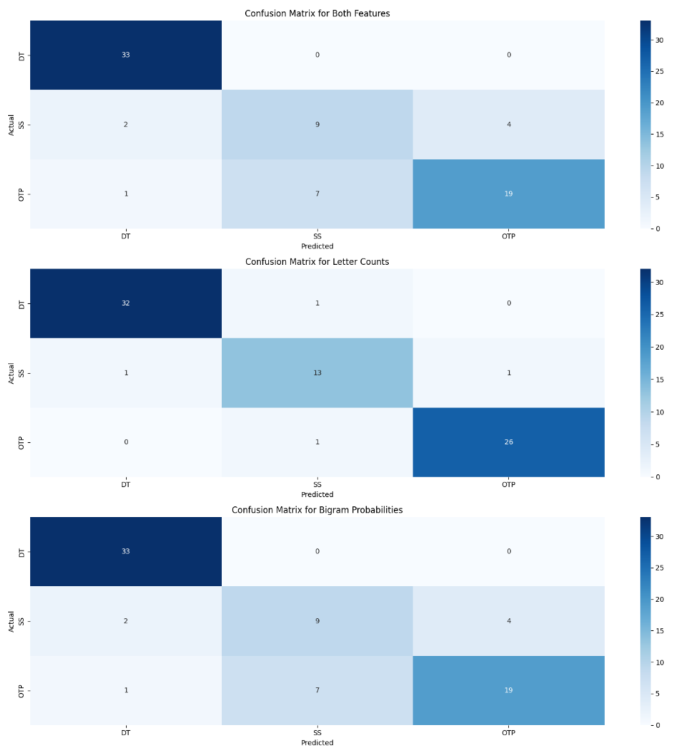


Figure 20: Confusion Matrix for all three Naive Bayes

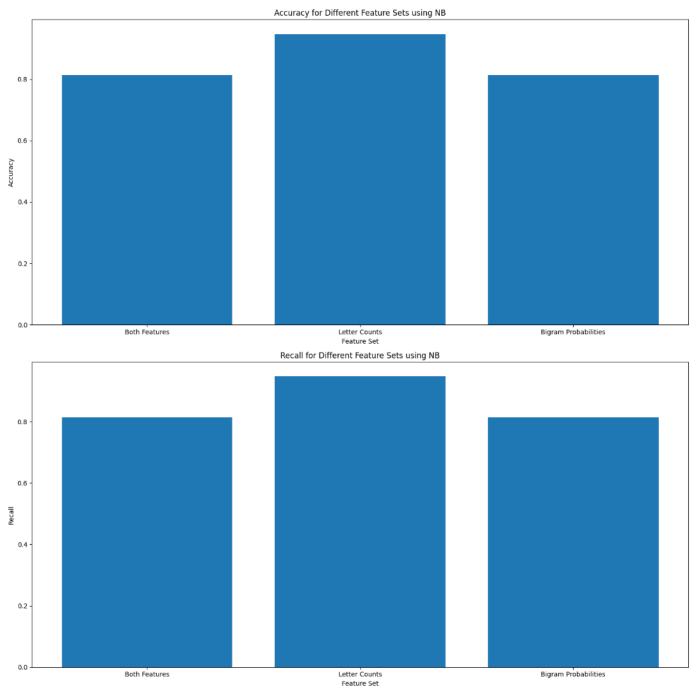


Figure 21: Histogram of Accuracy and Recall for all three Naive Bayes

5. Conclusion

While the study conducted did not produce any extraordinary results, we can still see a pattern in the result. The result of the study demonstrates that the best feature to classify the encryption method based on the cipher is the letter count. It is still unclear why this pattern is the only feature that produces consistently good results. Overall, the SVM is the technique that produced the most consistent result across all features. The best results were produced by the Naive Bayes which only had the letter count as a feature.

Technique	Accuracy
K-Nearest Neighbors - Letter Count	0.90
K-Nearest Neighbors - Bigram	0.73
K-Nearest Neighbors - Both Features	0.78
CNN	0.93
SVM - Letter Count	0.93
SVM - Bigram	0.87
SVM - Both Features	0.87
NB - Letter Count	0.95
NB - Bigram	0.81
NB - Both Features	0.81

Figure 22: Table of the Accuracy Result

For future work, research could see if the result pattern of letter count is reproducible across other types of encryption method. It could also be interesting to add new algorithms to the one already in this study.

References

[1]	Mark Stamp. 2011. Information security: principles and practice (2nd ed ed.). Wiley, Hoboken, NJ.
[2]	Nivedhitha Ramarathnam Krishna. Classifying classic ciphers using machine learning. Master’s thesis, Department of Computer Science, San Jose State University, 2019.
[3]	Brooke Dalton. Classifying World War II Era Ciphers with Machine Learning. Master’s thesis, Department of Computer Science, San Jose State University, 2023.
[4]	Mark Stamp. 2023. Introduction to machine learning with applications in information security (Second edition ed.). CRC Press, Boca Raton.
[5]	Charles R. Harris, K. Jarrod Millman, Stéfan J. Van Der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. Van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández Del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. Nature 585, 7825 (September 2020), 357–362. https://doi.org/10.1038/s41586-020-2649-2
[6]	F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research 12, (2011), 2825–2830.
[7]	Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Retrieved from https://www.tensorflow.org/