C111: Machine Learning For The Physical Sciences

Spam Email Classification With Machine Learning

Emma Allen

December 2023

Abstract

This project aims to classify emails as spam using five supervised learning algorithms to determine which one is the most accurate and precise. The algorithms used in this study are Logistic Regression, Decision Tree, K-Nearest Neighbors (KNN), Support Vector Machines (SVM), and Multi-Layer Perceptron (MLP) classifiers, implemented through Scikit-Learn and PyTorch. It is found that MLP classifier (Scikit-Learn) is the most accurate and precise for classifying emails as spam. This is closely followed by logistic regression and SVM.

Contents

1 Introduction

The first known spam electronic mail was sent on 3rd May 1978 to several hundred users by a marketing manager for digital equipment Corporation [1]. Over the past 40 years, the volume of spam emails has only increased, reaching a point where spam accounted for 48% of all emails sent in 2022 [2]. This project aims to use classify emails as spam through utilising five supervised learning algorithms to conclude which one is the most accurate and precise. The algorithms used are

1. Logistic Regression
2. Decision Tree
3. K-Nearest Neighbours (KNN)
4. Support Vector Machines (SVM)
5. Multi-Layer Perception (MLP) classifier, implemented in
   (a) Scikit-Learn
   (b) PyTorch

2 Data

The dataset utilised in this project is from the UCI Machine Learning Repository [3]. Created in 1999, the dataset encompasses 4601 instances, with each instance representing an email. Consisting of 57 continuous features and one binary class label that denotes whether an instance is spam, the dataset is an imbalanced distribution with spam instances accounting for 39.4%. Features of the dataset include word and character frequencies. The data was pre-processed by adding column names and turning into a pandas data frame. Figure 1 shows the whole distribution of spam across instances and Figure 2 shows the distribution for a single feature across all instances. This information allows one to determine which features are good differentiators of spam. Notably one finds that the word George and area code 650 are indicators of non-spam.

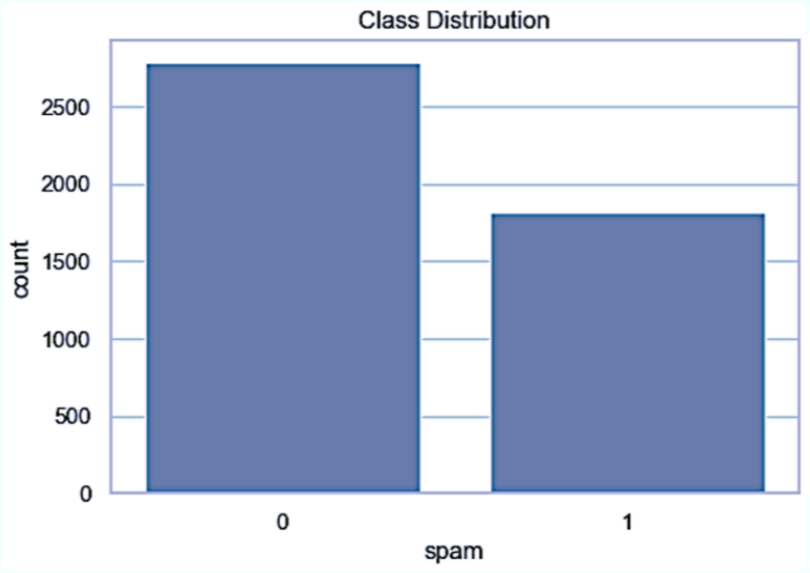

Figure 1: Distribution of spam within the dataset.

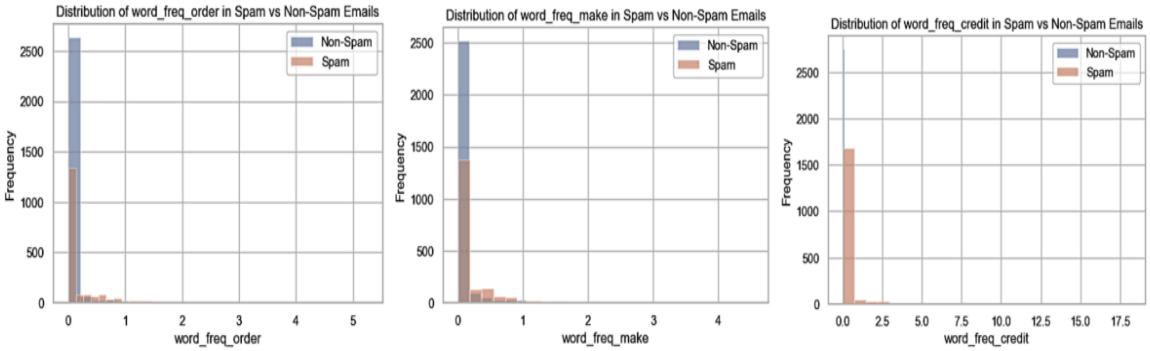

Figure 2: Distribution of Spam across three features

3 Performance Metrics

Throughout this project, the definition of accuracy is taken to be to be a measure of the overall correctness of the model across all classes and precision is defined as a measure of the accuracy of the positive predications made by the model [4]. Although the Scikit-Learn package includes an accuracy measure, this measure performs less well with an imbalanced dataset, furthermore due to the PyTorch model the decision was made to forgo this metric. Instead, the following metrics are used.

Learning Curve

The learning curve provides a graphical representation to a model's performance as it is exposed to more training data. This is useful to gauge whether a model has high bias or variance, hence shows whether a model is under or over fitting the data.

Confusion Matrix

The Confusion matrix shows one a visual breakdown of the true positives, true negatives, false positives, and false negatives. A successful algorithm is measured by having minimal false positive and false negatives.

Receiver Operating Characteristic (ROC) Curve
The ROC curve graphically represents the performance of a binary classification model [5]. The ROC curve plots the true positive rate (TPR) against the False Positive Rate (FPR) which are defined as

$$TPR = \frac{TP}{TP + FN},$$

$$FPR = \frac{FP}{TP + FN}.$$

Matthews Correlation Coefficient (MCC)
This metric evaluates the performance of binary classification models by considering the true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN) through the formula

$$FPR = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}.$$

A high MCC suggests the model effectively captures true positives while minimising false positives and false negatives. Because it considers all four quadrants of the confusion matrix, it is less sensitive to class imbalance. In these scenarios, the MCC penalises misclassification in both minority and majority classes, hence aligning with the project's precision definition.

Cohen's Kappa
It measures the agreement between predicted and actual classifications, through the formula [6]

$$\kappa = \frac{\text{Observed Agreement} - \text{Expected Agreement}}{1 - \text{Expected Agreement}}.$$

Ranging from -1 to 1 where 1 indicates perfect agreement and -1 signifies perfect disagreement. For an imbalanced dataset, this metric is useful as it evaluates the agreement between model predictions and true classifications while accounting for the possibility of random agreement, aligning with the definition of accuracy for this project.

4 Modelling Considerations

All models employ k-fold cross-validation to reduce over-fitting and ensure each fold has a representative distribution of minority and majority class samples.

Logistic Regression
Given the dataset's imbalance (39.4% spam), class weights are adjusted (0.6 for non-spam, 0.4 for spam). The Newton Conjugate Gradient Optimization Method is chosen as the optimisation algorithm due to its efficiency in moderate-sized datasets with many features and $O(kN^2)$ time complexity [7], where N is the number of iterations required for convergence and N is the number of features.

L2 regularisation is used as the penalty term. This prevents over-fitting by indicating an expectation that all features contribute. This ensures a balanced consideration of each feature's impact on classification.

Decision Tree Classification

After experimenting with various values, the `max_depth` parameter was established at 12, leading to optimal accuracy. This parameter governs the maximum depth of the tree (m), impacting its computational efficiency. This is shown by a time complexity of $O(np\log(p))$ for tree construction and $O(\log(m))$ for predictions [8], where n is the number of samples and p is the number of features. This adjustment was particularly significant as, prior to refining the model, execution times were too long.

To prevent over-fitting, the `ccp_alpha` parameter was set to 0.001. This cost-complexity parameter plays a key role in pruning the tree, ensuring a balance between model performance, and avoiding over-fitting.

KNN

Five neighbours were chosen, determined through trial and error, ensuring a balance between model complexity and predictive accuracy. This in combination with cross validation attempts to mitigate the curse of dimensionality [9].

The Manhattan distance metric was adopted for measuring the proximity between instances in the feature space, due to its suitability with non-uniformly scaled features. Furthermore, the algorithm for nearest neighbours was set to `ball_tree` due to its favourable time complexity and efficiency for this dataset.

SVM

The Radial Basis Function (RBF) kernel was chosen because of its ability to handle complex decision boundaries and high dimensions, particularly important for this dataset. The regularisation parameter C is set at 30 and the gamma parameter, which defines the influence of a single training example, was set at 0.0001. Together these prevent over-fitting and balance computational efficiency.

MLP - Scikit-Learn

By calculating two-thirds of the input layers, plus the output layer, then testing for optimality, the number of hidden layers was chosen to be 39. L2 regularization ($\alpha = 0.1$) is utilised to mitigate overfitting, and a batch size of 32 was chosen to control the number of samples used in each iteration, leading to efficient model training. To counteract overfitting, an L2 regularisation parameter and batch size of 32 was utilised.

MLP – PyTorch

A neural network architecture is designed with three hidden layers, each utilising the Rectified Linear Unit (ReLU) activation function to capture complex patterns. The output layer uses a sigmoid activation function, as it is well-suited for binary classification tasks by providing probability-like outputs.

To prevent over-fitting and enhance computational efficiency, early stopping is implemented, halting training when the number of epochs with no improvement reaches 35. We picked a binary cross-entropy loss function because it is good at measuring how different the predicted

probabilities are from the actual class labels.

5 Results

5.1 Learning Curve

Figure 3 shows learning curves for each model implemented through Scikit-Learn and Figure 4 shows the MLP Classifier (PyTorch) Model. In each case the testing and training curves converge at a high f1 score indicating there is no bias or high variance, hence no model is under or over-fitting the data

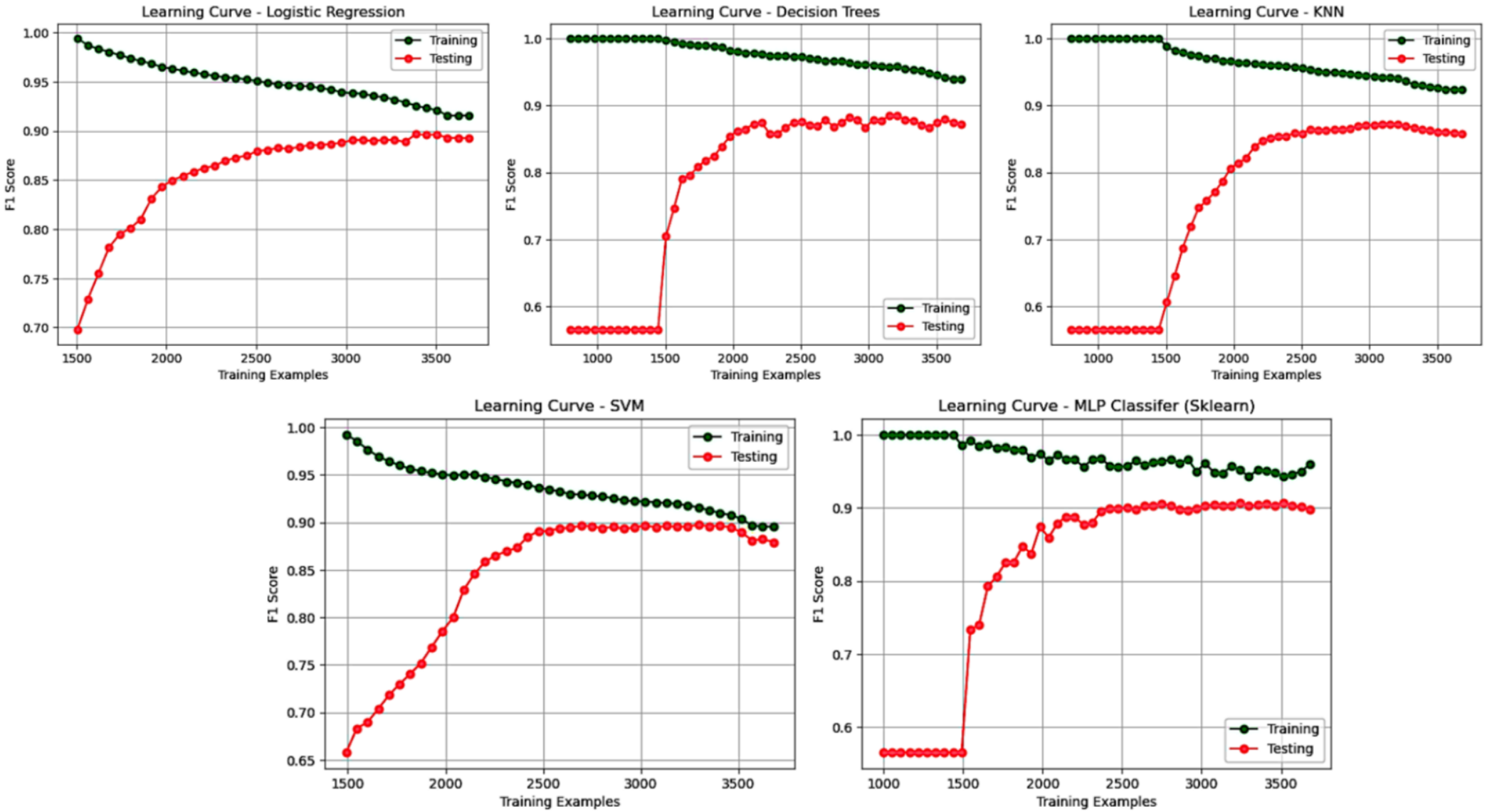

Figure 3: Learning Curves with F1 scoring for all Models with Scikit-Learn.

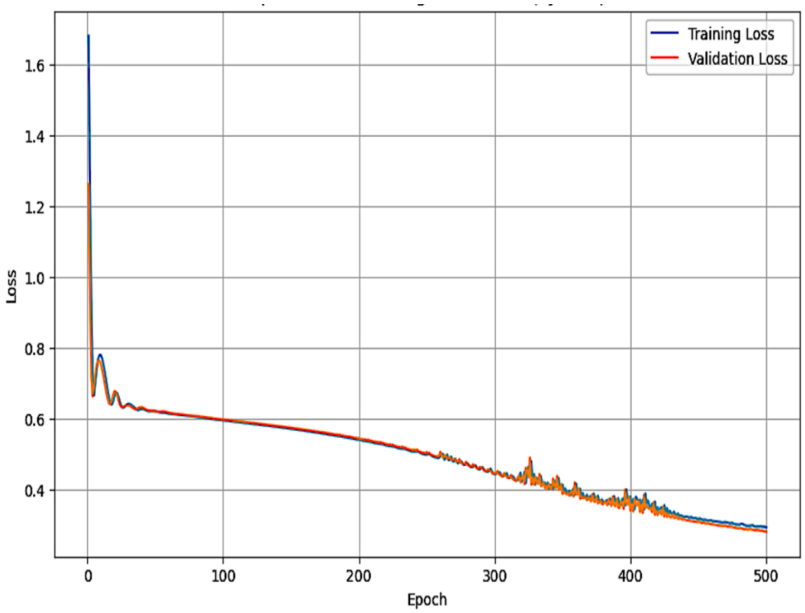

Figure 4: Learning Curves with F1 scoring for all Models with Pytorch.

5.2 ROC Curve

Figure 5 shows ROC curves for each algorithm, quantified by the AUC number. Decision trees have the lowest AUC, while the MLP classifier (Scikit-Learn) performs the best, though with no clear margin. This result indicates that this model is best in distinguishing between spam and non-spam. It is worth noting that the AUC metric does not consider the possibility of accidental correctness, therefore across all models is much higher than the MCC and Cohen's kappa.

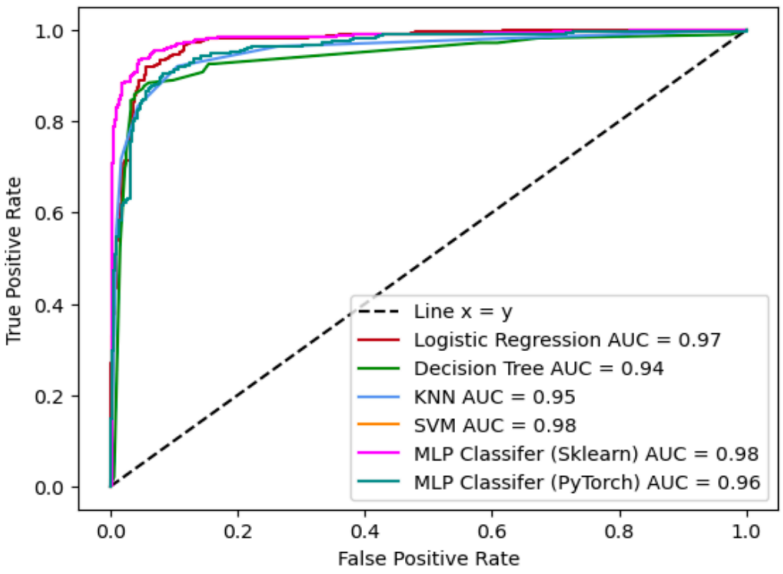

Figure 5: Learning Curves with F1 scoring for all Models with Scikit-Learn.

5.3 MCC

As shown in Table **??**, MCC figures range from 0.79333 to 0. 89540. While performing well, KNN shows lower values indicating that it has difficulties in capturing the overall agreement. MLP Classifier (Scikit-Learn) demonstrated the highest MCC demonstrating a strong overall agreement and highest precision.

5.4 Cohen's Kappa

As shown in Table 1, MLP Classifier (Scikit-Learn) achieved the highest score, hence is the most accurate model. SVM and logistic regression also performed well and KNN shows the weakest performance, indicating difficulties in capturing agreement beyond chance.

Algorithm	**Logistic Regression**	**Decision Trees**	**KNN**	**SVM**	**MLP (Scikit-Learn)**	**MLP (Pytorch)**
AUC	0.97342	0.94015	0.94897	0.97105	0.98377	0.96530
MCC	0.86424	0.83380	0.79333	0.83146	0.89540	0.81611
Cohen's Kappa	0.86422	0.83121	0.78691	0.82902	0.89502	0.81295

Table 1: Table displaying the AUC number, MCC coefficient, and Cohen's Kappa for each model. All values are rounded to five significant figures. Red shading indicates the lowest values in the row, green shading indicates the highest values.

5.5 Confusion Matrix

Figure 6 shows that across all divisions, SVM performs best, closely followed by all other models with KNN being the worst. Interestingly all models are worse at predicting true negatives than true positives, this likely is due to the class imbalance of the dataset.

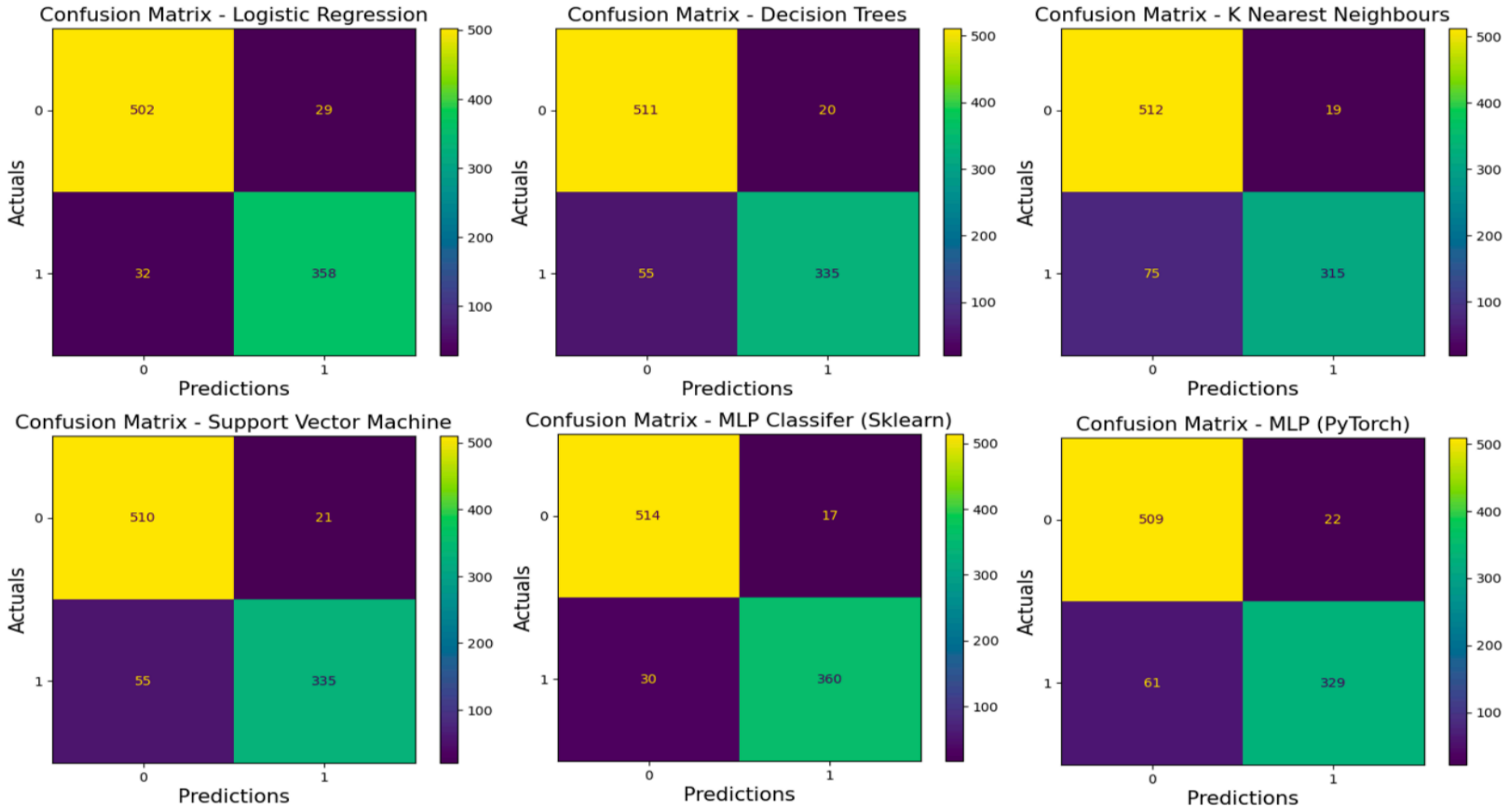

Figure 6: Confusion matrices for all algorithms. The colour scale indicates that yellow is highest and purple is lowest. From top left to bottom right the matrix shows true positive, false positive, false negative and true negatives.

6 Discussion

Overall KNN displays the weakest performance. This can be attributed to its sensitivity to high-dimensional spaces and the curse of dimensionality, where identifying meaningful relationships becomes challenging. Given the dataset consists of 58 features, this exacerbates the curse of dimensionality, implementing dimensionality reduction techniques could enhance the model's efficacy.

The low MCC and Cohen's Kappa scores can be attributed to its reliance on local decision boundaries, hence it is less effective in capturing global patterns. The fact these are significantly less than the AUC suggests KNN is poor at capturing agreement beyond chance.

Conversely, the MLP Classifier (Scikit-Learn) exhibits the most superior performance. The AUC score suggests that the non-linearity inherent in the data is adeptly modelled by the MLP, a feature lacking in decision trees and KNN. These latter models struggled to capture the intricacies of the decision boundary as well, leading to slightly lower scores.

Remarkably, SVM proved highly effective based on the AUC metric, indicating its proficiency in distinguishing spam instances. Although SVM models are generally less effective with imbalanced datasets, introducing class weighting may enhance the performance of this model.

Curiously, the MLP PyTorch performed worse than Scikit-Learn this is likely due to their different architectures and specific implementations, such as variations in hidden layer sizes and activation functions.

Interestingly all Scikit-Learn models demonstrate high variance and little improvement until a particular number of training examples are reached, typically around 1500 training examples. This could be because these models with their complexity require sufficient volume of data to generalise the varied nature of spam emails from the dataset. Overall, one can be confident that the models are not overfitting the data due to learning curves, cross validation and regularisation techniques employed throughout.

From looking at the learning curve for the PyTorch implementation, Although the graph has no indications of bias or high variance, one expects a negative exponential like shape, which has not occurred here, furthermore this would change on multiple runs of the code. This implies there is an implementation error, perhaps explain why this model performed worse than its Scikit-Learn counterpart.

7 Conclusions

AUC, MCC and Cohen's Kappa indicate a strong and reliable performance across all algorithms. One concludes that MLP classifier (Scikit-Learn) is the most accurate and precise for classifying emails as spam. This is closely followed by logistic regression and SVM.

KNN performs notably poorer due to the dimensionality of the data. These results were achieved using the spam base dataset. Since this dataset was created in 1999 it is unsuitable to utilise these models for modern day spam databases, due to changes in spam characteristics and patterns of legitimate emails.

Potential future improvements include integrating the current dataset with a more recent one for broader email classification. Exploring feature engineering on elements like email headers and sender information could enhance the models. Additionally, using feature scaling to emphasise known non-spam indicators, like "George" in this model, but it may not generalise as well to other datasets. Specific model improvements include

Logistic Regression
Introduce interaction terms between features to capture potential dependencies between them.

Decision Trees
Extend the model into a Random Forest, leveraging an ensemble of decision trees. This would combine predictions from multiple trees, potentially enhancing the model.

KNN
Utilise dimension reduction techniques, such as principal component analysis to transform original features into a lower dimensional space.

SVM
Implement additional measures to address imbalanced datasets, such as class weightings or adjusting the decision threshold.

MLP - Scikit-Learn
Write separate functions for the hidden layers and output layer. For instance, utilising a sigmoid activation function for the output layer as this function exhibits excellent binary classification features.

MLP – PyTorch
Change the architecture of the model, particularly increase the number of hidden layers. Batch normalisation could be implemented to address variability in results in each code run. The variability in results could also be address by changing the learning rate scheduling.

References

[1] World Economic Forum. *40 years on from the first spam email, what have we learned? Here are 5 things you should know about junk mail*. https://www.weforum.org/agenda/2018/05/its-40-years-since-the-first-spam-email-was-sent-here-are-6-things-you-didnt/. Retrieved from World Economic Forum. 2018.

[2] Statista. *Global spam volume as percentage of total e-mail traffic from 2011 to 2022*. https://www.statista.com/statistics/420400/spam-email-traffic-share-annual/. Retrieved from Statista. 2023.

[3] M. R. Hopkins. *Spambase*. https://doi.org/10.24432/C53G6X. Retrieved from UCI Machine Learning Repository. 1999.

[4] Evidently AI. *Accuracy vs. precision vs. recall in machine learning: what's the difference?* https://www.evidentlyai.com/classification-metrics/accuracy-precisionrecall. Retrieved from Evidently AI. n.d.

[5] Google Machine Learning. *Classification ROC Curve and AUC*. https://developers.google.com/machine-learning/crashcourse/classification/roc-and-auc. Retrieved from Google Machine Learning. n.d.

[6] Data Tab. *Cohen's Kappa*. https://datatab.net/tutorial/cohens-kappa. Retrieved from Data tab. 2023.

[7] Yue Xie and S. Jin. *Complexity of a Projected Newton-CG Method for Optimization with Bounds*. Tech. rep. Wisconsin Institute for Discovery at University of Wisconsin-Madison, 2023.

[8] SriharshAI. *What are the Time and Space Complexities of Decision Trees*. https://www.youtube.com/watch?v=P8jE6kYeXhU. 2022.

[9] A. A. Awan. *The Curse of Dimensionality in Machine Learning: Challenges, Impacts and Solutions*. https://www.datacamp.com/blog/curse-ofdimensionality-machine-learning. Retrieved from Datacamp. 2023.