MLP Experiments - relu		
Optimizer	**Learning Rate**	**Accuracy**
SGD	0.01	0.8032786885245902
SGD	0.001	0.819672131147541
Adam	0.01	0.7377049180327869
Adam	0.001	0.819672131147541

MLP Experiments - tahn		
Optimizer	**Learning Rate**	**Accuracy**
SGD	0.01	0.8852459016393442
SGD	0.001	0.8360655737704918
Adam	0.01	0.8852459016393442
Adam	0.001	0.8852459016393442

SVM Experiments	
Kernel Function	**Accuracy**
linear	0.868852
rbf	0.836066
sigmoid	0.852459

Upon completing the models for SVM and MLP, the following results regarding accuracy are demonstrated below via tables.

Based on the experiment outcomes, the MLP model with “tanh” activation function outperforms both linear and non-linear SVM models, reaching the best accuracy across all experiments (achieving an accuracy of 0.885). This indicates that the MLP model (tanh activation) is a greater fit for this dataset. One possible reason for MLP outperforming the SVM models could be its ability to learn complicated non-linear correlations between input instances and target output. The “tanh” activation function in the MLP model can simulate non-linear correlations between input attributes and target output, possibly making it simpler for the MLP model to capture more complex data patterns. Needless to say, SVMs are well renowned for their ability to learn linear decision boundaries. Nevertheless, if the connection among the input features and goal output is non-linear, the SVM model may not be as effective as the MLP model in capturing this relationship. The “tanh” activation function can aid in the network's learning of non-linear decision boundaries, potentially boosting its performance on the dataset. I would like to note that my SVM had pretty decent accuracy as well, in fact it performed better than “relu” activation function. The highest performance for the SVM model was via the linear kernel (achieving an accuracy of 0.868). The MLP model with “relu” activation function was not able to learn a decision boundary as effectively as the SVM model’s linear boundary, resulting in a lower accuracy rate.

I also noticed that “tanh” performed better with the learning rate of 0.01, while “relu” performed slighty better with the learning rate of 0.001 opposed to 0.01. This indicates that the optimal learning varies amongst them. In general, a high learning rate potentially results in instability or divergence, whereas a low learning rate can contribute to slower convergence. To sum this point, the learning rate plays a significant role in the performance of neural network models. The optimizer and learning rate utilized had a substantial influence on the performance of the MLP model, with the 'SGD' optimizer and a learning rate of 0.01 obtaining the maximum accuracy. The 'SGD' optimizer may have outperformed the 'Adam' optimizer in the presented tests since the dataset size was relatively small, and the 'SGD' optimizer can converge quicker on smaller datasets. However, it is important to know that an optimizer's performance might vary based on the particular dataset, network design, and hyperparameters that were utilized..

It is important to highlight that both models (SVM and MLP) have strengths and weaknesses.

SVMs can successfully handle high-dimensional data fairly well and non-linearly separable data via kernel functions. In fact the linear kernel performance was not too far off from that of the “tanh” activation function. SVMs are also easier to analyze than MLPs and can shed light on the decision-making process. On the other hand, MLPs are far more flexible and capable of learning more complicated mappings. However, MLPs may be prone to overfitting and necessitates careful selection of the hyperparameters. I want to note that the “tanh” function is symmetric, meaning that it gives opposite output values for opposing input values. This trait can potentially aid the model’s ability to converge and understand the underlying data patterns.

To further improve my results, I can try a different preprocessing technique, such as scaling features in a different way opposed to data normalization. I can also try to experiment with alternative SVM hyperparameters, such as using various cost values and gamma values to discover the optimum combination for the dataset. Additionally, I can also examine feature selection approaches to check if deleting less relevant features potentially improve model performance. I also want to appropriately tune my hyperparameters so it can improve the model's performance overall. I did have a bit of an issue in python. I am more familiar with tuning models in rstudio opposed to python, so this is something I can work on for future experiments.