Tic-Tac-Toe Project

Sheela, Dipali, Jimmy

Implementation

The Learning Application

Our learning application is a Jupyter Notebook file that reads in three datasets and performs cross validation using the following three machine learning models: **linear regression**, **k nearest neighbors**, and **multilayer perceptron**. Two of the datasets are classification problems, and one is a regression problem. The trained regression models were saved and used in our Tic-Tac-Toe game application that we describe in a later section.

Data Preprocessing

For the data preprocessing, we extract each of the datasets into a reusable structure that we call **data_splits**. Additionally, we divide each datasets into train / validation / test splits according to the following ratio **6:2:2**. Below is an example of how we use the **data_splits** structure to access the training splits for the **tictac_single.txt** dataset.

```
X_train = data_splits['single']['examples']['train']
y_train = data_splits['single']['labels']['train']
```

We performed one additional preprocessing step for the **tictac_final.txt** dataset. We changed all of the **-1** class labels to instead be **0**. In the original case where the class label is **-1**, we noticed that when the cross validation predictions are split between the **-1** and **1** classes, the prediction would be averaged to **0** which is an unallowed prediction. By changing the class labels to be **0** and **1**, all averaged results, when rounded, will be one of the two true labels. This fixed the problem with undefined predictions being made.

Data Visualisation:

For data visualisation, we implemented multiple functions for statistical analysis of all three datasets. One function plots the value distribution for each position in all the three datasets. For **tictac_final.txt** and **tictac_single.txt** the last graph represents label distribution over the entire corresponding dataset. For **tictac_multi.txt** dataset, we plotted the individual class distribution over entire data points as well as per label data points. We also plotted a correlation matrix for 9 tic tac toe positions for all three datasets. Co

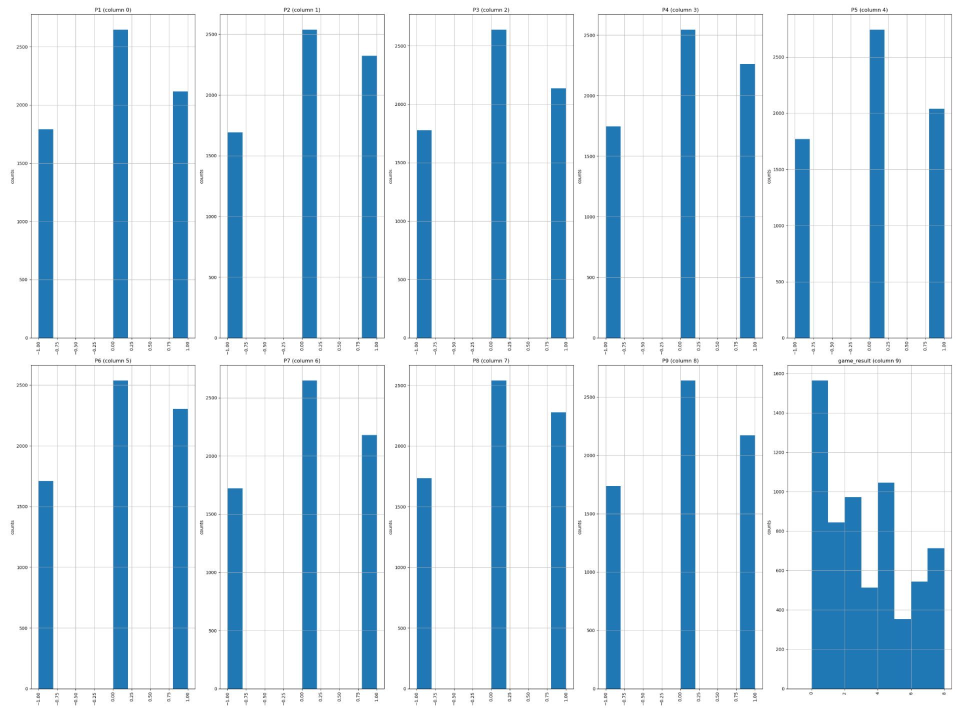


Fig 1: Histogram plot per tic tac toe position in **tictac_single.txt**

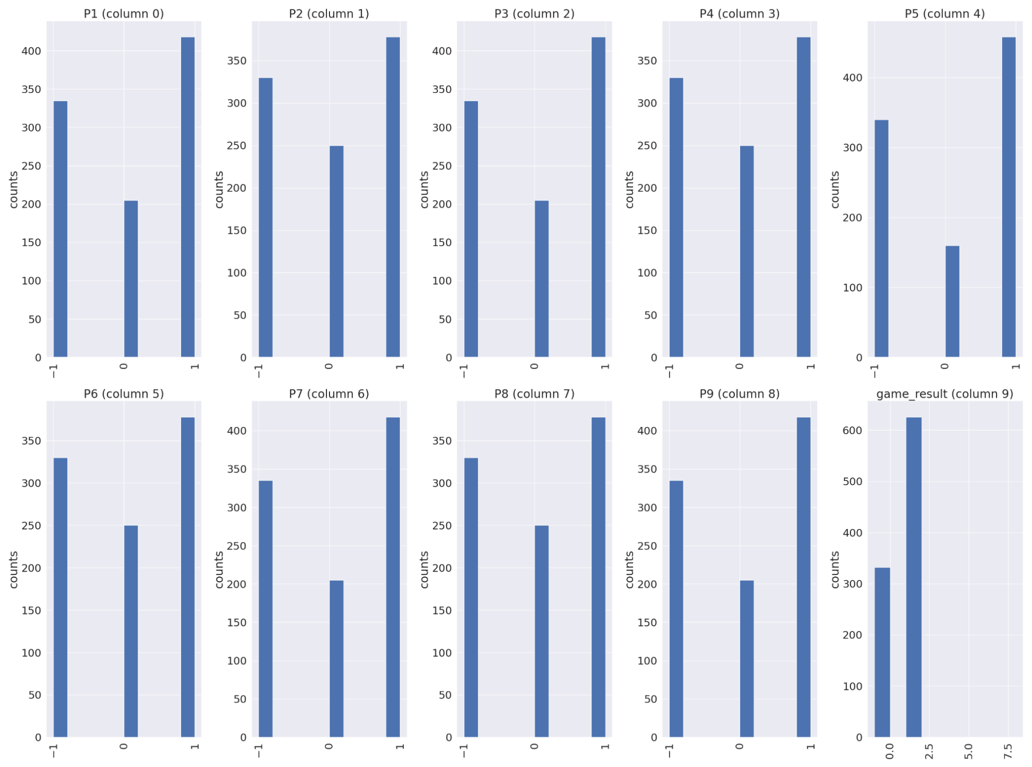


Fig 2: Histogram plot per tic tac toe position in **tictac_final.txt**

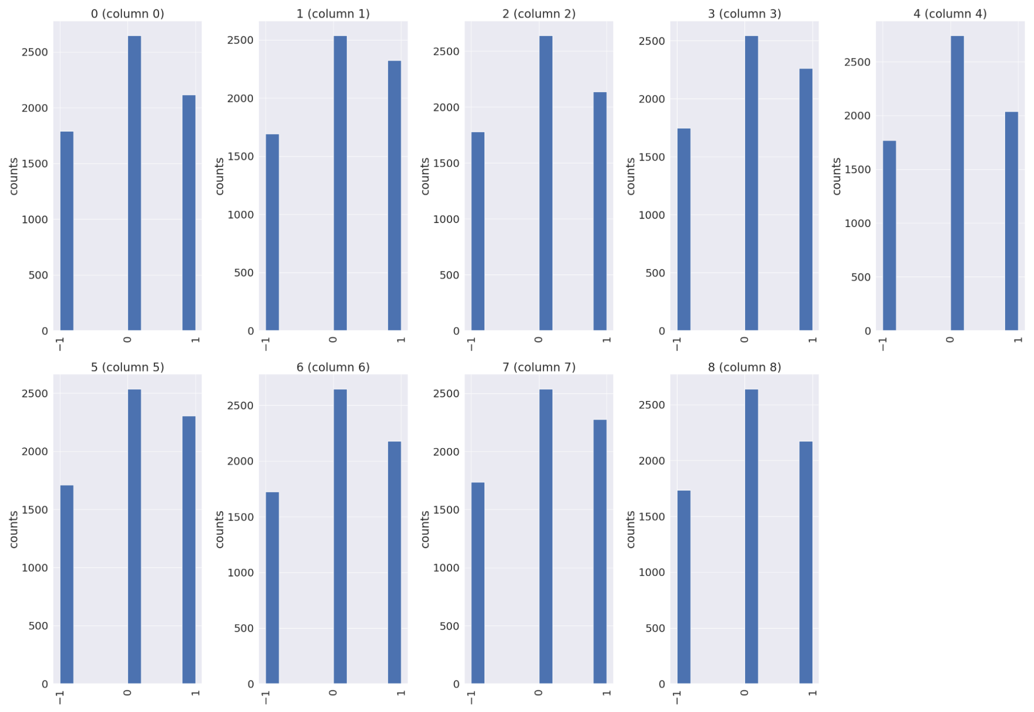


Fig 3: Histogram plot per tic tac toe position in **tictac_multi.txt**

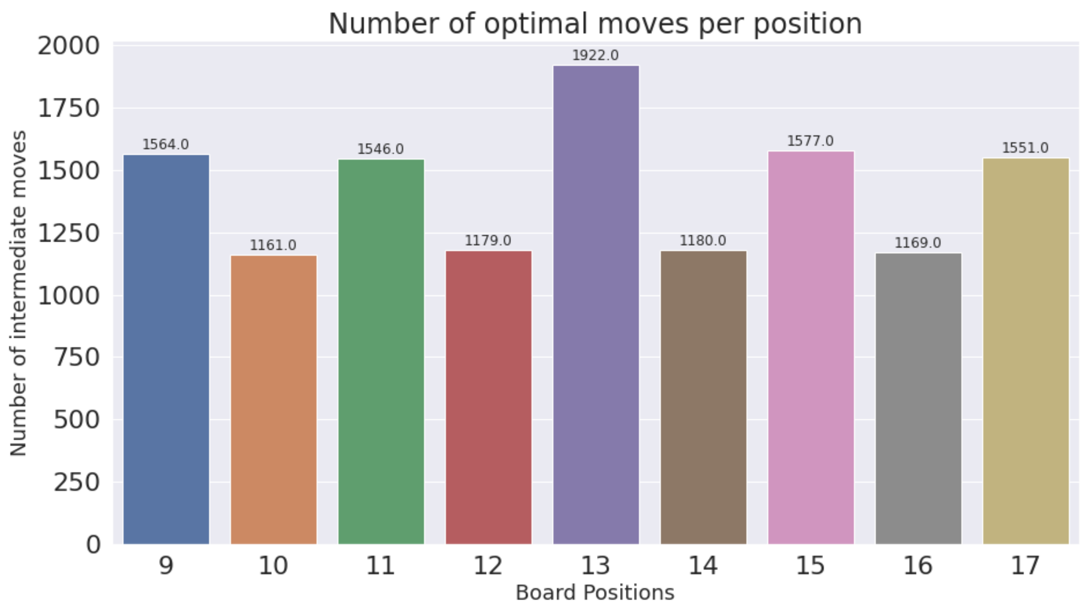


Fig 4: Histogram plot per tic tac toe class distribution over all labels in **tictac_multi.txt**

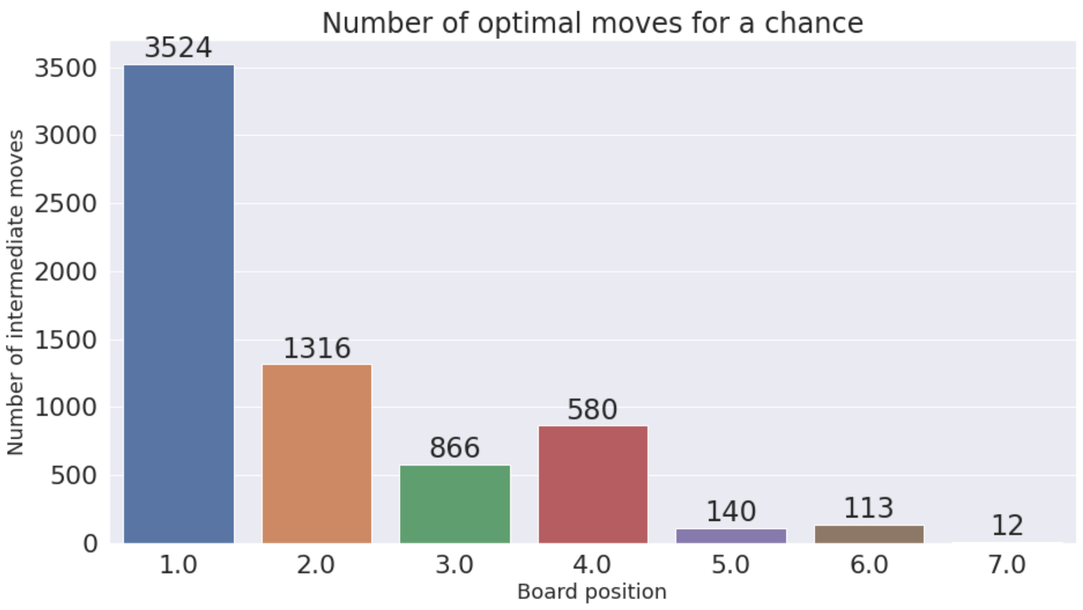


Fig 5: Histogram plot per tic tac toe class distribution per label **tictac_multi.txt**

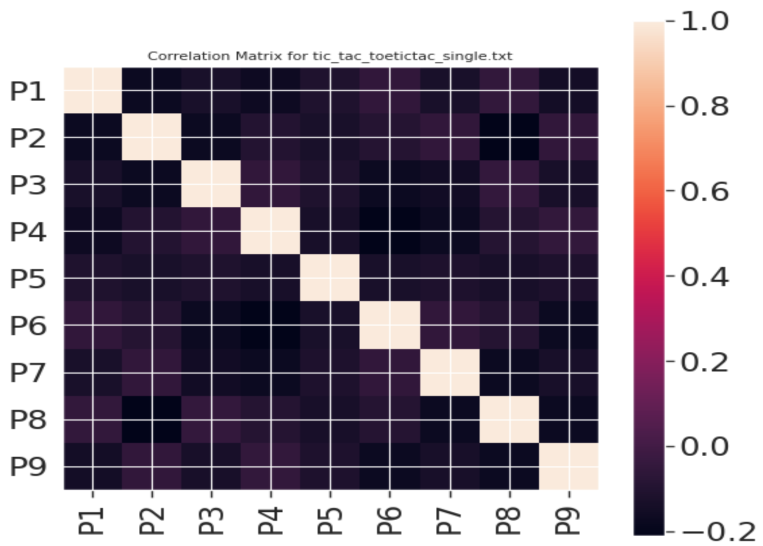


Fig 6: Correlation matrix for **tictac_single.txt**

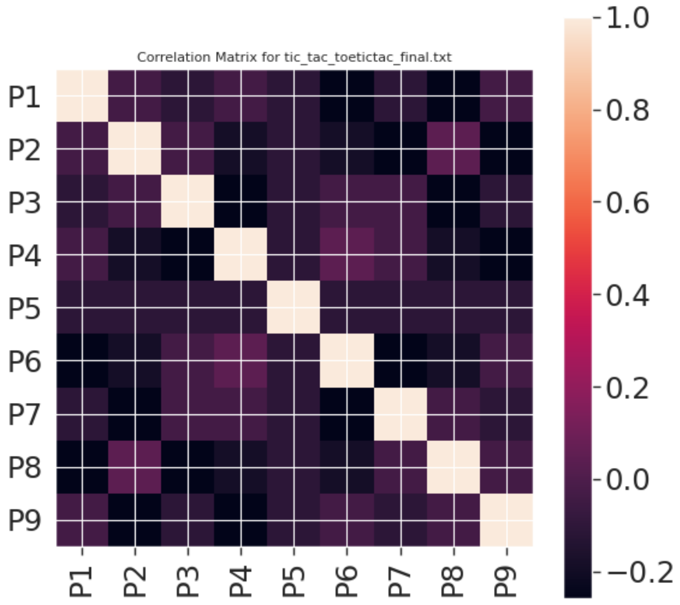


Fig 7: Correlation matrix for **tictac_final.txt**

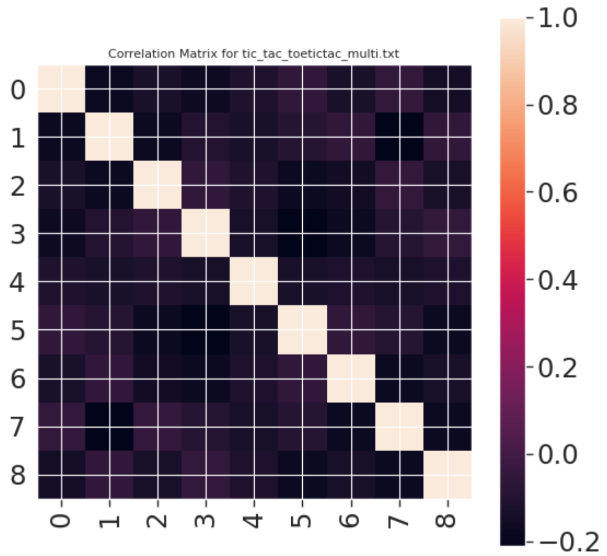


Fig 8: Correlation matrix for **tictac_multi.txt**

Metrics

We evaluated our models by charting the confusion matrices and by capturing the accuracies. Additionally, we tuned hyperparameters by evaluating the models' f1-scores.

Training

All models were trained using cross validation. For cross validation, we used the sklearn **cross_validate** function with 10 folds and shuffled the data.

Validation

We used the validation set to tune a few hyperparameters. We computed the f1-score model with various hyperparameter settings and chose the values that maximized the f1-score on the validation set.

All of our regression models rely on a threshold value to convert the regression prediction into a classification prediction. Our assumption was that a threshold value of **.5** would optimally round each regressor into a classifier. However we noticed that each regressor needed a different threshold value.

We used the validation set to test a variety of threshold values for each regressor. We found that the linear regressor needed a slightly higher than average threshold of about **.54**, whereas the KNN and MLP models needed a lower threshold value of about **0.39** and **0.48**, respectively.

We also tuned the number of neighbors for the KNN model. We tried values between **5** and **30**, and found **30** to be the optimal number. In fact, we found that the KNN model performed well on even larger numbers of neighbors, however we bounded the hyperparameter at **30** to avoid overfitting.

Model Implementations

We used the sklearn APIs for the following models: linear regression (regression only), k nearest neighbors (regression and classification), and multilayer perceptron (regressor and classification). For the linear regression classification problem, we implemented logistic regression from scratch using numpy. Our least squares normal equation python implementation is listed below.

```python
def linearNormEqn(X, y):
    """Normal equation linear regression function using least squares"""

    theta = np.matmul(np.matmul(np.linalg.inv(np.matmul(np.transpose(X),
X)), np.transpose(X)), y)

    return theta
```

Additionally, in our logistic regression implementation we used the sigmoid function to bound our regression predictions between 0 and 1.

The Game Application

The tic tac toe board has 9 positions (0 - 8) to fill with either X(human) or O(computer). The player is provided an option to choose KNN regressor, linear regressor or MLP regressor for the computer to predict and make the move. The human starts the game by choosing a position from 0 to 8 to make the first move. Once the player makes the move, the type of regressor chosen is used to predict the computers' move based on the player's move. The most accurate position is played by the computer if not already filled. If the position is already filled, it plays the next best move (by using argsort function). After every move by either the player or the computer, the whole board is checked to see if any player won the game. If the whole board is filled and there is no winner, a drawn match is declared.

Observations from the Gameplay

Out of all the games played from each position on the tic tac toe board against each regressor, the human won all the games against the linear regressor. It was either draw or lose against the MLP regressor and a mix of win, lose and draw against the KNN regressor. MLP predicted the best out of all the three models and linear regressor's performance was the least.

Results

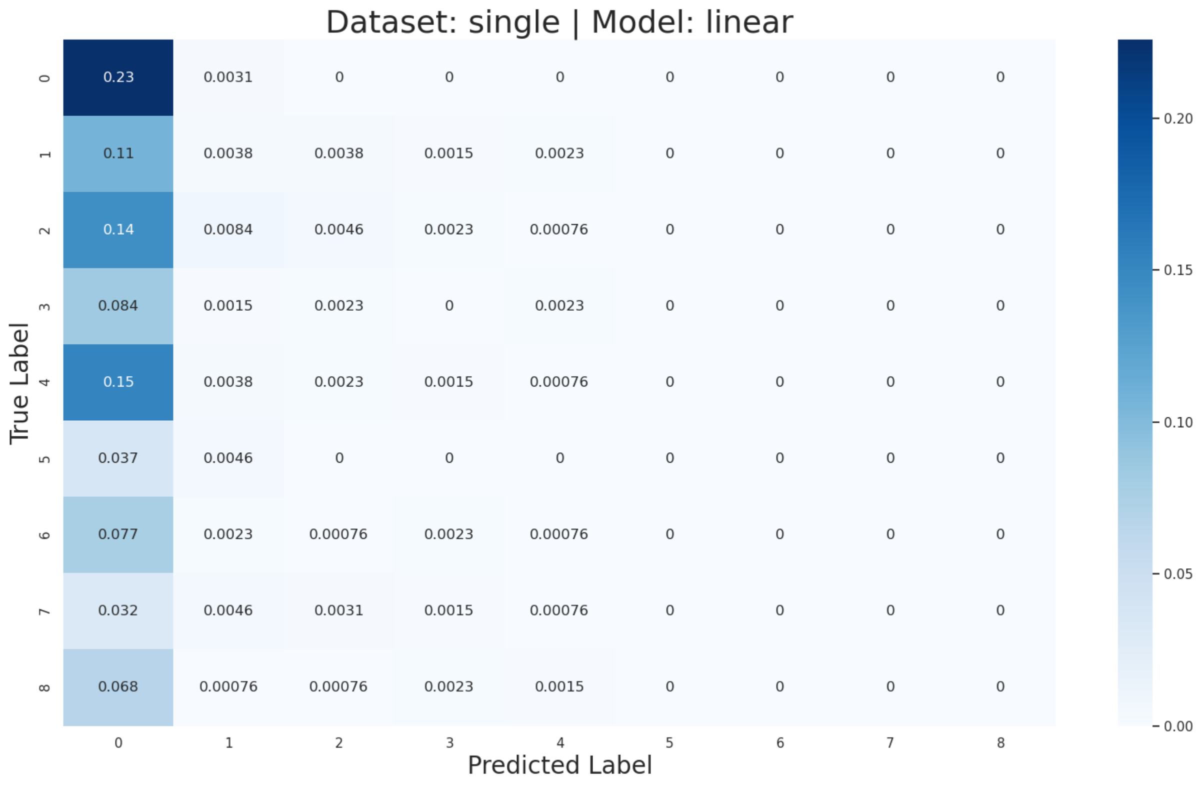
In this section, we display the confusion matrices and other matrices related to the performance of our trained models on the test set.

Confusion Matrices

Results on the **tictactoe_single.txt** dataset

Linear Regression

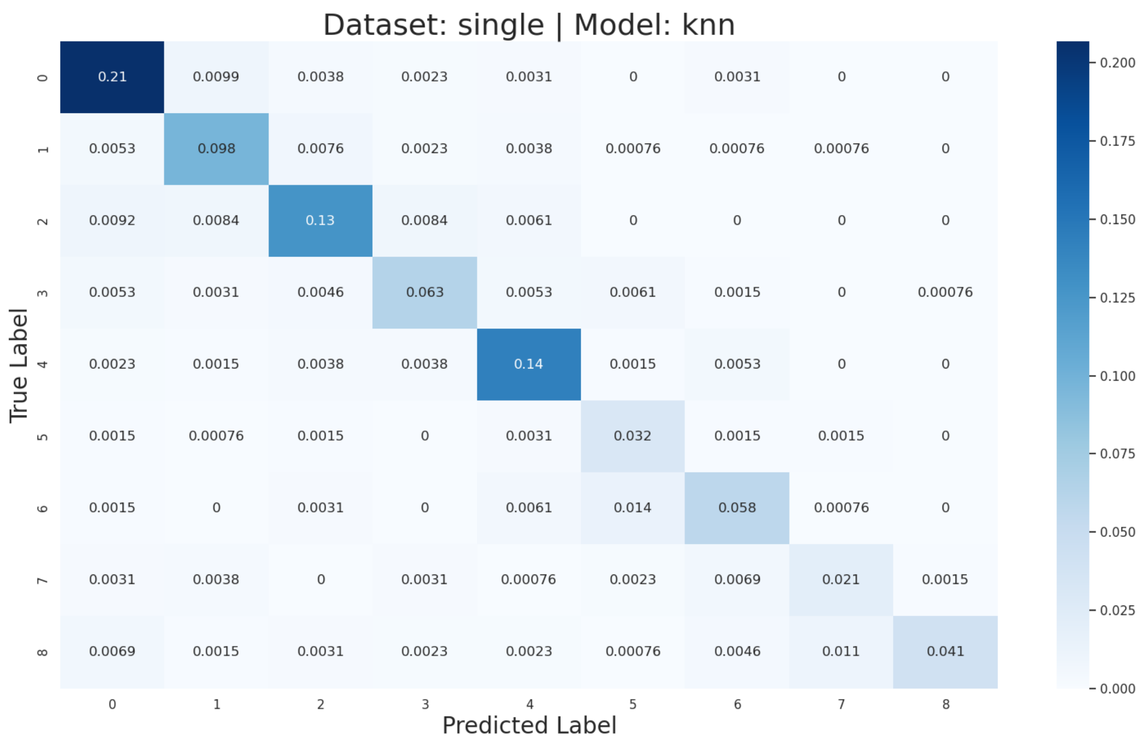
Linear regression performed the worst of all models on the **tictactoe_single.txt** dataset. It predominantly predicted a class of **0**. We believe this indicates that the dataset is not linearly separable.



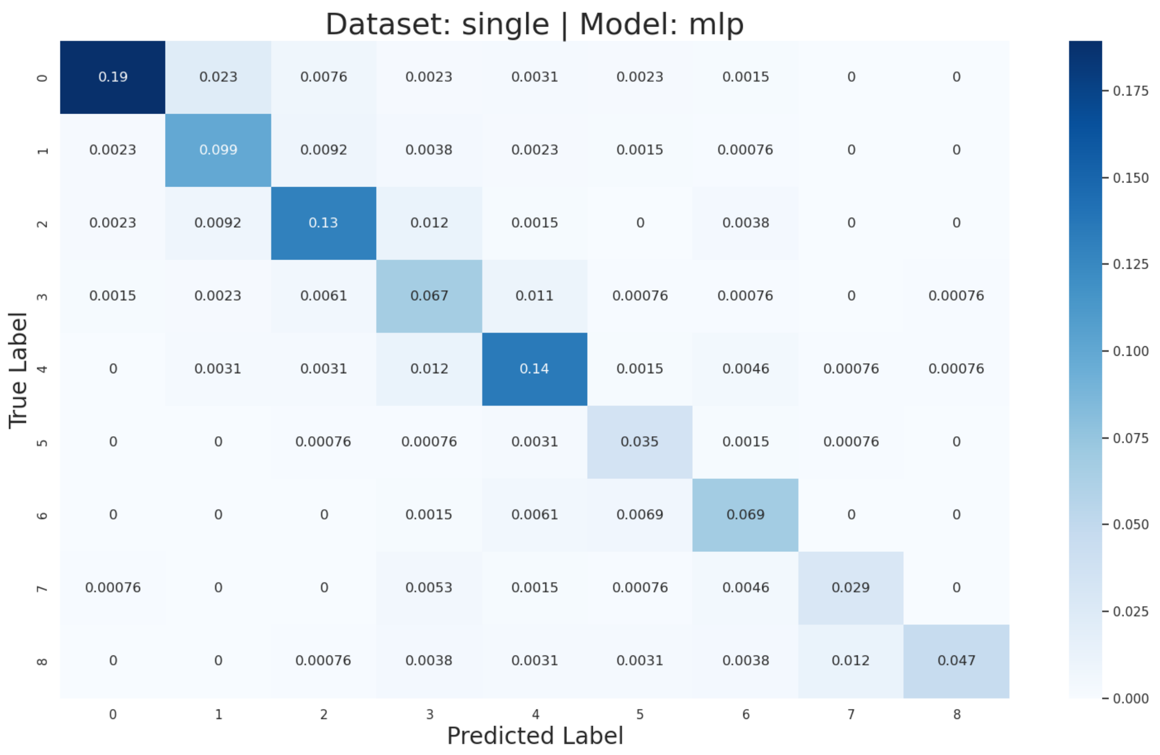
True \ Predicted	0	1	2	3	4	5	6	7	8
0	0.23	0.0031	0	0	0	0	0	0	0
1	0.11	0.0038	0.0038	0.0015	0.0023	0	0	0	0
2	0.14	0.0084	0.0046	0.0023	0.00076	0	0	0	0
3	0.084	0.0015	0.0023	0	0.0023	0	0	0	0
4	0.15	0.0038	0.0023	0.0015	0.00076	0	0	0	0
5	0.037	0.0046	0	0	0	0	0	0	0
6	0.077	0.0023	0.00076	0.0023	0.00076	0	0	0	0
7	0.032	0.0046	0.0031	0.0015	0.00076	0	0	0	0
8	0.068	0.00076	0.00076	0.0023	0.0015	0	0	0	0

K nearest neighbors

K nearest neighbors and multilayer perceptron performed much better than linear regression.

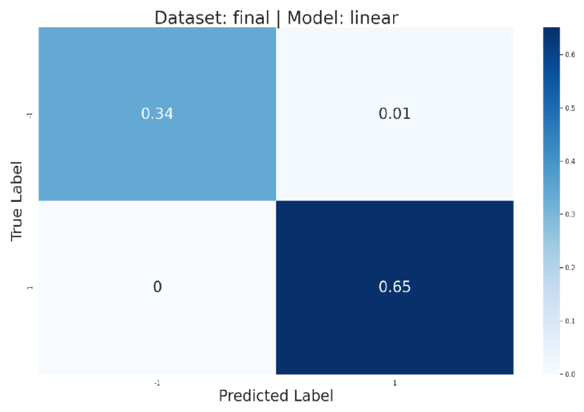


Multilayer perceptron

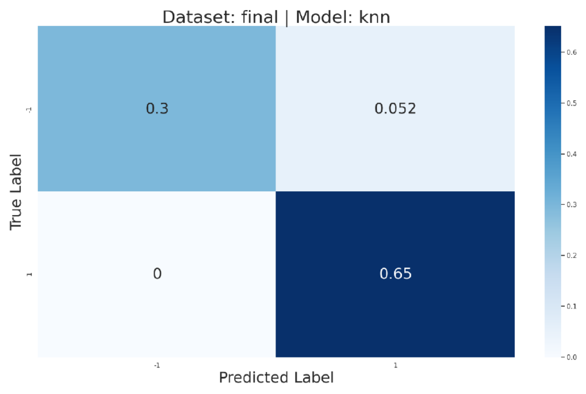


Results on **tictactoe_final.txt** dataset

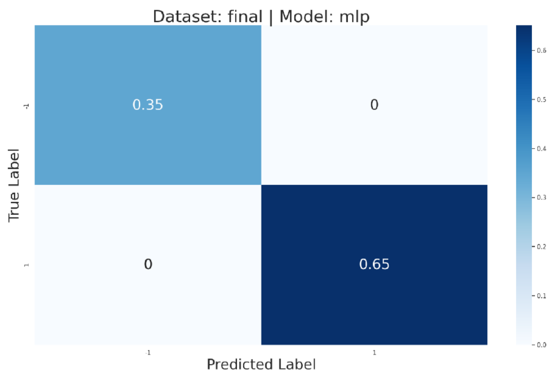
Linear regression



K nearest neighbors



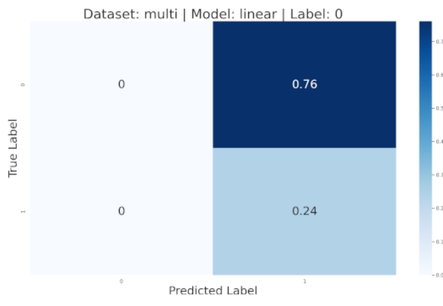
Multilayer perceptron

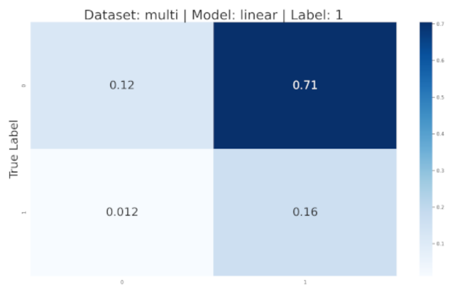


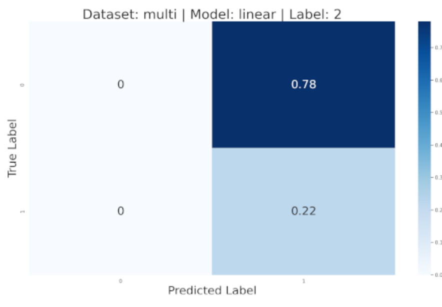
Results on the **tictactoe_multi.txt** dataset

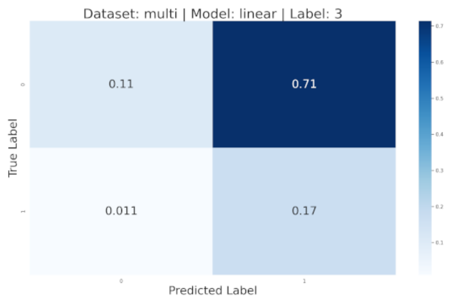
Linear regression

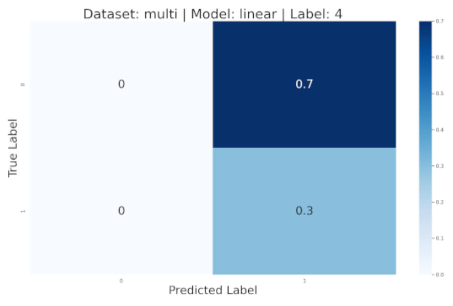
Confusion matrix for each of the 9 labels. Linear regression performed very poorly.

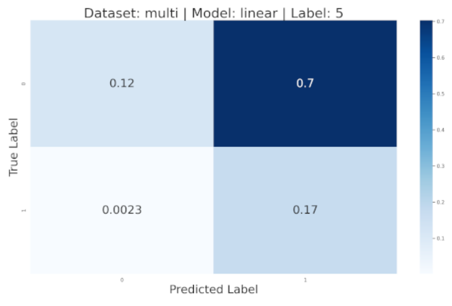


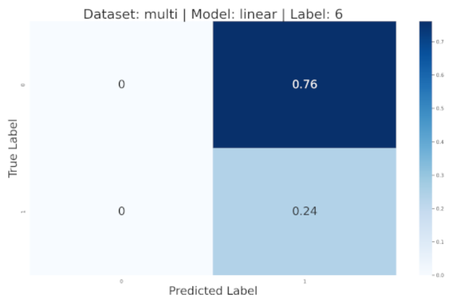


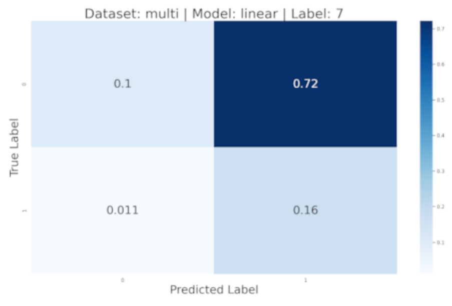


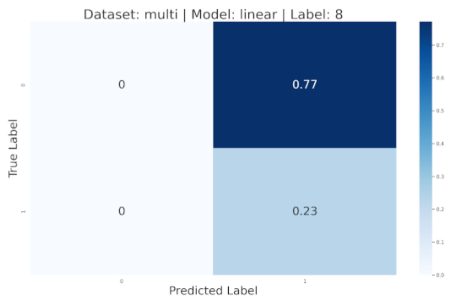






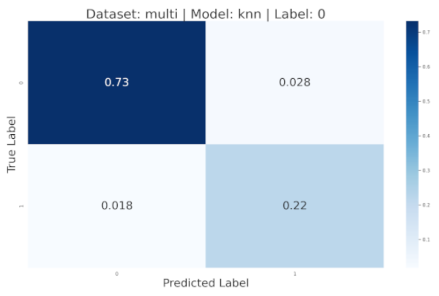


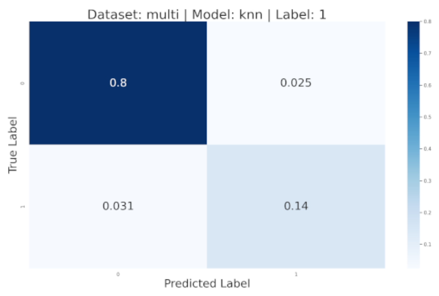


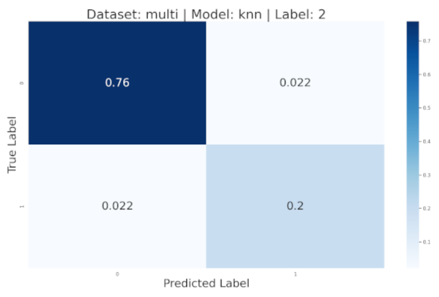


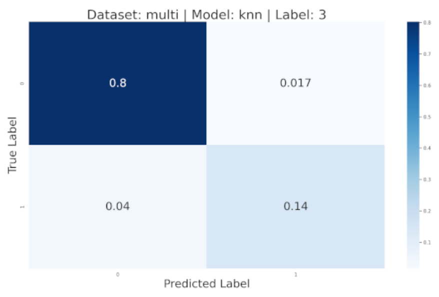
K nearest neighbors

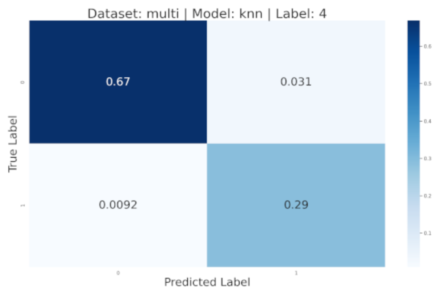
Confusion matrix for all 9 labels.. This is a large performance improvement over linear regression.

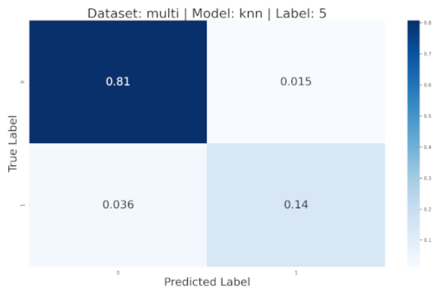


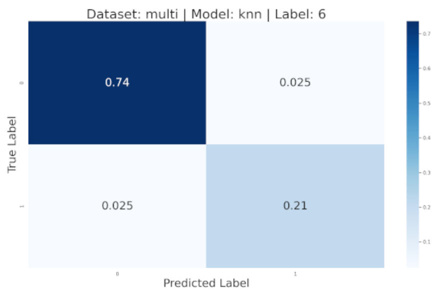


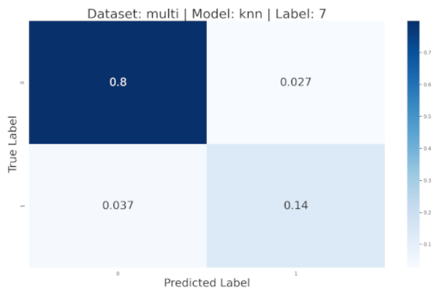


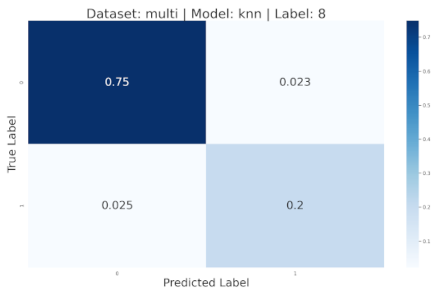






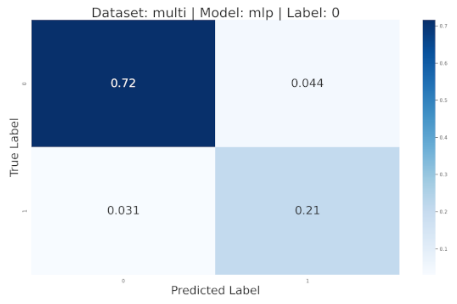


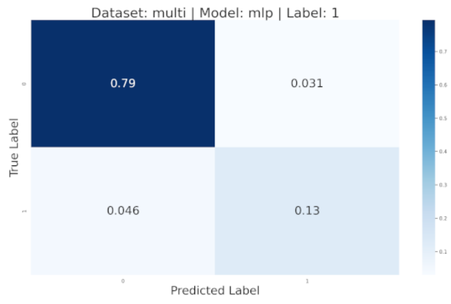


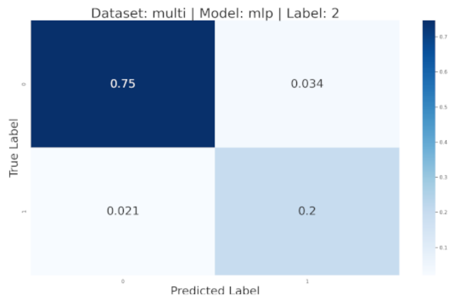


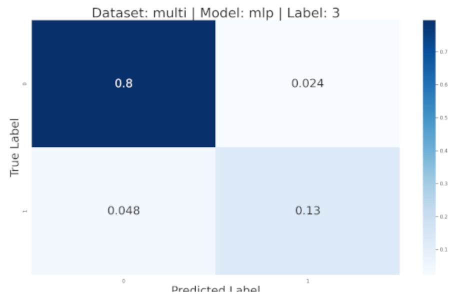
Multilayer perceptron

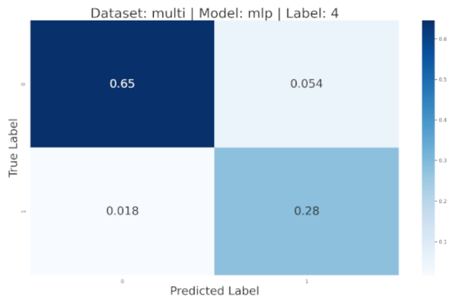
Confusion matrix for all 9 labels.

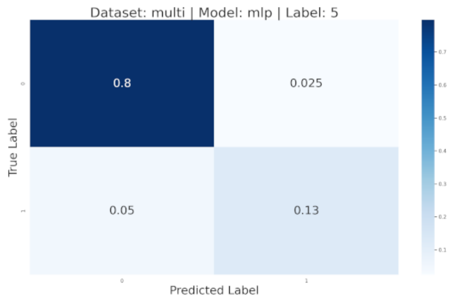


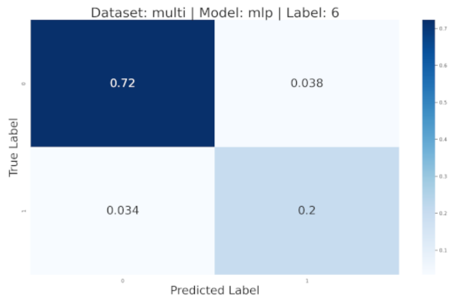


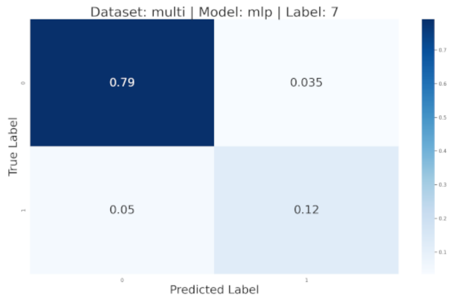


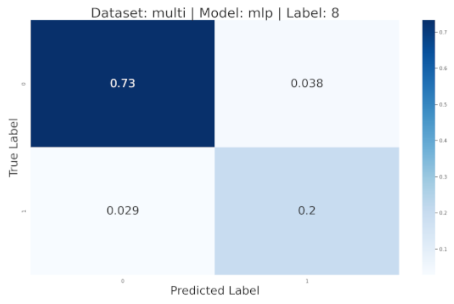












Accuracy, Precision, Recall and F1 Score

The following table shows the average accuracy, precision, recall, and F1 score over the cross validation models.

Results using whole datasets (with a 60:20:20 train:validation:test split)

Dataset	**Model**	**Accuracy**	**Precision**	**Recall**	**F1 score**
tictactoe_single.txt	Linear	23%	11%	8%	5%
tictactoe_single.txt	KNN	80%	73%	77%	74%
tictactoe_single.txt	MLP	80%	76%	76%	76%
tictactoe_final.txt	Linear	99%	99%	99%	99%
tictactoe_final.txt	KNN	95%	93%	96%	94%
tictactoe_final.txt	MLP	100%	100%	100%	100%
tictactoe_multi.txt	Linear	0%	98%	22%	36%
tictactoe_multi.txt	KNN	72%	87%	89%	88%
tictactoe_multi.txt	MLP	83%	91%	94%	92%

We also evaluated the models on one-tenth the size of the datasets. Below are the performance metrics using the smaller dataset.

Results using one-tenth the datasets (with a 60:20:20 train:validation:test split)

Dataset	**Model**	**Accuracy**	**Precision**	**Recall**	**F1 score**
tictactoe_single.txt	Linear	22%	12%	7%	7%
tictactoe_single.txt	KNN	46%	29%	35%	28%
tictactoe_single.txt	MLP	53%	43%	46%	42%
tictactoe_final.txt	Linear	100%	100%	100%	100%
tictactoe_final.txt	KNN	68%	60%	68%	59%
tictactoe_final.txt	MLP	89%	92%	89%	89%
tictactoe_multi.txt	Linear	0%	88%	21%	34%
tictactoe_multi.txt	KNN	10%	74%	50%	57%
tictactoe_multi.txt	MLP	31%	94%	60%	73%

Model Performance Comparison Graphs:

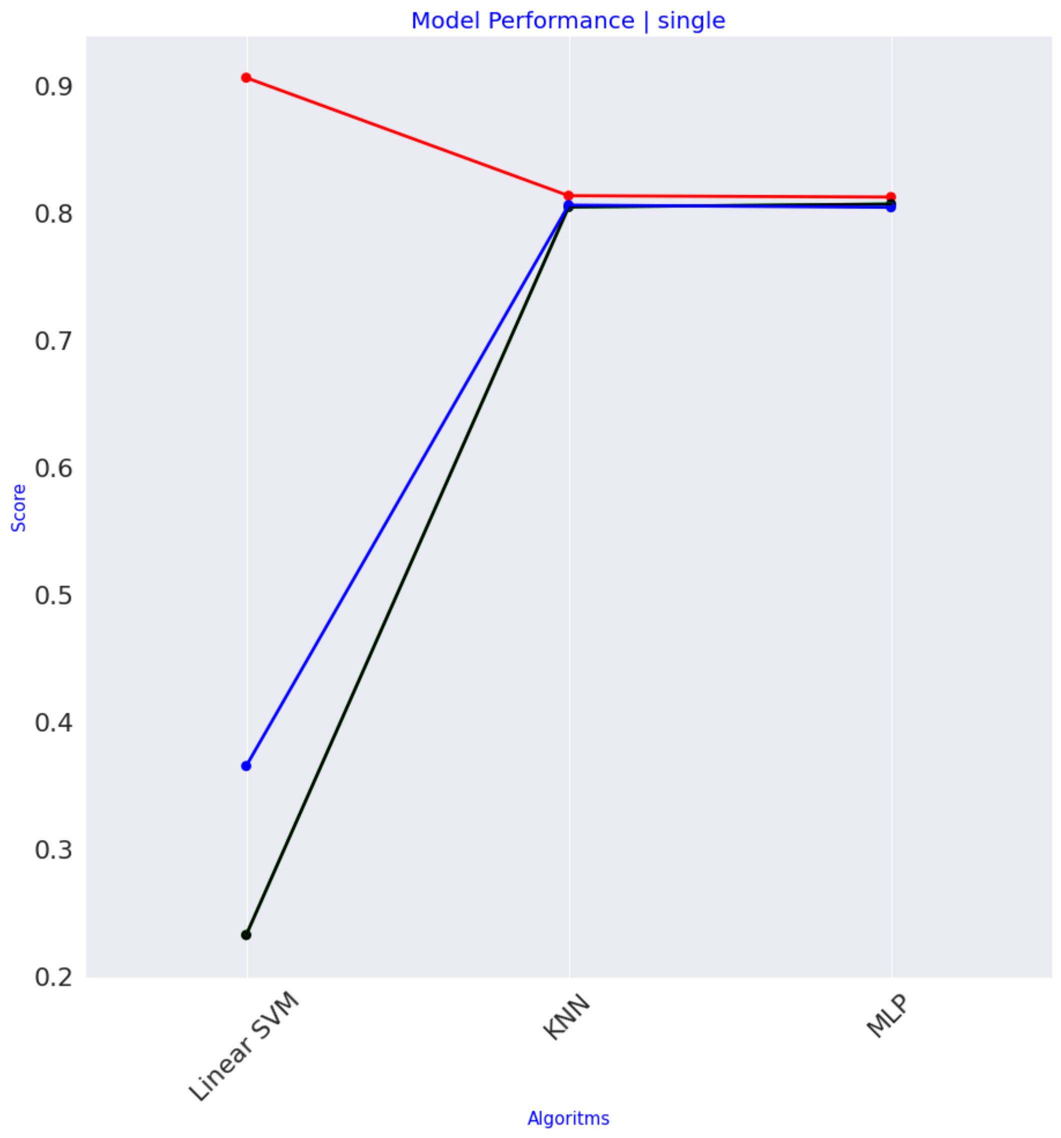


Fig: Performance evaluation parameters(precision[**Red**], recall[**Black**], f1_score[**Blue**]) for **tictac_single.txt**

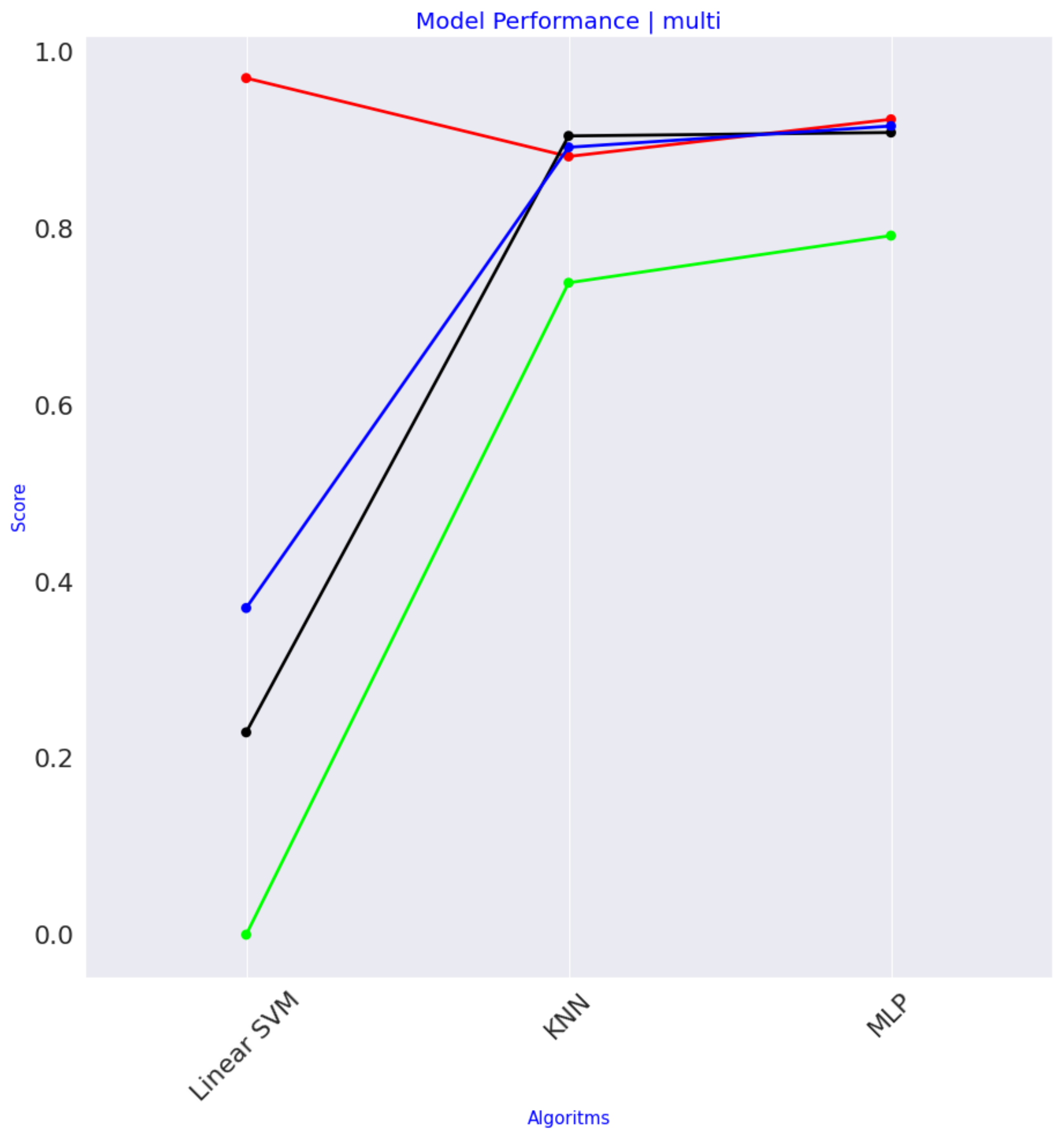


Fig: Performance evaluation parameters(accuracy[**Lime**], precision[**Red**], recall[**Black**], f1_score[**Blue**]) for **tictac_multi.txt**

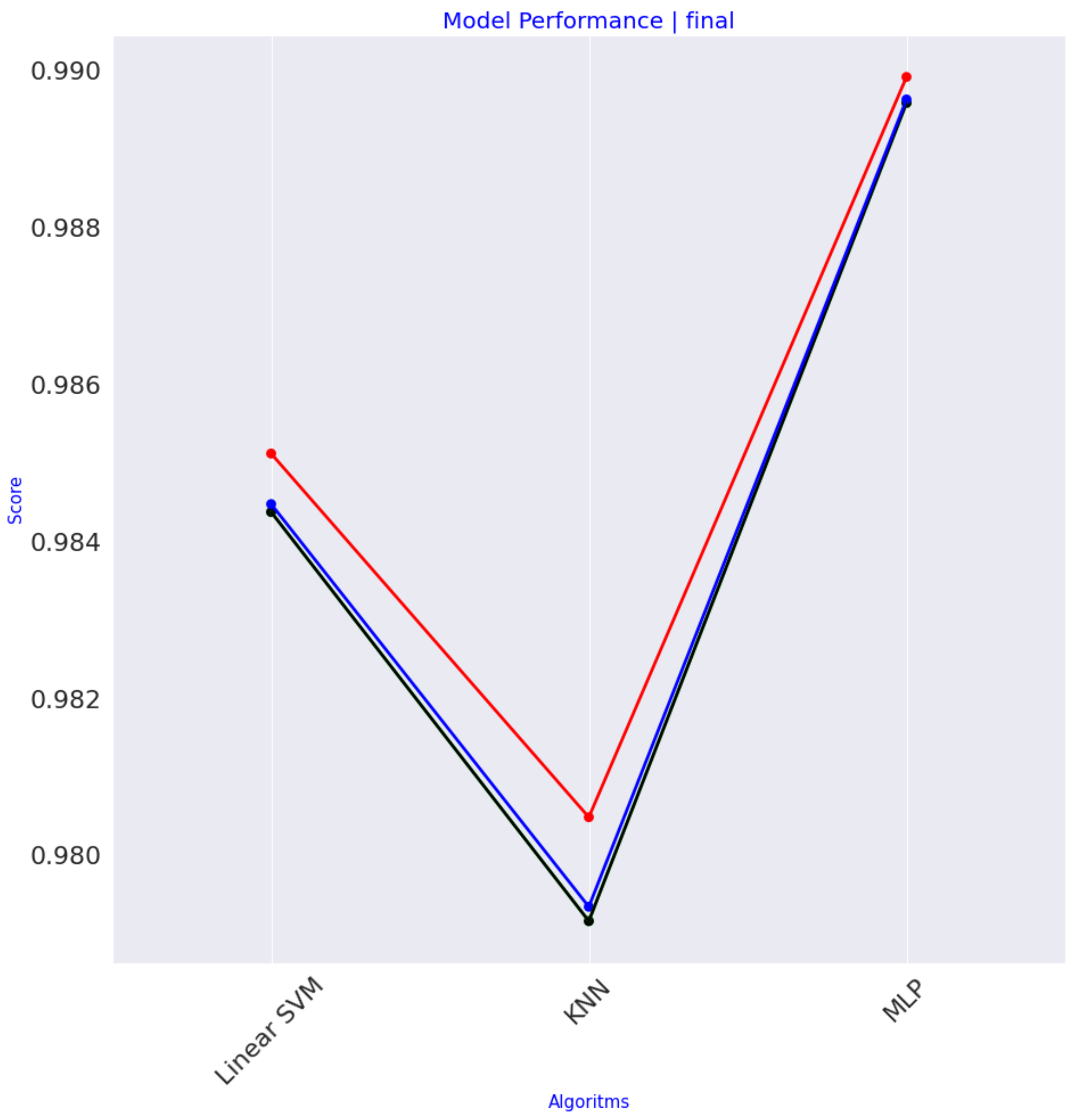


Fig: Performance evaluation parameters(precision[**Red**], recall[**Black**], f1_score[**Blue**]) for **tictac_final.txt**

Discussion

The model that performed best on the classification tasks was the **MLP**. On the **tictactoe_single.txt** dataset, MLP and KNN performed well and produced very similar metrics (f1 scores of 76% and 74%, respectively). The linear model, on the other hand, performed very poorly and only produced an f1 score of 5%. On the **tictactoe_final.txt** dataset, all three models performed well. The linear model performed unexpectedly well and outperformed the KNN model by achieving a 99% f1 score. Despite this performance, MLP once again won this round and produced a perfect 100% f1 score.

In the regression task on the **tictactoe_multi.txt** dataset, **MLP** produced the highest values across all our reported metrics. The linear model, on the other hand, achieved nearly 0% accuracy and very high precision. This indicates that the linear model predicted the positive class on nearly every example, and suggests that either the dataset is not linearly separable, or that there is a mistake in our logistic regression implementation.

We further tested the models' performance using only a tenth of the examples in the original datasets. Unsurprisingly, the results of the models trained with fewer examples is far less than the models trained on the whole dataset. On the classification dataset, MLP still did better than all other models, with one exception. The linear regression model did perfect on the **tictactoe_final.txt** dataset. We can only explain this by concluding that the dataset is highly linearly separable. On the regression task, **MLP** outperformed KNN using the smaller dataset by a wide marger (73% compared to 57% f1 score).

We suspected that MLP would produce the highest performance metrics, and indeed, that is what we observed. We note, however, that the performance difference between KNN and MLP is small. In fact, both KNN and MLP play great games of tic-tac-toe; so well that we weren't able to defeat either of the classifiers. We hypothesize that had we chosen a large train test split (80%, for example), the performance of KNN and MLP would have both exceeded 80% accuracy. Both MLP and KNN scaled well to the larger dataset and demonstrated significantly better metrics when compared to their counterparts trained on the diminished datasets.

How to Run Code

Unzip our project deliverable, then run the TicTacToe.ipynb file. This will read in the datasets (included in our deliverable), generate and save the models, and produce all of our metrics charts. Our deliverable already includes our trained models and generated plot metrics, however rerunning the TicTacToe.ipynb file will regenerate all of these files.

To play the command line tic-tac-toe game, open the Game.ipynb file and run it. The command line game will appear in the Jupyter Notebook cell output window.