Mine VS. Rock Summary

Prepared by Ahmed Ismail Abu Qahf (1223023698)

Observations

After applying the script to the dataset file, the highest accuracy achieved is **0.95** for seven components, and the confusion matrix is shown in Table 1 below.

Confusion Matrix

Table 1: Confusion Matrix

N = 63	**Class Rock (Predicted)**	**Class Mine (Predicted)**
Class Rock (Actual)	28	1
Class Mine (Actual)	2	32

Conclusions

From the above observations, the model predicts that my chances of surviving a real minefield are 95%. Moreover, from the confusion matrix, we can interpret that we have 63 records in the test dataset, **29** actual for *class Rock*, and **34** for *class Mine*. For the Mine data, we could identify a *False Negative* as the model predicted 2 out of them to be of class Rock, while they're actually of class Mine. As for the Rock data, we could identify a *False Positive* as the model one of the Rock data to be of class Mine, while it's actually of class Rock.

The maximum accuracy of 0.95 was achieved by only 7 components which means that only these components are the vital for deriving the model. That's why as we add more components, the model's accuracy keeps decreasing as all of the remaining components from (8-60) don't carry as helpful information as the first seven (1-7) for the model to use for prediction. As a result, if we take a look at the plotted graph in Figure 1, we find the accuracy keeps going up from 1 component until it reaches its maximum at 7 components then it begins to decline as we add more components.

The parameters I choose for the `MLPClassifier` are:

- `hidden_layer_sizes=(100)`
- `activation='logistic'`

- `max_iter=200`
- `alpha=0.00001`
- `solver='adam'`
- `tol=0.0001`
- `random_state=5`

To get the best combination of parameters to use, I created a `GridSearchCV` object and tried it with different combinations of parameters, and then used the method `best_params_` to get the best combination of parameters. The only parameter value I tweaked though is *`random_state`*, its default value is set to **None**, so I changed it to an integer value of 5. The reason for making this tweak is that if we used the default value of None for random_state, the function would produce different results for multiple calls which means we would have a different `MLPClassifier` for each number of components. Thus, we would have an inconsistent comparison between the number of components as for each loop iteration we would have two variable values: the number of components and the `MLPClassifier`. To solve this issue, we need to make the value of `MLPClassifier` unchangeable, so we set the value of `random_state` to an integer value which for multiple function calls, would return the same result for the `MLPClassifier`, and we would only be left out with one variable: the number of components.

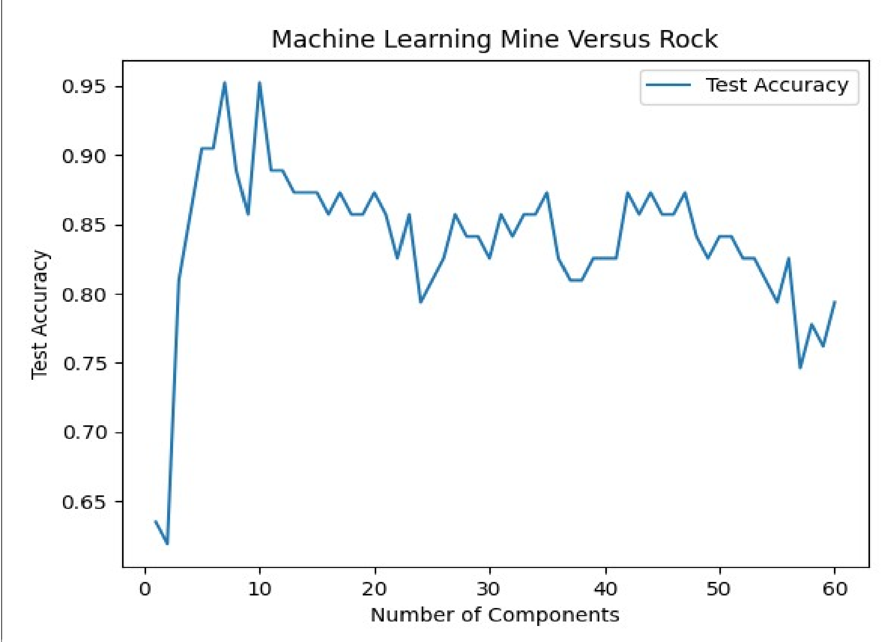

Figure 1: Test Accuracy VS. Number of Components