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 **0.92** for thirty-seven components which means that my chances of surviving a real minefield are 92%, 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)	24	5
Class Mine (Actual)	0	34

Conclusions

From the confusion matrix, we could identify the total number of samples in the test set to be **63**, **29** for *Class Rock* and **34** for *Class Mine*. To be able to interpret the confusion matrix well, first we need to identify two vital terms:

- False Positive means that we predicted a mine, but it was actually a rock.
- False Negative means that we predicted a rock but it was actually a mine.

Based on these two terms, we could argue that the number of false negative cases, for this dataset, is actually more important than the false positive. This is mainly because if we predicted a mine but it was actually a rock, that's fine because no one would get hurt. However, if we predicted a rock but it was actually a mine, that's a disaster because people would get hurt or die. Therefore, the purpose of training the model should be minimizing the number of false negatives as much as possible. Back to our confusion matrix result, we could see that our model managed to successfully identify all the mines which is fantastic. On the other hand, the model failed to classify 5 rocks, so this means there is still room for improvements.

The maximum accuracy of 0.92 was achieved by only 37 components which means that these components have high correlation to class and are vital for training the model. On the other hand, the remaining components have low correlation to class, and that's why as we add more components, the model's accuracy keeps decreasing as all of the remaining components from (38-60) don't carry as helpful information as the first thirty-seven (1-37) for the model to use for prediction. As a result, if we take a look at the plotted graph in Figure 1, we would find the accuracy keeps going up from 1 component until it reaches its maximum at 37 components, and then it begins to decline as we add more components.

The parameters I choose for the `MLPClassifier` are:

- `hidden_layer_sizes=(200, 100)`
- `activation='relu'`
- `max_iter=200`
- `alpha=0.0001`
- `solver='adam'`
- `tol=0.001`
- `learning_rate='constant'`
- `random_state=1`

In order to get the best parameters, I created a `GridSearchCV` object with different combinations, and then used the method `best_params_` to get the best parameters. One final note, I had tweak the default value of `random_state`, as 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, the comparison wouldn't be fair between the number of components as for each loop iteration, we would have two variable values: the number of components and the `MLPClassifier`. To overcome 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. Now, we can make the comparison.

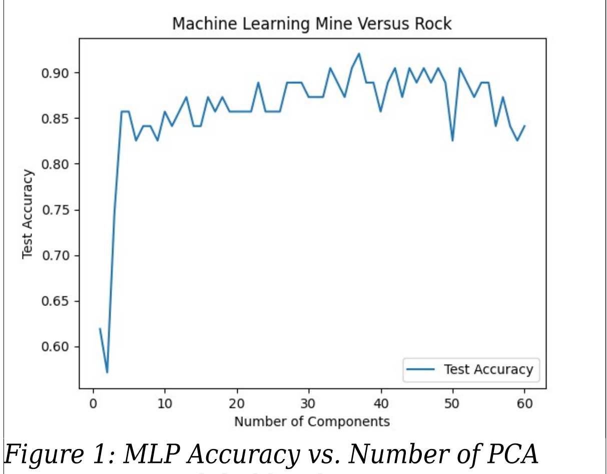

Figure 1: MLP Accuracy vs. Number of PCA components with hidden_layer_sizes=(200,100)