RT 1: Clustering

To cluster the Covertype dataset subset with k-means and a Gaussian Mixture Model (GMM), I made specific choices regarding scaling, initialization, and other parameters to enhance stability and performance without using any class label information. Here is the rationale behind these choices:
I scaled the data using StandardScaler before running k-means and GMM. Since features in the Covertype dataset have varying ranges, standardizing them ensures that each feature contributes equally to distance calculations in both algorithms. This is particularly important for k-means, which relies on Euclidean distances, and beneficial for GMM, as scaling can improve convergence in multimodal distributions. Additionally, random_state=42 was maintained for all models, in order to make empirical testing easier, increasing reproducibility.
For the k-means algorithm, I used the k-means++ initialization to enhance clustering quality. The k-means++ method spreads out the initial cluster centers, reducing the likelihood of poor convergence and often leading to a more stable result than random initialization. This initialization should require fewer iterations to converge compared to random. Additionally, I set n_init=10 since running the algorithm multiple times with different centroid seeds reduces the likelihood of convergence to suboptimal solutions.
For GMM, I once again chose the k-means++ initialization for init_params rather than random to leverage k-means clustering to initialize cluster centers more accurately. This decision was made for similar reasons as before. I set n_init=10 for similar reasons as explained above and decided to keep the default covariance_type=full, to properly model the full covariance matrix for each component. I kept max_iter=300 (the default) to balance sufficient iteration time with computational efficiency, and tol=1e-3 (the default again) as a convergence threshold to ensure that the model refines its cluster probabilities until there's minimal improvement, since it balances precision and efficiency for convergence. I thought about including warm_start but I ended up omitting it because, given the scope and scale of this coursework, iterative fitting would not be necessary (especially given the fact that we are supposed to make reasonable choices for this part of the coursework, and not deeply evaluate our models).
By setting a random_state=42 for both k-means and GMM, I ensured that the algorithms produce consistent results, allowing for reliable analysis of cluster performance across multiple runs. In a coursework setting, this reproducibility is advantageous as it enables clear assessment and comparison without relying on or affecting class label information.

Through these configurations, I selected initialization methods and parameters that are suited for clustering without label information, thereby achieving reproducibility and stable clustering quality suitable for unsupervised learning tasks.

RT 2: Clustering

The total number of pairs of datapoints with the same class labels was 18840016, k-means had 13804797 errors, GMM had 13419928 errors, and random baseline had 16150142.

RT 3: Clustering

The first thing to note about these errors is that they are much higher than expected for all models. It is of course possible to fine-tune and optimise k-means and GMM based on these results, but I did not opt to perform any of this given the instructions for this coursework. It is also noteworthy that the errors themselves also depend quite heavily on the random subset that has been chosen and can vary quite a bit between runs.

The random baseline produces the highest number of errors, as expected, due to its random uniform clustering which fails to capture relationships between datapoints. Interestingly, however, it performs much closer to the other models than I expected for many runs, suggesting that likely separating classes based solely on feature space is inherently challenging in this dataset. I expected GMM to outperform K-means by quite a bit, especially given its ability to capture more nuanced and complex cluster structures through its use of Gaussian Distributions, and the ability to alter the covariances, but the two models had a similar number of errors for most runs. This could be due to the number of clustering challenges posed by the high dimensionality of the Covertype data set (curse of dimensionality), which makes Euclidean distances (used by K-means) less meaningful, coupled with the fact that the data may not fit Gaussian Distributions, or spherical clusters. The marginally better performance of the GMM could be due to K-means struggling with outliers, while full covariance gives GMM better flexibility.

RT 4: Classification

Although training an SVM classifier on the Covertype dataset seems like a good choice due to the data's high dimensionality, it would prove quite difficult, and would pose several challenges:

1. Computational Complexity: First of all, kernel-based methods would be computationally expensive due to the large size of the dataset (hundreds of thousands of data) and given the fact that their time complexity can range from quadratic to cubic. Even with a very efficient implementation, SVMs might run into memory or time constraints. The choice of kernel would feed into this, since using linear kernels (which are computationally cheaper) would lead to underfitting, but using nonlinear ones (e.g. RBF) would be much more computationally expensive.

2. Class Imbalance: The Covertype dataset contains 7 classes, which are likely imbalanced. In the case of imbalances, SVM's margin optimisation would likely favor the majority class, performing poorly on minority classes. This is something that can be mitigated using the class_weight parameter, but it would require very careful tuning and management of hyperparameters, which adds extra complexity to the training process.

3. Feature Scaling: SVMs rely on distance metrics for optimisation and are sensitive to feature magnitudes. To efficiently run SVM's on this dataset, one would need to scale the features in an optimal manner, otherwise features with larger magnitudes would dominate the optimisation process, skewing the decision boundaries. This, of course, is possible, but it adds another layer of complexity.

Thus, given the dataset's size, class imbalances, and nonlinear relationships, in order to use SVMs, one would need to make big decisions regarding optimisation and performance, likely needing to sacrifice one for the other at points. Given the dataset characteristics, it can be much more sensible to tackle a problem such as this with an ensemble method (e.g. Random Forest) to get more efficient modeling of feature relationships instead of opting for SVMs.

RT 5: Classification

random_state=42 was maintained for all models, in order to make empirical testing easier, increasing reproducibility.

1. Logistic Regression: First of all, features fed into the logistic regression were scaled. I chose to do this since logistic regression relies on distance metrics during optimisation, and unscaled features could dominate the optimisation process, leading to poor convergence. I set solver='lbfgs' because it is efficient for multiclass classification problems and performs well on large datasets. I also tried 'liblinear' and 'newton-cg', but they were both outperformed by lbfgs. Additionally, I set max_iter=1000 to allow the solver enough iterations to converge. I initially tested lower values, starting from 100, and also tested

larger values up to 5000, but 1000 seemed to be the sweet spot. I decided to leave the rest of the settings on the defaults because alternate values did not seem to grant much of a performance or overhead boost.

2. Decision Tree Classifier: Feature scaling was not necessary for the decision tree because it is less sensitive to it, and thus I used the unscaled data. I set max_depth=None, which is the default. I was surprised at first due to the test set accuracy (~93%), and thought some form of strange overfitting might be occurring, but after employing some cross-validation to test the results, it truly seems like this is the ideal value (cross validation still got ~92-93%). I tried many values, and anything under 10 introduced underfitting, while values over 15 still appeared to perform worse on the test set. I left the rest of the settings on default, including using the gini index because there was no noticeable gain in tweaking them (I tried entropy, but there was no noticeable gain), and model efficiency was higher using them.

3. Ensemble Method: I trained both a Random Forest and a Gradient Boost ensemble method on the Covertype dataset, but I settled on the Random Forest due to its much more computationally efficient nature, coupled with its ability to model nonlinear relationships. This choice was reinforced given that Gradient Boosting's sequential training process was significantly slower on this large dataset. Similarly to the decision tree, I used the unscaled data for this model and set n_estimators=100 to maximise performance. Setting it to anything less was a bit faster, but the results were slightly worse, and anything over this value gave very marginal to no improvements, but significantly increased training time. I set max_depth=None for similar reasons as above, cross-validating these results as well, and I used max_features='sqrt' to introduce randomness in feature selection and reduce overfitting. I also tested 'log2' and None, but the former reduced accuracy, and the latter increased overfitting.

RT 6: Classification

There were quite significant differences in the performance of the three classifiers. Logistic Regression performed the worst, achieving the lowest accuracy of the three models (~72.4 for most runs). This result is consistent with the model's assumption of linear decision boundaries, which does not effectively represent the dataset. Thus, Logistic Regression is not well-suited for high-dimensional, nonlinear datasets such as this one, as expected.

On the other hand, both the Decision Tree (~93.8%) and Random Forest (~95.5%) models performed very well on the test set, with Random Forest outperforming Decision Tree, as expected. This reflects both the model's ability to capture nonlinear relationships, while setting max_depth=None allowed all trees to grow fully, capturing the complex dataset structure. However, given how high these numbers are, the model's tendency to overfit must be considered. Although both models performed excellently on both the training and test datasets, it is possible that they will fail to generalise to unseen data. Cross-validation was used to test this, however, and they both scored very close to their test set performance across the board (~93.2 and ~95.0 respectively), indicating that they likely generalise well, and overfitting might be minimal. The reason Random Forest outperforms Decision Tree is likely due to the fact that it mitigates overfitting better (bagging, feature randomness due to 'sqrt'), and that it aggregates predictions from multiple trees to further emphasise its robustness.

RT 7: Regression

1. Linear Regression: For the Linear Regression, it was chosen to fit the parameters using maximum likelihood estimation (equivalent to minimising the squared error), since it is the most straightforward approach to regression and serves as a baseline model for comparison. There is not much of substance to say here regarding parameter choices.

2. Neural Network: The NN architecture was designed to capture potential nonlinear relationships in the data. This includes 4 layers with increasing and then decreasing hidden units (64->128->64) chosen based on empirical testing, chosen to capture nonlinearity while avoiding overparameterisation, the ReLU activation function due to its computational efficiency and avoidance of gradient saturation (the sigmoid function was tested as well, but it was severely outperformed), and the Adam optimiser with a learning rate of 0.001 to lend stable convergence without overshooting (chosen after testing higher and lower values), due to its adaptability and performance in minimising the loss function. Mean squared error was used as a loss function, given that the task is regression, and 450 epochs were set. This seems like a very large number given the dataset, but through thorough empirical testing, including cross-validation, it seems to be the best possible epoch number for the dataset, balancing learning and preventing overfitting. Given the small dataset, to prevent overfitting, smaller batch sizes of 16 were chosen, together with shuffle. Dropout and Early stopping were tested as well, but both seemed unnecessary, neither improving the performance on the cross-validated folds, so it was concluded that this, simpler model, was more suitable.

3. Bayesian Regression: Bayesian regression was implemented with weakly informative priors (N(0, 100)). A cubic model was chosen after analysing the data, which appeared to display a cubic relation. A simpler quadratic model was considered as well, but it was outperformed. A Half-Normal prior for σ was selected to constrain the scale to positive values and provide a reasonable range of variability. A uniform sigma was evaluated as well, but a half-normal one ended up being more suitable. Additionally, Student's t-distribution was opted for for the likelihood, to account for the data's uniform noise which deviates from normality. A Gaussian Distribution was examined, but student's t-distribution seemed to fit the data better overall, and through empirical testing and comparison of the two, it seemed more apt. The sampler used was NUTS (auto-selected by PyMC during runtime), although the Metropolis sampler was tested as well.

RT 8: Regression

Neural Network test set mean squared error: 10395773.309670439

Linear Regression test set mean squared error: 22848693.320845652

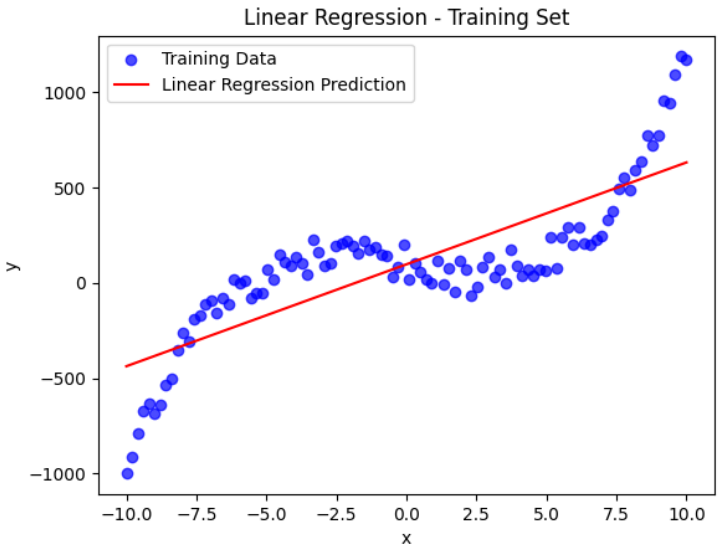

Figure 1: Linear Regression Training Set

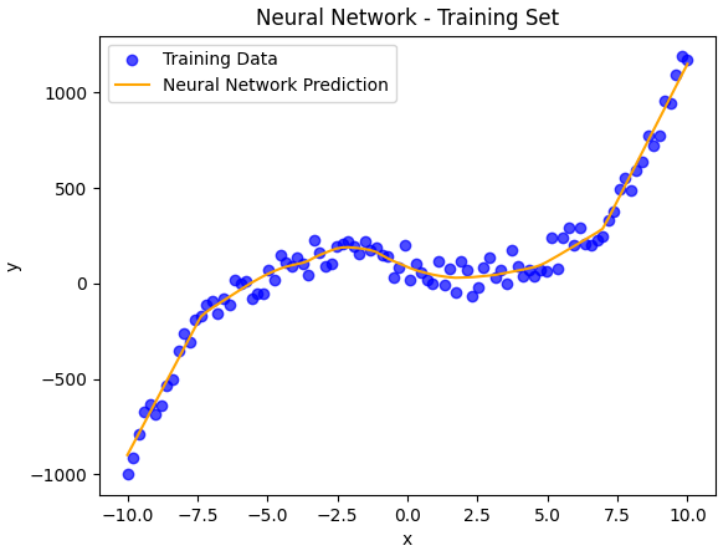

Figure 2: Neural Network Training Set

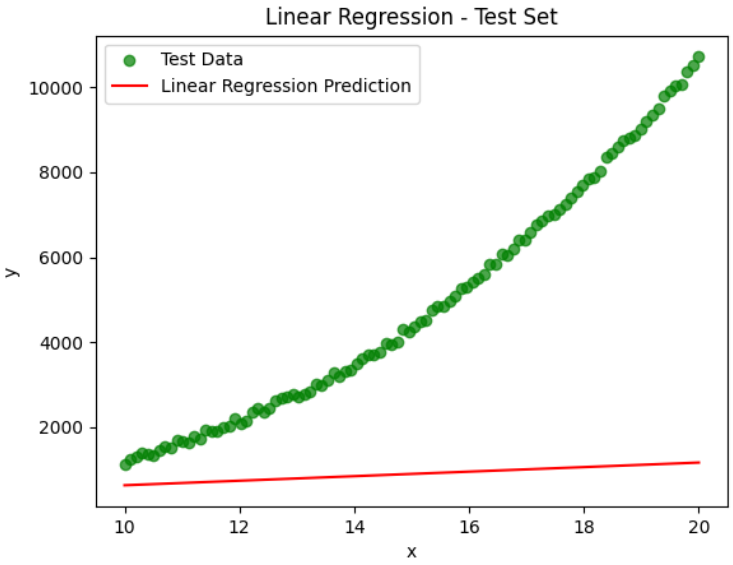

Figure 3: Linear Regression Test Set

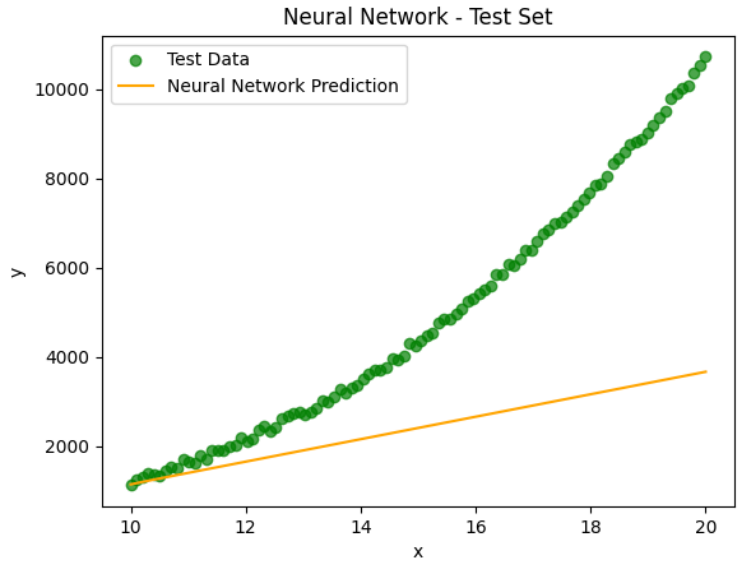

Figure 4: Neural Network Test Set

RT 9: Regression

Looking at the above figures and the MSE of each model, we can analyse their respective performance. The Linear Regression Model achieves a significantly higher MSE compared to the Neural Network. Looking at Figure 3, linear regression clearly fails to capture the nonlinear relationship of the data, as expected since its predictions are constrained to a straight line. This leads to substantial errors, and very poor overall performance on the test set. In contrast, the Neural Network achieves a much better test set MSE (less than half of that of linear regression), however it still displays quite a poor performance in Figure 4. Its predictions still seem quite linear, which is in stark contrast to its performance in the training set (Figure 2). The near linear predictions of the NN on the test set likely stem from the test data falling outside the range of the training data. Neural Networks are not inherently good at extrapolation, and in this case, it seems like the model reverts to a simpler, linear relationship when making predictions for unseen data. Although neither model performs exceptionally well, both modeling near linear relationships, linear regression performs poorly because it is fundamentally limited to modeling linear relationships, whereas the NN struggles to generalise effectively to the test set, causing its predictions to exhibit linear tendencies and reduced complexity. This is likely due to limited training data, and its potential overfitting to the training set, coupled with the test set covering a significantly different range. It

is still worth noting that, although neither model makes accurate predictions on the test set, the Neural Network still severely outperforms the linear regression model.

RT 10: HMM

Comparing the two HMM models, where one was given the true transition probabilities, and the other was not, yields a few interesting observations. When the model did not have access to the true probabilities, the learned transition matrix significantly deviated from the true values, and it failed to match the true grid structure. Some transitions that should not be possible (e.g. between non-neighbouring states) had non-zero probabilities. Without true transition probabilities, the model heavily relied on the observed sequences, leading to irregular patterns in the transition matrix that do not represent the underlying grid structure accurately. On the other hand, providing the model with the true transition probabilities led to a more accurate representation of the 3x3 grid, where transitions were limited to neighbouring states. This led to the model producing more reliable initial state probabilities, in turn removing ambiguity in learning and enforcing alignment with the true grid structure.

The above was reflected in the learned initial state probabilities as well. The model without true transition probabilities inferred state 4 as the most likely starting point based solely on the data, while the model that had access to the true probabilities distributed the initial state probabilities more evenly, although state 4 and 7 seemed to dominate. The emission probabilities in both tasks were deterministic (1.0) for all states, which is expected given the nature of the dataset (each state emits a single distinct symbol).

When it comes to confidence in estimated parameters, the fact that the learned transition probabilities deviate substantially from their true values in the model not provided with them, indicates that the model struggled to infer the correct structure of state transitions, in turn heavily biasing the initial state probabilities toward state 4. This suggests overfitting due to observed sequences. In contrast, the fixed transition matrix in the second model ensures accurate modeling of state transitions, making the estimated parameters more reliable. The learned initial state probabilities are also more reasonable, reflecting the constraints imposed by the true transition structure. Overall, I am much more confident in the parameters estimated by the model provided with the true probabilities due to the incorporation of prior knowledge. This comparison, in general, reveals a significant advantage of using true transition probabilities when possible.

References

1. Scikit-learn Documentation: https://scikit-learn.org/stable/
2. PyTorch Documentation: https://pytorch.org/docs/stable/index.html
3. PyTorch Tutorials: https://pytorch.org/tutorials/
4. hmmlearn Documentation: https://hmmlearn.readthedocs.io/en/latest/
5. PyMC Documentation: https://www.pymc.io/projects/docs/en/stable/learn.html
6. Stack Overflow – Generating Unique Pairs In Python: https://stackoverflow.com/questions/70413515/get-all-unique-pairs-in-a-list-including-duplicates-in-python
7. GeeksforGeeks - Hidden Markov Models: https://www.geeksforgeeks.org/hidden-markov-model-in-machine-learning/