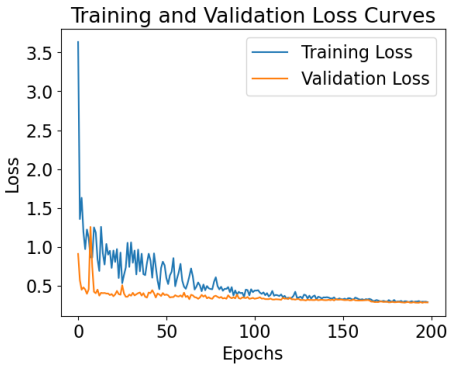
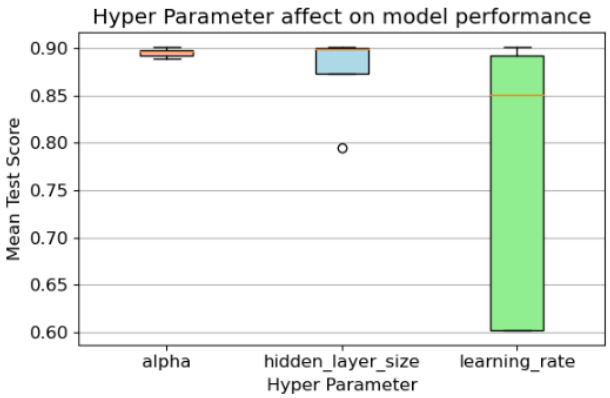
Classification



I trained a neural network MLP classifier on the activity recognition data set, firstly I applied a randomised search on my hyper parameters to identify which values produce a model with the highest accuracy on the given training data. In the hope of reducing overfitting, I utilised cross-validation when training the model to expose the model to mulitple training and validation sets. The graph displaying the training and validation loss curve depicts the relation between the model and data. The graph shows the model continues to performs significantly better on the training data until it reaches around 150 epochs, where it starts to plateau and little to no change occurs. The validation loss curve on the other hand shows a relatively flat curve when compared to the training loss curve and the flatness around a value of 0.5 indicates the model performs consistently well without any indication of over/underfitting.

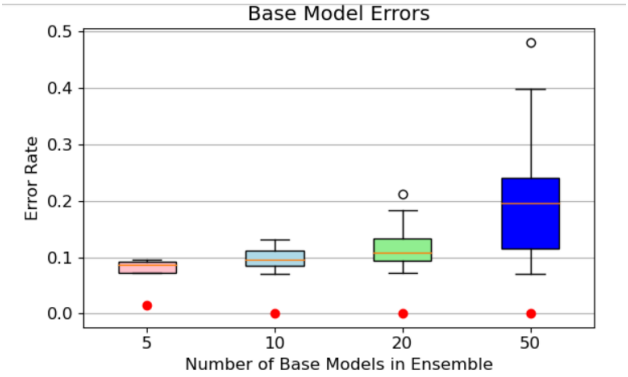


When evaluating the score of my model, it achieves an accuracy of 89.98% on the training data and 89.01% on the testing data. The high accuracy score in both the training and test data suggest the model performs well and neither under or over-fits the training data. To investigate performance further, I created box plots to show the affect each hyper parameter has on the performance of the model. I used the default parameters h as the baseline parameters. The box-plots clearly display the effect of the 'learning_rate_init' parameter, which influences the speed of convergence, having the largest affect on the models mean test score. The number of hidden layers appears to have a smaller affect on the performance than the learning rate but the outlier shows that a certain values for the hidden_layer_sizes can have a significant affect on the models performance. The regularisation parameter appears to have the smallest impact out of the tested hyperparameters on the model's performance. Due to the large range of the learning rate it is clear it has the highest affect on the model, followed by the hidden layer size.

Decision Tree Ensemble

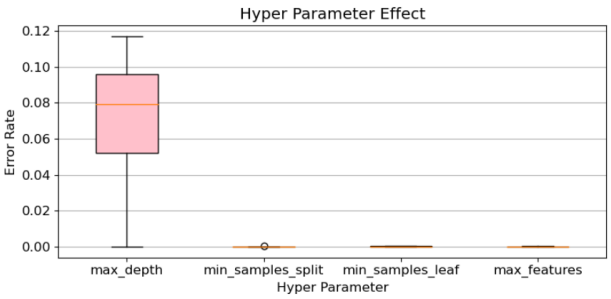
Ensembles ultimately reduce the error by taking the average output from multiple decision tree models, this theory is derived from the law of large numbers, that the average of multiple independently trained models will converge to the true underlying distribution. I decided to implement a Boosting ensemble (using the AdaBoost algorithm). Each base model is trained sequentially, where the next model focuses on learning from the mistakes of the previous model, aiming to significantly improve performance over other ensembles and even more so than just one decision

tree model. Instead of random sampling, I weighted the data points in the training set according to the performance of previous base models, these weights are chosen to try and minimise the exponential loss that the model gives us. The results from my ensemble show the accuracy on the data test set ranges from 98.44% to 99.99% depending on the number of models used within the ensemble.



The graph of box-plots shows the error rate for the individual models within the ensemble on the test data. The box displays the interquartile range and the red line resembles the mean. As expected, the more models in the ensemble, the wider range of error produced. The red dot is used to mark the overall error from the ensemble, when compared to the average individual model error, there is a clear, significant improvement in error especially as number of models increases. However, after 10 models the change in ensemble error is minimal, which may be valuable when training time becomes a desirable value when training a classifier, being able to achieve high accuracy with less models to train.

When exploring how sensitive an ensemble is to different hyper-parameters I used an ensemble consisting of 20 individual models and default parameters. The graph clearly displays that the ensemble model is incredibly sensitive to maximum number of levels in the tree (the max_depth parameter). Increasing the depth allows for a more complex model, enabling it to identify more complex relationships. Having a value too large for max depth increases the possibility of overfitting occurring. This can be counteracted by the minimum number of samples required to split a node, min_samples_split, ensuring enough samples are provided at each split. Increasing this parameter too much may cause the model to be too simplified, subsequently leading to underfitting.The minimum number of samples required to be at a leaf node, min_samples_leaf, has a similar effect to min_samples_split, reducing the risk of overfitting but may lead to overly generalised predicitons. The best model performance occurs when max_depth is set to None and the other parameters have little to-no effect.



Sequence Labelling

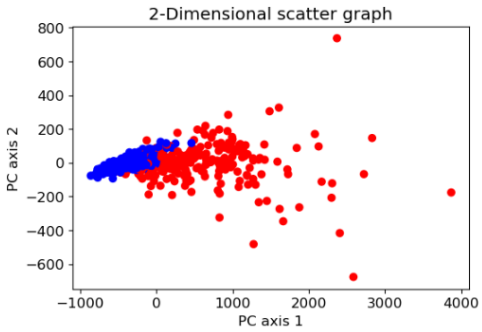
To train a sequence labeller, I decided to use a Gaussian HMM model, a type of neural network effective in capturing complex, non-linear relationships in the data. This takes into account the sequential nature of the data unlike the previous methods in task 1 and 2. The model I trained achieved an accuracy score of 90.50% on the test set of data. Due to the lower score from a

HMM model compared to the decision tree ensemble and similar score to the MLP classifier, it may suggest the sequential order of events is less informative for prediction than the aggregate or static features of the data.

The transition matrix highlights the likelihood from transitioning from one state to another (current state to new state). I normalised the results for better interpretability. Dominant transitions are shown by the red colour, this transition matrix clearly identifies the relationship between states and the intense red shows there is a dominant transition between states. By analysing the means I found that features 1,2 and 4 show a lot of variability, in particular feature 4, this suggests that it could be an important feature when predicting states. When reviewing the covariance, the features with the lowest variance across states are more likely to be a consistent indicator when predicting states. Features 1,2 and 3 show consistent variability, whereas feature 4 has high variance, suggesting it's not a reliable feature. Considering both, features 2 and 3 appear to be the most informative and stable features for predicting states.

When comparing the performance of all classifiers we can take into account multiple factors: training time, accuracy and interpretability. Starting with accuracy there is a clear winner, the AdaBoost Ensemble. Due to the significantly higher accuracy rate it would be the clear answer to the classifier with the highest accuracy, the HMM model would come second with the MLP model following closely in third. In terms of interpretability, the AdaBoost Ensemble is not as simple as the single decision tree but still quite good. The MLP classifier's, the black box, interpretability is significantly worse, providing a complex procedure in order to understand the decision making. The HMM model is very interpretable especially when using sequential data. The model can provide its states along with transition and emission probabilities, all of which can be examined to determine the decision making process of the model.

PCA

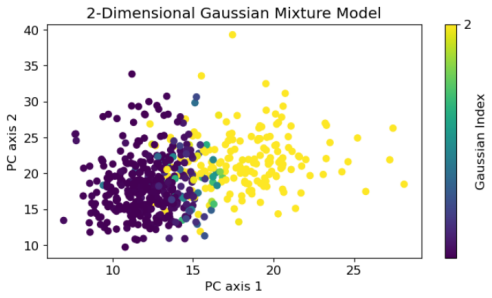


I used sklearn's implementation of PCA to reduce the dimensionality of the breast cancer data set down to 2 principle component axis. The variance explained by the first principle component is equal to 442188.2 and 7169.7 for the second principle component. To make this easier to interpret, principle component one explains 98.23% of the variance. The red and blue scatter graph represents the data transformed by PCA, the blue dots represent benign diagnosis' and the red for malignant.

The second graph which features a colour bar on the side represents the responsibilities of each data-point, those data-points belonging to certain percentages of either distribution 1 or 2, yellow and purple

respectively. The Gaussian Mixture Model applied to the reduced data applies soft clusters to each of these data points, determining the likelihood of that data-point belonging to a specific Gaussian, the soft nature of clustering explains the need for a spectrum of colour rather than a binary 2. The second graph displaying the responsibilities of each data-point show the data's clustering and the probabilistic nature of these clusters. Whereas, the first graph displays data along axes which correspond to directions of maximum variance and data has not been clustered in anyway, hence the difference in the spread of data-points.

These new axes allow us to see the overall spread of data and the directions of variance. The newly obtained axis from PCA with reduced data dimensionality reduces the number of parameters needed by the Gaussian Mixture Model to cluster the data, resulting in a more reliable and stable performance.



SVM classifier

I trained my SVM classifier using hyper-parameters found by a grid search. The hyper-parameters I obtained were: and they produced a result of 93.22% on the test set. I then trained an SVM on the dimensioanlly reduced data from appling PCA. I obtained new hyper-parameters: and these gave a result of 94.07% accuracy on the test set. There is a slight improvement using the 2-dimensional approximation to the original data lead me to believe it helps reduce the SVM model overfitting, the fewer features can simplify the model whilst retaining the most significant features. The reduced dimensionality may have reduced the noise within the data, further removing insignificant information from the data.

Bayesian Linear Regression

In order to prepare the data for linear regression I made a few changes. First of all I extracted information from the date, including the day and month. I excluded the year as this was unnecessary due to the data set only containing 1 year's worth of data. I then encoded the seasons and functional day columns from string values to integers. Next I changed the hour parameter from integer to a sin and cosine, capturing the cyclical relationship of these values. I applied the same procedure for the days feature, next I removed the holiday parameter as this was the direct inverse of functioning day so I deemed it irrelevant. I assigned the rented bike values to y and the predictors to X, converting all values in X to floats.

I chose my priors based on the effect I believed each predictor in X to have on the rented bike count. For example, the effects I believe winter, rainfall and snowfall would have on the number of bikes rented would be negative, therefore I assigned them a normal distribution with a negative mean. Because I was confident about these assumptions I assigned a relatively low standard deviation on 1 and an upper bound of 0. For other predictors I assigned normal distributions, where I was confident I assigned a lower standard deviation and the believed effect was represented in the magnitude of the mean. For the flat priors, the parameters I was uncertain about I set the mean to zero and added a high standard deviation of around 15.

The results arviz summary results:

	Mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	rhat
beta_temp	55.17	1.74	51.94	58.49	0.03	0.02	3014.05	3166.27	1
beta_humidity	-3.79	0.34	-4.41	-3.16	0.01	0	3496.52	3337.51	1
beta_wind	39.13	4.74	30.21	47.51	0.06	0.04	6716.52	2777.56	1
beta_visibility	0.03	0.01	0.01	0.05	0	0	5362.14	3126.2	1
beta_dew	-23	1.69	-26.18	-19.75	0.03	0.02	2956.08	2685.22	1
beta_rad	-91.9	7.61	-106.29	-77.69	0.1	0.07	5604.15	3068.72	1
beta_month	11.53	1.58	8.54	14.5	0.02	0.01	5924.72	3241.68	1
beta_spring	34.41	8.11	18.5	48.88	0.1	0.07	6864.83	2859.45	1
beta_summer	-17.1	8.73	-34.16	-1.58	0.09	0.08	8825.1	3051.12	1
beta_func_day	2.06	1	0.26	3.93	0.01	0.01	8704.98	2711.34	1
beta_hour_sin	-13.2	1.94	-16.72	-9.43	0.02	0.02	7495.04	2629.38	1
beta_hour_cos	0.1	1.95	-3.46	3.93	0.02	0.04	8096.14	2750.49	1
beta_day_sin	-0.59	0.99	-2.38	1.34	0.01	0.01	8453.22	2950.05	1
beta_day_cos	0.46	0.99	-1.34	2.36	0.01	0.02	8174.32	2622.37	1
beta_rain	-4.16	0.98	-6	-2.36	0.01	0.01	7263.75	2537.73	1
beta_snow	-1.19	0.75	-2.5	-0.01	0.01	0.01	3347.47	2021.04	1
beta_winter	-1.57	0.87	-3.05	-0.01	0.01	0.01	2833.36	1845.92	1

Due to the long list of parameter values I only included those that were standardised for readability. From the table provided above we can cleary see all ”rhat” values are equal to 1 suggesting good convergence for each parameter. Then estimates sample size (ess) bulk and tail are mostly sufficiently large compared to the size of the data set suggesting a good number of effective samples. These values infer that MCMC sampling has generated reasonable approximations to the posterior distributions.

Some surprising posterior mean values are the wind speed(beta_wind) being positive, summer(beta_summer) being negative, which I initially believed would have been the opposite for each. Other surprising posterior means are the temperature, dew, winter and snow (beta_temp, beta_dew, beta_winter, beta_snow), for temperature and dew, both have significantly higher means than expected, however this could be due to the low estimated sample size. I believe this could be the same reason for the low posterior means of winter and snow.

Due to the relationships in the data being complex, there are significant interactions between predictors, I believe linear regression might oversimplify these relationships. Additionally, linear models may not adequately capture the cyclical nature of time-related variables (like hours and days) or the unique characteristics of binary variables I used to encode the seasons. In conclusion, while linear regression provides a straightforward approach to modeling and can offer valuable insights, it’s essential to consider its limitations. A more complex model might be necessary for capturing the underlying patterns and relationships between predictors more effectively.