Machine learning report:

2. Classification and Sequence Labelling:

Classification: Classification is a type of supervised learning task in where the goal is to assign input data points to one of N categories/classes. The algorithm learns from a labelled training dataset, which consists of input-output pairs. The First 2 Tasks will analyse pertinent classifier models namely a Neural Network and an Ensemble method (utilising decision trees as its base model).

Task 1 - Neural Networks: A neural network is comprised of a set of interconnected nodes organized in layers with a set of weights and an activation function describing the connections between the nodes. Neural networks are generally feedforward where the output of one layer is the input of the next layer. The weights are often updated in a two step-process called Gradient Descent & Backpropagation. The purpose of the gradient descent algorithm is to minimize the loss which is achieved through backpropagation at each iteration. This results in an equation of the form: $w_{ji} = w_{ji} - \Delta w_{ji}$

1a) For low train sizes (i.e., 500) Training Accuracy was 0.997, and a corresponding Validation accuracy of 0.567 showing that the model was overfitting the model. Overfitting occurs when the model memorizes the training data but fails to generalize well to new data.

Training Score	0.957 (to 3d.p.)
Validation Score	0.906 (to 3d.p.)
Testing Score	0.900 (to 3d.p.)

Table 1: Accuracy scores for a train size of ≈ 26000

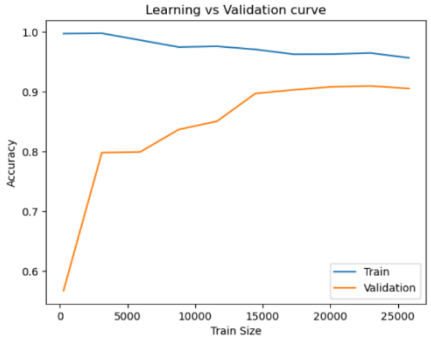


Figure 1: Learning vs Validation Curve for different train sizes.

As the train sizes I observed a differing trend for train and validation scores, where as the train size increased, the train size dropped to a score of 0.957 and the validation score increasing to 0.906 (Table 1), showing both that the model has learned the patterns in the training dataset, indicating a good degree of generalization to new, unseen data.

1b) A test score of 0.900 (Table 1) indicates that approximately 90.0% of the test samples were correctly classified. The test score is consistent with the validation score suggesting that the model is maintaining its performance on new, unseen data. The consistency between the validation and test scores, also suggests that the model is not overfitting the validation set, and the performance observed is likely to carry over to new, independent (related) datasets.

1c) Hyperparameters are configuration settings external to the model that cannot be learned from the training data. They are parameters that need to be set prior to training a model and separate from model parameters (which are learned during training). Hyperparameter tuning is crucial for several reasons, such as: optimising model performance, avoiding overfitting or underfitting, reducing training time etc.

My method for performing hyperparameter tuning was using Grid Search on each hyperparameter I deemed would result in better performance. These include max_iter, alpha, hidden_layer_sizes, solver, and learning_rate_init. Despite this, there were only 2 hyperparameters that led to interesting results and consequently a strong effect on model performance. These were namely: alpha and hidden_layer_sizes:

(a) (b) (c) (d)

Figure 2: Plots for Train and Test Scores for different values within a hyperparameter

alpha: A L2 regularization term which adds a penalty term to the loss function based on the square of the magnitude of weights, helping prevent overfitting. New loss: $Loss = Original\ Loss + \alpha \sum_{i=1}^{n} w_i^2$,

where α = alpha and w_i = the weights

Trend: As the value of alpha increases, the test score initially improves (Figure 2a), peaking at alpha = 0.1, however beyond this point, the test score starts to decrease. This indicates that too much regularization might be hindering the model's ability to capture underlying patterns in the data, resulting in a decrease in generalization to new data. A similar pattern occurs when training the data (Figure 2b [blue line]) but occurs at a slower rate and not as rapidly.

Conclusion: Lower values of alpha (closer to 0) might allow the model to overfit the training data, while higher values of alpha (beyond 0.1) might lead to underfitting (where the model is too constrained to learn the underlying patterns).

hidden_layer_sizes: The number of neurons in each hidden layer (In our case 1 hidden layer).

Trend 1: As the size of the hidden layer increases, the test score initially improves, reaching its peak at (32,) with a test score of 0.907 and then gradually decreasing to a test score of 0.893 for hidden layer size (512,) (Figure 2c).

Trend 2: On the other hand, train scores for hidden layer sizes showed markedly different behaviour. A larger hidden layer size allows the model to better fit the training data, resulting in higher training scores such as 0.970 for an HLS of (512,) (Figure 2d).

Conclusion: The divergence between the training and test scores, especially with larger hidden layer sizes, suggests a potential risk of overfitting. The model may be memorizing the training data without generalizing well to new data.

Task 2 - Ensemble methods: An ensemble is a combination of multiple models that is often more robust and accurate due to combining the strengths of one model which can compensate for the weaknesses of another, leading to improved overall performance. An example of one of these methods is boosting. This involves sequentially training weak learners and giving more weight to datapoints that were misclassified in the previous iterations. This way, subsequent models pay more attention to challenging instances.

An example of a popular and widely used implementation of this algorithm is AdaBoost with a key benefit of this approach being can use any weak learner (/machine learning method).

2a + b)

Table 2: Accuracy scores comparing an ensemble to a single model.

Classifier	Training Accuracy	Validation Accuracy	Testing accuracy
Single Model (DecisionTreeClassifier)	1.000 (to 3 d.p.)	0.867 (to 3 d.p.)	0.845 (to 3 d.p.)
Ensemble (AdaBoostClassifier)	1.000 (to 3 d.p.)	0.860 (to 3 d.p.)	0.852 (to 3 d.p.)

Both the Single Model and the Ensemble Model achieve perfect training accuracy (1.000), indicating that they have overfit / memorised the training data (Table 2). This is also evidenced by the lower validation score of 0.867 and 0.860 for a single model and ensemble respectively (Figure 3a,3b).

Despite this, the Ensemble Model, exhibits a slightly higher testing accuracy (0.852) compared to the Single Model (0.845) (Table 2). This improvement suggests that the ensemble approach (specifically AdaBoost) has enhanced the model's generalization performance on unseen data.

(a)

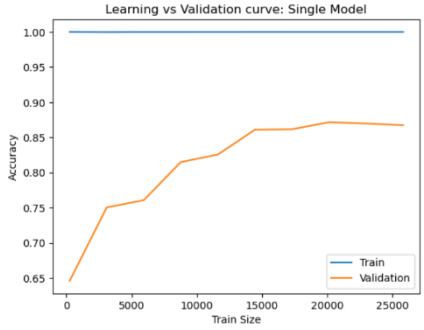
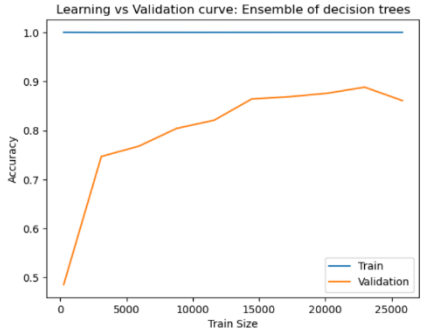


Figure 3: Plots for Train and Validation scores for a single decision tree vs an ensemble of decision trees

(b)



2c) To understand if improves performance we most first set out some definitions to better understand this.

Error rate of an ensemble: $E_{COM} = \mathbb{E}_X\left[\left(\frac{1}{M}\sum_{m=1}^{M}(y(x) - y_m(x))\right)^2\right]$

Error rate of an individual model: $E_{AV} = \frac{1}{M}\sum_{m=1}^{M}\mathbb{E}_X\left[(y(x) - y_m(x))^2\right]$

$\Longrightarrow E_{COM} = \frac{1}{M}E_{AV}$

and for $E_{COM} = \frac{1}{M}E_{AV}$ to hold, 2 assumptions must be satisfied. These are:

1) The errors of each model have zero mean

2) The errors of different models are not correlated

As discussed in the course these are quite extreme assumptions, as errors - in practice, are highly correlated and biased. For example, AdaBoost, involves adjusting weights based on the performance of previous models, introducing a form of correlation in the errors.

In Theory however we can reduce the inequality above to $E_{COM} \leq E_{AV}$ by Jensen's Inequality which states: $f(\mathbb{E}[X]) \leq \mathbb{E}[f(X)]$

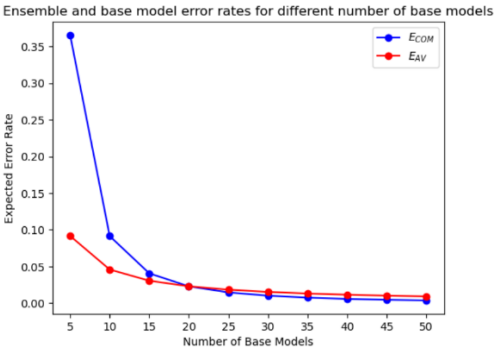


Figure 4: E_{COM} against E_{AV}

We can reach this inequality by letting $f(X) = X^2$ which in turn $\Longrightarrow (\mathbb{E}[X])^2 \leq \mathbb{E}[X^2]$, which we can in turn apply to our error rates to get.

$E_{COM} = \mathbb{E}[(ensemble\ error)^2]$ and $E_{AV} = (\mathbb{E}[(individual\ model\ error)])^2$ and consequently: $E_{COM} \leq E_{AV}$

From running the code above with the formulas for E_{COM} and E_{AV}, (where in my code I substituted $(y(x) - y_m(x))$) for error rate: $\varepsilon = 1 - accuracy$), I am able to graphically show: $E_{COM} \leq E_{AV}$ (Figure 4).

As the number of base models (Decision Trees) increases, the error rate of the ensemble (AdaBoost) consistently decreases, with the inequality is met by "number of base models" = 20, where $E_{COM} = 2.285 * 10^{-2} \leq E_{AV} = 2.290 * 10^{-2}$ (Table 3).

As the number of models keep increasing the magnitudes of improvement (reduction in error rates) become substantial, with the last reading for "number of base models" = 50, $E_{COM} = 3.655 * 10^{-3}$ and $E_{AV} = 9.159 * 10^{-3}$ (Table 3) which is a 250% decrease in error rate (Since $2.5 * E_{COM} = E_{AV}$).

No. of models:	5	20	35	50
E_{COM}:	0.36553	0.02285	0.007460	0.003655
E_{AV}:	0.09159	0.02290	0.013085	0.009159

Table 3: Ensemble and individual model error rates (intervals of 15)

Conclusion: From the data collected, I can say that Ensembling does improve performance as the number of base models in the ensemble increases.

2d) To evaluate how sensitive the AdaBoost classifier is to different hyperparameters, I first had to tune its base classifier. Here I found that, despite tuning hyperparameters such as: max_features, min_samples_leaf and min_samples_split, the only Hyperparameter to have a meaningful effect on test performance was max_depth for the DecisionTreeClassifier.

From (Figure 5a) we can see, that increasing max_depth leads to worse test performance. If the tree becomes too deep, it may capture noise and outliers in the training data, leading to overfitting which is what we are potentially seeing.

For the AdaBoost classifier, (after carrying over the best Hyperparameters found for the base classifier using Grid Search), the hyperparameter to have the biggest improvement in test performance was, the n_estimators hyperparameter. Increasing this value lead to an increase in performance from 0.854 (to 3d.p.) (for n_estimators = 5) to a peak of 0.896 (to 3d.p.) (for n_estimators = 85) (Figure 5b).

(a)

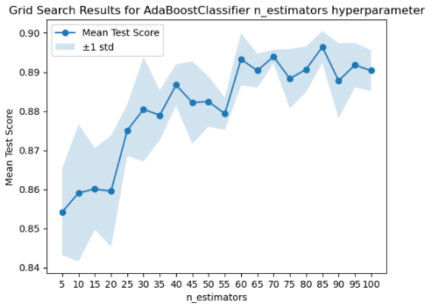
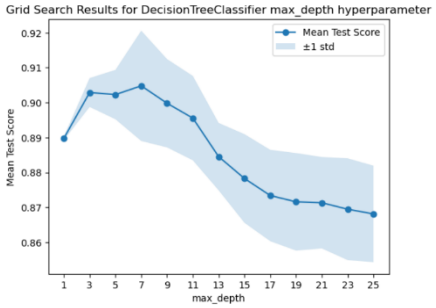


Figure 5: Test Scores for different values within a hyperparameter



(b)

Also, worth noting that the change in improvement investigated from a Single model to a tuned AdaBoost Classifier is 0.896 - 0.845 = 0.051, a meaningful enhancement in the model's ability to make accurate predictions on new data and showing improved generalisation.

Task 3 - Sequence Labelling: A sequence labeller assigns a categorical label to each data points in a sequence. Unlike traditional classification tasks where each input is treated i.i.d, sequence labelling models consider the sequential nature of the data, often using a (1st Order) Markov Chain. An example of such a state space model is a HMM, where the latent variables (z_n) are discrete and the observations (x_n) may be discrete or continuous. Alongside sequence labelling, a HMM can be used to predict the next latent state or even predict the next observation, using the Viterbi algorithm (a max-sum algorithm). The complete HMM model can be defined by the joint distribution over observations and latent states: $p(X, Z|A, \pi, \phi) = p(z_1|\pi)\prod_{t=2}^{T} p(z_n|z_{n-1}, A)\prod_{t=1}^{T} p(x_n|z_n, \phi)$. This can be marginalised by summing over all the possible states (z) and have the log of it taken: $\ln\big(p(X, Z|A, \pi, \phi)\big) = \ln(\sum_Z\{p(X, Z|A, \pi, \phi)\})$ to give us the log likelihood, which we can use in the Forward-backward algorithm to estimate the HMM parameters: A - Transition matrix, π - Initial state probabilities and ϕ – Parameters of the emission distribution. The forward-backward algorithm has a runtime is $O(k^2N)$ which is better than going through each of the possible sequences of length n with k latent states, which would give a runtime of $O(k^n)$

3a) The model's ability to achieve a testing accuracy (0. 855) higher than the training accuracy (0.815) is a positive sign of generalization. It indicates that the model has learned patterns that are applicable beyond the specific sequences in the training set, something further evidenced by its ability to get high accuracy scores for sequences lengths of all lengths in the test set, including long ones (Table 4).

Table 4: Example scores for sequences in the test split.

Sequence Length	No of correct predictions	Accuracy
4563	4223	0.925
2542	2161	0.850
3339	2651	0.794

3b) Interpretation of the matrix:

Diagonal Elements (Self-Transitions):

- The diagonal elements represent the probabilities of staying in the same state (label) from one time step to the next.
- For most of the states the next most likely state is to stay in is its current state, indicating that the actions / labels take place occur over a long duration of time and sequences tend to persist in the same activity.

Off-Diagonal Elements (Transitions between Labels):

- The off-diagonal elements represent the probabilities of transitioning from one state to another.

Label 3: (Label that is marginally different from other labels)

- Lowest self-transition probability compared to other labels
- Moderate probabilities of transitioning to sitting on the bed (Label 0) and sitting on a chair (Label 1) (Table 5) indicating potential patterns in transitions from ambulating to these activities.

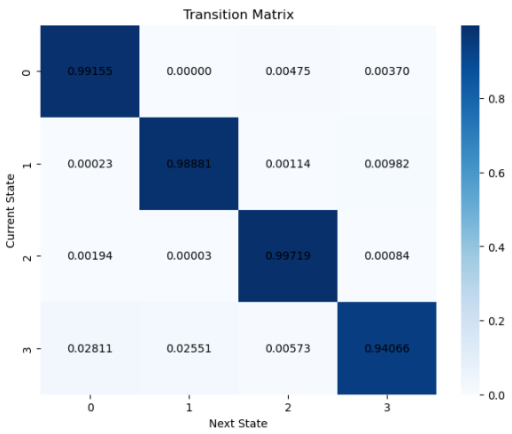


Figure 6: Transition matrix for the sequence labeller

Conclusion: The high probabilities of self-transitions suggest temporal continuity in activities.

3c) Taking the largest two means for a feature for each state which happens to be features 1 and 2 which correspond to frontal acc, vertical acc respectively (Table 6). The magnitude of features 1 and 2 as well appearing consistently across different states shows that these are particularly informative when predicting activity labels.

State 0 has the highest mean (1.26), indicating that positive values of vertical acceleration are more likely in this state.

Label	0	1	2	3
Description	sit on bed	sit on chair	lying	ambulating

Table 5: Labels and their corresponding semantic meaning

State	0	1	2	3
Feature 1	-1.18	-0.48	0.72	-1.69
Feature 2	1.26	1.03	-0.78	1.29

Table 6: States and the means from the emission distributions for features 1 and 2

3d) For there to be a fair comparison but each of the classification methods, the same random seed is used when generating the train_test_split so that the training and test data used is the same. Also, all the X_train and X_test data has been scaled to have $\mu = 0$ and $\sigma = 1$ using StandardScalar which employs: $z = \frac{x - \mu_X}{\sigma_X}$. Good as impact of each feature is not influenced by its scale.

Models (with Hyperparameter Tuning)	Runtime (Averaged over 20 runs)	Test Accuracy
MLPClassifier	2.635s (to 3.d.p)	0.909 (to 3.d.p)
AdaBoostClassifier	4.133s (to 3.d.p)	0.896 (to 3.d.p)
GaussianHMM	0.528s (to 3.d.p)	0.855 (to 3.d.p)

Table 7: Overview of different classifier models

Performance: MLPClassifier tends to offer high performance but comes at the cost of potentially longer training times. AdaBoost offers a good balance between performance and training time. GaussianHMM is specialized for sequential data but may not perform as well on certain tasks due to the assumption that the hidden states are homogeneous, which may not always be the case in real-world scenarios.

Training Time: GaussianHMM is the fastest in terms of training time, followed by AdaBoost and MLPClassifier.

Interpretability: AdaBoost can provide insights into feature importance, allowing some level of interpretability. GaussianHMM also provides interpretability through hidden states but assumes homogeneity. MLPClassifier also known as "black-box models" are less interpretable due to its neural network architecture making it challenging to interpret the learned features and decisions.

3. Clustering and dimensionality reduction:

Principal Component Analysis (PCA) is a dimensionality reduction technique used to transform high-dimensional data into a lower-dimensional space while retaining as much of the original data's variability as possible. PCA first calculates the principle components (defined as the eigenvectors of the sample covariance matrix ordered by eigenvalue) then represents each datapoint x_n (where $x_n \in \mathbb{R}^D$) as a linear combination of these principle components: $x_n = \sum_{i=1}^{D} (x_n^T * u_i)\, u_i$. From this formulation, we typically only keep $M < D$ of these dimensions and we can simply do this by changing the Upper limit of the summation to M as the Principal Components (u_i) are ordered by variance contribution. For the example below we pick $M = 2$.

Task 4:

Percentage of variance explained by PC1: 98.2% (to 3 s.f.)
Percentage of variance explained by PC2: 1.62% (to 3 s.f.)
Cumulative percentage of variance explained by PC1 and PC2: 99.82% (to 4 s.f.)

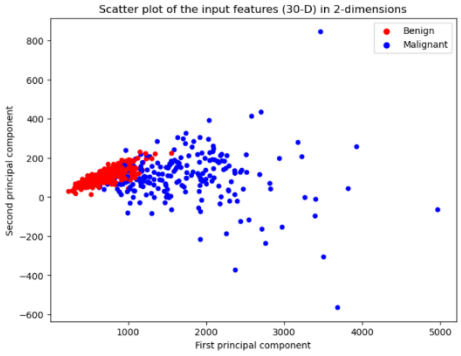


Figure 7: Scatter plot of 2-D data derived from PCA.

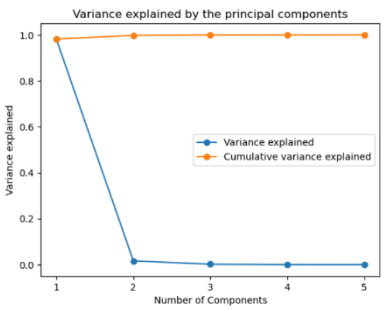


Figure 8: Line graph showing variance explained by PC's.

Task 5:

A Gaussian Mixture Model (GMM) is a probabilistic model that represents a mixture of multiple Gaussian distributions. GMMs provide soft assignments, meaning that each data point is associated with multiple Gaussian components with associated probabilities. The pdf for a GMM is given by the weighted sum of individual Gaussian component distributions:

$p(x) = \sum_{k=1}^{K} \pi_k * \mathcal{N}(x|\mu_k, \Sigma_k)$. For GMM to work well, we must find Gaussian parameters (μ_k and Σ_k) that maximise the log likelihood: $\ln\big(p(X|\pi,\mu,\Sigma)\big) = \sum_{n=1}^{N} \ln\{\sum_{k=1}^{K} \pi_k * \mathcal{N}(x_n|\mu_k, \Sigma_k)\}$, using the E-M algorithm on the formulation: $\ln\big(p(X|\theta)\big) = \mathcal{L}(q,\theta) + KL(q||p)$, where $q(Z)$ is any distribution over the latent variables.

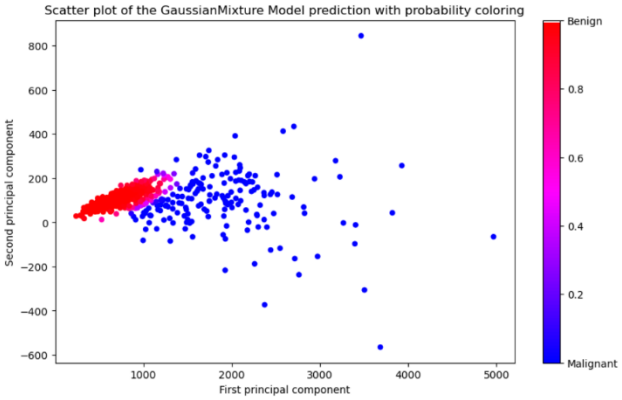


Figure 9: Scatter plot of 2-D data derived from GMM.

Task 6:

Main differences between graphs:

- The GMM plot gives us soft clustering, where data points may have varying degrees of colour intensity, indicating the likelihood / uncertainty of belonging to different clusters.

- GMM assumes that data points within a cluster are generated from a mixture of Gaussian distributions. As a result, the clusters can have different shapes and orientations due to having different covariance structures. For example, in Figure 9 the cluster for 'Benign' data has an ellipsoidal, non-isotropic shape.

4. Classification with SVMs:

SVM aims to find an N-Dimensional hyperplane that best separates the data into different classes. The optimization objective of SVM is to maximize the margin (using support vectors) while minimizing the classification error. We can classify a new datapoint x using: $y(x) = \sum_{n=1}^{N} a_n t_n k(x, x_n) + b$.

In the equation the term $k(x, x_n)$ is a kernel function, which defines a degree of similarity between two arguments and is an essential part for both learning parameters in the classifier as well as using it classify new points. SVM's employ what is known as the kernel trick which allows the algorithm to implicitly map the input data into a higher-dimensional space without explicitly computing the transformed feature vectors. Commonly used kernels include: The Linear Kernel: $k(x, x') = x^T x'$ and the RBF kernel: $k(x, x') = \exp(-\gamma \parallel x - x' \parallel)$, where the implicit feature space is ∞-Dimensional. Also, for a kernel function to hold true, they are required to be symmetric, and any Gram Matrix K must be a positive semidefinite matrix.

Task 7:

I performed GridSearch on SVM using a Cross Validation value of 5. Cross-validation is a statistical technique used to assess the performance and generalization ability of a machine learning model. It works by dividing the dataset into multiple subsets, training the model on some of these subsets, evaluating its performance on the remaining validation data. This process is repeated multiple times, and the performance metrics are averaged to provide a more robust estimate of the model's performance.

Table 8: SVM maximisation of test set score on full data

Kernel Function	Optimal Hyperparameters	Test Score
Linear	{'C': 5}	0.933 (to 3 d.p.)
RBF	{'C': 25, 'gamma': 0.0001}	0.916 (to 3 d.p.)

Task 8:

Table 9: SVM maximisation of test set score on 2D data

Kernel Function	Optimal Hyperparameters	Test Score
Linear (Table 8)	{'C': 5}	0.924 (to 3 d.p.)
Linear (New)	{'C': 0.01}	0.933 (to 3 d.p.)
RBF	{'C': 25, 'gamma': 0.0001}	0.916 (to 3 d.p.)

Task 9: The formula for the objective function of a linear SVM with regularisation is:

$$\min_{w,b,\xi} \frac{1}{2} \| w \|^2 + C \sum_{n=1}^{N} \xi_n \text{ subject to } t_n(w^T \phi(x_n) + b) \geq 1 - \xi_n, \xi_n \geq 0, \forall n = 1 \ldots N$$

The 2-dimensional approximation led to a decreased test accuracy if the same regularisation value (C=5) was used for the Linear SVM. A high value of C imposes a stronger penalty for misclassification, meaning the model is more focused on fitting the training data accurately which may lead to a decision boundary that closely follows the training data, possibly resulting in overfitting as evidenced by a lower test score of 0.924. When a lower regularisation value was used (C=0.01), the test score increased back to 0.933 (Table 9). This is possibly because a low value of C allows for a more relaxed regularization, allowing the model to have a smoother decision boundary that is less influenced by individual data points and is more generalized.

5.Bayesian Linear Regression:

Bayesian Linear Regression is a probabilistic approach to linear regression that incorporates Bayesian principles. Unlike classical linear regression, which provides point estimates for model parameters, BLR provides a full probabilistic distribution over the parameters. This allows for uncertainty quantification in both the model parameters and predictions and enables tractable sampling conditioned on observed values. In this task we utilise PyMC which is a probabilistic programming framework which features Markov Chain Monte Carlo sampling algorithms such as the 'No U-Turn Sampler' (NUTS). Whilst doing MCMC we also:
1) Throw away (earlier) samples as we go along as datapoints early on are usually distant from the target.

2) Running independent chains to check for convergence to the target distribution. For the tasks below I use 4 independent chains.

Task 10:

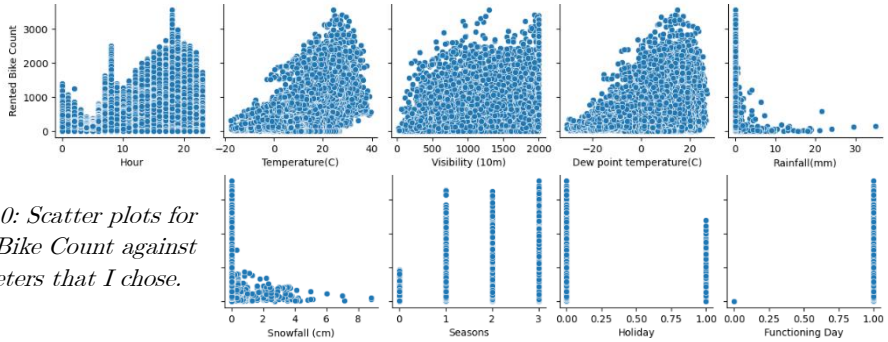


Figure 10: Scatter plots for Rented Bike Count against parameters that I chose.

Data that I have not utilised and the reason: Date: Not Relevant, Rented Bike Count: Thing we're trying to predict shouldn't use it in the model. Humidity, Wind Speed and Solar Radiation: No real relationship, data appears "random".

Data that I chose: reason and {mapping} (if any):

- Hour: Decent indicator, largest number of rented bike's occur at key hours, bimodal with peaks at Hour 8 and Hour 18
- Temperature, Dew Point temperature: Good indicator, more cycles rented when the temperature is higher
- Visibility: Slight relationship, where the higher the visibility the more bikes rented (clearer close to 0)
- Rainfall and snowfall: Decent indicators when values greater than 0 as the number of cycles rented are much lower when there is a non-zero presence of either of those values.

Seasons: Decent indicator, like temperature shows more cycles are rented during "hotter" seasons. {'Winter': 0, 'Autumn': 1, 'Spring': 2, 'Summer': 3}

Holiday: Decent indicator, shows more people rent cycles when it's not a holiday possibly for commute to work or education etc. {'No Holiday': 0, 'Holiday': 1}

Functioning day: Great indicator, as when it is not a functioning day the number of cycles rented is 0. {'No': 0, 'Yes': 1}

The data was also standardised (using StandardScaler) to allow the chains to converge more efficiently. This lead to a reduction of runtime from ≈ 60 mins to ≈ 10 mins.

Task 11:

Setting prior distributions for all predictors as $X \sim \mathcal{N}(0, 20)$ bar sigma which had a prior distribution of $p(\sigma) = U(0,20)$. This was done due as the choice of a wide Gaussian prior can make the model less sensitive

to outliers in the data. The prior encourages the model to place less emphasis on extreme parameter values, which can be beneficial in the presence of noisy or outlying observations.

Task 12:

Posterior distribution over the model parameters:

$$P(intercept, Hour, Temp, Visibility, DewPointTemp, Rainfall, Snowfall, Seasons, Holiday, FunctioningDay, \sigma|D)$$

$$y_{est} = intercept + Hour * x_i + Temp * x_i + Visibility * x_i + DewPointTemp * x_i + Rainfall * x_i + Snowfall * x_i + Seasons * x_i + Holiday * x_i + FunctioningDay * x_i$$

	intercept	Hour	Temperature(C)	Visibility (10m)	Dew point temperature(C)	Rainfall(mm)	Snowfall (cm)	Seasons	Holiday	Functioning Day	sigma
mean	704.52	196.06	517.54	47.17	-202.06	-74.22	-4.25	0.06	0.41	-1.98	20.0
sd	0.21	0.23	0.66	0.25	0.64	0.22	0.22	0.01	0.00	0.01	0.0
r_hat	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.0

Figure 11: Using arviz.summary to generate summary statistics for the posterior distributions for each parameter in your model

The $\hat{R}$ quantity is the Gelman-Rubin statistic which measures how similar our Markov chains are. A value close to (say less than 1.1) is evidence that the chains have converged as shown here.

Task 13: From the trace plots generated we can see insights into the convergence indicating whether the MCMC sampling has generated reasonable approximations to the posterior distribution. These include:

Stationary chains that fluctuate around a stable mean and consistency across multiple chains converging to the same point.

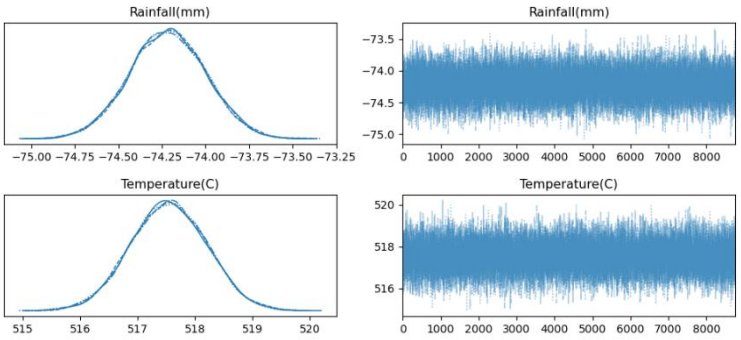


Figure 12: Scatter plots for Rented Bike Count against parameters that I chose.

Task 14: Generally, the posterior mean values from figure 11, are as expected, with values such as Hour, Temperature and visibility all having strong positive values indicating a positive relationship with the predictor variable. As well as Rainfall, Snowfall and Functioning day having strong negative value indicating a negative relationship with the predictor variable.

The Dew point temperature with a value of (-202.06) was surprising however, as from the data and plots, there was evidence to suggest that there was a positive relationship with the number of bikes sold. But the posterior mean value suggests an inverse relationship meaning an increase in dew point temperature is associated with a decrease in the number of bikes hired. My hypothesis for why this was the case was because this variable is strong related to Temperature, leading to temporal autocorrelation. The reason this is bad is because it can lead to incorrect assessments of parameters and its relationship with other model parameters, as seen here.

Task 15: By using arviz's built in az.r2_score, I observe $R^2 = 0.51476$. This shows that approximately 51.48% of the variance in the dependent variable is explained by the Bayesian linear regression model. This is a moderate level of explanatory power.

Furthermore, $R^2 = 0.51476 \implies R = 0.717$ (to 3 d.p.) which suggests a positive correlation between the predicted values from the linear regression model and the actual values and a moderate correlation. The conclusion I can draw from this is that the BLR model is a decent predictor, and moderately suitable for the task provided based on the posterior distribution I used.