# MACHINE LEARNING COURSEWORK REPORT

## 1 Classification and Sequence recognition

The dataset used in this section is Activity Recognition of Healthy Older Adults Using Battery-Free Wearable Sensors. Neural network classification, Ensemble tree, and sequential labelling were applied respectively.

**Neural Network Classifier** Neuron Network Classification is used for feature categorisation, which has three different types of layers - an input layer, a hidden layer, and an output layer. The input layer is responsible for receiving data and transmitting them to the hidden layer. The hidden layer consists of multiple layers, each containing multiple neurons responsible for processing and learning the data. This structure is also known as Multiple-Layer Perceptron (MLP). The output layer produces the result from the hidden layer. In this task, we use the 64 hidden layer neuron network to classify the activity recognition dataset that the *StandardScaler* function has preprocessed to ensure that the input features are standardised, which helps to improve the training efficiency and performance of the model.

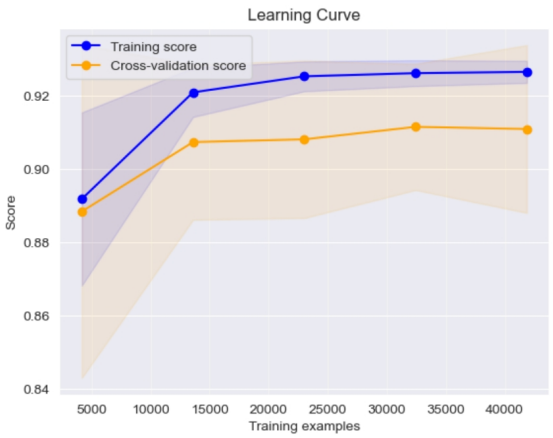



Figure 1: Learning Curve of MLP

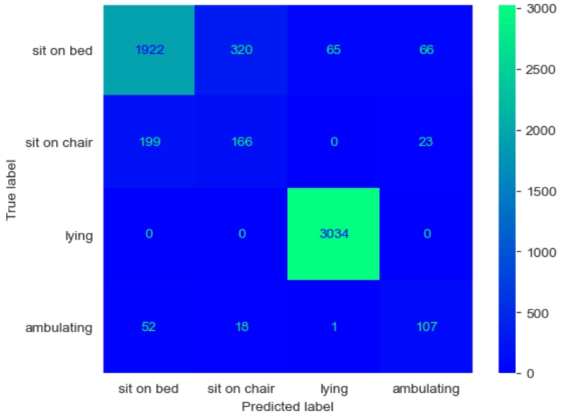


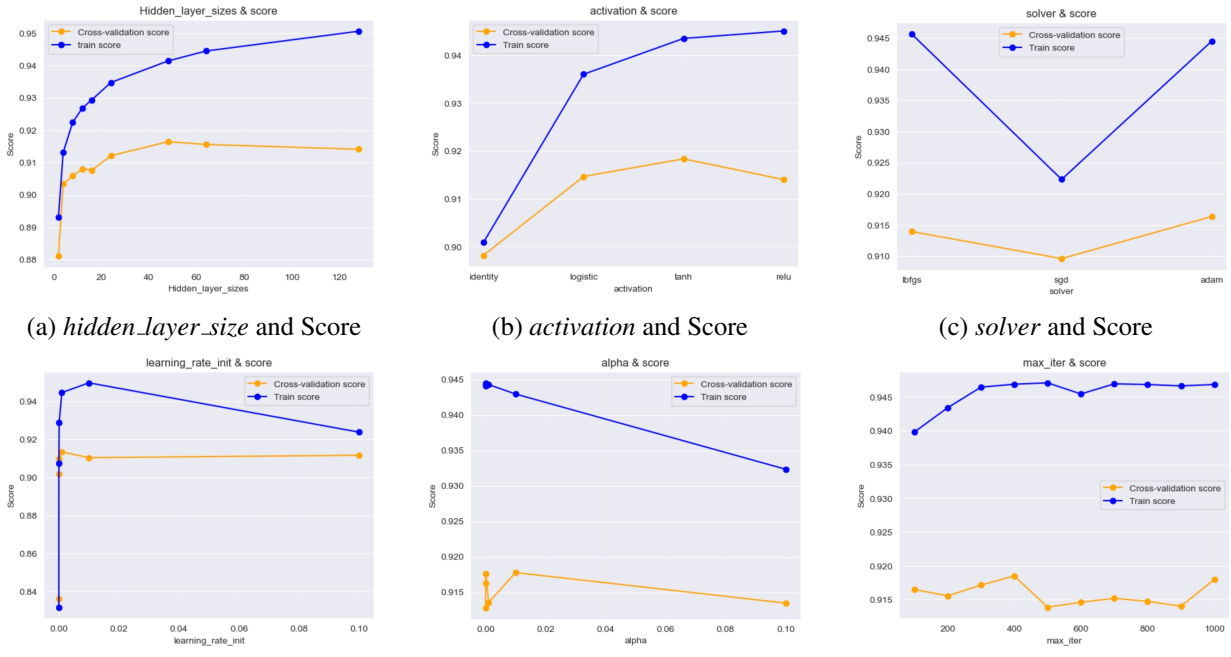

Figure 2: Confusion Matrix of MLP

**The learning curve depicted in Figure 1** demonstrates a clear trajectory of improvement in model performance with increasing data. The blue line represents the model's training accuracy, and the orange line indicates the performance on the cross-validation. Initially, the training and cross-validation scores are lower, but they ascend notably with more samples, especially between 5,000 to 15,000 training examples. Post this range, the score increment plateaus, indicating diminishing returns on learning from additional data. The gap between training and test scores remains narrow, even with more data, pointing to a **model that generalises well without significant overfitting**. This is further evidenced by the decline of the standard deviation, represented by the shaded areas. The convergence of the score lines, along with the reduction in the shaded areas for the training score, highlights the model's increasing stability by adding more training examples. It is notable, however, that the shaded area around the cross-validation score remains comparatively larger than that of the training score throughout the learning curve. This suggests that while the model's predictions are becoming more consistent, there is still a degree of uncertainty in how well the model generalises to unseen data. Despite this, the lack of an increase in the size of the orange-shaded area, along with the consistently narrow gap between the training and cross-validation scores beyond 15,000 samples, implies that the model maintains its generalisation performance without overfitting as more data is introduced.

**Figure 2 shows the confusion matrix** of the model to demonstrate the results of the neural network model. Overall, the accuracy value is 0.88. In this case, state *lying* is predicted very accurately with a 0.98 precision score, while *sitting on the bed* and *ambulating* state are also predicted relatively accurately. On the other hand, state *sitting on a chair* performs relatively poorly; many features were incorrectly identified as *sit on chair*. Probably because the data features were very similar, which caused the model to misjudge them; also, some were identified as *lying* and *ambulating*, caused by the model not fully learning the features. This may be the main reason why the training score is around 0.92.

**Impact of hyperparameters on the model** The Training scores and Cross-validation scores of the model with six different hyperparameters are shown in Figure 3. The hidden layer is where the data is processed; therefore, normally, the size of the hidden layer increases the more the model will learn from the training set. As demonstrated in Figure 2(a), the blue line representing the Train score has been growing, but the gap between it and the orange line representing the Cross-validation score is getting bigger, indicating a tendency for the model to be overfitting. The impact of different activation functions: *identity*, *logistic*, *tanh* and *relu* are shown in Subgraph (b). For the *identity* activation function that returns no operation and the *logistic* that returns a sigmoid function, the *tanh* and *relu* are more suitable for this training set. The *tanh* function, a scaled version of the logistic function, can deal with negative inputs more effectively. The *relu* function is less susceptible to the vanishing gradient problem, allowing the model to learn faster and perform better. The choice of solver, the optimisation algorithm for weight adjustment, also shows varied effects. The graphs suggest that the *lbfgs* performs consistently on training and cross-validation. However, both *sgd* and *adam* have better training performance than *lbfgs*, but not as well on cross-validation. This could mean they may be overfitting to the training data and not generalising as well as *lbfgs*. The *learning_rate_init* determines the step size of the model weight update. Subgraph(d) implies that an initial learning rate set too high may cause the model to fail to converge effectively. The *alpha* parameter controls the regularisation strength; it helps prevent overfitting by penalising larger weights. Subgraph(e) indicates that as *alpha* increases, the score slightly decreases on both data sets. This suggests that higher *alpha* may be causing the model may become too simple, losing the capacity to capture patterns in the data. The last graph reflects the relationship between the maximum number of iterations and model performance; the training score is relatively stable as the number of iterations increases, while the cross-validation score fluctuates. In general, more iterations mean that the model has more opportunities to learn the data, but if the number of iterations is too high, it may lead to overfitting.

**In summary**, *Activation* and *Alpha* have the greatest impact on the learning of the model. Too much tuning of most of the parameters leads to model overfitting. The choice of hyperparameters can significantly affect the model's ability to learn and generalise.



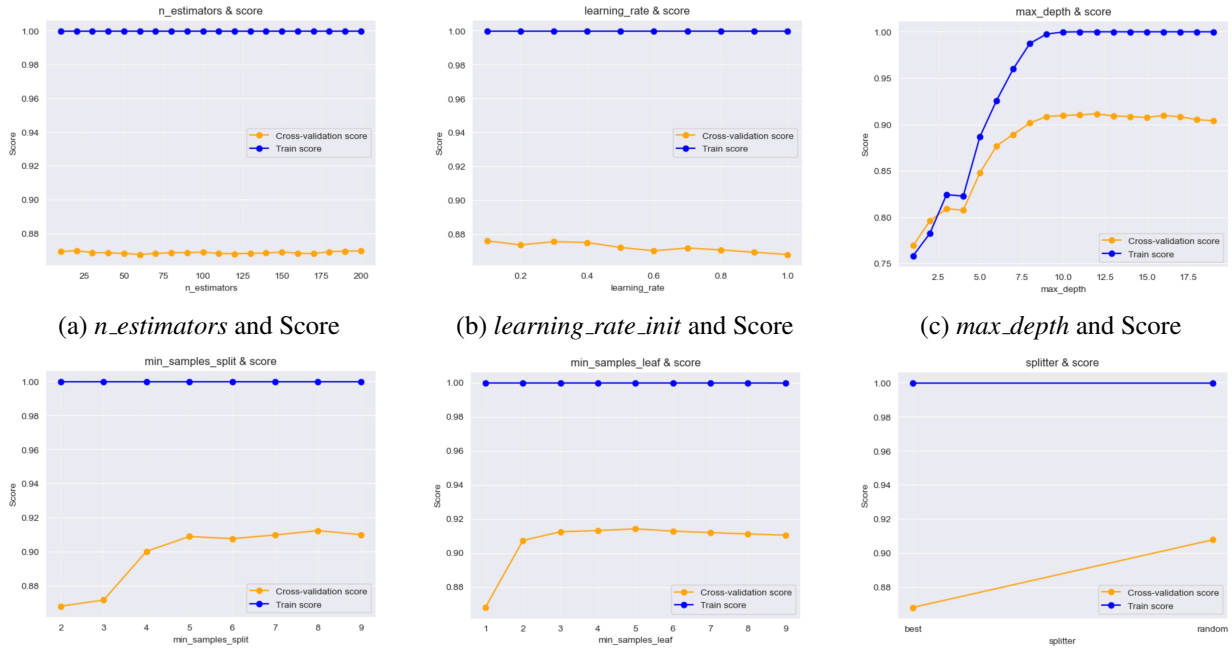

(a) *hidden_layer_size* and Score (b) *activation* and Score (c) *solver* and Score

(d) *learning_rate_init* and Score (e) *alpha* and Score (f) *max_iteration* and Score

Figure 3: Impact of hyperparameters on model scores

**Tree and ensemble tree** In this task, we explore the characteristics and performance of a Decision Tree and an Ensemble Tree, specifically employing the **AdaBoost** algorithm. A Decision Tree is a widely used classification method characterised by its tree-like structure where each internal node signifies a test on an attribute, each branch denotes the outcome of the test, and each leaf node corresponds to a class label. The training dataset was first applied to the *DicisionTreeClassifier* as a control group. The decision Tree achieved a training accuracy of 0.999, indicating potential overfitting given its closeness to 1. Its test accuracy of only 0.845 fur-

ther substantiated this suspicion, suggesting a diminished ability to generalise to new data. Subsequently, we applied the AdaBoost algorithm, an ensemble-based approach, to the same dataset. AdaBoost is a commonly used ensemble tree that focuses on misclassified datapoints by adding weights $w_n^{(1)} = \frac{1}{N}$ to each datapoints and updating the weights of the datapoints in each iteration. The error rate $\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)}[y_m(x_n) \neq y(x_n)]}{\sum_{n=1}^{N} w_n^{(m)}}$ keeping them constant or increasing them according to the correctness of the model predictions. After all the classifiers are trained, they are combined by weighting to form the final model. The weights of classifier m are determined by minimising the exponential loss function. The weighting formula is $\alpha_m = \ln\left(\frac{1-\epsilon_m}{\epsilon_m}\right)$; therefore, the lower error rate will be given a higher weight, which means that a well-performing classifier will have a greater impact on the final prediction. So, the AdaBoost algorithm has various advantages, including multiple iterations of datapoint weights to emphasise hard-to-classify datapoints and reduce the tendency to overfitting, resulting in higher applicability of the model. This is also verified through experiments. The *AdaBoostClassifier*, while achieving a training accuracy comparable to the Decision Tree (0.9998), demonstrated superior generalisation capabilities with a test accuracy of 0.9060. This marked improvement in test performance over the single Decision Tree underscores the AdaBoost model's enhanced proficiency in handling and adapting to unseen data. Additionally, compared to the decision tree, AdaBoost provides a more balanced F1 score across all categories, which may mean it is more balanced in dealing with all types of errors.




(a) *n_estimators* and Score (b) *learning_rate_init* and Score (c) *max_depth* and Score

(d) *min_samples_split* and Score (e) *min_samples_leaf* and Score (f) *splitter* and Score

Figure 4: Impact of hyperparameters on model scores

**Sensitivity of the model to different hyperparameters** The six subplots in Figure 4 collectively illustrate the impact of various hyperparameters on the model's training and cross-validation scores. For most hyperparameters, except for *max_depth*, the training accuracy remains consistently high, suggesting that the model is capable of fitting the training set well across a range of hyperparameter values. However, this does not necessarily indicate insensitivity, as it is crucial to consider the corresponding cross-validation scores for a complete picture of the model's generalisation performance. The cross-validation scores improve with the increase of certain hyperparameters, although this trend does not hold for the *learning_rate_init* and *splitter*. For these two parameters, the cross-validation scores do not show a similar improvement, implying that they do not contribute as effectively to the model's ability to generalise when varied within the tested range.

The model demonstrates significant sensitivity to the *max_depth* hyperparameter. As *max_depth* increases, we observe a sharp rise in the training and cross-validation scores, indicating that deeper trees capture more complex patterns. Nevertheless, there is an inflexion point beyond which the cross-validation score plateaus and even decreases, signaling the onset of overfitting. This trend emphasises the importance of selecting an optimal maximum depth that maximises cross-validation performance without incurring over-complexity costs.

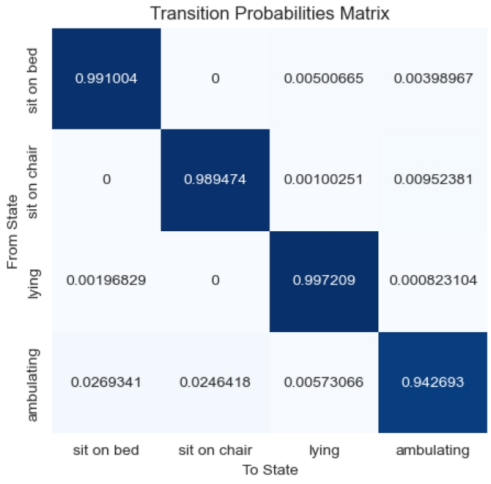
**Error rate of the ensemble and base model**
Figure 5 shows an orange line representing the SINGLE model error rate and a blue line representing the AdaBoost error rate. Most notably, Adaboost's error rate is lowest when the number of model bases is 2, which shows that AdaBoost effectively reduces the error rate even when there are only a few base models. As the number of models increases, their error rates fluctuate, but AdaBoost's is lower. For the same number of models, the error rate is lower than that of a single model by about 0.03. This indicates that the error rate of the integrated model is lower than that of the single model and remains relatively stable as the number of base models increases. This means the integrated model generalises better, reduces the risk of overfitting, and achieves more stable performance over multiple base models.

Figure 5: Error Rate in different number of base model

**Sequence labelling** In this task, a Gaussian Hidden Markov Model (HMM) was implemented. This model assumes that the observed data sequences are generated by a finite number of hidden states with Gaussian emissions.

**Training accuracy** The accuracy of the HMM on the test set was approximately 0.779, indicating a reasonable overall accuracy of the model. However, it shows significant variations in different classes; the model performs well on (0 and 2) but struggles on (1 and 3), indicating a class imbalance in the model.

**Transition matrix** With the transition matrix (Figure 6), the matrix is close to a value of 1 on the diagonal, which indicates that each state is highly likely to remain in the same state in the next step. The transition probabilities for states *sit on bed*, *sit on chair* and *lying* to remain in the same state are close to 1, but state *ambulating* is slightly less stable, with a 94.25% probability of remaining unchanged. This is because this state is also the main articulation of the other states; for example, it's not usually possible to switch directly from *sit on bed* to *sit on chair*, which also doesn't fit our common sense. So there is a need for an overstate, *ambulating*, so this state has horizontally and vertically values in its matrix position.




Accuracy: 0.7793403649757241

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.88 | 0.57 | 0.69 | 2373 |
| 1 | 0.19 | 0.38 | 0.25 | 388 |
| 2 | 0.98 | 1.00 | 0.99 | 3034 |
| 3 | 0.22 | 0.71 | 0.34 | 178 |
| accuracy | | | 0.78 | 5973 |
| macro avg | 0.57 | 0.67 | 0.57 | 5973 |
| weighted avg | 0.87 | 0.78 | 0.80 | 5973 |

Figure 6: Transition Matrix and report

**Features for predicting states** Figure 7 shows the average of different features on different states. The *frontal acc* has a significantly higher mean in state *lying* than in the other states, which suggests that it may be particularly informative in predicting state *lying*. The mean value of *vertical acc* in state *lying* is close to zero, which may indicate that it is useful in distinguishing state *lying* from the other states. *RSSI* in state *ambulating* has a significantly higher mean than the other states, which suggests that it can be a good way to distinguish state *ambulating*. *lateral acc*, *phase*, and *frequency* do not have much difference in their means across states, which may suggest that they are not particularly informative in distinguishing states.

| State | frontal acc | vertical acc | lateral acc | RSSI | phase | frequency |
|---|---|---|---|---|---|---|
| sit on bed | 0.357468 | 0.933800 | 0.057307 | -58.992142 | 3.157205 | 922.742689 |
| sit on chair | 0.551372 | 0.866495 | 0.066515 | -58.164914 | 3.488912 | 922.653706 |
| lying | 1.092375 | 0.000897 | -0.026741 | -58.423987 | 3.322364 | 922.799698 |
| ambulating | 0.223427 | 0.965195 | 0.022937 | -54.190101 | 3.148087 | 922.885264 |

Figure 7: Mean Feature Values by Activity States

**Comparison of three classifiers** To summarise, neural networks stand out for their high accuracy and reasonably fast training but are limited by their complexity and potential for overfitting. Decision trees offer a balance

of fast training and interpretability but may lack stability. HMMs are well-suited for sequence prediction tasks despite their lower accuracy. While more accurate than a single decision tree, the AdaBoost ensemble requires significantly more time to train.

The comparison of classifiers in Figure 8 shows that the neural network achieves the highest accuracy, indicating its effectiveness in capturing patterns within datasets. The decision tree and AdaBoost ensemble exhibit moderate accuracy, while the HMM has the lowest accuracy. Regarding fitting time, the decision tree is the fastest, closely followed by the HMM and neural network, with no significant time difference between them. The AdaBoost ensemble's fitting time is considerably longer, exceeding 20 times that of the other models. For Interpretability, neural networks are less transparent due to their intricate hidden layers, making them difficult to interpret. Decision trees offer high interpretability because their decision paths can be visualised, which is beneficial for transparency. HMMs offer moderate interpretability since their states and transitions can be inspected, although they may not be as intuitive as decision trees. The AdaBoost ensemble combines multiple weak learners and has lower interpretability than a single decision tree.


Figure 8: Accuracy and Fit time on three classifiers

## 2 Clustering and dimensionality reduction

In this section, Principal Component Analysis (PCA), Gaussian Mixture Models and Support Vector Machines will be implemented in order to classify Breast Cancer Wisconsin (Diagnostic) dataset.

**PCA** Principal Component Analysis is one of the most widely used data dimensionality reduction algorithms. The main idea of PCA is to map n-dimensional features onto k-dimensions. These k-dimensions are brand new orthogonal features, also known as principal components, which are k-dimensional features reconstructed based on the original n-dimensional features. On this dataset, we reduced the 30-dimensional data to 2 dimensions (Figure 9). Yellow scatter indicates that the tumour is malignant, and pink indicates that the tumour is benign. The first component explained variance is about 13.45, and the second component explained variance is about 5.65. This means the first principal component contains more information and contributes more to the data set's variability. The ratio of variance for the first principal component is about 0.4472, and the ratio of variance for the second principal component is about 0.1880, which means the first principal component explains about 44.72% of the variability in the data, while the second principal component explains about 18.80%. Together, these two principal components explain 63.52% of the total variance in the data.


Figure 9: The scatter plot for the dataset after PCA

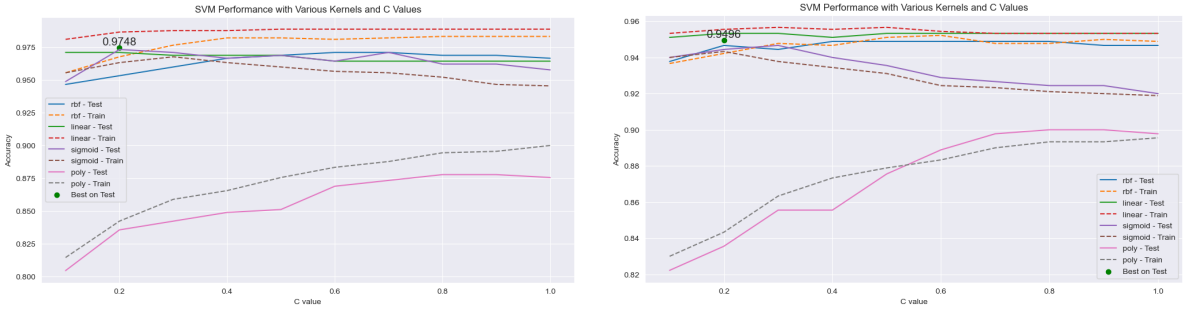
**Gaussian Mixture Model soft clustering** The Gaussian Mixture Model(GMM) is a weighted sum of several Gaussian distributions. Each of them is called a component with its own mean and covariance. Specific Gaussian mixture distributions can be obtained by maximum likelihood estimation. Soft clustering allows a data point to belong to multiple clusters with a certain probability. Figure 10 illustrates that there are only two data types in the dataset, but the graph shows more than two colours, with the two most coloured - yellow for malignant tumours and pink for


Figure 10: GMM soft clustering scatter plot

benign tumours. Those datapoints in between: the closer the colour is to yellow, the more likely it is to be malignant, and the closer it is to pink, the more likely it is to be benign.

**The difference between PCA and GMM** First, the most obvious difference between the two scatterplots is the colour coding, PCA uses discrete colours to signify the actual categories in the original dataset, whilst GMM uses colour gradients to denote the probability of belonging to different clusters. This is because the PCA and GMM are two distinct techniques, with the former being utilised for dimensionality reduction and the latter for clustering. PCA identifies the main directions of data by computing the covariance matrix $S = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T$ where $N$ is the number of datapoints, $X_n$ is each datapoint and $bar\mathbf{x}$ is the mean of the data. It linearly transforms the data to a new coordinate system to illustrate the structure and separation of the dataset. Conversely, GMM employs a probabilistic model $p(x) = \sum_{k=1}^{K}\pi_k\mathcal{N}(x|\mu_k, \Sigma_k)$ where$K$ is the number of Gaussian distributions and $pi_k$ represents the mixing coefficient with $sum_{k=1}^{K}\pi_k = 1$, and $mu_k$ and $Sigma_k)$ are the mean and covariance matrix of the $K$-th Gaussian component respectively. This estimates how datapoints are organised into a mixed Gaussian distribution.

**Support Vector Machine** SVM is a supervised learning algorithm mainly used for binary classification problems. Its core idea is to find an optimal hyperplane between datapoints to maximise the classification interval. Specifically, SVM uses a kernel function to map the original data into a higher dimensional space so that data that is originally linearly indivisible in a lower dimensional space becomes linearly divisible in a higher dimensional space. In this part, on the 30-dimensional original dataset and the dataset that has been reduced to 2-dimensions by PCA, attempts are made to adjust the kernel function and degree of regularisation to explore the maximisation of test set accuracy.




(a) SVM with various kernel functions and C-value combinations on the 30-D dataset

(b) SVM with various kernel functions and C-value combinations on the 2-D dataset

Figure 11: SVM performance with different kernel and C-value combinations on datasets of different dimensions

**SVM on different hyperparameters and dimensional dataset** Hyperparameter optimisation was conducted using *GridSearchCV*, exploring forty combinations across four kernel functions (*RBF*, *linear*, *sigmoid*, and *polynomial*) with ten different regularisation parameter *C*. The results are illustrated in Figure 11. For the original dataset, the *GridSearchCV* algorithm determined the optimal model utilised a *sigmoid* kernel with a *C* value of 0.2. This model achieved a cross-validation accuracy of 0.9733. When applied to the test set, comprising the last 118 datapoints not seen during the training phase, the model exhibited an accuracy of 0.9748, slightly outperforming other combinations. Notably, the *sigmoid* kernel, often less common in practice compared to *RBF* or linear kernels, provided superior results in this high-dimensional space, possibly indicating a non-linear but not overly complex decision boundary. Conversely, for the PCA-reduced dataset, the *linear* kernel with a *C* value of 0.2 was identified as the most effective, obtaining a cross-validation accuracy of 0.9533. Upon validation with the test set, this configuration yielded an accuracy of 0.9496. The preference for the *linear* kernel in the reduced-dimensional space suggests that PCA may have distilled the dataset's features into a form where the decision boundary is more linearly separable. **SVM on different dimensional dataset** Both SVM models were trained with their respective optimal hyperparameters and were subject to the same preprocessing steps, including feature scaling, to maintain consistency. The slightly superior accuracy on the 30-dimensional dataset indicates that while PCA is effective for dimensionality reduction, consideration must be given to the trade-off between simplicity and the potential loss of informative variance.

**Effect of PCA reduced dimensionality on SVM** As listed in the previous section, there are several important data: the variance of the data after PCA dimensionality reduction is only 63.52% of the original data; and the accuracy of the model on the test set has decreased from 0.9748 to 0.9496, which suggests that some information that is crucial for accurate classification may have been lost during the PCA process. The regularisation parameter C remains the same, but the kernel function changes from *sigmoid* to *linear*, suggesting that the decision boundary is also slightly different. This may account for the increase in the actual false positive rate in the confusion matrix, as shown in Figure 12. The prediction accuracy for class "M" was 0.82, with the model predicting six false positives but no false negatives. This indicates that while still valid, the classifier has decreased accuracy after dimensionality reduction. In conclusion, although the use of the 2D PCA approximation affected the accuracy of the test set to some extent, the model's accuracy remained relatively high.

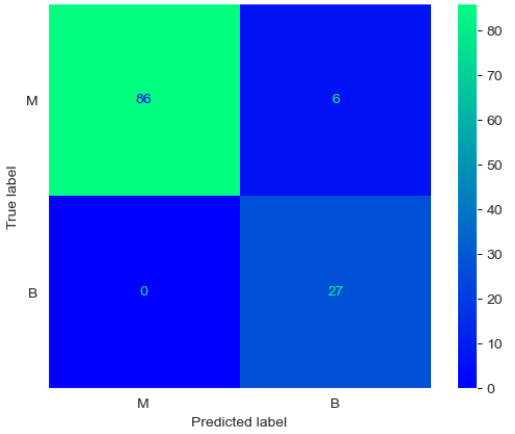


Figure 12: Confusion Matrix for SVM after PCA

## 3 Bayesian Linear Regression

In this section, Bayesian linear regression (BLR) is used to predict the number of bicycle rentals in Seoul. The initial step entails preprocessing the dataset to render it suitable for the BLR model. The raw dataset contains non-numeric attributes such as dates and categorical variables like seasons and holidays. Numerical mapping is applied to these features, mapping these categorical variables to numeric values, assigning binary digits to represent holidays (0 for 'No Holiday', 1 for 'Holiday') and integers 1 through 4 for seasons (1 for 'Spring', and so forth). There are also contain two zero-inflation feature: *Rainfall* and *Snowfall*. This binary representation can be more informative for the model, especially if the primary concern is whether it rained or snowed rather than the quantity of precipitation. Lastly, removed the *date* columns but kept the *IsWeekend* attribute of the dates. Then, the dataset was split into training and test sets, and the *StandardScaler* method was applied for standardisation, contributing to the stability and convergence of the model optimisation algorithms. Finally, after ensuring no data is lost, the data was converted to a uniform *np.float32* format for performance improvement and uniformity of data format. The processed data can then be applied to BLR.

**Prior distribution selection** Firstly, a linear model with the same number of $w$ as features was used to fit the dataset. Its posterior predictive plot is much like a Gaussian distribution, so next, the Gaussian distributed model is used to fit the dataset. The intercept $\alpha$ represents the expected value of bicycle rentals when all features are at their mean level. Due to standardisation, a normal prior distribution centred at 0, $N(0, 10^2)$, is chosen for $\alpha$, permitting the model to adjust the level of bicycle rentals around the mean with a reasonable degree of uncertainty. For the regression coefficient $beta$, which corresponds to the quantification of the effect of each feature on the target variable, the parameter selection is consistent with $\alpha\ N(0, 10^2)$, which assumes allows the data to inform their values. The standard deviation of the error term $\sigma$ captures the unexplained variability within the model. A half-normal distribution is applied with the scale parameter set to 10, reflecting an initial assumption of a relatively large noise level, which can be refined based on the observed data. The initial attempts to model the data did not yield satisfactory results, as evidenced by the divergence between the observed values

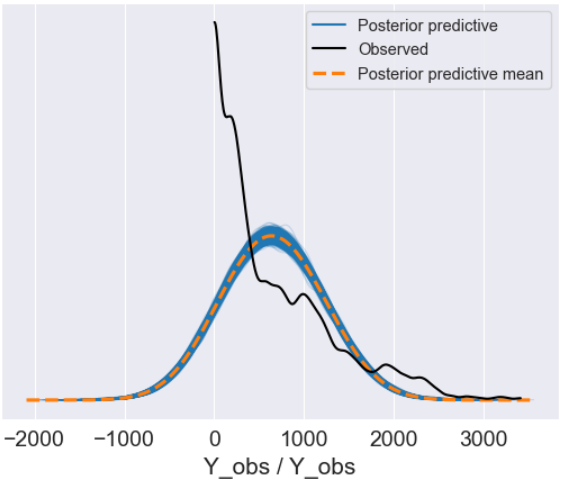


Figure 13: Posterior predictive for Gaussian model

and the posterior predictive distribution, depicted in Figure 13. The model's inability to capture the tails of the distribution (both the left and right) in the data suggested the need for an adjustment.

**Log transformation** After looking at the target values for the dataset, they are spread out, with extremely large scale. To address these issues, a log transformation was applied to the target variable,$y_{\text{train}}$, the transformation $y_{\text{train\_log}} = \log(y_{\text{train}} + 1)$ to stabilise the variance and make the distribution more symmetric. Furthermore, the log transformation provides a more symmetrical distribution, which is advantageous for the subsequent Bayesian analysis. Subsequently, the priors for the regression coefficients $\beta$ were adjusted, with a normal distribution centred at 0 and a smaller standard deviation, $N(0, 1^2)$. For the transformed intercept, $\alpha$, a normal distribution with a tighter standard deviation, $N(0, 0.5^2)$, was selected, reflecting the centred nature of the log-transformed $y_{\text{train\_log}}$. The standard deviation of the error term $\sigma$ was modelled using an exponential distribution, Exponential$(1)$, allowing the model to adjust its uncertainty about the data's variability on the log scale. After applying, as shown in Figure 15. The posterior predictive distribution now closely follows the observed data, with the posterior predictive mean tracing the central tendency of the distribution effectively. However, a noticeable spike at the left end corresponds to zero or near-zero counts in the original data. This spike indicates a significant number of days with very low bike rental counts. This could result from operational issues, weather conditions, or other factors leading to very few or no bike rentals.


Figure 14: Summary statistics

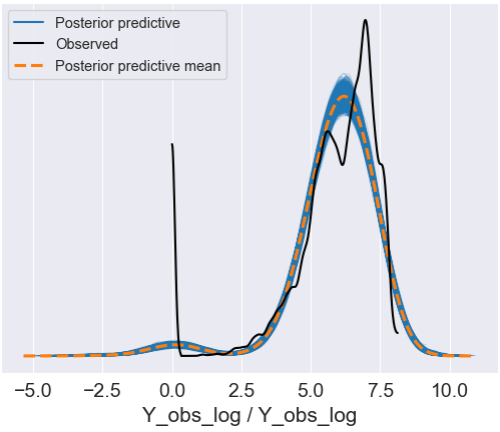



Figure 15: Posterior predictive log transformation

**MCMC Sampling Quality** The convergence of Markov Chain Monte Carlo (MCMC) sampling by looking at several statistics in the heatmap shown in Figure 14, The $r_hat$ values are important; value close to 1 indicate good mixing and coverage of the chains. The ess_bulk and ess_tail show the effective sample size for the bulk of the distribution and the tail, respectively. It suggests that the MCMC sampling has generated reasonable approximations to the posterior distributions.

**Interpretation of Posterior Mean Values** The implications of these features for leasing are largely consistent with common sense. From Figure 14, the posterior mean values suggest that most features have a negligible effect on bike rentals, including wind speed, seasons, etc. It indicates that they do not significantly influence the demand for bike rentals. As anticipated, negative correlations are observed for adverse weather conditions like Rain and Snow, reflecting expected decreases in rental frequency under these conditions. The biggest influencing factor is the functioning_day, which is as common sense as we live, it causes people to travel more, so the demand for bike rental increases.

**Suitability of Linear Regression** The PPC plot provides evidence of the model's suitability. If the predictive distribution captures the variability in the observed data well and the mean prediction follows the observed data closely, it suggests that linear regression could be a reasonable model for this data. This can also be assessed by examining the data relationship. An alternative method, such as transformations or non-linear models, might be more appropriate if the data is non-linear.