COMS30077: Machine learning coursework 2023

Ram Larg (2120398)

1 Classification I

Task 1(a) For this task, we use a Multi-layer Perceptron classifier, a neural network model that trains using backpropagation. MLP trains iteratively; at each time step, the partial derivatives of the loss function are computed to update the parameters. We can use a trained neural network model to predict labels for unseen data (our test set) and score them using actual test labels.

The validation score for our classifier improves over time, shown in Figure 1, indicating the model is learning from the training data. The graph shows the low validation and high training score initially, suggesting the model is overfitting but gradually converges to an improved score. The model does not seem to underfit significantly, as both the training and validation scores converge to a relatively high value. The error also steadily decreases for our validation, reaching a low value upon convergence.

Task 1(b) On the test set, the mean validation score is 0.82. We calculate the score by predicting the mean accuracy of each label set against each test sample. The train and test accuracy are both high, indicating the model generalises well.

Task 1(c) From plotting validation curves for several hyperparameters, we can tell whether the MLP-Classifier estimator is overfitting or underfitting to some hyperparameter value. Thus, we can determine which hyperparameters might strongly affect the model's performance and whether it would reduce or improve our score. We keep the parameters defined in Figure 1 constant for the following models.

From Figure 2, we see if the alpha parameter is too low, the model will overfit, so the training score is high, but the validation score is low. As alpha is increased and allows for a stronger L2 regularization, our model generalises better, as seen by the convergence of train and validation lines. However, as alpha is further increased, the score is also reduced, suggesting the model may start underfitting. Alpha can be said to have a strong effect on model performance as lower values can cause the model to overfit, but larger values allow the classifier to perform well at the cost of the model score.

The validation curve for Figure 3 shows overfitting for neurons 32 through 128. The train score seems to improve with the number of neurons in the hidden layer despite this, suggesting a higher number of neurons allows the model to fit the complexity of the dataset better. The validation score decreases minimally with the increase in the number of neurons, implying the model may be beginning to overfit to the increasingly complex network. We can conclude changing the number of neurons in the network for one layer might not have the strongest effect on model performance.

Figure 4 shows us the score for our classifier with learning_rate_init being changed. It is clear from the graph that our model is overfitting until approximately a value of 10^{-1}, from where it begins to converge. The plot suggests the model learns more accurately in incremental steps at lower rates, as higher score values show. Higher learning rates may cause the model to overshoot the minimum, as the lower score shows. Divergence is also indicated by the wider error bars as the step size is increased. The learning rate is similar to alpha as it strongly affects model performance. Lower values can cause the model to overfit, but larger values allow the classifier to perform well at the cost of the model score.

We can tune hyperparameters by randomly selecting a combination of parameters to train and evaluate model performance. We then use cross-validation where train data is split into k folds. The model is trained on $k-1$ folds and evaluated on the last fold. We repeat this k times, so each fold is used for validation. RandomizedSearchCV gives us the following best parameters with a test score of 0.91:

solver: sgd, n_iter_no_change: 20, learning_rate_init: 1, learning_rate: adaptive, hidden_layer_sizes: 128, alpha: 0.01, activation: logistic

From CV, an initial learning rate of 1 was chosen, following from Figure 4. Alpha is also chosen at 0.01, providing a high score, but is overfit on the validation graph. This could be because the alpha parameter converged at a higher score than the one shown in Figure 2 when using cross-validation.

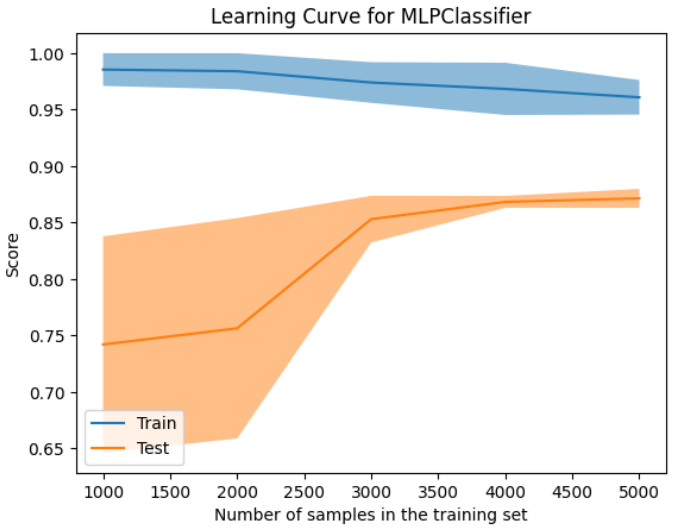

Figure 1: MLP classifier with default: parameters `max_iter=1000, random_state=42`

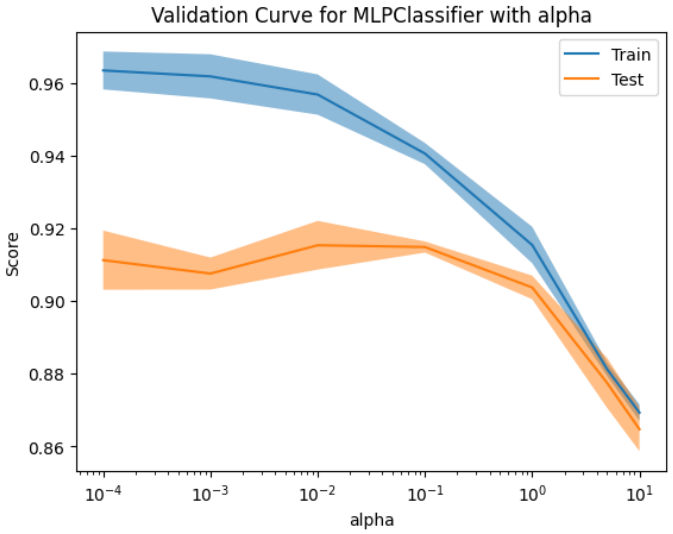

Figure 2: MLP classifier with an `alpha`

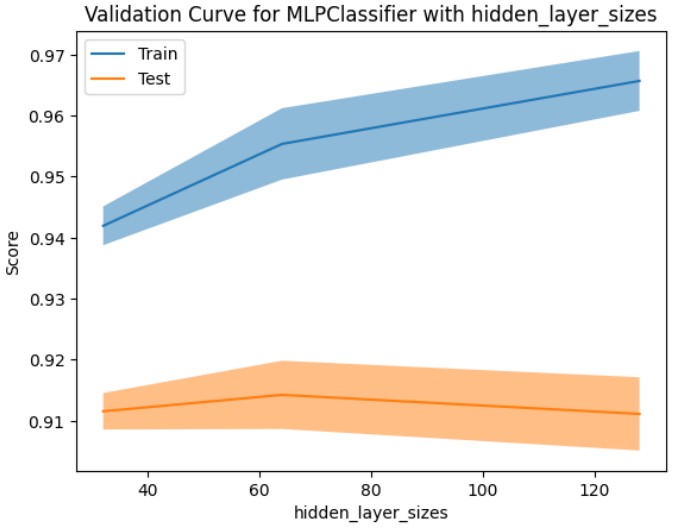

Figure 3: MLP classifier with `hidden_layer_size`

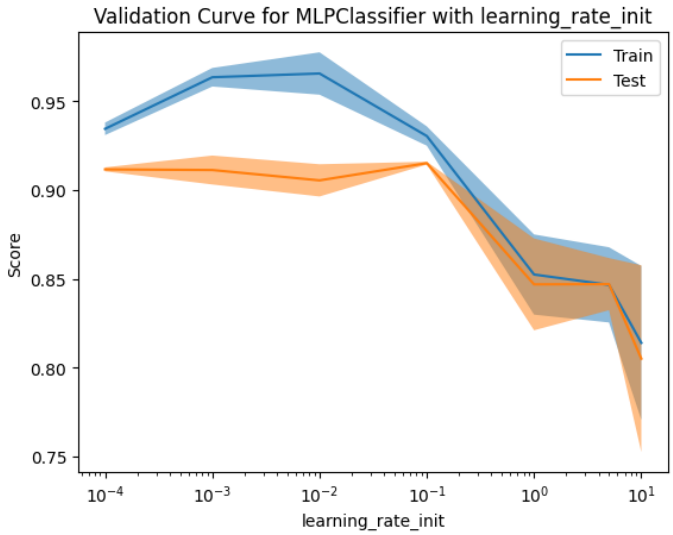

Figure 4: MLP classifier: `learning_rate_init`

2 Classification II

Task 2(a) & (b) A decision tree classifier 9 creates a model that can predict target variables by learning decision rules deduced from features in the data. Gini impurity can be used as a criterion to split nodes, where we choose the best split by maximising the Gini gain by subtracting the weighted impurities of the branches from the original impurity. Gini Impurity is calculated by:

$$G = \sum_{i=1}^{C} p(i) * (1 - p(i))$$

With C being the number of classes and $p(i)$ being the probability of randomly selecting an element of class i [5].

After creating a single DecisionTreeClassifier and fitting that classifier to the X training and y training data, we achieved an average score of 0.84 on the test set. After training an ensemble of decision trees, we achieved a higher average accuracy on the test set at 0.87. Decision trees allow us to mitigate slight variations in the data which might have been generated from a different decision tree. Each decision tree in the ensemble is trained by fitting base classifiers on random subsets of the original dataset. This can reduce overfitting by aggregating their individual predictions, allowing the classifier to generalise better. Sampling is also drawn with replacement.

Task 2(c) Figure 5 shows us the error rate of the bagging ensemble, the error rate of an individual base model, and the number of estimators or base models in the classifier. We can see from this graph that the error rate reduces if we increase the number of estimators; therefore, ensembling does, in fact, improve

performance. The performance for a single individual base model stays at a higher constant error rate.

Task 2(d) Figure 6 shows us the accuracy of the ensemble as we increase the maximum depth of the tree. We see the graph varies quite significantly as we vary max tree depth, suggesting the model is very sensitive to this hyperparameter. Our test set accuracy decreases as the max depth increases, suggesting the model begins to overfit and does not generalise well. The accuracy increase in the training set also backs this up. Figure 7 shows accuracy varying quite significantly as we change the minimum impurity value. This value controls if a node will be split if the split reduces the Gini Impurity by greater than or equal to this value. As we can see, the accuracy is nearly halved on the test set when this value is increased from 0.0 to 0.5. This value can modified to decrease overfitting. Figure 8 shows a similar result to that of 6, where peak accuracy for the test set is found towards the start and then steadily decreases. This parameter determines the maximum number of leaf nodes within a tree. Similarly to 6, the model may start overfitting as the max-leaf nodes increase, thus decreasing accuracy. The training accuracy also increases with the leaf nodes, suggesting overfitting the to the train set. Thus, the model is likely very sensitive to this parameter.

RandomizedSearchCV gives us the following best parameters with a test score of 0.90:

'estimator__splitter': 'best',
'estimator__min_samples_split': 8,
'estimator__min_samples_leaf': 5,
'estimator__min_impurity_decrease': 0.0,
'estimator__max_features': 'sqrt',
'estimator__max_leaf_nodes': 11,
'estimator__max_depth': 11,
'estimator__criterion': 'log_loss'

From CV, a max tree depth of 11 was chosen, which does not correspond to Figure 6. According to the plot, we would expect the maximum depth to be around 5. However, with the addition of other changing hyperparameters in CV search, there exist combinations which might allow us to choose a lower depth without overfitting. This is similar to max_leaf_nodes, which obtained a value of 11 from CV, compared to the plot in Figure 8 where we would expect the max-leaf nodes to be around 7. The reason for this is likely similar to max depth, where CV allows us to explore a set of hyperparameters specialised to get the highest score.

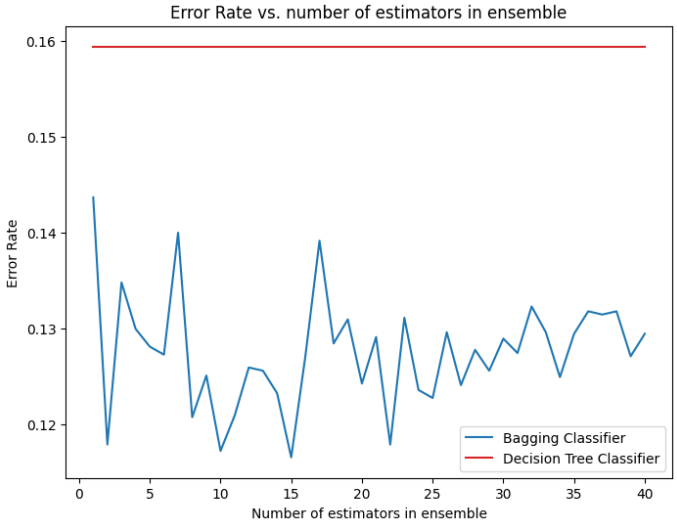

Figure 5: BaggingClassifier v. single decision tree

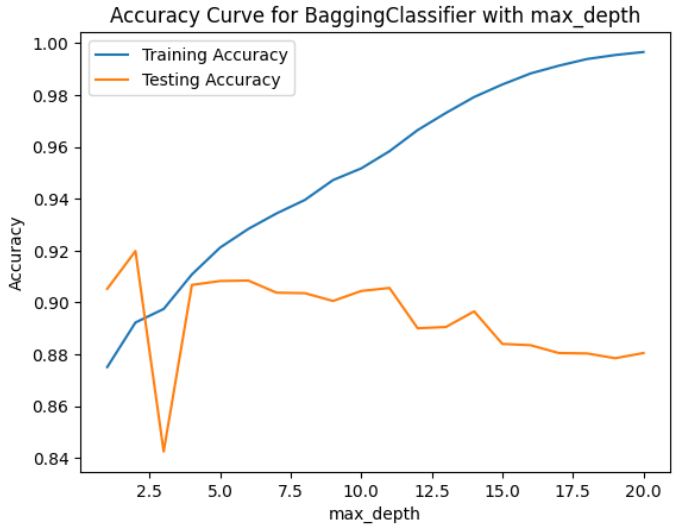

Figure 6: BaggingClassifier with max tree depth

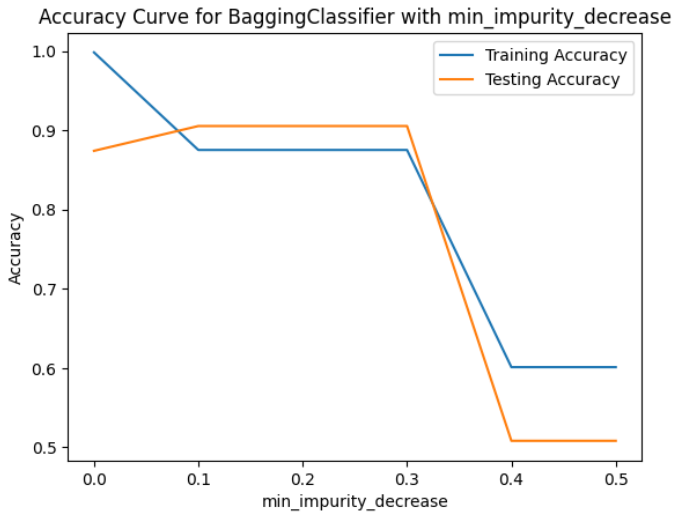

Figure 7: BaggingClassifier with min impurity

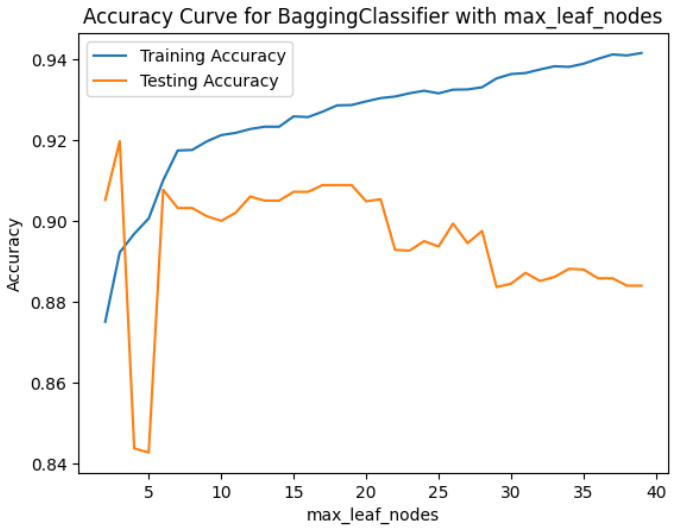

Figure 8: BaggingClassifier with max_leaf_nodes

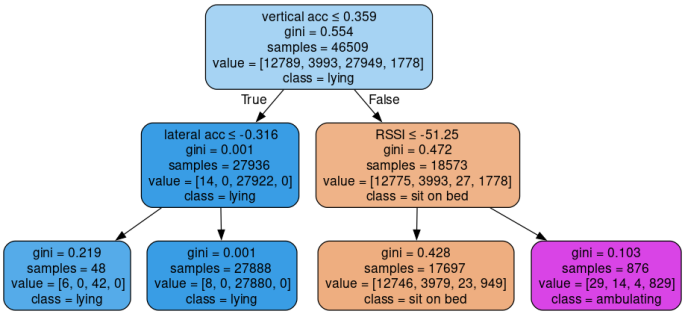

Figure 9: Decision tree of max_depth=2

3 Sequence labelling

Task 3(a) We can use Hidden Markov Models (HMMs) to create a sequence labeller. The HMM can be used to assign a label to each member in an observed sequence. The sequence of internal hidden states

The accuracy score achieved by the hidden test states on true test labels is 0.81. The X_train sequences are first flattened before being used to estimate the model parameters for the Gaussian HMM. We then use this model to make predictions on the X_test sequence, which we flatten to obtain the test hidden states. We then obtain the true test labels by flattening the true test labels sequence. From this, we can calculate an accuracy score between the hidden test states and the true test labels.

Task 3(b) This transition matrix shows the probability of transitioning from a state along the y-axis to a state along the x-axis. The transition matrix in Figure 10 suggests that if a person is doing one of the activities, 1: sit on bed, 2: sit on chair, 3: lying, or 4: ambulating, they are very likely to continue doing that activity.

Task 3(c) Assuming the shown means in Figure 13 of Gaussian emission distributions are the means for the activities, the columns would represent the features. We can then use the diagram from Figure 11 to understand the sensor data. From this, we can tell a person is likely lying on a bed if they have low acceleration in the vertical axis, suggesting they are not moving up or around in the bed. A high acceleration in the frontal axis may also suggest they are lying down. Low acceleration in the frontal axis may suggest ambulating (standing or walking). A relatively high lateral acceleration may suggest sitting on a chair. Comparatively, A low positive lateral acceleration may indicate sitting on a bed.

Task 3(d) The strength of the sequence labeller over the neural network is that we can predict the most likely sequence of hidden states (activity labels) that might follow given a sequence of observations. In addition, the sequence labeller allows us to interpret the emission and transition probabilities. These help us understand how likely the model is to move from one state to another and the probability of the possible observations. This type of interpretation is more difficult for the neural network as the weights and biases 12 alone do not explain why the model might be making certain predictions.

The advantage the neural network might have over an HMM is the capability to learn non-linear models. The neural network can achieve better accuracy than the HMM, with the HMM only achieving a score of 0.81 on the test set, in comparison to the neural network's highest of 0.91. However, our sequence labeller, on average, trained much faster than the neural network at (0.4 seconds compared to 10.0 seconds), allowing analysis to be performed much faster.

The decision tree ensemble performs better than the HMM model, with a score of 0.90. Also, the decision tree is arguably more interpretable than the HMM, as we can visualise the tree like in Figure 9 to understand why it makes certain predictions. Comparatively, the hidden states in an HMM are not observed directly and are less interpretable in this aspect. Our ensemble training time was, on average, shorter than the neural network, taking around 3.7 seconds for 10 estimators. However, this is still longer than the sequence labeller.

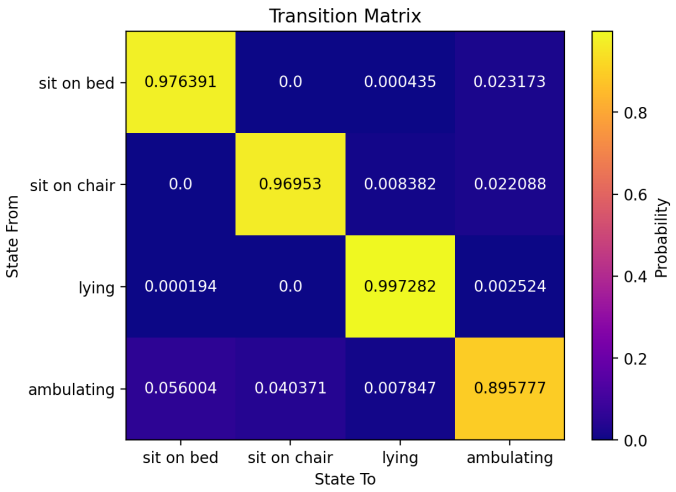

Figure 10: Transition matrix for different activities

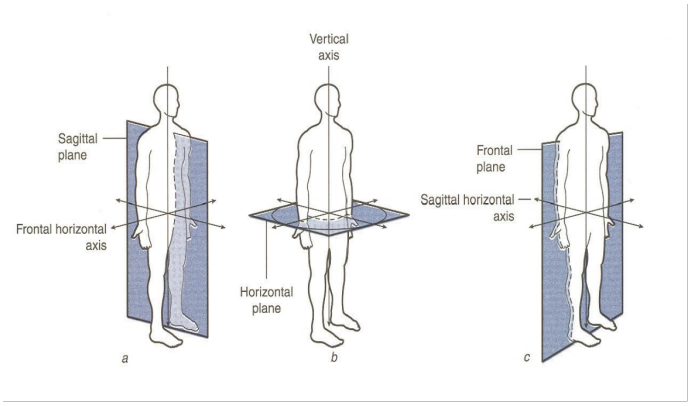

Figure 11: Axes of movement: Behnke 2000 [1]

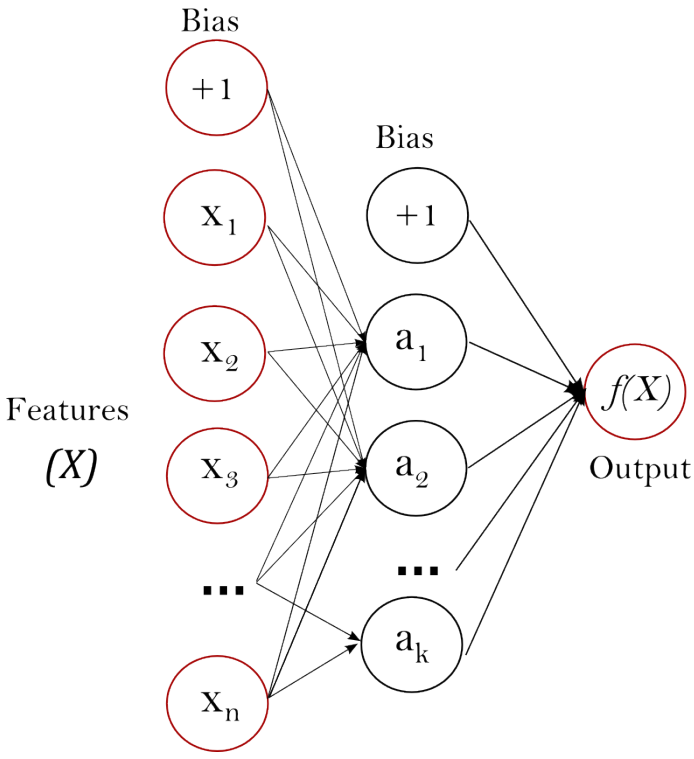

Figure 12: MLP with one hidden layer: scikit-learn [3]

	frontal acc	vertical acc	lateral acc	RSSI	phase	frequency
sit on bed	0.315452	0.968343	0.039093	-58.413312	3.238497	922.749516
sit on chair	0.659959	0.828318	0.082466	-59.173712	3.349652	922.693184
lying	1.091676	0.003837	-0.025791	-58.431152	3.320219	922.798317
ambulating	0.156979	0.963522	0.059106	-57.035387	3.045852	922.773597

Figure 13: Emission means: Features to activity label

4 Clustering and dimensionality reduction

Task 4 Principal components analysis (PCA) is a method to mitigate the curse of dimensionality. PCA projects data to a lower dimensional space while trying to preserve variance. If one feature varies more than the others, PCA would determine that feature dominates the direction of PCA components. Thus, we first standardise the data using StandardScaler. We can then plot a PCA-reduced dataset in a scatter plot as shown in Figure 14. The variance explained by the first principal component is 13.3, and the second is 5.7. This is equivalent to the pc1 accounting for 0.44% of the variance around the principal components and pc2 accounting for 0.19% of the variance.

Task 5 We can use the Gaussian mixture model (GMM) to estimate the parameters of a Gaussian Mixture distribution. The GMM is a probabilistic model that assumes data points are generated from a mixture of several Gaussian distributions. We can display the soft clusters of the PCA-reduced dataset by plotting the responsibilities for each point. This is done by defining the responsibility component k has for explaining an observation $\mathbf{x}$. The colours represent the responsibilities $\gamma(z_{nk})$ associated with datapoint $\mathbf{x}_n$. The data points with a greyish colour in Figure 15 could be in either cluster for $k = 1$ or $k = 2$.

Task 6 The main difference between the first and second scatter plots is the first represents the true classes 14, whereas the second scatter 15 shows the responsibilities (component density for each sample). These refer to the probabilities that a given data point belongs to either the malignant 'M' or benign 'B' component of the Gaussian Mixture. The responsibilities scatter plot requires a trained GMM model, whereas the true class labels plot does not. The clusters in the scatter plot for true class labels show the benign points grouping towards the left side of the plot. Although there are some outlier benign points that the GMM does not capture in the upper half of the graph, it captures the area where the benign classifications are most dense. This suggests the GMM captures the cluster in the data but not perfectly. It may misclassify some benign points as malignant.

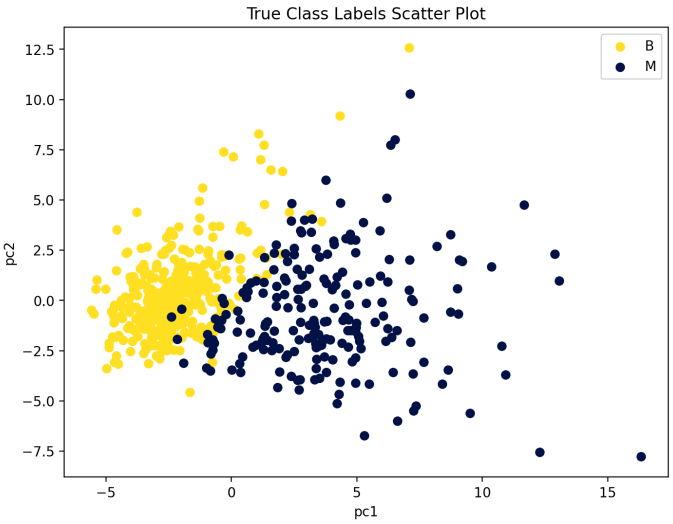

Figure 14: True class labels for first and second principal components

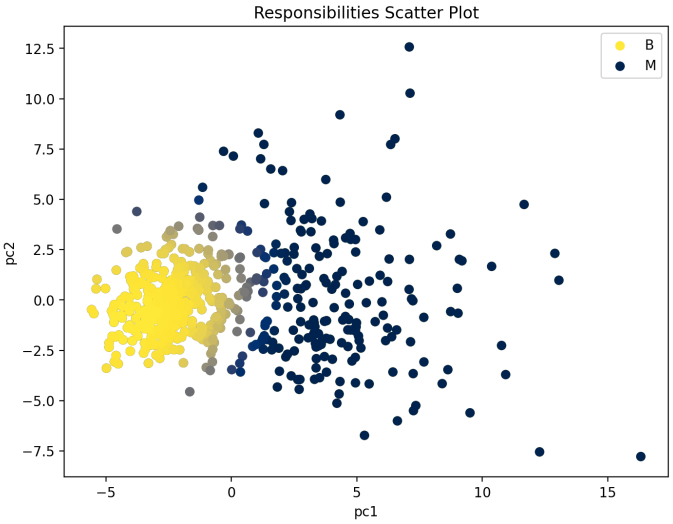

Figure 15: Responsibilities for first and second principal components

5 Classification with SVMs

Task 7 & 8 The SVM algorithm creates a set of hyperplanes in a high-dimensional space that can be used for classification. The best hyperplane is the one that creates the largest separation (margin) between the two classes. The support vectors are the ones which lie closest to the decision boundary.

Test accuracies for SVM

SVC accuracy for 'linear' kernel on test set: 0.97
SVC accuracy for 'poly' kernel on test set: 0.98
SVC accuracy for 'rbf' kernel on test set: 0.98
SVC accuracy for 'sigmoid' kernel on test set: 0.91

Test accuracies for SVM with 2D dataset

SVC accuracy for 'linear' kernel on test set: 0.95
SVC accuracy for 'poly' kernel on test set: 0.97
SVC accuracy for 'rbf' kernel on test set: 0.96
SVC accuracy for 'sigmoid' kernel on test set: 0.90

Task 9 Using the 2D dataset does not seem to affect test set accuracy significantly; the original data only has slightly higher accuracies. We could set a higher number of principal components to capture a higher variance and obtain a higher accuracy.

6 Bayesian Linear regression

Task 10 With Bayesian linear regression, we try to estimate distributions over the parameters and predictions. From this, we can model uncertainty in the predictions.

Using the Bike Sharing data given, we remove the first row so the variables in the second row become the columns. We drop all days where the 'Functioning Day' of the bike sharing system is 'No'. There are no bikes rented on these days. We can then drop the column as it becomes redundant. We also replace the seasons with an integer, choosing 'Winter' as 0, 'Autumn' as 1, 'Spring' as 2, and 'Summer ' as 3. We also change holidays to '0' if the value is 'No Holiday' otherwise, '1' for 'Holiday'. In addition, we can also use pd.to_datetime to convert the 'Date' column to a suitable format. Then, using pandas series.dt.date, we add 3 columns to our dataframe, which represent the day, month and year values. This allows us to see if we can use the date to help predict the number of bikes rented. We also standardise our values by subtracting the mean from the data points and dividing by the standard deviation.

We also want to ensure our independent variables are all linearly independent (no multicollinearity) [4]. We can do this by measuring the variance inflation action factor (VIF). We find that temperature and dew point temperature both have large VIF values at 89.9 and 116.7, respectively. So we can also drop this column.

Task 11 With Bayesian linear regression, we try to obtain a posterior distribution over the model parameters from our data. We do this for our predicted outcomes Y with standard deviation σ^2 and expected value μ that is a linear function of each of our predictors X_i.

$$Y \sim \mathcal{N}(\mu, \sigma^2)$$

$$\mu = \beta_i X_i + \alpha$$

Where σ is our observation error, β_i is the coefficient for the predictor X_i, and α is the intercept.

Our priors are:

$$\alpha \sim \mathcal{N}(0, 20)$$

$$\beta_i \sim \mathcal{N}(0,\, 20)$$
$$\sigma \sim |\mathcal{N}(0,\, 20)|$$

We have chosen priors which are uninformative about the data so we do not bias the data in a certain direction. The half-normal distribution is chosen for our observation error as σ should not be negative. The priors are considered relatively flat with a standard deviation of 20 centred around 0.

Task 12 Figure 17 and 18 show the summary statistics for our BLR model. In addition, we can look at our model structure in Figure 16 to show the relationships between our priors.

Task 13 We can see whether the MCMC sampling has generated reasonable approximations by looking at our 'r_hat' in 18. This is the Gelman-Rubin Statistic, which is an approach to monitoring the convergence of our MCMC sampler. The multiple chains, which start with different values, should converge onto a single distribution. By comparing the variation between chains, we can tell if the chains have converged if the variation is close to 0. As chain variation tends to 0, r_hat tends to value 1. As our r_hat values in 18 are all 1.0, we have evidence that our approximations are reasonable [2].

Task 14 Figure 18 and 17 gives the posterior distribution of coefficients, which can tell us about what variables might affect the number of bikes hired. The temperature and dew point temperature are high and positive, suggesting warmer temperature conditions increase the number of bike hires. The hours are also high, suggesting that people are more likely to rent bikes later in the day (likely past 12). Humidity is large and negative, meaning more humid (%) conditions result in fewer people renting bikes. Solar radiation is negative, suggesting that people may opt out of riding bikes in clearer, sunny conditions to avoid getting sunburnt. This is similar to rainfall, where higher rainfall results in fewer people renting bikes. One surprising value is snowfall, where a snowfall suggests a slight increase in bike rentals. As previously mentioned, we set no holidays to 0 and holidays to 1. We can see holiday days are negative, suggesting people get more bicycles on the lower end of this value when it is a non-holiday. Our seasons are given values of 0 through 3 for winter, autumn, spring, and summer. As the season mean is high, this suggests a tendency towards higher bike rentals in non-winter months as it is warmer outside. It is difficult to interpret the date other than that bikes may be more likely to get rented in the early days of the month and the latter months of the year.

Task 15 Linear regression becomes a usable model for this data if we transform the categorical data into numerical data. We changed the values associated with 'Date', 'Seasons', and 'Holiday Day' to numbers to be able to use the BLR model. For linear regression, we also assume each observation is independent of others. However, this was not the case as the Dew point temperature was highly correlated with normal Temperature, so it had to be dropped. Sigma is also large, implying the errors have a high standard deviation. Our model may be too simple, and instead, a polynomial regression model may be more suited for the data.

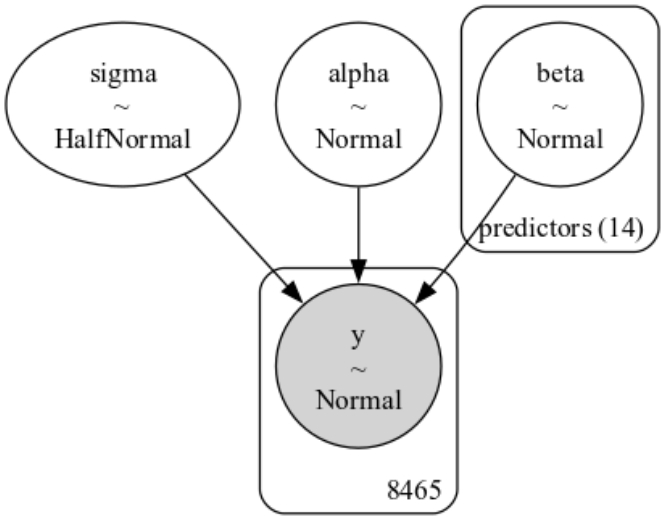

Figure 16: BLR Model structure graph

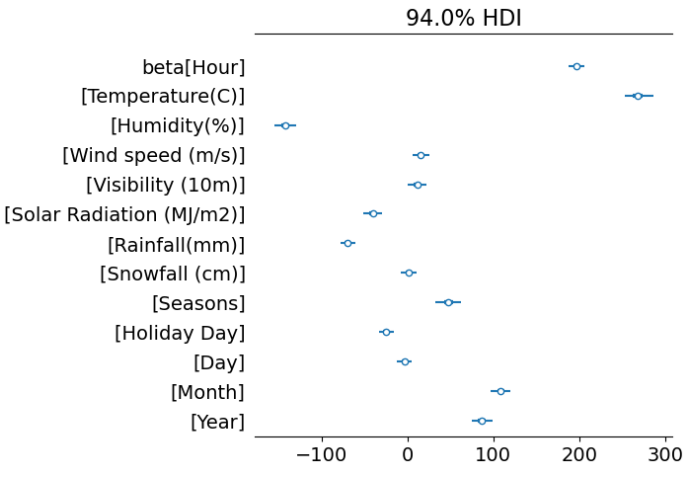

Figure 17: Forest plot statistics for each parameter

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
alpha	690.95	4.61	682.69	699.94	0.06	0.04	6457.29	3162.44	1.0
beta[Hour]	196.86	4.89	187.64	205.87	0.07	0.05	5266.18	3521.77	1.0
beta[Temperature(C)]	268.59	8.72	253.30	286.34	0.17	0.12	2703.27	2837.93	1.0
beta[Humidity(%)]	-142.30	6.91	-155.49	-129.71	0.11	0.08	3853.26	3588.66	1.0
beta[Wind speed (m/s)]	15.36	5.25	5.75	25.29	0.07	0.05	6049.52	3561.35	1.0
beta[Visibility (10m)]	11.27	5.69	0.34	21.82	0.08	0.06	5005.27	3437.89	1.0
beta[Solar Radiation (MJ/m2)]	-40.39	5.93	-51.04	-29.13	0.10	0.07	3783.27	3251.02	1.0
beta[Rainfall(mm)]	-70.03	4.79	-78.49	-60.67	0.06	0.04	6774.88	3137.63	1.0
beta[Snowfall (cm)]	1.36	4.94	-7.67	10.88	0.06	0.07	6504.69	3357.17	1.0
beta[Seasons]	47.67	7.86	32.95	62.05	0.14	0.10	3379.23	3023.06	1.0
beta[Holiday Day]	-24.86	4.71	-33.55	-15.79	0.06	0.04	6770.97	3051.02	1.0
beta[Day]	-3.39	4.72	-12.28	5.33	0.06	0.06	7413.48	3314.59	1.0
beta[Month]	108.71	6.13	96.90	119.67	0.10	0.07	3557.57	3299.67	1.0
beta[Year]	86.44	6.25	75.02	98.63	0.09	0.07	4448.71	3298.90	1.0
sigma	432.02	3.08	426.18	437.71	0.04	0.03	6744.98	3350.65	1.0

Figure 18: Summary statistics for approximate posterior distributions for each parameter in BLR model

References

[1] Behnke. *PhysicalSolutions - Understanding Planes and Axes of Movement*. https://www.physical-solutions.co.uk/wp-content/uploads/2015/05/Understanding-Planes-and-Axes-of-Movement.pdf. (Accessed on 12/04/2023). 2000.

[2] PyMC. *7. Model checking and diagnostics — PyMC 2.3.6 documentation*. (Accessed on 12/07/2023). URL: https://pymcmc.readthedocs.io/en/latest/modelchecking.html%5C#convergence-diagnostics.

[3] scikit-learn. *1.17. Neural network models (supervised) — scikit-learn 1.3.2 documentation*. https://scikit-learn.org/stable/modules/neural_networks_supervised.html. (Accessed on 12/07/2023).

[4] Skipper Seabold and Josef Perktold. "statsmodels: Econometric and statistical modeling with python". In: *9th Python in Science Conference*. 2010.

[5] Victor Zhou. *A Simple Explanation of Gini Impurity - victorzhou.com*. (Accessed on 12/07/2023). URL: https://victorzhou.com/blog/gini-impurity/.