Machine Learning Report

2.1.1

To begin we will use the principal component analysis (PCA) tool from the Sklearn library to perform dimensionality reduction for our dataset. PCA is a dimensionality reduction tool that picks the highest eigenvalue of the eigenvector as the first principal component and picks the highest eigenvalue of the eigenvector that is orthogonal to the first principal component and so on. For the proprecessing part, I standardize my dataset as it will affect my PCA result. This is because features with high scale might dominate other features. After transforming the fashion MNIST dataset containing 784 features to only 2 using PCA, I look at the explained variance ratio of my principal components as it is useful because it tells us how much information is retained in our new features. The variance ratio for the two principal components are 22.08% and 14.41%. This shows that we lost much information regarding the dataset when transforming it from 784 features to only 2. This is the tradeoff of using PCA, as we might lose information of our dataset however make it easier to train our model and visualize our dataset.

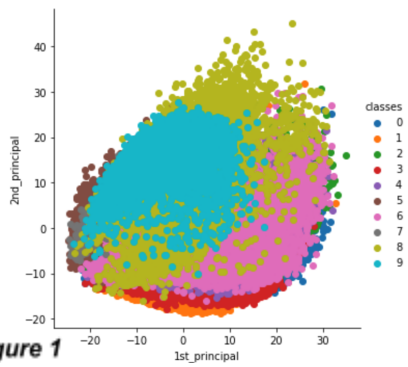


Figure 1

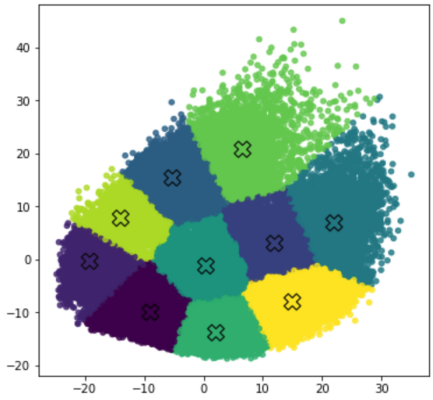
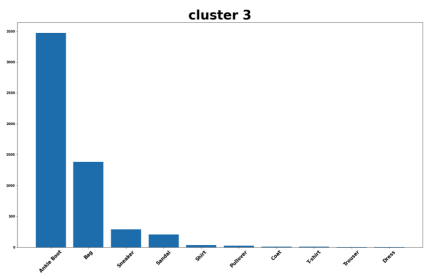


Figure 2

I conduct an exploratory data analysis on the fashion MNIST dataset as it will allow us to draw some patterns and make assumptions for K-means clustering. This method is important because we can ensure the validity of the result generated by our Machine Learning model. The correct labels are {0:T-shirt, 1:Trouser, 2:Pullover, 3:Dress, 4:Coat, 5:Sandal, 6:Shirt, 7:Sneaker, 8:Bag, 9:Ankle Boots}. Overall, the plot shown in figure 1 suffers visual issues. This is because we have too much overlapping data in the plot because PCA removes the distinct feature of our dataset. However, we are still able to draw some insight from this plot. If we look at class 9, we can see that it mainly appears in the region around (-20,10) on the 1st_principal axis and (0, 25) on the 2nd_principal axis. Besides that, we also observed that class 6 and 8 are in the middle region of the plot. However, both these classes suffer from high variance as the data points are spread around in a large region which might not cluster well when performing K-Means. I also noticed that class 1 and 3 are mainly in the bottom region but due to too much overlapping data, we are unable to conclude whether these two labels only appear in the lower region. It is sensible that these two labels appear together, as Trousers and Dresses are similar. I observed that class 5 and 7 are mainly located near the region of class 9. This is sensible as sandals and sneakers share similar features with ankle boots. Overall, we can only conclude that class 9 will cluster well in K-Means clustering.

Figure 3



In this part, I use the K-Means clustering tool provided by the Sklearn library as our model for clustering the dataset. K-Means is an algorithm that iteratively assigns data points to the nearest centroid and computes the new centroid value until there are no changes in the centroid value. I assign 10 clusters for our model as instructed in the coursework. After that I plotted the result returned by our model in a 2-d plot as shown in figure 2 to show the centroid locations and the data that is assigned to each centroid. As you can see from the plot in figure 2, our model is able to provide 10 clusters in our dataset however we are not sure whether it is able to cluster each class accurately just by looking at this plot. Hence I will be using bar plots to show what items are inside these clusters.

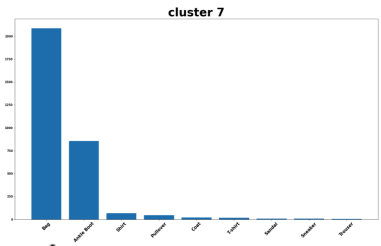


Figure 4

I decided to only extract bar plots that are able to provide meaningful observation of our model. If we look at figure 3, we observed that in cluster 3 our model performs quite well in clustering the Ankle Boots as it occupies a very high proportion in this cluster. This does prove our assumption that K-Means clustering will perform very well for ankle boots when doing exploratory data analysis. Besides cluster 3, we can also see from figure 4 that K-Means are also able to cluster Bags quite well as cluster 7 mainly consists of Bags. One of the reasons that it is able to cluster both of these items well is mainly because both have a very distinct look compared to other items in the dataset.

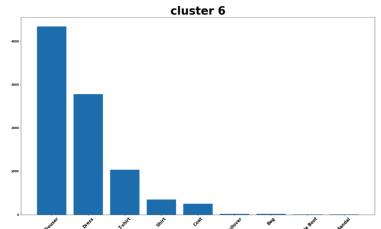


Figure 5

In figure 5, our model captures a large amount of data points on Trousers. However, ⅓ of this cluster also contains the data points on Dress. This can be explained from the analysis above as I found that the data points of trousers and dress do appear closely to each other in the bottom region. This causes our model to be unable to differentiate between these two classes. This also happens in the cluster shown in Figure 6 as it also consists of a high amount of sandals and sneakers' data points. One reason that our model clusters them together is because PCA removed information that differentiated them .

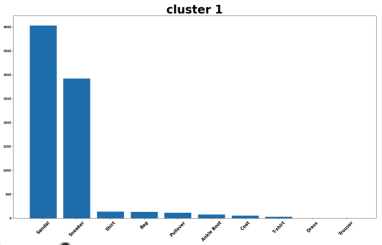


Figure 6

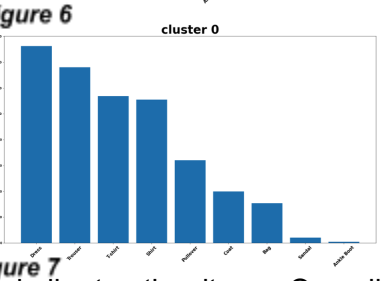


Figure 7

Cluster 2,4,5,6,8 and 9 have a very similar plot as the diagram of cluster 0 shown in figure 7. These clusters consist of many different types of items, and, we can conclude that K-Means performs badly as it couldn't differentiate the items very well. For example, when looking at cluster 0, it has almost the same amount of data points for Dress, Trousers, T-Shirt, Shirt. This happened mainly because after performing PCA, the items in the dataset have lost their distinct features and appear to be similar to other items. Overall, K-Means clustering couldn't cluster each of the items properly as it only focuses on grouping data points that are near to the centroid. Only the Bags and Ankle Boots will remain quite different from other items.

ANN Classifier

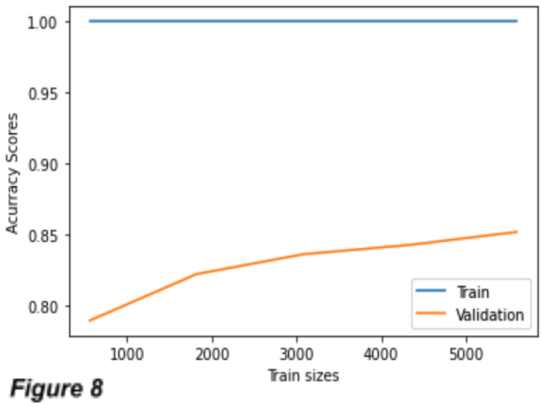


Figure 8

In this part of the coursework, I will be using the MLPClassifier model from the Sklearn library to train the fashion MNIST classification dataset. The MLPClassifier performs a forward pass to calculate the output and uses the output to compute the loss function of our model. In order to optimize the model, it uses backward pass to backpropagate the loss and update the model parameter by using gradient descent. For the preprocessing phase, I subsample the dataset from a size of 70000 to 10000 as it would take too long for the model to be trained. At the same time, I checked the number of data points in each label to ensure there is no imbalance in the number of labels as this issue might affect my model's accuracy. Besides that, I also standardize the dataset as bigger scaled features might dominate when training the model and affect the model accuracy. In the last step, I splitted the dataset into a training and testing set where 30% of the data will be used to evaluate my model performance by using train_test_split function from Sklearn library.

We used the learning curve tool from Sklearn library to plot our model's learning curve. This tool is able to evaluate the accuracy score on training and test sets for different training sizes and allows us to

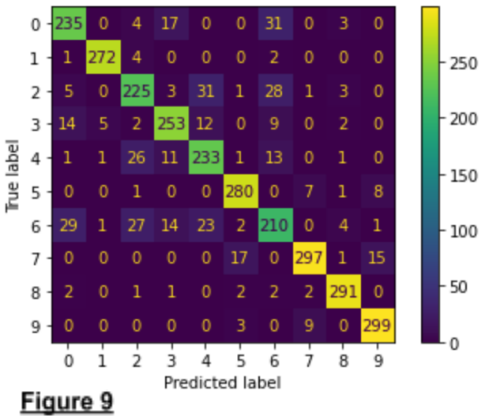


Figure 9

perform K-fold cross validation to ensure all of the data will appear in the training and test set. Based on the learning curve that we plotted in figure 8, I noticed that the training data always has a perfect accuracy score. Whereas the validation data shows that the accuracy score slowly increases as the number of data increases. It shows some sign of overfitting in our model as the gap between training and validation curve are quite far apart.
I will be using the accuracy score to evaluate my model as it is a standard metric of evaluation for classification. My model achieved an accuracy score of 0.855 on the testing dataset. However, I decided to plot the confusion metric to have a better understanding of my model's performance. From figure 9, we can see that our model performed very well in classifying item 1,5,7,8 and 9 with few misclassifyings occurring. However, we can see that our model did a poor job in item 6 as it frequently misclassified it as other items.

In this section, I will be analyzing the influence of a single hyperparameter on our model performance. I will be using Sklearn's validation curve since it is used to evaluate the training and test scores when varying the hyperparameter values. It also allows us to apply K-fold Cross Validation to ensure all data will appear in the training and test set. I decided to first look at how the 'hidden_layer_sizes' hyperparameter will affect our model performance. From figure 10, I noticed that the number of neurons in each hidden layer has an obvious effect on my model performance. For example, when using only 50 neurons in our model the accuracy does drop compared to 100 and 150 neruons. Besides that, I also look at how different algorithms for weight optimization will influence our model performance which is shown in figure 11. Overall, different algorithms do not have any effect on the training score while on the cross-validation score I noticed that using 'adam' algorithms does improve my model's performance compared to other algorithms. For the 'learning_rate_init' hyperparameter, I decided to use a table to analyze it as using a plot would not visualize it well because the range goes from 0.00001 to 1. From table 1, I found that when we increase the value of 'learning_rate_init', it brings a negative effect to the train and cross-validation score. However, when the value ranges from 0.00001 to 0.005 it does not show much effect on our model performance.

Figure 10

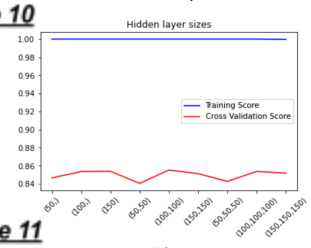


Figure 11

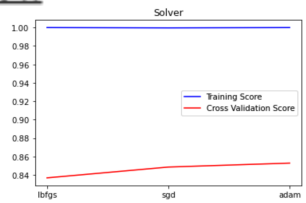


Table 1

Learning_rate_init	0.00001	0.00005	0.0001	0.0005	0.001	0.005	0.01	0.05	0.1	0.5	1
Train Score	0.996	1.000	1.000	1.000	1.000	0.996	0.956	0.880	0.869	0.889	0.859
Validation score	0.853	0.853	0.851	0.853	0.852	0.849	0.825	0.800	0.798	0.792	0.786

For the alpha hyperparameter, we analyze it using a table as well. In table 2, I noticed that changing the values of alpha has almost no effect on our model's performance. When using a very low value such as 0.00001 still has almost the same effect on our model when using a very high value such 1.

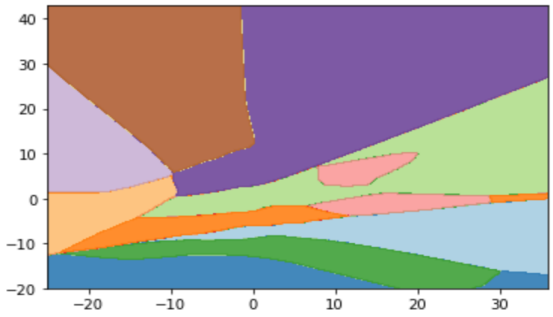
Table 2

Alpha	0.00001	0.00005	0.0001	0.0005	0.001	0.005	0.01	0.05	0.1	0.5	1
Train Score	0.996	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.999	0.995	0.988

Validation score	0.852	0.851	0.849	0.853	0.851	0.854	0.853	0.853	0.849	0.855	0.856

After analyzing how different hyperparameter values on each hyperparameter will affect our model's performance, I found out that different values on 'learning_rate_init' has the most effect on my model's performance. The 'hidden_layer_size' and 'solver' hyperparameter only have very little effect on the prediction accuracy of unseen data. Lastly, the 'alpha' hyperparameter shows almost no effect when different values are used.

Figure 12



In order to plot the decision boundary I used PCA to transform all features into two in order to allow us to plot the decision boundary in a 2-d plot. In figure 12, I plotted the decision boundary of our model on the fashion MNIST dataset. It allows us to understand how our model is able to separate the whole feature space into different classes. I noticed that our MLPClassifier is able to separate each class into different regions.

Support Vector Machine

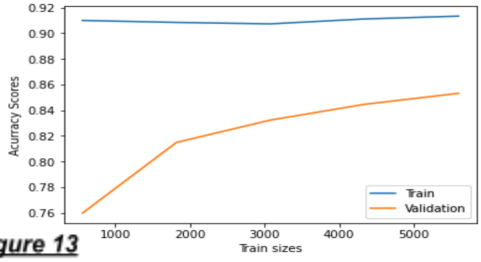


Figure 13

For SVM, I will be using Sklearn's SVC model on the same dataset that I used in MLPClassifier. The objective of SVC is to separate classes using a hyperplane that maximizes the margin, which is the maximum distance between different classes of data points. It uses a kernel trick to perform non-linear classification. I specified my kernel type to be 'rbf' in my model and kept other hyperparameters as default values. Figure 13 shows the learning curve from our model. We can see that the accuracy on unseen data is increasing as the number of training data is increasing. However we still see quite a large gap between our training and validation curve in terms of accuracy score. Hence, we can conclude that our model does suffer from overfitting to some degree, however it is still better than the MLPClassifier model. Our model has a 0.857 accuracy score on the test dataset and I have plotted the confusion matrix as shown in figure 14 to observe which labels have better accuracy when using SVM. I noticed that class 2,3,5,7,8 and 9 do classify well when using SVM by having only a few misclassification errors. Besides that, we also can see that class 6 still has the problem of misclassifying it as other classes, which also happened with MLPClassifier.

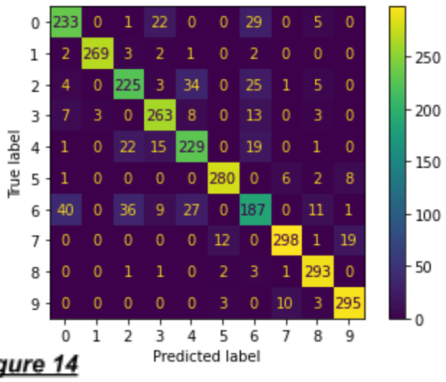


Figure 14

Figure 15

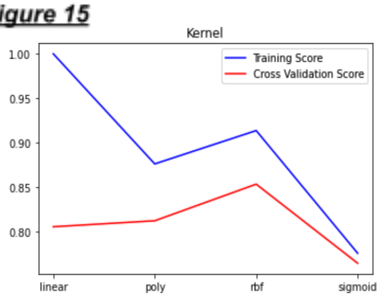


Figure 16

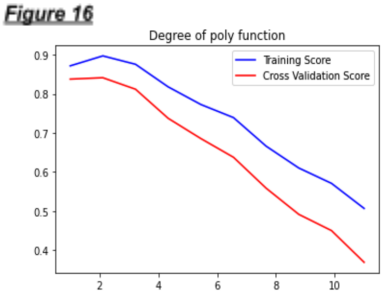
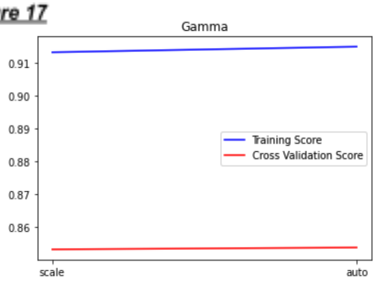


Figure 17



In this part, I will be analyzing how the change of the values of hyperparameters will affect our model performance using the validation_curve function from the Sklearn library. Firstly, I analyze the effect of using different kernel functions on our model and it is shown in figure 15. I noticed that different kernel

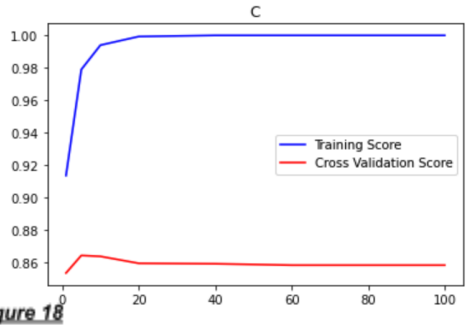


Figure 18

functions do influence our model significantly. When using the 'sigmoid' kernel function, our model's performance drops drastically on both training and cross validation sets. Besides that, when using the 'linear' kernel function, our model has a decent accuracy score on training and cross validation set but our model suffers overfitting issues as the gap between training and validation curve is large. Overall, 'rbf' kernel function has the best effect on our model as the accuracy on unseen dataset is the highest and our model has a good fit.

The 'polynomial' kernel function does not have a better effect on our model's performance compared to 'rbf'. However, it can be tuned further by changing the degree hyperparameter which is shown in figure 16. I noticed that when the number of degrees gets higher, our model's performance tends to drop. This can be explained by setting higher degrees makes our model too complex and unable to predict unseen dataset well. Besides that, I observed that when the degree value is set to 2, our model's performance does improve and has quite a good fit.

Furthermore, I also analyze how different settings of hyperparameter gamma will affect my model. Based on figure 17, I noticed that changing the setting of gamma has no significant effect on my model as the accuracy on cross-validation sets remains the same when using 'scale' and 'auto.

Lastly, I also analyze the hyperparameter C, which is the regularization parameter and this hyperparameter is the most common hyperparameter to be tuned in SVC. From figure 18, I noticed that using a high value of C leads to having a perfect accuracy score on my training dataset which shows that our model becomes overfit because the gap between the training and cross validation curve are far apart. I found that using a value around 5 to 10 does improve the accuracy of the unseen dataset.

In conclusion, the hyperparameter kernel and degree of polynomial kernel function have a very strong effect on my model's performance.

Figure 19

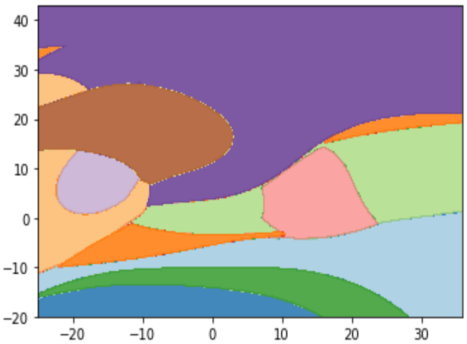


Figure 19 shows the decision boundary by our model, I noticed that the SVC has smoother decision boundaries between each label compared to the MLPClassifier model. We also see that the SVC does have a lot of classes having different regions in the feature space. For example, we see that the orange region appears in three different regions in our decision boundary.

In the final part, we compared the performance of SVC and MLPClassifier in terms of their accuracy score on the testing set and the time taken to train the model.

Based on the barplot in figure 20 and 21, we see that both have roughly the same accuracy score as each other but it is obvious that SVC took almost more than half of training time when compared with MLPClassifier. In this case, we can conclude that MLPClassifier performs better as it takes less computational power to get roughly the same accuracy as SVC.

Figure 20

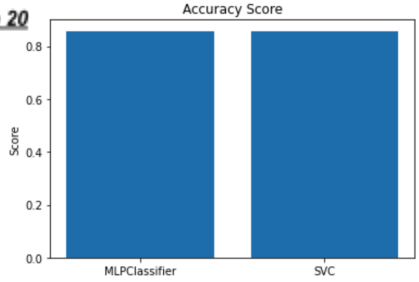
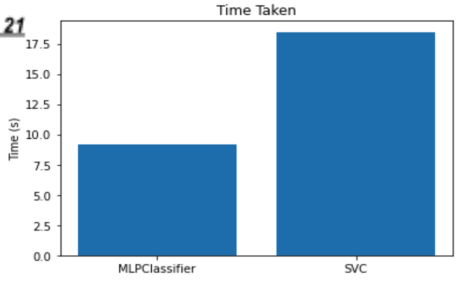


Figure 21



Bayesian Linear Regression

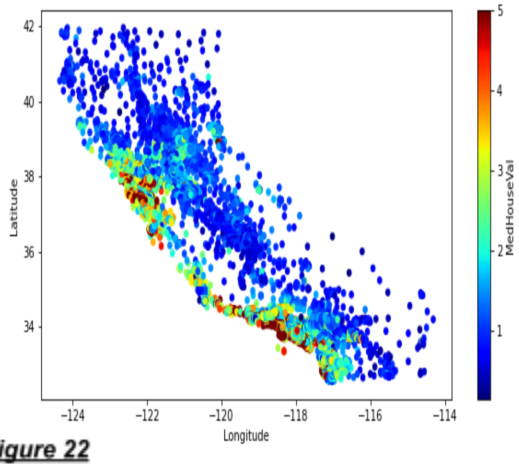


Figure 22

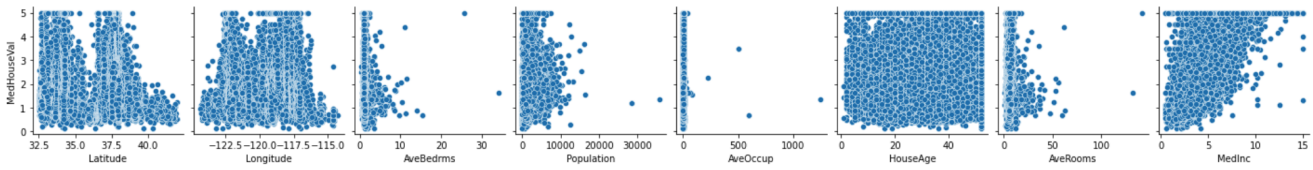
For this part of the coursework, I will be using the PyMC3 library to perform Bayesian Linear Regression for the California housing dataset. Bayesian Linear Regression is able to compute the posterior probability distribution by using the Markov Chain Monte Carlo sampling method. First, we will analyze how latitude and longitude will affect the median house price as shown in the figure 22. For latitude, we noticed that most of the expensive houses are located at a lower latitude, but at the same time there are a lot of cheaper houses in the lower latitude as well. Besides that, I noticed that when the latitude is around 39-42 the median house prices are mainly lower. For longitude, it appears that the expensive houses are mainly located at around -123 to -117 region, but the same problem also occurs when there are a lot of cheaper houses in that region as well. If we look at the latitude and longitude altogether, we can conclude that most of the expensive houses tend to be in the lower left region. Since this plot is an actual map of California, it makes sense that the expensive houses are mainly located in the region closer to the ocean.

Next we will be exploring the dataset to determine whether we have to transform/clean the dataset. I used the info function provided by Pandas to inspect the dataset's data type and number of non null values as shown in figure 23. I noticed that there is no missing data which means I do not have to do imputation or deletion for the dataset. Besides that, all the data appeared to be in float data type, which means there is no categorical data to deal with and I do not have to perform one hot encoding to transform a categorical dataset into numerical form. Since our main goal is to apply Bayesian linear regression to our dataset, we have to analyze the value of each feature to determine whether we need to standardize the features or not. Hence, I plotted the relationship between each feature to the median house value as shown in figure 24 . I found out that the value for each feature has large differences. For example, longitude ranges from around -120 to -115 but average bedrooms mainly range from 0 to 10. Hence, standardization of the features will be applied as it will affect the accuracy of our model.

Figure 23

```
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   MedInc       20640 non-null  float64
 1   HouseAge     20640 non-null  float64
 2   AveRooms     20640 non-null  float64
 3   AveBedrms    20640 non-null  float64
 4   Population   20640 non-null  float64
 5   AveOccup     20640 non-null  float64
 6   Latitude     20640 non-null  float64
 7   Longitude    20640 non-null  float64
 8   MedHouseVal  20640 non-null  float64
dtypes: float64(9)
```

Figure 24



For my Bayesian Linear Regression, I will be using the No-U-Turn Sampler method which is a more efficient sampling method in MCMC. For prior, I assigned a weakly informative Gaussian distribution prior to my model parameter which has a large standard deviation. This is because I do not have any expert knowledge or strong belief on my parameter. Hence, the observed data will definitely dominate my model parameter's posterior distribution. For sigma, I used uniform distribution as the value must be a positive value.

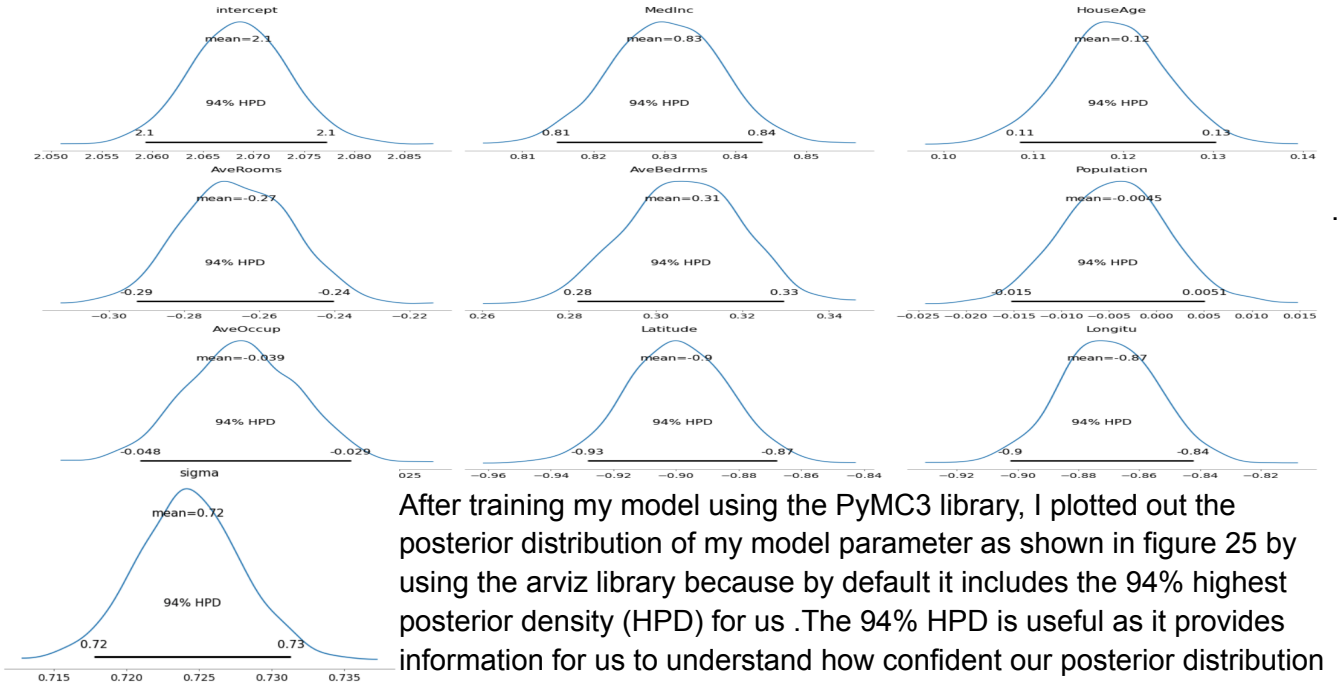


Figure 25

After training my model using the PyMC3 library, I plotted out the posterior distribution of my model parameter as shown in figure 25 by using the arviz library because by default it includes the 94% highest posterior density (HPD) for us .The 94% HPD is useful as it provides information for us to understand how confident our posterior distribution are. As we can see that almost all our model parameters have a very good result as the range in the 94% HPD is very small. However, I found that for the population feature the 94% HPD ranges from -0.015 to 0.0051 which shows that it is not entirely sure whether it has a positive or negative effect on our model. I also plotted out the correlation between features and median house value as shown in the figure 26 to evaluate my posterior result. I noticed that in our posterior distribution, the MedInc feature has the strongest effect as the value is the highest compared to other features. This is accurate based on our correlation plot, we see that MedInc has the highest correlation with the median house value. Besides that, I also noticed that the positive and negative effect of my features on the median house value do match with their correlation. However, only the AveBedrms feature performs badly as in our posterior distribution it has a positive effect on median house value but when looking at their correlation, it shows a weakly negative correlation with median house value. If we look at the posterior distribution for sigma, the 94% HPD ranges from 0.72 to 0.73 indicating a very little uncertainty in the median house value. Overall, we can conclude that our posterior distribution for model parameters does produce a good approximation to the desired distribution.

Figure 26

	MedHouseVal
MedInc	0.688075
HouseAge	0.105623
AveRooms	0.151948
AveBedrms	-0.046701
Population	-0.024650
AveOccup	-0.023737
Latitude	-0.144160
Longitude	-0.045967

After that, we sampled our dataset to sizes 50 and 500 to compare the result we obtained from running PyMC3 with 20640 datapoints. When using only 50 and 500 data points, the 94% HPD range and standard deviation becomes larger. This indicates that our models are uncertain about the parameter values .Besides that, some of the model parameters show a different effect to the median house value. For example, when using only 50 data points, the HouseAge feature shows a negative effect on the median house value where the actual correlation should be positive. This happens to the population feature when using only 500 data points as well. The possible reason that contributes to this problem is because our prior belief is very uninformative and when the data is limited the model parameter will be highly uncertain.

CART Decision Tree

CART Decision Tree is a learning algorithm that is used for classification or regression predictive modeling. The objective of CART Decision Tree is to minimize the error where on regression will be the sum of squares error and classification will be the Gini index. The structure of a decision tree consists of root nodes, internal nodes and leaf nodes. The CART Decision Tree used the greedy algorithm to

find an optimal local choice at each node. The algorithm starts from the root node which represents the whole dataset and runs an exhaustive search over each possible variable and threshold for a new node. For each variable and threshold it first computes the average value of the target variable for each leaf and proposed new node and computes the error as well. Then it will choose the variable and threshold that minimize the error and add it as a new node. It repeats doing the exhaustive search until there are only n data points associated with each leaf node. Lastly, prune back the tree to remove branches that do not reduce by more than a small tolerance value.

Figure 27

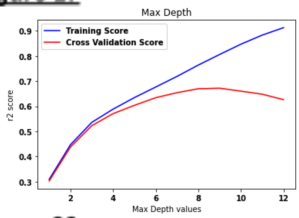


Figure 28

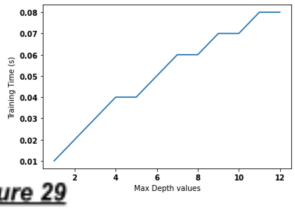


Figure 29

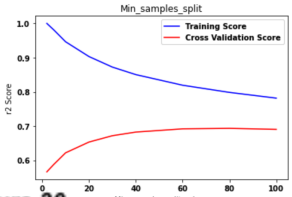


Figure 30

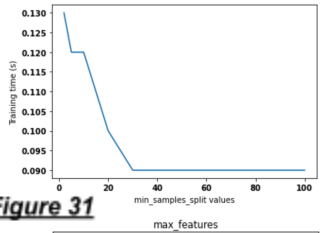


Figure 31

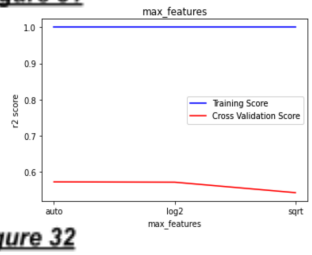
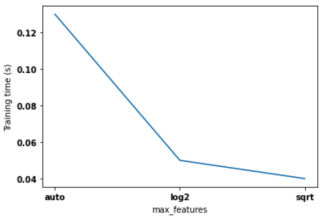


Figure 32

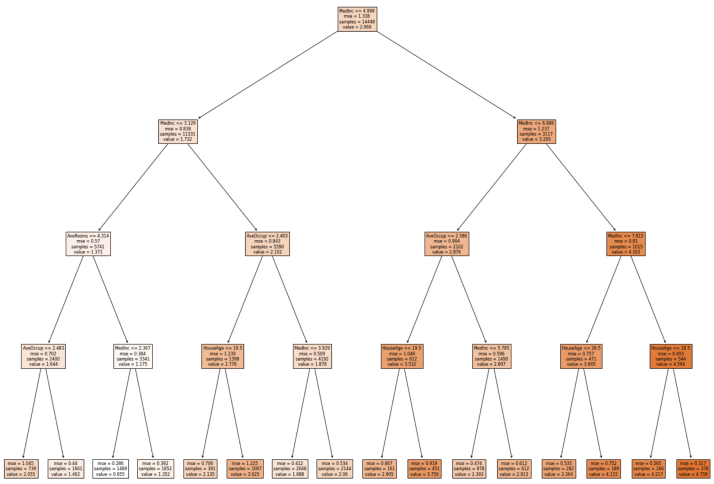


For the CART Decision Tree, I will be using the model DecisionTreeRegressor from the Sklearn library and the GridSearchCV function which is also from the Sklearn library to perform model selection in order to optimize my model's hyperparameter. Basically, GridSearchCV uses an exhaustive search method to select the best combination of hyperparameter values. I specified a grid of hyperparameter values for each hyperparameter and GridSearchCV will train our model on each hyperparameter combination. It will then evaluate the model for each different combination using the cross-validation method to output the best combination. Besides that, it uses K-fold cross validation which ensures that all data will appear in the training and test set. As our model is a regression task, I will be using r-squared (r2) to evaluate my model performance, which the formula is 1 - sum((y-predicted)^2) / sum((y-mean(y))^2).

In this section, I will be looking at how different hyperparameter values will affect my model performance and training time. At the end of the analysis, I will make a conclusion on whether the training time will affect my choice on the hyperparameter value. Firstly, I analyze how the change in value of 'max_depth' hyperparameter will affect my model by using the validation_curve function from the Sklearn library. In figure 27, I noticed that when the value of 'max_depth' increases, our model's performance does change a lot. The cross validation score ranges from around 0.3 to 0.65 when we use different values of 'max_depth'. Besides that, when using a large value, our model shows more signs of overfitting as the gap between training and cross-validation curve becomes wider. Based on figure 28, I noticed that a higher value of 'max_depth' does increase the training time but it still takes a very short time for our model to be trained. Next, I will be looking at the effect of changing the value of the hyperparameter 'min_samples_split'. From figure 29, I noticed that when the value gets higher, the training score tends to decrease while the cross validation score gets higher and flatten when the value reaches 40. When using a low value for 'min_samples_split', my model has a strong sign of overfitting issue as the gap between training and cross validation curve are far apart. From figure 30, I see that using a higher value of 'min_samples_split' took a shorter time for our model to be trained. This is mainly because the number of split nodes decreases when the minimum number of samples required for a split is higher. When looking at the validation curve and training time for my model on different values of the hyperparameter 'min_samples_leaf', it shows roughly the same result as 'min_samples_split' which can be seen in figure 29 and 30 . Hence I will not include the plots for 'min_samples_split' in my report. Next I will be analyzing the hyperparameter 'max_features'. This hyperparameter decides the number of features to consider for the best split. From figure 31, I noticed that when 'max_features' is set to 'auto', our model performs the best but is just a slight improvement compared to 'log2'. However, we can conclude that using 'sqrt' for 'max_features'

has quite an obvious negative effect on our model performance. When looking at the training time on figure 32, I noticed that ‘auto’ uses longer time for the model to train compared to ‘log2’ and ‘sqrt’. Overall, I found that using different values of ‘max_depth’ affected my model’s performance the most. As you can see from the plot in figure 26 that when the values range from 1 to 100, the r2 score actually changes from around 0.3 to almost 0.7. Hence, I concluded that the hyperparameter ‘max_depth’ has the strongest effect on my model. Besides that, when conducting the analysis on time taken for our model to train, I noticed different values for each hyperparameter does affect the training time. However, the CART Decision Tree has a very fast training time. Hence, the training time would not affect my decision on the value of each hyperparameter as the model itself does not take much computational resources.

Figure 33



After applying GridSearchCV on my model to obtain the best combination of the hyperparameter. My model is able to achieve a 0.713 r2 score on the test set. If I compare it with my previous model that uses the default hyperparameter values which has a 0.609 r2 score on the test set, my tuned model has improved a lot. However, our model still shows that the CART Decision Tree algorithm is not good enough to predict the unseen dataset as the r2 score is only 0.713.

In the last part, our project requires us to visualize our tree and it is shown in figure 33. Due to visibility reasons, I decided to only have a max_depth of 4 in my model. I noticed that the feature MedInc appears to be split multiple times, this can lead us to the conclusion that MedInc is the most important feature. This is because the CART Decision Tree selects the new node based on which feature that minimizes the error the most.

Random Forest

Random Forest is a supervised machine learning algorithm for classification and regression tasks. It is an ensemble learning method as it trains multiple decision trees and takes the mean vote of the result. Random Forest is similar to the bagging algorithm as it bootstraps samples from the training set. Bootstraps sample is a resampling technique that involves random sampling of a dataset with replacement. However, Random Forest draws random subsets of features to train individual trees and takes the mean vote as the prediction result. This makes each individual tree to have correlation with each other. This method is better than the usual bagging algorithm because uncorrelated models produce more accurate ensemble predictions.

Figure 34

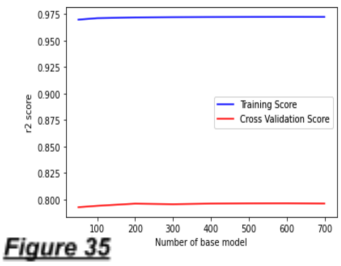
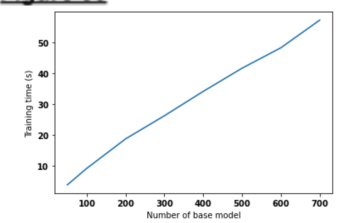


Figure 35



In this part, I will be using the RandomForestRegressor model from the Sklearn library to perform a regression task on the California housing dataset. Firstly, I have to find the optimum value for the number of base models. This can be done by looking at how different values of the hyperparameter ‘n_estimators’ will influence my model’s performance. I used the validation curve function from Sklearn library to carry out this analysis. Based on figure 34, I noticed that the performance of our model does not change much when the value of ‘n_estimators’ changes. However, there is a slight increase when using 200 as the ‘n_estimators’ value. In figure 35, we plot the training time of our model with different values of ‘n_estimators’. It shows that when using more number of base models, the training time also increases. Based on these two figures, I decided to

set the hyperparameter ‘n_estimators’ to 200 as it takes a reasonable amount of training time and has a slight improvement on our model’s performance.

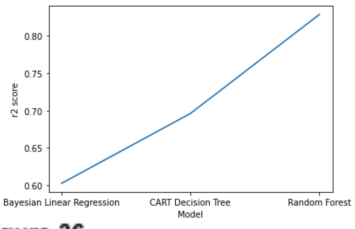


Figure 36

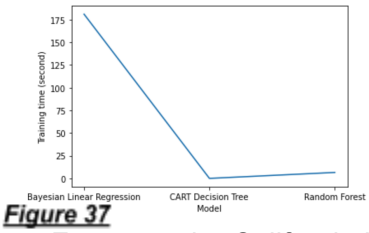


Figure 37

For my chosen setup for my model, I used the GridSearchCV function for model selection by looking at different combinations for hyperparameter ‘max_depth’, ‘min_samples_split’, ‘min_samples_leaf’, ‘max_features’ and ‘bootstrap’. I found out after performing model selection, the r2 score of my model does increase from 0.81 to 0.83 when compared to the model that uses 200 base models. Hence, I will be using the model that is tuned since it has a better performance. Overall, I can conclude that using Random Forest for my California housing dataset does perform quite well as it is able to achieve a 0.83 r2 score on unseen data.

For the last part of the coursework, I compare the performance and training time between Bayesian Linear Regression, CART Decision Tree and Random Forest on the California Housing dataset by plotting the r2 score and training time for each model to analyze. Figure 35 shows the r2 score for each model, I noticed that Bayesian Linear Regression has a very bad performance at predicting the unseen data as it only has around 0.61 r2 score. One possible reason is that linear regression is too simple and unable to fit our dataset well. Besides that, I noticed that Random Forest has a better performance than Decision Tree. This is because Random Forest is an ensemble learning algorithm that uses multiple Decision Trees to compute the prediction which result in better performance. On figure 36, I noticed that Bayesian Linear Regression uses a lot of computational power for training the dataset as it took 181 second for training. Besides that, Decision Tree has the lowest training time compared to the other model. Based on these two figures, I am able to conclude that Random Forest is the best model as it took slightly more time for training compared to Decision Tree but the performance does improve a lot, which is around 10% increase compared to Decision Tree and 20 % compared to Bayesian Linear Regression.