COMS30069 - Machine Learning Coursework

Ismail Nafaal Ibrahim - University of Bristol

1 Introduction

This is coursework is done for the module COMS30069 - Machine Learning at the University of Bristol.

2 Tasks

2.1 Analysing fashion-MNIST

2.1.1 PCA

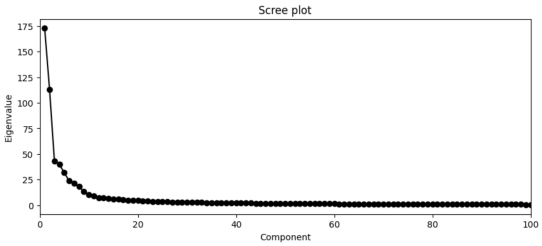
Principal component analysis is a method used in reducing the number of dimensions in large data sets by condensing a large collection of variables into a smaller set such that, it retains the majority of the large set's information. The fundamental concept of PCA is based on finding an orthogonal and linear projection of higher dimensional data $x \in \mathbb{R}^D$ to a low dimensional subspace $z \in \mathbb{R}^M$ where $M < D$ whilst being a good enough approximation of the original data. It does this by constructing and selecting principle components which approximates a value for the most relevant data.

Before we calculate these principal components, we will perform standardisation on the data such that the analysis benefits equally from each data point. To do this we import StandardScaler() from sklearn.preprocessing. Firstly, the covariance matrix for the input data is calculated. Using this matrix, we now calculate and extract the most significant M number of its eigenvalues and eigenvectors. A new feature matrix will be made using the selected principal components. This new feature matrix can now be used to approximate the values of the original data by recasting the data alongside the principle component axis.

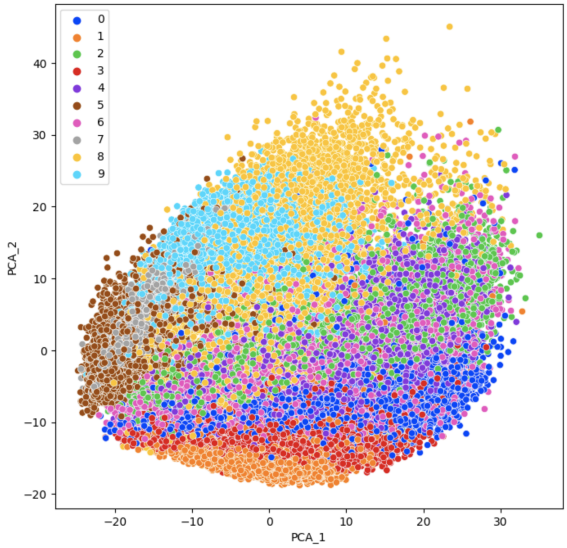
When looking into the fashion-MNIST we observe that each sample is 28x28 pixels, giving us a total of 784 features. When we perform principal component analysis on the dataset we find the following results for explained variances as most significant.

PC1	PC2	PC3	PC4	PC5
0.221	0.144	0.055	0.051	0.041

This shows that the just the first 2 principal components out of 784 can be used to represent 36.5% of the data. That significance can also be seen by plotting the scree plot on the eigenvalues.



If we project the original dataset onto the first two principle vectors i. e. fit them, we can plot the resulting scatterplot, with each class labelled.



From this we can see that some classes are very distinctly clustered, especially labels 1 and 5. We can also see that classes 2, 4 and 6 are overlapping with each other significantly, and would therefore be harder for any model to classify.

2.1.2 Gaussian Mixture Modelling

Gaussian Mixture Modelling is a form of unsupervised learning which is used to identify clusters or patterns within a data set. It assumes our data points are generated from a mixture of many other Gaussian distributions which can be represented as the following

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\mu_k, \Sigma_k).$$

where k is the number of classes, μ_k and Σ_k are the means and covariances of the k-th gaussian respectively. Thus, in order to do gaussian mixture modelling we need to obtain the best values for π, μ and Σ.

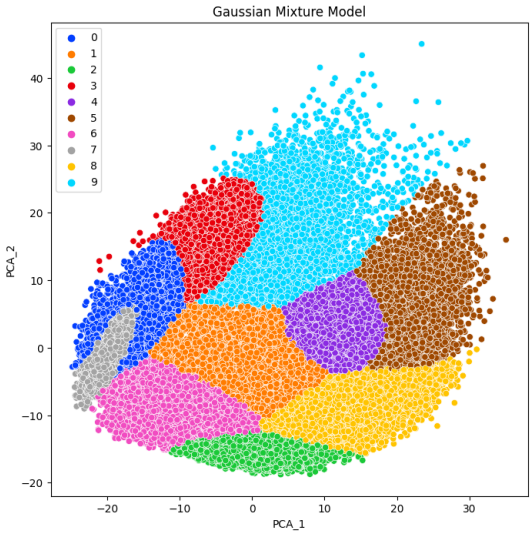
We do this by maximising the probability $\mathbb{P}(X|\pi, \mu, \Sigma)$ where X is a NxD matrix with N samples and D features. Given this, we want to maximise the probability of seeing the data X given π, μ and Σ.

$$\mathbb{P}(X|\pi, \mu, \Sigma) = \prod_{n=1}^{N} \left[\sum_{k=1}^{K} \pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k) \right]$$

To maximise this, we take the derivative of the log probability with respect to π_k, μ_k and Σ_k and solve for their optimal values. Each data point is then associated with a hidden latent variable z_n being an element

of [1...D] which specifies the identity of the mixture component which was used to generate x_n.

These identities can be found by computing a posterior over them using Bayes rule. These identities are known as the responsibility of a certain point n being in cluster k.



When plotting the clusters, with reduced dimensions from the previous section, we can notice that the model does correctly cluster some of the targets, such as 0, 2, 3, and 7 (Blue, Green, Red and Gray), but since there was overlapping data in the middle, it has done a poor job in clustering the classes around the center.

2.2 Classifiers

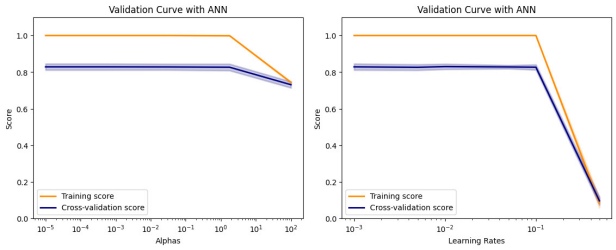
2.2.1 Artificial Neural Networks

Artificial neural networks are models of computation which try to mimic the operation of neurons in the human brain. It consists of node layers, containing an input layer, with one or more hidden layers. Data is passed through the layers if meets a certain required threshold, which is determined by some activation function.

To model an artificial neural network we will be using MLPClassifier from sklearn.neural_network. As before we scale our data before splitting them into train and test data.

There are some parameters we can set whilst training the neural network. The number of hidden layers, the activation function, learning rate and the value for alpha can be initialised. The learning rate is amount the weights are updated during a step in training. Selecting too big of a value for the learning rate would mean that the learning jump made would be over the local minimum, and might not be able to converge. Selecting a value which is too small would also cause problems as this would lead to the neural network taking a long time to train the data. Alpha is the regularisation term which penalises overfitting.

We initially set the number of hidden layers to be 128, the learning rate and the alpha value to 0.001. After that, we can plot the validation curves to see whether or not the model is underfitting or overfitting based on a range of any given hyperparameter.

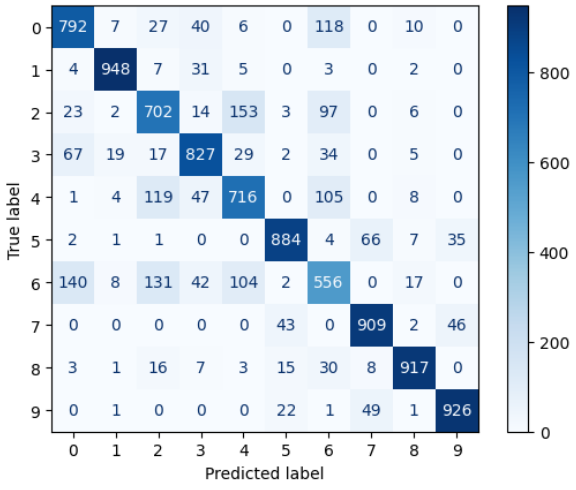


We can see from this that, for sensible values for alpha and learning rates the training and cross validation curves show that the data is very accurate for training data and good enough for the test data. However, when alpha and learning rates increase, the model becomes more simple (underfits) and fails to converge respectively, leading to worse scores on both sets.

For the initial fit of the neural network on both the training and testing data we see the following R scores

Training	Testing
1.000	0.8177

We see that our model overfits on the training data, and we get a score of 81.7% accuracy on test data unseen to the model. We can analyse the result of the testing data further by plotting the confusion matrix of the testing dataset.



A big part of the accuracy is being lost on misclassifications by the 0th, 2nd, 4th and 6th classes, especially on the 6th class. These classes are T-Shirt/Top, Pullover, Coat and Shirt respectively. We get this as these classes look similar to each other.

The hyperparameters can be modified to optimise our model and achieve a better test score. To find the best value for both alpha and the learning rate, we will use RandomizedSearchCV. This is used so that we could find a good set of hyperparameters much faster than GridSearchCV.

RandomizedSearchCV is initialised with a cross validation of 5, and 10 iterations at first. For the selection of alphas and learning rates, a range from 0.0001 to 1 and 0.001 to 1 is provided respectively. The result we get is as follows

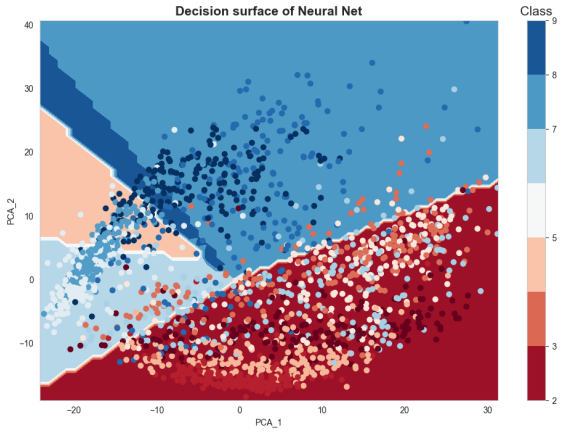
	Alpha	Learning Rate	Score
Initial	0.001	0.001	0.8177
Tuned_10	0.1072	0.0657	0.8234

By doing RandomizedSearchCV, we are able to find slightly better hyperparameters which are able to increase our model score by 0.5%. We could try and run the model for 30 iterations to see if the scores get any better.

	Alpha	Learning Rate	Score
Initial	0.001	0.001	0.8177
Tuned_10	0.1072	0.0657	0.8234
Tuned_30	0.1072	0.1232	0.8228

It seems that the hyperparameter values we have gotten in the initial run of RandomizedSearchCV were better than those we got on the second try. It is possible for that to happen with RandomizedSearchCV, as parameters are guessed at random. In order to get the absolute best parameters, GridSearchCV can be run at the cost of time and more computing resources.

We can take this model, and then plot its decision boundaries. To do that we first compute the principle vectors, so that we can reduce our dimensions to something we can represent visually. We make a mesh grid or xx and yy values, and use the principle components generated to project these 2D points into 784 dimensions, which are then used to calculate the predictions using our model. The resulting plot of prediction boundaries looks as the following



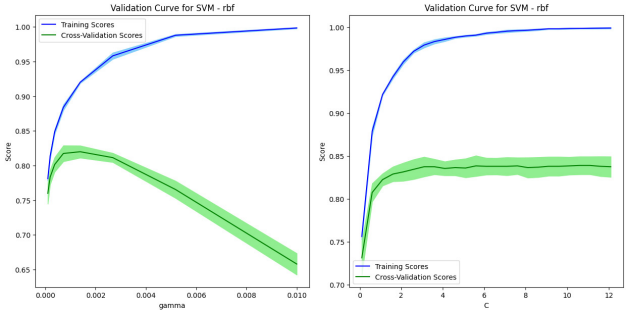
As we can see, not all the classes are captured in the decision boundary, however, we can see some boundaries being very clearly defined, such as classes 3 and 9.

2.2.2 Support Vector Machines

A Support Vector Machine - SVM is a supervised machine learning algorithm which can be used for classification and regression. It makes use of data points or support vectors to find the best hyperplane which separates the target classes.

For the classification of the Fashion-MNIST data using Support Vector Machines, we first import SVC from sklearn.svm. This is used to train a classifier using the first 2000 samples. Our SVC is initialised with a regularisation parameter of 2, whilst keeping everything else at default.

The validation curve for this initial support vector classifier is as follows



We can see that our model over-fits for gamma values greater than 0.001, and having a regularisation parameter greater than 2 leads to over-fitting, but with around 83% accuracy on unseen data

We can now test different kernels for the support vector machine and determine which kernel has the best performance

Kernel	C	Gamma	Train Score	Test Score
rbf	2	0.0013	0.9545	0.8264
linear	2	0.0013	1.000	0.7943
poly	2	0.0013	0.9125	0.7736
sigmoid	2	0.0013	0.7975	0.7555

Not surprisingly, rbf has the best score as it can combine multiple polynomial kernels multiple times before using a hyperplane to separate data projected onto a higher dimensional space. We also see that polynomial and sigmoid kernels do not over-fit as much as rbf and linear, even though their testing scores are a little bit worse.

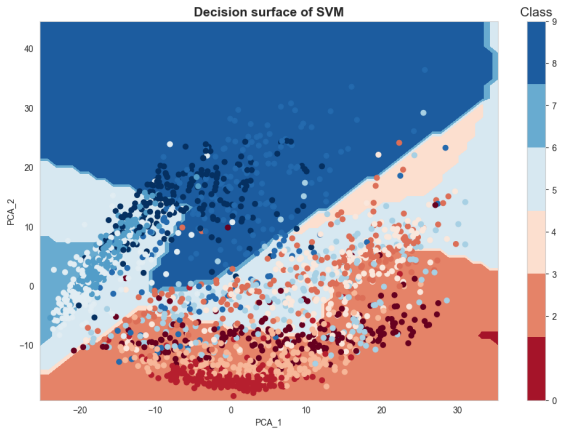
GridSearchCV is used to figure out which hyperparameters can be used to tune the model. For the values of gamma and C (Regulariser) we pass in the values between [0.0001 ... 0.01] and [0 ... 12] respectively and we obtain the following results.

Tuned parameters and updated scores

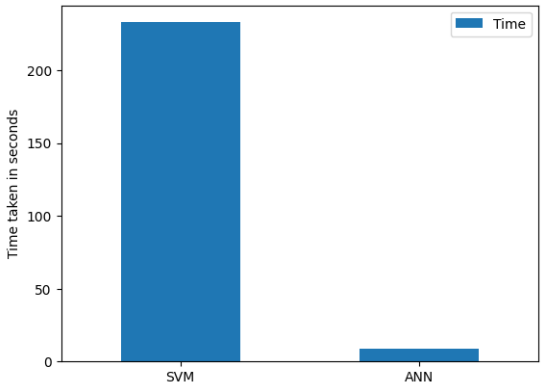
Kernel	C	Gamma	Train Score	Test Score
rbf	5.1	0.0014	0.9745	0.8960
linear	2	0.0013	0.9955	0.7965
poly	3.1	0.0026	1.000	0.8162
sigmoid	12.1	0.0004	0.9125	0.8118

By tuning the SVM, we see higher accuracy for all the kernels, except for the linear kernel. Testing scores for rbf, polynomial and sigmoid have increase by 7%, 4.3% and 5.6% respectively. However, for the polynomial kernel is overfitting as it increased from a training score of 91% to 100%.

To plot the model decision boundaries we compute the data's principle components, which is multiplied by the xx and yy of the mesh grid, so that we can use our support vector machine to make predictions over 784 features. The calculated predictions plotted look as the following.



Here we see that the decision boundary is not as clear as the decision boundary of the neural network. Furthermore, we see that even fewer classes are captured in comparison to the neural network.



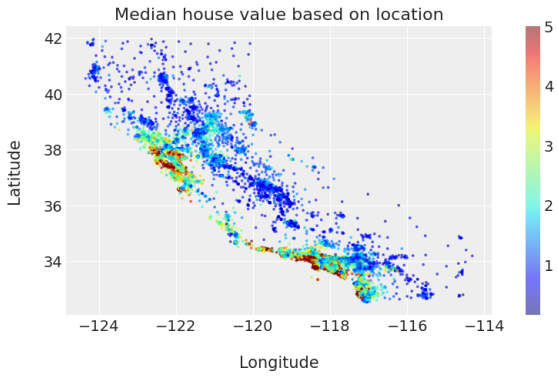
The SVM took an astounding 232.8 seconds to fit the data, whilst the artificial neural network only took about 9 seconds. This is because SVM's are based around a kernel function, which requires the computation of a distance function between each point in the data set. This is very memory dependent, and based on the number of samples, it would take a very long time to compute.

Model	Training Score	Testing Score
SVM	0.9386	0.8870
ANN	0.7523	0.7431

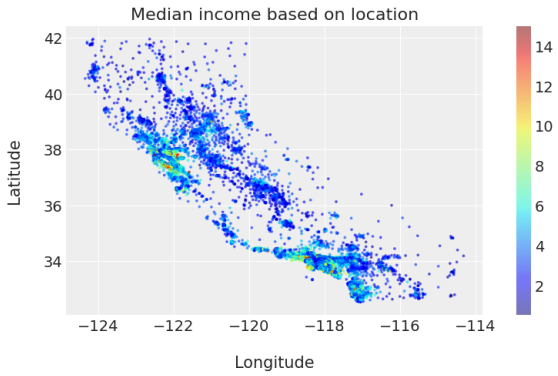
However, the SVM outperforms the neural network in testing scores as seen above.

2.3 Bayesian linear regression with PyMC

When we plot the location of California houses we observe the following

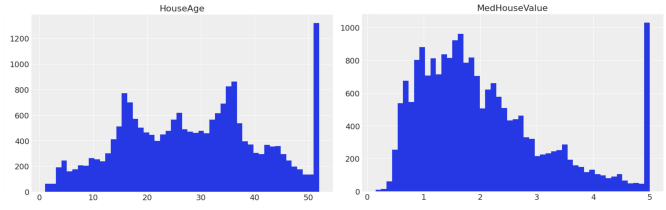


We see that there are two clusters which are prominent around the south side. When observing a map, we find that these locations are San Francisco and Los Angeles. The prices for houses near the sea, are more sought after and therefore become more expensive.



When median income is plotted in relation to the location we also see that the areas with higher house values also have people with higher incomes.

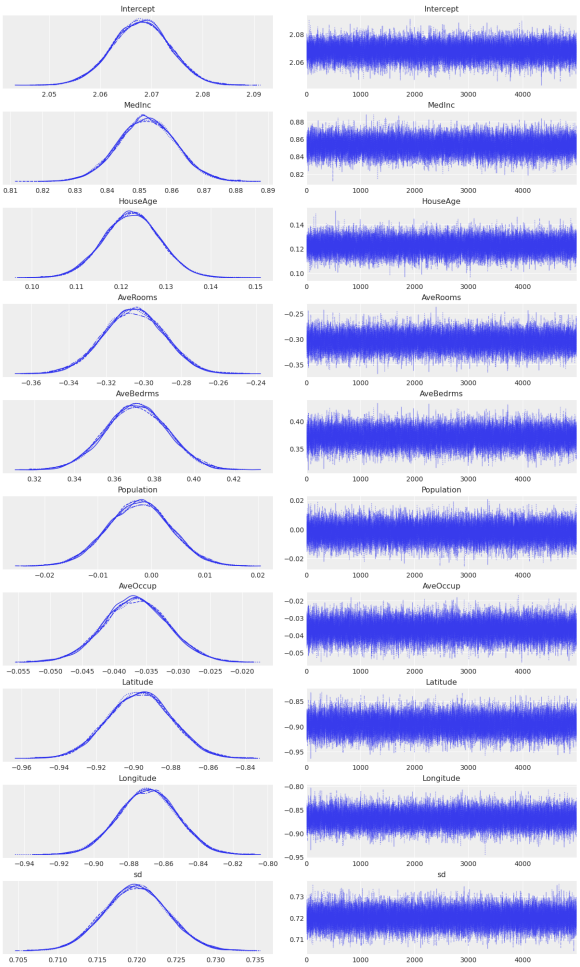
Outliers are abnormal data points, which could skew machine learning models, and lower their accuracy. However not all outliers are necessarily bad, and all outliers should not be removed. Even though removing outliers may increase the accuracy of our machine learning model, the outlier data might actually be true, in which case might contain very crucial information about the data collected.



When all the features of the California Housing Dataset are plotted on a histogram we notice that in median house age and median house value we have some outliers. As mentioned before, we need to take a sensible decision on what to do for such outliers. Checking for nulls does not reveal any missing information.

Bayesian Linear regression is a probabilistic approach to linear regression, where we use probabilities to predict data. It is a slight departure from basic linear regression where we work with exact values and find values for the weights and biases using discrete values. Using a Bayesian approach to linear regression we are able to take into account the unreliability of our parameter estimates.

We set our priors to be gaussian, and with a combination of all features dependent on y, we sample from the posterior and get back the following traceplot.



From full dataset

	Mean	Standard Deviation
Bias	2.07	0.01
Median Income	0.85	0.01
House Age	0.12	0.01
Avg Rooms	-0.31	0.02
Avg Bed Rooms	0.37	0.02
Population	-0.00	0.01
Avg Occupancy	-0.04	0.01
Latitude	-0.90	0.02
Longitude	-0.87	0.02
sd	0.72	0.00

This is the resulting means and standard deviations we got from sampling the posterior.

We can sample from a posterior estimated with lesser data points to see what effect has on the resulting samples.

From 50, 500 and full dataset

	sd_50	sd_500	sd_full
Bias	0.08	0.03	0.01
Median Income	0.16	0.05	0.01
House Age	0.09	0.03	0.01
Avg Rooms	0.36	0.09	0.02
Avg Bed Rooms	0.66	0.08	0.02
Population	0.08	0.03	0.01
Avg Occupancy	1.17	0.32	0.01
Latitude	0.29	0.09	0.02
Longitude	0.28	0.09	0.02
sd	0.06	0.02	0.00

Bayesian linear regression results in a distribution of possible model parameters based on the data and the prior we set. Since it is based on the data, the fewer data points we have, the more spread out our posterior would be. Thus we would get a higher variance on points sampled, as we can see from the table above.

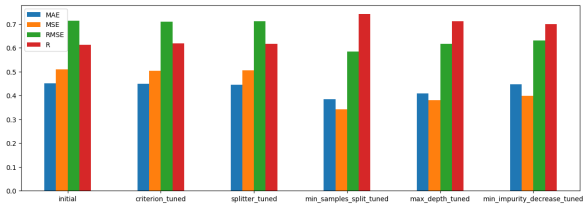
2.4 Trees and ensembles

2.4.1 CART Decision Trees

CART is a type of tree which can be used in classification and regression in machine learning problems. In CART, the training set is the root node, which is divided into two by taking the best attribute and threshold value into account. Additionally, the subsets are divided according to the same logic. This continues until the tree has either produced all of its potential leaves or discovered its last pure sub-set.

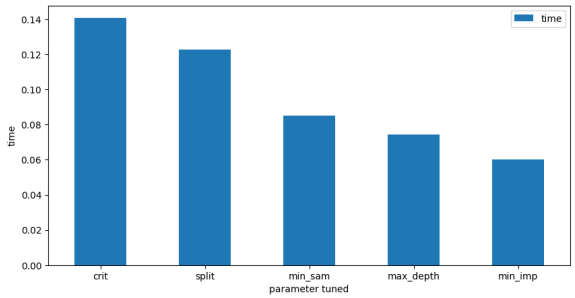
To do the predictions we will be using DecisionTreeRegressor from sklearn.tree. The dataset is fitted to the model with the default hyperparameters.

We obtain the following results against the test data.



From the following plot, we can see that tuning min_samples_split had the biggest impact on the Mean Absolute Error, Mean Squared Error and the accuracy R of the plot compared to other hyperparameters. We achieve an accuracy score of around 0.741. The next best parameter which improved the model score was max_depth, which after tuning resulted in an R score of 0.710.

After finding the optimal hyper-parameters, we plot the time taken for each Decision tree to fit our data, and we get the following results



hyperparameter values

criterion	friedman_mse
splitter	best
min_samples_split	60
max_depth	10
min_impurity_decrease	0.001

Just by tuning the min_impunity_decrease to 0.001 we get save over half the time it takes to fit a model which tuned only on criterion. We also see the time taken to fit a model tuned on max_depth to be low as well.

Decision trees can be used when there are tricky relationships between the features and the output variables, whilst linear regression would be more suited if there is a linear relationship between the features to the target variable

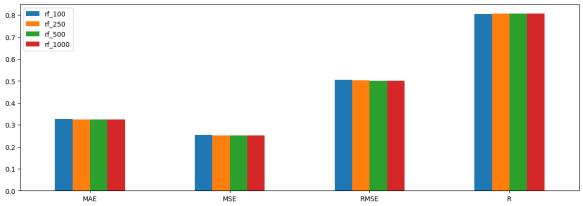
2.4.2 Ensemble Methods

For this task, Random Forest Regressor is used to model the data. It uses a bagging technique that grows multiple decision trees which are merged together for a more accurate prediction. It is done on the basis that multiple uncorrelated decision trees would perform significantly better than they would by itself.

Since RandomForestRegressor is a bagging ensembler it increases the amount of decision trees used in the prediction. The RandomForestRegressor has been fitted with 100, 250, 500 and 1000 n_estimators (decision trees) and the result is the following

	rf_100	rf_250	rf_500	rf_1000
MAE	0.326648	0.325310	0.324930	0.324834
MSE	0.254972	0.252253	0.251868	0.251531
RMSE	0.504948	0.502248	0.501865	0.501529
R	0.805425	0.807500	0.807794	0.808051

We can observe that increasing the number of estimators does not have a negligible change on either MAE, MSE or even the accuracy. This could mean that all the trees give an average prediction which has low variance. This can be further visualsed by plotting a table as seen below



References

[1] Scikit-Learn.

[2] Mohammed Alhamid. Ensemble Models: What Are They and When Should You Use Them?, 2022.

[3] Adrien Biarnes. Gaussian Mixture Models and Expectation-Maximization (A full explanation), 2020.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1 edition, 2007.

[5] Jason Brownlee. Understand the Impact of Learning Rate on Neural Network Performance, 2019.

[6] Niklas Donges. Random Forest Classifier: A Complete Guide to How It Works in Machine Learning, 2022.

[7] Bhumika Dutta. A Classification and Regression Tree (CART) Algorithm, 2021.

[8] Jim Frost. Guidelines for Removing and Handling Outliers in Data, 2018.

[9] Geeks4Geeks. CART (Classification And Regression Tree) in Machine Learning, 2018.

[10] IBM. Neural Networks, 2022.

[11] Zakaria Jaadi. A Step-by-Step Explanation of Principal Component Analysis. 2022.

[12] Julia Kho. Why Random Forest is My Favorite Machine Learning Model, 2018.

[13] Kishan Maladkar. Why Is Random Search Better Than Grid Search For Machine Learning, 2018.

[14] Medium.com. Decision Tree with CART Algorithm, 2021.

[15] Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022.