COMS30035 - Machine learning

1. Introduction

This report introduces various machine learning methods used during the Machine Learning Unit. It will cover various algorithms such as: PCA, GMM, ANN, SVM, Bayesian Linear Regression and CART decision trees. The Fashion MNIST dataset is used in sections 2.1 and 2.2. In sections 2.3 and 2.4 the California Housing dataset is used. In the next paragraph I will be talking about some of the key concepts in Machine Learning.

1.1 Supervised vs Unsupervised Learning

Supervised and unsupervised learning are two main categories of machine learning algorithms. In supervised learning, the algorithm is trained on a labelled dataset, where the correct output for each input is provided. In unsupervised learning, the algorithm is not provided with labelled training data, and must instead discover the underlying structure of the data through self-organization and pattern recognition. Supervised learning is often used for tasks such as classification, while unsupervised learning is often used for tasks such as clustering. Supervised learning is generally more accurate but requires more labelled data, while unsupervised learning is more flexible but may not produce as accurate results.

1.2 Underfitting vs Overfitting

Underfitting and overfitting are two common problems that can arise when training a machine learning model. Underfitting occurs when the model is too simple and cannot capture the underlying patterns in the data, while overfitting occurs when the model is too complex and has learned specific patterns in the training data that do not generalize to unseen data.

2. Analysing Fashion-MNIST

To analyse the Fashion MNIST dataset, you need to load and explore the data, pre-process it, train a machine learning model, and evaluate its performance. You can use techniques such as data normalization, data augmentation, and cross-validation to improve the model's performance, and use metrics such as accuracy, precision, and f1 score to evaluate the model's performance. You can also create visualizations to gain a better understanding of the data and the model.

2.1.1.1 Principle Component Analysis

Principal component analysis (PCA) is a dimensionality reduction technique that is commonly used in machine learning. It reduces the number of dimensions or features in a dataset while preserving as much of the variation and information in the data as possible. PCA is useful for addressing the curse of dimensionality and for visualization.

2.1.1.2 PCA Algorithm

To apply the principal component analysis (PCA) algorithm to a dataset, you need to perform the following steps:

1.Perform scaling and standardization on data.
2.Compute the covariance matrix of the data, which represents the correlations between the different features.
3.Compute the eigenvectors and eigenvalues of the covariance matrix, which correspond to the principal components and their relative importance.
4.Choose the number of principal components to keep, based on the amount of variation and information you want to preserve in the data.
5.Use the principal components to transform the original dataset into the lower-dimensional subspace, where each data point is represented by a smaller number of features.

2.1.1.3 Variance of first and second principal components

The first two principal components explain 36.5%. (22.1% for 1st PC and 14.4% for 2nd PC).

```
22.083547297081754 %
Text(0, 0.5, '$x_2$')
```

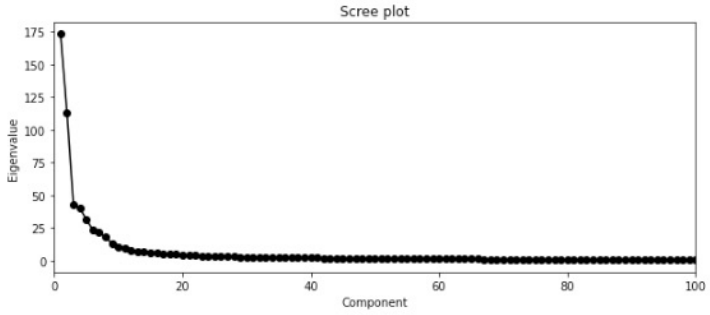


Figure 1: Scree plot to visualise eigenvalues

This result was obtained by following the steps mentioned in 2.2. The explained variance of a principal component is the ratio of the component's variance to the total variance of the original data. It can be used to assess the importance of each component in representing the data, with a high explained variance indicating a significant component and a low explained variance indicating a less important component.

2.1.1.4 Scatter plot of the data projected onto the first two principal components

The basis vectors of PCA are the eigenvectors of the covariance matrix of the data. These vectors define the directions along which the data varies the most and are used to project the data onto a lower-dimensional space.

```
[[ 1.         -0.00670663]
 [ 0.          0.99997751]]
```

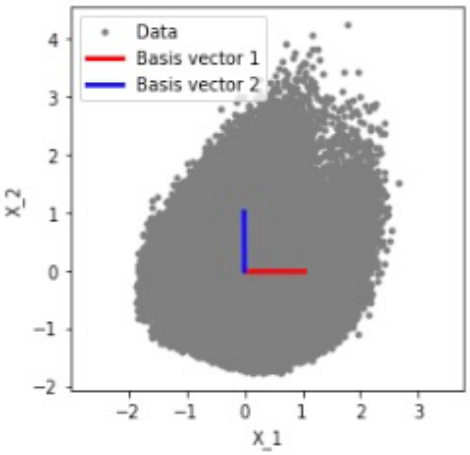


Figure 2: Plot of basis vectors

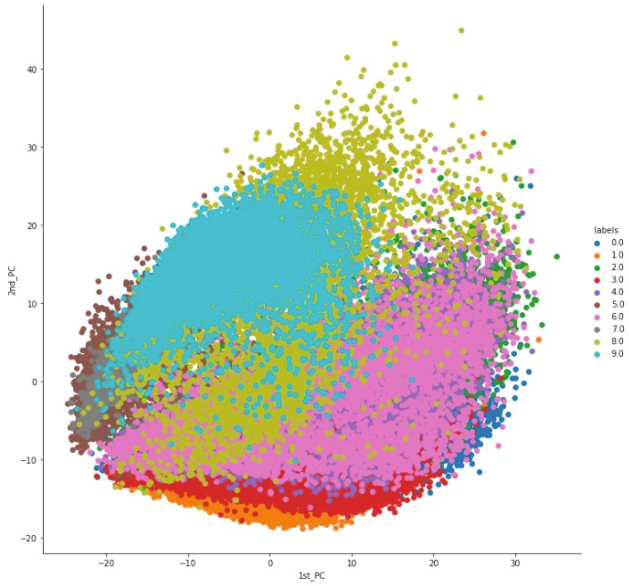


Figure 3: Scatter plot of projected data with class assignments

Figure 3 is a plot of the data projected onto the first two principal components. As you can see there is quite a bit of overlap between the class assignments and the boundaries are not clearly visible.

2.1.2 Gaussian Mixture Modelling

Gaussian mixture modelling is a method in machine learning that assumes a dataset is generated from a mixture of different Gaussian distributions. This method can be used for clustering, density estimation, and other types of data analysis. It allows the model to capture more complex patterns in the data and to more accurately describe the data.

I was required to use GMM to do soft clustering of the data using only the first two components from the PCA analysis above. However, before I did this, I decided to do K-

means clustering, which is hard clustering to compare the distribution of data and how classes are assigned.

Below in Figure 4 you can see each cluster is represented by the centroid of the data points in that cluster. The algorithm works by first initializing the centroids of the clusters, and then iteratively assigning each data point to the closest centroid and updating the centroids to be the mean of the data points assigned to them.

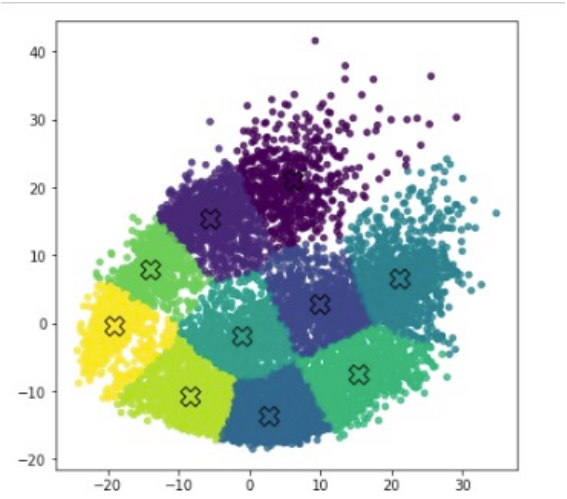


Figure 4: K-means clustering of first two principal components

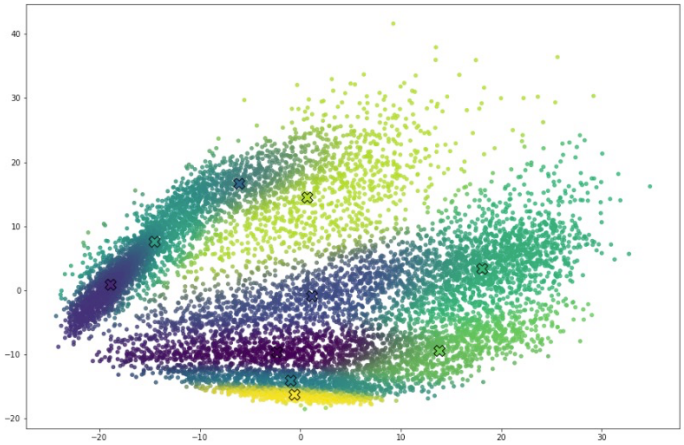


Figure 5: GMM soft clustering using first two principal components

In soft clustering using a Gaussian mixture model (GMM), the goal is to assign each data point to one or more clusters, but with a degree of uncertainty or "softness" rather than with absolute certainty.

The steps I took were as follows:

1.Fit a GMM to the data by estimating the parameters of the mixture of Gaussian distributions that best describe the data.
2.For each data point, compute the posterior probabilities that the data point belongs to each of the clusters. These probabilities reflect the degree of uncertainty or "softness" in the cluster assignments.
3.Assign each data point to one or more clusters based on the posterior probabilities. For example, a data point may be assigned to all clusters with probabilities greater than some threshold, or it may be assigned to the cluster with the highest probability.
4.Use the cluster assignments and posterior probabilities to summarize the data.

As you can see from the above figure, unlike K-means the cluster centres do not have hard edges and the datapoints are plotted probabilistically therefore it is possible that a datapoint with a probability above a certain threshold can belong to more than one class.

2.2 Classifiers

Artificial neural networks (ANNs) and support vector machines (SVMs) are two types of machine learning algorithms. ANNs are composed of many small processing elements called neurons that are connected to each other in a network and are designed to recognize patterns in data. SVMs are used for classification and regression tasks and find the line or hyperplane that best separates the different classes in the data. ANNs are better at recognizing patterns, while SVMs are better at finding lines or hyperplanes.

2.2.1 Artificial Neural networks

To train ANN I first took a subset of the training data to speed up optimisation, then standardized the training data, I then fitted it to a MLP Classifier with multiple different hyper parameters such as activation, solver, alpha and learning rate. Activation is the activation function for the hidden layer, it defaults to 'relu' which is the rectified linear unit function. Solver is the solver for weight optimization. This was set to 'adam' which refers to a stochastic gradient-based optimizer. Next alpha is the strength of the L2 regularization term. The L2 regularization term is divided by the sample size when added to the loss. Finally learning rate init is the initial learning rate used. It controls the step-size in updating the weights. The alpha and learning rate are the two hyper parameters which I will be adjusting.

2.2.1.1 Training and Validation curve

From running the training and test set on the MLP Classifier I got these results:

```
Training accuracy:  1.0
Testing accuracy:  0.85
```

This shows overfitting as Training accuracy is perfect however the Testing accuracy is not as good. Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the mode -l. The problem is that these concepts do not apply to new data and negatively impact the model's ability to generalize.

We can also visualise this by plotting a training and validation curve as shown on figure 6.

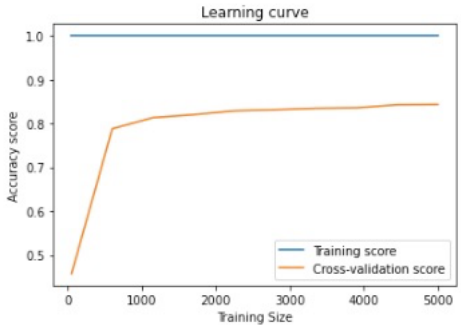


Figure 6: Training learning curve

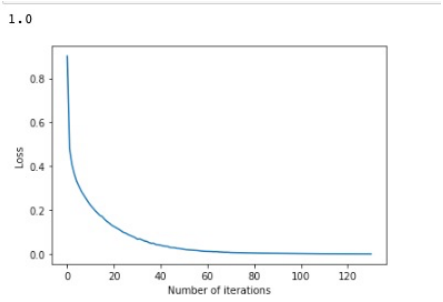


Figure 7: Training set loss curve

2.2.1.2 Hyperparameter tuning

To improve the accuracy and reduce overfitting I had to adjust the hyperparameters of the MLP Classifier. I used the RandomizedSearchCV to iterate through several different values for the parameters; alpha and learning rate. Then I used the best. params_ func to get the best parameters for the classifier to perform at its best.

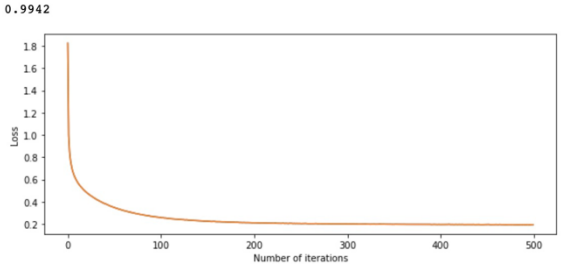


Figure 7: Training loss curve after hyperparameter tuning.

Learning rate	Training set	Testing set
0.0002	0.9942	0.8579
0.001	0.746	0.7303
0.01	0.8967	0.8411
0.1	0.9902	0.8559

Table 1: Training and testing accuracy depending on learning rate.

After adjusting the parameters, the Training accuracy has decreased and Testing accuracy has increased by a very small amount this means that it is not overfitting as much anymore, as training

data is not learning everything in the dataset such as noise, thus allowing the test set to increase in accuracy.

2.2.1.3 Decision Boundary

A Decision Boundary separates the data points into specific classes, where the algorithm switches from one class to another. Figure 8 shows a contour-based decision boundary of the classes of the dataset. This helps give a rough idea of how ANN classifies the datasets.

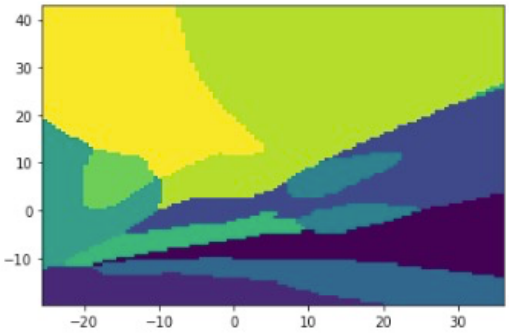


Figure 8: ANN Decision Boundary

2.2.2 Support Vector Machines

A support vector machine (SVM) is a supervised machine learning model that uses classification algorithms for two-group classification problems. After giving an SVM model sets of labelled training data for each category, they're able to categorize new data. Compared to newer algorithms like neural networks, they have two main advantages: higher speed and better performance with a limited number of samples (in the thousands).

2.2.2.1 Training and Validation Curves

To plot the training and validation curve of an SVM we follow a similar process to the one described in ANN, however we now fit the data to an SVM. I used an SVC with 'rbf' kernel and a C value of 4. This returned an accuracy as shown below for Training and Testing respectively.

```
SVM classifier accuracy: (0.9714, 0.8647)
```

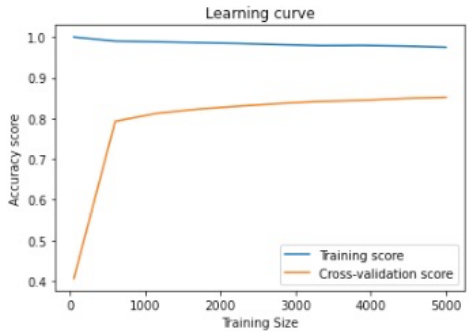


Figure 9: Training learning curve

As you can see there is still some overfitting as the training score is much higher than the testing score.

2.2.2.2 Hyperparameter Tuning

To improve the accuracy of the SVM I used GridSearchCV to find the best parameters for C and gamma and then fitted the training data to the SVC with 'rbf' as the kernel and the best parameters. The results I got for the training and testing accuracy are as follows:

```
SVM classifier accuracy: (0.9661, 0.8632)
```

This shows that the training and testing accuracy has decreased very slightly

C(RBF)	Training set	Testing set
5	0.9661	0.8632
1	0.908	0.8516
10	0.9847	0.8644

Table 2: Training and testing accuracy depending on C value.

Gamma(RBF)	Training set	Testing set
0.001	0.9661	0.8632
0.01	1.0	0.7368
0.1	1.0	0.1538

Table 3: Training and testing accuracy depending on gamma value

2.2.2.3 Decision Boundary

Because the fashion MNIST dataset consists of grayscale images of clothing items, the decision boundary in this case would be a complex, multidimensional surface, rather than a simple line or hyperplane. This is because the decision boundary would need to account for the different pixel intensities in the images, which would require a more complex, multidimensional surface to properly separate the data points into their respective classes.

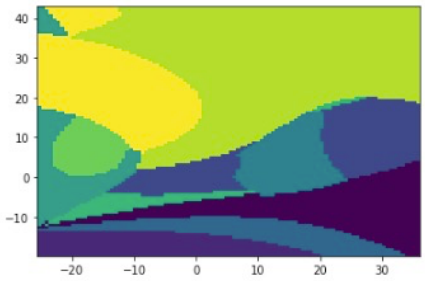


Figure 10: SVM Decision boundary

2.2.2.4 Comparing ANN and SVM performance

This bar plot shows the difference in training times for SVM when compared to ANN. As you can see SVM is considerably faster. Also, the accuracy of the SVM is also slightly higher for the Test set.

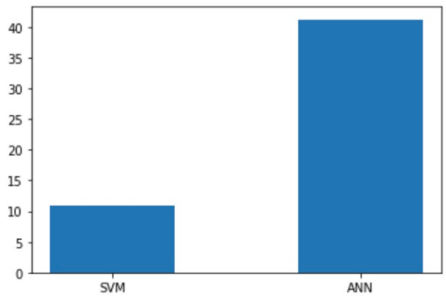


Figure 11: Training time of SVM vs ANN

2.3 Bayesian Linear Regression

Bayesian linear regression is a type of linear regression that uses Bayesian inference to estimate the model's parameters. In Bayesian inference, we use probability distributions to represent our beliefs about the model's parameters, and we update these distributions based on the data we observe. PyMC is a Python library for building and fitting Bayesian models. PyMC includes a module for implementing Bayesian linear regression. Using this module, you can specify the model and its priors, and then fit the model to data using Markov Chain Monte Carlo (MCMC) sampling methods.

2.3.1 Median House value

To work out how longitude and latitude affects median house price I first had to import the California Housing dataset then standardize it before I could plot it. By doing this I have ended up with figure 12. From this figure we can see that if the latitude is larger and the longitude is lower and if the latitude is lower and the longitude is higher the median house price tends to be lower.

Figure 12: Median House Price by Location

The higher median house price is at lower longitudes and latitudes.

2.3.2 Transforming data

No, the data does not need to be cleaned as using the. isnull().sum() func, I have table which shows there are no null values. This means that the data is reliable and is ready for analysis without the need for further processing.

2.3.3 Using PyMC to get approximate posterior distributions over each model parameter

In this task I chose a prior distribution for all the model parameters of the dataset named alpha, beta and sigma. I gave alpha and beta a mean of 0 and a standard deviation of 20. I chose these fairly 'flat' priors (high variance) because it represents a lack of prior knowledge as to the true values of the parameters.

The results I received after running PyMC are as shown in table 4 and figure 13.

	Mean	SD
alpha	2.07	0.01
beta	0.83	.01
sigma	0.72	0.00

Table 4: Posterior values

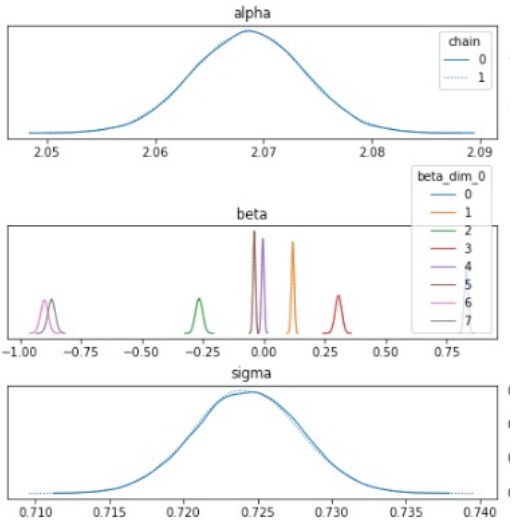


Figure 13: alpha, beta and sigma

2.3.4 How did PyMC perform

Yes, my run of PyMC did succeed as it produced good approximations for the posterior distribution. I know this because by looking at figure 13, we get a smoothed estimate of the posterior distributions and I can see that in alpha, the two samples from the two chains overlap each other almost perfectly.

2.3.5 Running PyMC on random samples of data

In this next task, to evaluate if sample size made a difference to the posterior distributions produced, I conducted the same PyMC on a sample of size 50 and a sample of size 500.

	Mean	SD
alpha	2.07	0.01
beta	0.83	.01
sigma	0.72	0.00

Table 5: 500 samples

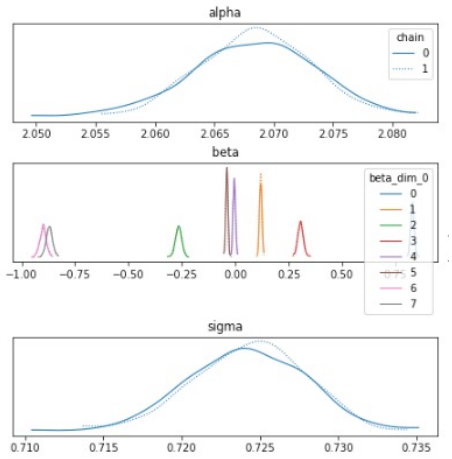


Figure 14: 500 samples

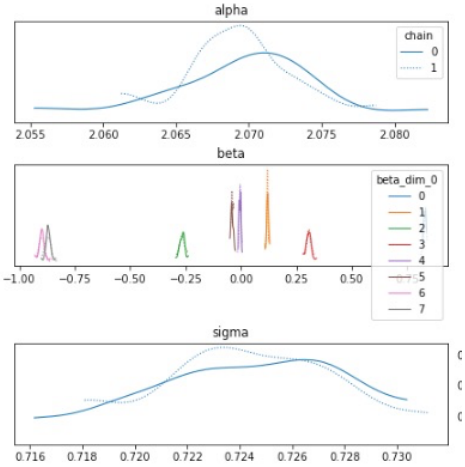


Figure 15: 50 samples

	Mean	SD
alpha	2.07	0.00
beta	0.83	.01
sigma	0.72	0.00

Table 6: 50 samples

From the tables the values of the mean and SD have not changed significantly however by looking at figure 14 and 15, we can indeed conclude that reducing sampling size does negatively impact the posterior distribution. This is because there is a more divergence between the two chains when sample size is reduced.

2.4 CART Decision trees

A CART decision tree is a type of algorithm that is used to build decision trees for classification and regression tasks. It uses a greedy approach to split the data into smaller subsets based on the input feature values, with the goal of minimizing the Gini impurity of the resulting subsets. CART decision trees have several advantages, such as their speed and efficiency, their ability to handle different types of data, and their interpretability.

2.4.1 Optimising hyperparameters

I first had to import the California dataset then I had to split it into training and testing sets. Once this was done, I fitted the training data to the Decision Tree Regressor to get initial values for training and testing accuracy, which were as follows:

```
Training set accuracy: 1.0
Test set accuracy: 0.6398164244654199
```

This shows a lot of overfittings as we have a high training accuracy but very low testing accuracy. To fix this I used GridSearchCV to find the best hyperparameters for the model that would provide the best training and testing accuracy. The hyperparameter that has the biggest effect on performance is th e max depth due to the fact with more depth there is more learning occurring.

We can see this represented on a 3D plot

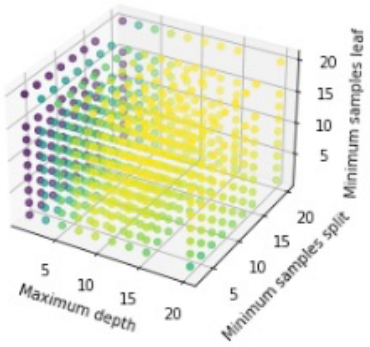


Figure 16: Best parameters

From the plot we can see that the points shaded in yellow are the best parameter values for the model to increase its performance

2.4.1.1 Training time

To see how the hyperparameters affected training times, I plotted the following graphs to visualise how the value of each parameter affected training time. From figure 17 we can see that as the max depth increased so did the training time, however it is the opposite for min_samples_split and min_samples_leaf (figure 18 and 19 respectively)

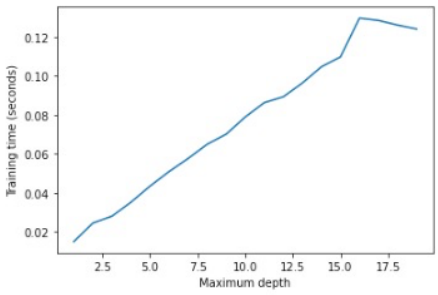


Figure 17: Max depth

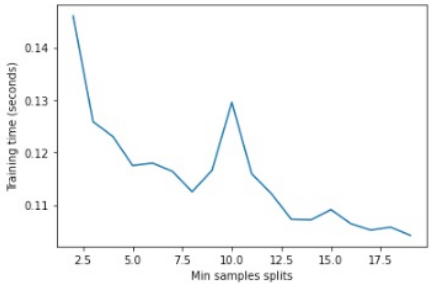


Figure 18: min_samples_split

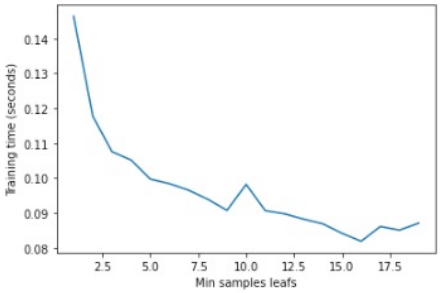


Figure 19: min_samples_leaf

2.4.1.2 Test set

For the test set the best parameters produced these results:

```
Total training time: 0.09 seconds
Training set accuracy: 0.8199284692631424
Test set accuracy: 0.7469826024292674
```

This shows that the model is no longer overfitting thus the hyperparameters of the model have bee n optimised.

2.4.1.3 Decision tree vs Linear Regression

A decision tree may be a better choice than linear regression in situations where the data has a non-linear relationship, contains outliers or missing values, or requires a model that is easy to interpret and explain. The best choice between these two models will depend on the specific c haracteristics of the data and the goals of the analysis.

2.4.1.4 Incorrect classification

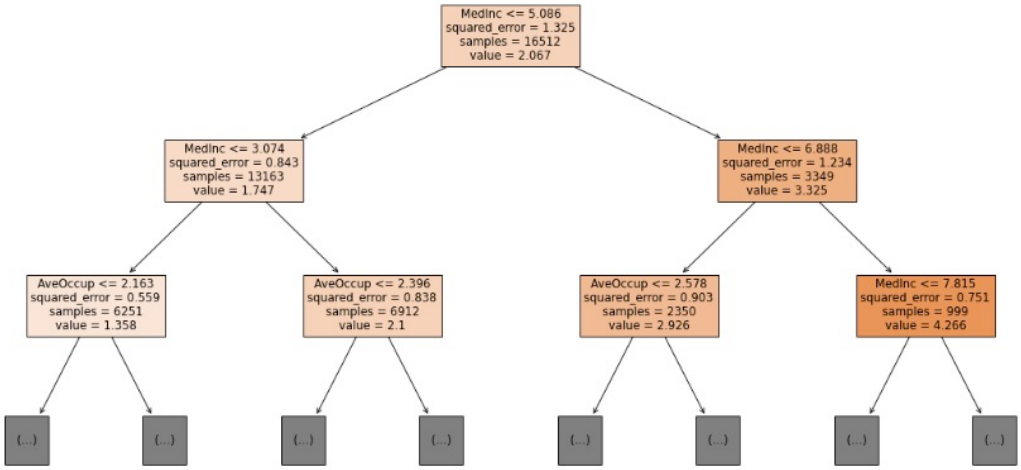


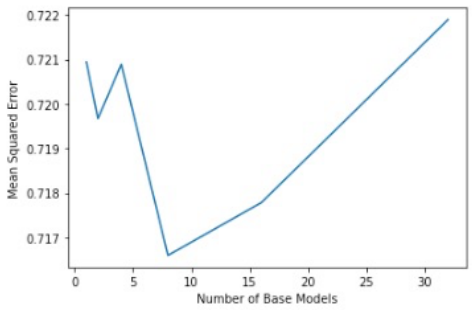
Figure 20: Decision tree

The tree is built using a top-down, greedy search through the space of possible branches with no backtracking. Thus, it is possible it can classify a datapoint incorrectly

2.4.2 Ensemble Methods

Stacking is a method that combines the predictions of multiple base models using a meta-model. It offers several benefits, such as improved predictive accuracy, increased robustness to overfitting, and the ability to combine models of different types. Stacking works by training multiple base models on the same dataset and then using a meta-model to combine their predictions on a new dataset. This allows it to capture a wider range of patterns and relationships in the data and improve the overall performance of the ensemble.

2.4.2.1 How is Stacking affected by number of base models



In this scenario with this specific dataset and the two models I decided to use for stacking; as seen in figure 21, as the number of models increase, the mean squared error increases, this means that using the models, decision tree regressor and linear regressor for stacking does not produce a significantly better testing accuracy and thus increasing the models has a negative effect.

Figure 21: Increasing number of base models

2.4.2.2 Testing accuracy

The results I obtained from the stacking model using decision tree and linear regressor are as shown in table 7. From this we can conclude that stacking does indeed improve the testing accuracy compared to decision tree or linear regressor on their own.

Table 7: Comparing different models

Model	Training	Testing
Stacking	0.91	0.72
Decision Tree	1.00	0.62
Bayesian LR	0.6062	0.6061

2.5 Conclusion

A total of 8 machine learning algorithms were used on the Fashion MNIST and California Housing data. Each section looked at the algorithms and applied it to the data. The limitations of these algorithms were also investigated. Moreover, accuracy could be improved with more runtime.