**Principle components** analysis or PCA, is a technique for reducing the dimensionality of datasets. The aim is to increase a databases interpretability while minimizing information loss. This is achieved by creating new uncorrelated variables that successively maximize variance. The dimensionality of the data is 784, consistent of 28x28 pixels. PCA is performed here to understand whether this reduction significantly impacts our information retention or not. a good reduction will aim to keep most of the information while discarding as many dimensions as possible. Here, we learn some quantities about these data namely different components and cumulative variance of the components. These represent the principal components of the data.

PCA was applied to all data and the pca object of the first 10 principles were returned. With this analysis we intend to keep higher variances. The output tells us that the cumulative variance decreases rapidly between first 3 components. Furthermore, to understand how many dimensions would be ideal to retain, Cumulative Variance Ratio per Principal Component is calculated. It can be observed that the first "51" principal components can explain 80% of the variance in this dataset. so, reducing the datasets to 51 dimensions is ideal.

The dimensionality of this data was reduced to 2 components. the data is then visualized as we plot the first component against the second component. Cumulative variance of the two components is printed. Chart, scatter chart

Description automatically generated

The first **principal components** are "19.80980567" and "31.92201614" showing the gap between the two components are high. This tells us that reducing to 2d from 780+ dimensions are not a good idea since all the classes keep clustering into each other and there is no clear separation between the classes and most of the datapoint are lost and we are underfitting our data.

**Gaussian mixture model** is a probabilistic model that is assembled by different gaussian distributions with each having their own mean and standard deviation. They are then used to find probabilistic cluster assignments. Predictions are then scattered into a plot for better visualization. Furthermore, the cluster centroids are represented by a grey X marker within the scatters. Z-score for this model is "-5.047949428065528" with some minor overlapping happening between the classes. Data's standard deviation is "3.995089742715105" and the mean is "0.2550780733852641". Since there is a large gap between the mean and standard deviation, it suggests that our data are scattered and not close to each other. It is worth mentioning that the random states in set to zero hence it is possible that the expectation maximization step has missed globally optimal solution.Chart, scatter chart

Description automatically generatedChart, histogram

Description automatically generated

To calculate how **accurate these predictions** are compared to the true values within the dataset, prediction labels are compared with actual labels. "accuracy()" function takes 2 inputs, compares them and returns an overall similarity percentage. Furthermore, how many different classes our predictions included that were true is also calculated. Accuracy of the prediction are very low, using all the training data (60000), it is only "7.948333333333333%". Moreover, it is observed that classes 3, 4, 8 and 10 were not at all predicted correctly while class 6 had the most correct predictions. Overall, the soft clustering on a 2d reduction does not output adequate predictions as with this reduction, majority of our information is lost. As mentioned before, reducing to 24 dimensions will ensure about 80% retention of information which is recommended.

**ANN** was applied using the multiple layer classifier from sklearn python library. The loss curve for both test and training data was plotted and it was observed that they were both extremely similar. This was expected due to logistic regression's score for both training and testing data being so close to each other. This ANN model does not seem to be overfitting as the loss for both datasets are low. Also, the model's loss curves, do not climb after the initial dip which is a sign that the model is not overfitting. Furthermore, our data is large and was standardized before applying the classifier which reduces overfitting. To understand this model's accuracy better, a confusion matrix is plotted as well as figuring out accuracy of our datasets.

Graphical user interface, application, calendar

Description automatically generatedChart

Description automatically generated

**Scores**, or the accuracy mean, of the model against testing data is 82.7% while it is 98.2% against training data. Our network model performs well on both datasets however the gap between the two datasets seems to be a bit high than ideal suggesting a slight overfitting. Furthermore, from the confusion matrix it is seen that this model made the most mistake when attempting to distinguish between classes 0(t-shirts/top) and 6(shirt), 4(coats) and 2(pullover), 6(shirt) and 2(pullover) and so on. It can be understood keeping in mind the similarities between these classes however the model did very well with over 82% accuracy across all the data points.

Our model's performance is dependent on our **hyperparameters. Optimizing** them will lead to better performance by our model and is a crucial part of machine learning. Ultimately, the goal is for the accuracy score to be as high as possible, with minimal overfitting. “randomizedsearchCV” is utilised to perform hyperparameters tuning. The cross validation split is 3, each having 10 iterations for performance optimization. Initial learning rate and alphas or regularisation term are also optimised. Alpha here refers to regularization term's strength which is divided by the sample size and added to the loss. This parameter controls the weighting of our regularization within cost function to better control overfitting. Note that this took a long time to compute.

From the output of this search, initial learning rate of " 0.00031992671377973844" and alpha of "0.006579332246575682" are observed. A new neural network with these tweaked hyperparameters is modelled to see if the accuracy of our data changes. All other hyperparameters stay unchanged. This ANN, will have 64 layers and 1800 iteration over 3000 datasets just as the previous classifier.Shape, square

Description automatically generated

**The changes done on these hyperparameters** does slightly improve our accuracy against testing data. Our accuracy is 83.1% with the new tuned hyperparameters which is an improvement of 0.4%. The loss curve was plotting to check for overfitting and the results are similar and there is no sign of significant overfitting. For further changes, a new neural network is modelled with more layers and a different solver. Previously, 64 hidden layers and the 'sgd' solver were utilised. SGD or stochastic gradient descent is mostly used because of simplicity and how fast it runs since it is linear. Adams’s solver has faster run time and need lesser parameters for optimisation. For hidden layers, increasing them can lead to higher accuracy however it can be very time consuming and expensive to run as well as risk overfitting.Chart

Description automatically generated

These two changes made the accuracy go up to 83.2%, which is a 0.1% improvement. The loss curve here for both training and testing data set changes slightly as it has more bumps through the descending gradient. Our model might be overfitting slightly more compared to other curves however it is insignificant. For further tuning, we employ a gridsearchCV for tuning “activation method. For a given model, Grid Search is a cross-validation technique used for finding optimal hyperparameters when these are in each grid and predictions are made. Activation method is responsible for deciding whether a neuron will be activated, and its input is important within the neuron network. During the grid search, there are no further improvements to our accuracy. The activation parameter did not improve our performance the adam solver was proven to be the best. 'ReLU' or rectified linear activation is the best which was the default parameter.

**Support Vector Machines** (SVM) are nonparametric model where the number of parameters is not fixed and increase appropriate to the size of our data. The size of both the testing and training dataset will be 5000, the regularisation parameter "c" will be 1 (default) (same as ANN classifier) and kernel used will be the default 'rbf'.

**Leaning curves** shows both validation and training scores as the number of our samples increase with blue being learning and orange as validation. It is a tool for estimating whether we benefit from adding more training data and how much our SVM is affected by variance or bias error. Learning curve here is plotted for accuracy against sample size.

Chart, line chart

Description automatically generatedFor validationcurve, we can see from it whether our model is overfitting or underfitting. Scores for both training and testing datapoints are calculated and plotted against a range of regularisation factors "C". High training scores paired with low validation score, suggests the SVM is overfitting. For this, function (plt\_valid\_curve) is introduced taking in an estimator (SVM in this case) and training and target samples, to plot validation curve of regularisation against accuracy of the training samples. **The SVM model's score** against our testing dataset is 84.1% while it is 90.0% against training set. From the plot output, the SVM model could not benefit from additional training data as the gradient for testing rapidly decreases as we reach all 5000 training size. So the testset error is not likely to decrease furthure than 12%. As we reach maximum training size, the error gap between testing and training data is 6% which is a not too large, suggesting the model does well. Furthermore, with small datapoints, the gap between training and test set's performance is high. This can be addressed by increasing the regularisation value and the number of training samples, in which here we increased the input size. As the number of data increases the error for both testing and training dataset decrease showing that our model is performing well. Moreover, the accuracy of the model increases as the number of data points increase, the lowest value with 500 datapoints is around 76% while the highest with 4000 datapoints is about 85%.Chart, line chart

Description automatically generated

From the validation curve, it is observed that both training score and our cross-validation score are very similar. suggesting that our model is working fine and is not overfitting or underfitting. In general, for small sample data, training score for SVM is bigger than validation score. This can be addressed by adding more sample data. As the regularisation term increases with our sample size, the score for both training data and validation increases which means our model is performing well.

“GridsearchCV” is utilised to tune our **hyperparameters**, the regularisation term "C" and different kernels are fitted 300 times with 3 cross validations for 5000 datapoints. The kernel, is "Linear" since it is the most basic and the fastest type of kernel and works best when there are a lot of features present like in fashion dataset.

From this search, our best regularisation factor is "0.20202020202020202" and "linear" as the best kernel. These hyperparameters improve our training accuracy to 95% from 90% (using rbf kernel, and c=1) while they become less accurate against testing data with 82.2% accuracy from 84.1% suggesting a very slight overfitting.

The “kernel” parameter and how sensitive SVM is to them is tested by plotting learning curve and printing their accuracy. The tuned regularisation parameter (0.20202020202020202) and gamma 0.01 for all kernel types will be utilised. (linear, Sigmoid, rbf)

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generatedChart, line chart

Description automatically generated

From the plots and final accuracy score, we can conclude that the best kernel is the linear kernel since it has a high accuracy score and better validation. The sigmoid kernel is the worst performing out of all since it has a very low score of 41.5% and loses accuracy as we increase our input samples. This kernel improves in accuracy with small sample size and maxes at 60% with about 1200 datapoints but beyond that rapidly declines in score. "rbf" kernel does well with both score (79.6%) and learning curve however it is outperformed by linear model (82.2%) by a margin of 12%. In conclusion, the best kernel is the "linear" and for it the regularisation factor "c" of "0.20202020202020202" gives us the best performance.

**SVM vs ANN:**

These two classifiers both have comparable accuracy and perform well on the given data. These parametric classifiers both can accommodate non-linear data with usage of Kernels in SVM and activation in ANN. To the contrary, ANN has the edge over SVM due to higher accuracy and overall better performance. Given enough data, and training, ANN usually outperforms SVM and it was seen in our case. Given we have enough computational power, if the input samples and number of layers within the ANN were to increase ANN will far outperform SVM. This is due to additional complexity ANN offers with hidden network layers. However, we don’t always have infinite computational power. SVMs are in general very fast to train since they only use a subset of data and rely on support vectors for decision boundaries. Large number of samples for ANN will required long time commitments making them a very expensive classifying method. This is further shown below with a bar chart, comparing time taken to train 5000 samples for both classifiers. SVM took only 1.3 seconds while ANN took 39.2 seconds to train, and it is expected to increase even further as the training size increases. This shows us that tuning our parameters and re-training a NN is far more expensive.

Chart, bar chart

Description automatically generated

In conclusion, ANN performs better than SVM since it has higher accuracy against testing data however, they are time consuming to train. Depending on available resources, time and accuracy needed, we can use either classifier to train our data. In this case, using only a subset of data available to us mean time and computational power were not an issue therefore using ANN is the preferred method of classifying.

**Task 3: Bayesian linReg: 2.3**Chart, scatter chart

Description automatically generated

From the scatter plot of all datapoints we observe that lower longitude values paired latitude values lower than approximately 39, will have the higher median values. This can be understood since California is a warmer climate location and the locations are near or next to the beach and seaside are more valuable. From this, an initial theory can be that as the houses get closer to the seaside, the higher the house values will be.

**Data transformation** is a technique to make the data more in-line with others and/or have it been more organised. They are also simpler for computers to use and make it easier to do various analysis such as plotting. Since we are dealing with linear regression, linearity is the objective so transforming the data is a common practice

Using "PyMc", an approximate **normal distribution** with our scaled data is produced. Each features data are used then used to define 6 different normal priors using their mean and standard deviation and a whole uniform standard deviation is defined with upper max of 20 and lower min of 0 for getting the likelihood function. Sample are taken from these priors and the posterior distribution of this model is produced. There are ten cores and two chains. Sampling method chosen here is "no U turn sampler".Chart, line chart

Description automatically generated

From the output, we see posterior distribution given our priors and likelihood. The mean for each prior is near 0 except "AveRooms" and "AveBedrms" whcih both are -0.001, and standard deviation: ['0.045','0.034','0.085','0.080','0.034','0.032']. For all our data, the mean is '4.58' with standard deviation of '0.022'.

**2.3.4:**

These distributions summarise uncertain quantities in Bayesian analysis and provides new information from our data. The R-hat value is an indicator that signals if we have achieved stationarity with our chains. Here, 2 chains are used and the r-hat value for all is "1.0". A good value for this is between "1.2-0.9" therefore these distributions are a decent approximation since these outputs are evidence that all chains have converged. The "ess\_bulk" value is used to understand sampling efficiency for the bulk of posterior. For features: "HouseAge", "AveOccup" and "Population" this value has a large gap with other features, meaning that we will need higher sampling points to have more effective rank normalisation in split chains and there is not much that can be extracted from them. The monty carlo standard error mean for each prior is near zero with standard deviation being zero at all time. This shows that the accuracy of the chains is high and the model is performing well.

**50 sample**:Chart, line chart

Description automatically generated

**500 samples**Diagram

Description automatically generated**2.3.5:**

For all 3 different sample distributions, the R-hat value for 50 samples, range between "1.00-1.05" suggesting although each chain did not fully replicate the posteriors, they did converge to an equilibrium. The posterior distribution for both 50 and 500 samples look different and less certain. This is due to the gradient of the slope for each prior suddenly and constantly changing. This is far more visible with 50 sample points and less so with 500. Reasoning for it is due to available sample points as with the full dataset, the distributions look more certain. Furthermore, "ess\_bulk" and "ess\_tail" values for 50 datapoints indicate that using this small sample size is not ideal. The same can be observed with 500 sample points. Finally, the monte Carlo standard error for both 50 and 500 sample size are relative to full dataset high, meaning the model's performance is not ideal for either sample sizes. These observations were expected as the accuracy of a model would vary based on how much evidence is available for initiating modelling.

**2.4.1.1: Trees**

CART or classification and regression tree is a predictive model, which predicts different variable's outcome values, with respect to other matters. Its algorithm starts by taking the best split point for each input, then takes the splits and further finds another split. Then split a chosen input based to the new best split point. This process continues until there is no further best split is possible. The first split is accomplished by utilising a threshold value of an attribute, then the nodes are split into sub-nodes. The sub-nodes are further split based on two different variables, best attribute and threshold value.

**2.4.1.2: Hyperparameter Tuning**

Parameters criterion, splitter, max\_depth, min\_samples\_split, min\_samples\_leaf, and min\_weight\_fraction\_leaf is tuned using grid search. For optimisation, only 4000 datapoints were used in the search.

From the output, best accouracy was 50% with parameters, absolute\_error as cirterion, 25 as max\_depth, 1 as min\_samples\_leaf, 0.2 as min\_samples\_split and best as splitter. To understand what hyperparameter this model is sensetive towards, the accuracy of the model with all default parameters will be obtained and compared to all other models. For all other models, all hyper parameters except the one being tested will be left as default. Chart, bar chart

Description automatically generated

From the bar chart, it is observed that the hyperparameters, criterion when set to "poisson" and splitter when set to "random" hurt the accuracy of the model the most. The model is most sensitive towards the "min\_weight\_fraction\_leaf". This parameter improved the accuracy relatively high compared to the other parameters.

**2.4.1.3**

For comparison, each model's runtime is collected when only specifically changning one hyperparameter. For example, when changing criterion, all other hyper parameters will be set to their previously tuned value. A bar chart is used to understand which performs the fastest. Chart, bar chart

Description automatically generatedChart, bar chart

Description automatically generated

From these plots, it is shown that the hyperparameter that most impacts our runtime is min\_samples\_leaf. Criterion has the least effect in the runtimes as there is no difference between four different inputs. The parameter “min\_sample\_leaf” takes the highest runtime while at the same time better improving the accuracy of the model however, the runtime was not too high so tuning this parameter would not be an issue. As for the others, tree depth and weight fraction parameter do not offer too much while taking a relatively long time. Criterion and splitter do not affect the runtime much hence using the best parameters there would be ideal.

**2.4.1.4**

The score of each model against the testing dataset is compared with each other. Default tree is considered as a base line, while tuned “should” have the best accuracy. The interesting comparison is seen between the models "tree\_tuned" and "tree\_Wf\_leaf" since the single tuned parameter provides more accuracy to our model than the fully tuned version. This can be due to the GridsearchCV being only ran on only a subset of all our data, as well as tree model being overfit. Overall, the best accuracy is 53.6% for "tree\_wf\_leaf" model and the worst is for "tree\_random".

Decision trees should be used when dealing with non-linear data. This dataset the data is quantitative, continuous, with many features present so linear regression will perform better than decision trees, however if we have categorial independent variables, decision trees are a better choice.

**Ensemble methods**

Decision trees tend to overfit and be sensitive to noise and a way of addressing them is using the bagging method. **Bagging** utilises different samples of training set, to train several different models with replacement, then it combines predictions by taking the mean prediction of each model. This reduces the variance in predictions using the additional data created. Usually applied when the classifier is unstable and has high variance. A maximum of 3000 models are created using the bagging ensemble method and their accuracy is being compared and the length of the sample size (20640). The plot below, compares the model’s accuracy as the number of model’s increase.Chart, scatter chart

Description automatically generated

The best accuracy against the testing data set was 0.577 or 57.7%. This is higher than our singular tuned decision tree regressor before and it is improved by 4.1%. Still the performance of the decision tree is not improved massively. The plot shows no significant relation between these two parameters as it is fairly scattered around. The model's accuracy does not necessarily improve with increased number of models. Overall, this ensembles method's performance is better than the previous singular decision tree employed. However, these two models, are both outperformed by the Bayesian linear regression.