1 Introduction

This paper introduces various machine learning methods for supervised and unsupervised learning. The paper will cover the following machine learning algorithms: PCA, K-means, ANN, SVM,Bayesian linear regression, Random forest regression, and stacking. Section 2,3,4, and 5 uses MNIST dataset and Section 6,7, and 8 uses California housing data set. Before delving into the algorithms, there are some concepts to understand.

1.1 Supervised vs Unsupervised Learning

- Regression (supervised) is a problem of predicting successive value based on features of a certain data. The data and label are provided: Bayesian linear regression and random forest regression
- Classification (supervised) is a problem of classifying given data into a set of categories. The data and label are provided: ANN and SVM
- Classification (unsupervised) is a problem of clustering similar data into certain class/label whereas labels are not provided: K-means

1.2 Overfitting vs Underfitting

Overfitting is a phenomenon which the model fits the training data too well, eventually failing to explain new data.
Underfitting is a phenomenon which the model does not sufficiently describe the training data, eventually failing to predict the testing data as well.

1.3 Gradient descent algorithm

Gradient descent algorithm is the algorithm used to reduce the cost J. It aims to find the global/local minimum and the corresponding weights θ_0 where the derivative of the cost function J is zero.

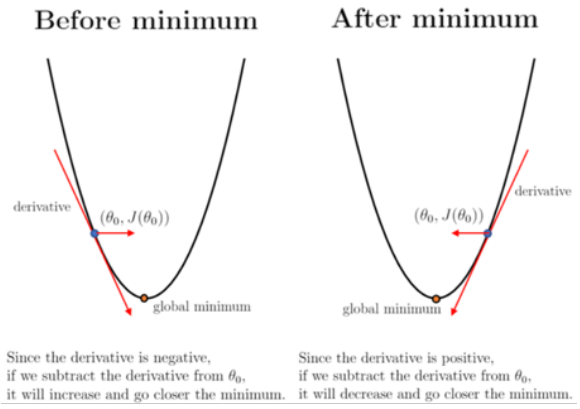

Figure 1: Gradient descent algorithm [1]

$$\theta_0 := \theta_0 - \alpha \frac{\partial J(\theta_0)}{\partial \theta_0}$$

where α is the learning rate and J is a cost function. The above example dealt with one θ however could be dealt with more. For example, the cost function $J(\theta_0, \theta_1)$ looks like figure2:

$$\theta_0 := \theta_0 - \alpha \frac{\partial J(\theta)}{\partial \theta_0}$$

$$\theta_1 := \theta_1 - \alpha \frac{\partial J(\theta)}{\partial \theta_1}$$

where θ_0 and θ_1 gets updated simultaneously. This topic will be further discussed in section4.

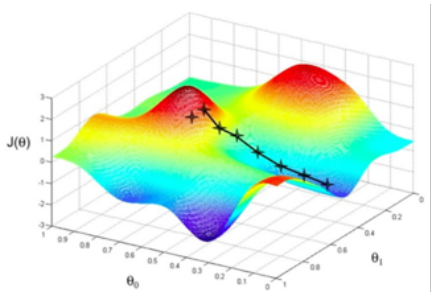

Figure 2: Gradient descent algorithm example. The cost funtion is not convex [1]

2 Principal Component Analysis

The PCA algorithm transforms from high-dimensional data (original) into low-dimensional data by finding the principal axis that preserves the variance of high-dimensional data as much as possible. The principal axis can be found through feature selection by finding the eigen-value of the original data. Preprocessing which make the data mean equal to zero and standard deviation equal to 1 is necessary when performing PCA because the principal component are chosen based on the standard deviation of the features. The higher the standard deviation of the specific feature, the more likely it is to be selected as a principal component.

2.1 PCA algorithm

Step1: Perform feature scaling and mean normalisation to the data
Step2: Find covariance matrix of the pre-processed data generated by Step1
Step3: Find eigen-values and eigen-vectors by using covariance matrix
Step4: Generate a new matrix with selected eigen-value and its corresponding eigen-vecotrs.
Step5: Perform PCA with original data and the matrix that has been computed by Step4.

2.2 Applying PCA algorithm to MNIST data-set

Step1, perform feature scaling/normalisation by using $sklearnStandardScalar()$.
Step2, find covariance matrix of the scaled data.
Step3, find eigen-values and eigen-vectors and list eigen-values in descending order with corresponding eigen-vecotrs. (the below plot is just to help with the visualisation of eigen-values)

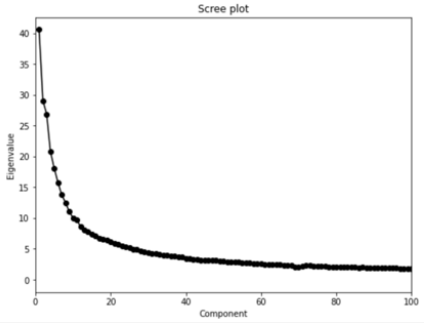

Figure 3: The actual value of eigen value of each component

The first two eigen-values/principal components (PC) explains 9.68% by calculating the following equation. (5.64%

for the first PC and 4.04% for the second PC)

$$\frac{eigenvalue_0 + eigenvalue_1}{\sum_{n=1}^{784} eigenvalue_i} \times 100$$

Step4, the shape of the generated new matrix with eigen-vector is: (784×2), each column containing eigen-vector. The first column contains eigen-vector with the biggest corresponding eigen-value. The second column contains eigen-vector with the second biggest corresponding eigen-value. The eigen-value/principal components could be visualised by figure4.
Step5, the reduced dimension data could be computed by the following equation:

$$X \cdot T = Z$$

where X is the scaled original data, T is the matrix acquired at **Step4**, and Z is the reduced-dimension data. Z could be visualised by figure5. Note that by calculating

$$Z \cdot T^{-1} = X$$

it is possible to acquire the scaled original data. By looking

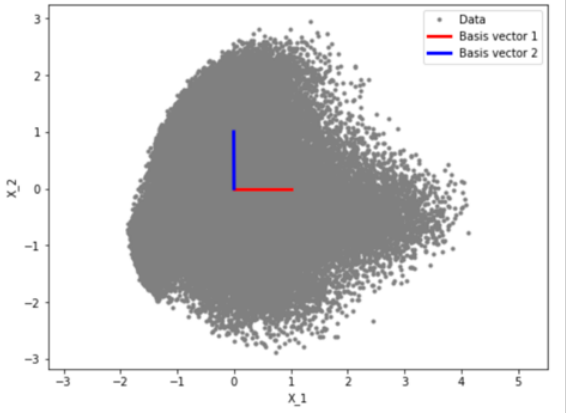

Figure 4: X_1 is the first principal axis and X_2 is the second principal axis

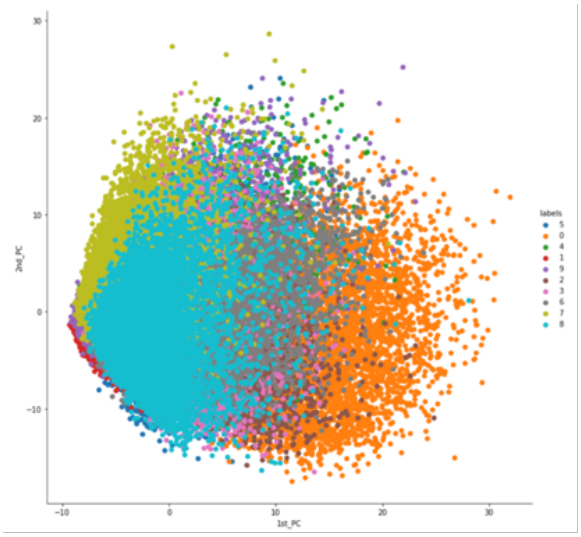

Figure 5: PCA transformed data on MNIST

at figure 5, the boundary of each digits is not clearly visible except for some digits such as 0, 7, 8, and 1 although PCA is performed. In the next section, the paper will discuss how the reduced dimension data could be clustered with the Kmeans algorithm.

3 Kmeans

Kmeans algorithm is a one of the representative algorithm among unsupervised learning. The algorithm separates similar data into the same group and dissimilar data into different groups.

3.1 K-means algorithm [2]

Step1: Randomly initialise cluster centroid. The model tries to classify MNIST data into 10 clusters; therefore, $K = 10$.
Step2: Perform cluster assignment step. Assign every training example $x^{(i)}$ to its closest cluster's centroid. This can be expressed in an equation:

$$c^{(i)} := j \text{ that minimises } \|x^{(i)} - \mu_{(j)}\|^2$$

where $c^{(i)}$ is the cluster's index which is closest to $x^{(i)}$. $c^{(i)}$ can be $c^{(1)}$ . . . $c^{(70000)}$ since there are 70000 samples and each $c^{(i)}$ is equal to a certain cluster index value (1..10). And $\mu_{(j)}$ is the coordinates of $j-th$ centroid. $\mu_{(j)}$ can be $\mu_{(1)}$ . . . $\mu_{(10)}$ since there are 10 clusters. Cluster assignment step is also called as Expectation step $(E - step)$.
Step3: Perform move centroid step. Based on the samples where each sample is designated to a certain cluster performed by Step2, update the coordinates of the cluster. This step is also called as Maximization step $(M - step)$. This can be expressed as an equation:

$$\mu_k := \frac{1}{|C_k|} \sum_{n=1}^{70000} x^{(i)}$$

where μ_k is $k-th$ cluster centroid coordinates and $|C_k|$ is the number of samples that assigned to $k-th$ cluster. Considering the equations from Step2 and Step3, the optimisation objective of the cost function J is:

$$J(c^{(1)}..c^{(70000)}, \mu_{(1)}...\mu_{(10)}) = \frac{1}{70000} \sum_{n=1}^{70000} \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

$$\min_{c^{(i)},\mu_k} J(c^{(1)}..c^{(70000)}, \mu_{(1)}...\mu_{(10)})$$

Step4: Fix μ and optimise c like Step2 and Fix c and optimise μ like Step3 until the centroid converges (does not move). This is so called as EM (Expectation-Maximisation) algorithm.

3.2 Result

The result is shown on figure6. Note that the cluster index is not related to the actual digits since Kmeans is unsupervised learning and assigns cluster randomly. However it was possible to find the most frequent digits in each of the cluster and this can be observed by Table1.

CL	MFD	CL	MFD
0	0	5	7
1	1	6	2
2	5	7	6
3	0	8	2
4	7	9	4

Table 1: CL: Cluster label, MFD: Most frequent digit

By looking at figure6 and Table1, it is obvious that Kmeans algorithm did not perform well to classify data. By comparing

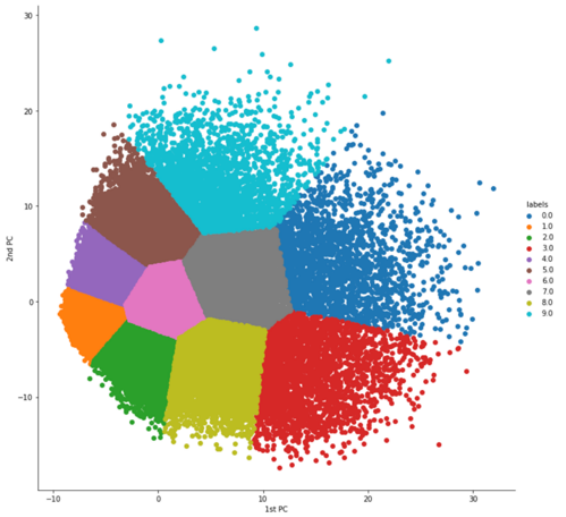

Figure 6: Result of Kmeans algorithm on reduced 2D MNIST

figure5 and figure6, the algorithm was able to classify digits such as cluster 0 and 1; however, the majority of the cluster could not distinguish digits well. This can be also observed by looking at the following cost value curve depending on the total cluster number. It is obvious that the function still decreases dramatically after K=10 considering 1e6 on the top left. The error rate(cost) will reach zero when there are 70000 clusters.

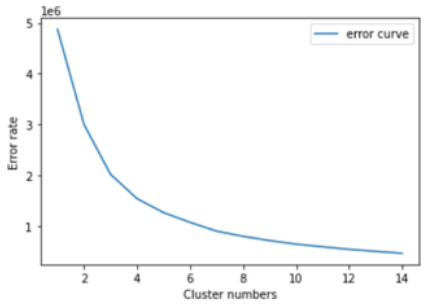

Figure 7: Error rate(cost) depending on the number of clusters

3.3 Limitations of Kmeans algorithm to classify MNIST dataset

Firstly, Kmeans algorithm can end up converging to different solution depending on the random initialization (which performed in Step1 in section3.1), eventually ending up at local optima. Secondly, it was not possible to classify MNIST dataset by using Kmeans (high error rate observed by figure7). Therefore, other machine learning algorithm should be implemented to classify the data better: ANN and SVM

4 Artificial Neural Network

Artificial neural network, also known as neural network (NN) is a model that imitates neurons in the brain. The neural network model is inspired by the fact that dendrite gets the input and axon outputs a spike to the other neurons.

4.1 ANN algorithm [2]

The training of the NN with MNIST 70000 samples could be divided into a few steps by following (training:testing split = 8:2 ratio).

Step 1: Compute forward propagation to get $h_\theta(x^i)$ where θ is a weight matrix and x^i is $i-th$ training example.

Step 2: Compute cost function $J(\theta)$

Step 3: Compute back-propagation to compute $\frac{\partial}{\partial\theta}J(\theta)$

Step 4: Use gradient descent with $\frac{\partial}{\partial\theta}J(\theta)$ to minimize cost function $J(\theta)$.

4.2 Deeper understanding of each step

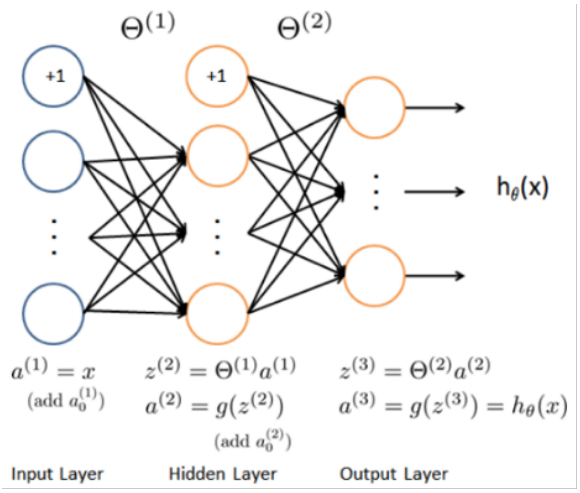

Figure 8: Neural network used for MNIST data set classification [2]

Step1: Take a look at figure8. A single sample of the MNIST data set is displayed as a 28×28 matrix. There are a total of $70,000$ samples; thus, the shape of the total sample is $70000 \times 28 \times 28$. Before the input enters the neural network, it is necessary to be flattened ($70000 \times 28 \times 28 \longrightarrow 70000 \times 784$) which then could be expressed by a total of 784 features per sample. The shape of the input layer is $(784+1)$. $+1$ is the bias term. The hidden layer was set to 256 in the model, and the output (final) layer should be 10×1 because the model currently classifies numbers from 0 to 9. The function h is a relu function and g is a sigmoid function. $\theta^{(1)}$ has a shape of $256 \times (784+1)$ and $\theta^{(2)}$ has a shape of $10 \times (256+1)$. The model starts training with random weights (θ).

Step2: Regularization is needed to prevent over-fitting/under-fitting. The λ term adjusts theta to prevent the model from over-fitting or under-fitting. If the λ is too large, it may lead the model to be under-fitted. If the λ is too small, it may lead the model to be over-fitted. Therefore, choosing appropriate λ term is essential. The regularization hyper parameter (λ) is chosen to be 1.11 which is acquired by using *sklearn.model_selection RandomizedSearchCV*. The cost function with regularization is:

$$J(\theta) = \frac{-1}{56000}\sum_{i=1}^{56000}\sum_{k=1}^{10}[{y_k}^{(i)}log((h_\theta(x^{(i)}))_k)+ (1-{y_k}^{(i)})log(1-(h_\theta(x^{(i)}))_k)]+ \frac{\lambda(=1.11)}{2\times 56000}\sum_{l=1}^{2}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}({\theta_{ji}}^{(l)})^2$$

where s_l is the number of units in layer l except bias term. $s_1 = 784$, $s_2 = 256$, $s_3 = 10$, and y_k is either 0 or 1. If current training example matches with class k, then $y_k = 1$. If current training example does not match with class k, then $y_k = 0$. The cost function gives a penalty/cost for the wrong prediction. The goal is to minimize the penalty.

Step3: The neural network minimizes the cost by back-propagation. For each training example, firstly, find error δ of the last layer by computing $\delta^{(3)} = a^{(3)} - y$ where a and y

are a column vector 10×1 ($h_\theta(x) = a^{(3)}$). Secondly, By using $\delta^{(3)}$, compute the previous the error term $\delta^{(2)}$. It is possible to find previous layer's error by a chain rule ($\delta^{(1)}$ is not needed because the first layer is the input layer; thus, only $\delta^{(2)}$ and $\delta^{(3)}$ are needed to be computed):

$$\delta^{(l)} = ((\theta^{(l)})^T \delta^{(l+1)}) g'(z^{(l)})$$

$$\delta^{(l)} = ((\theta^{(l)})^T \delta^{(l+1)}) a^{(l)}(1 - a^{(l)})$$

Thirdly, The derivative of J can be obtained from the following formula.

$$\Delta_{ij}{}^{(l)} := \Delta_{ij}{}^{(l)} + \alpha_j{}^{(l)} \delta_i{}^{(l+1)}$$

$$(\Delta_{ij}{}^{(l)} = 0 \text{ at the beginning})$$

$$D_{ij}{}^{(l)} := \frac{1}{56000}(\Delta_{ij}{}^{(l)} + \lambda \theta_{ij}{}^{(l)}), \text{ if j} \neq 0$$

$$D_{ij}{}^{(l)} := \frac{1}{56000}\Delta_{ij}{}^{(l)}, \text{ if j} = 0$$

regularisation (λ) is not needed for the bias term (j=0)

$$D_{ij}{}^{(l)} = \frac{\partial J(\theta)}{\partial \theta_{ij}{}^{(l)}}$$

where $\alpha_j{}^{(l)}$ denotes $j-th$ node in $l-th$ layer, $\delta_i{}^{(l)}$ denotes $i-th$ node in $l-th$ layer, and $\theta_{ij}{}^{(i)}$ denotes $i-th$ row and $j-th$ column element in $l-th$ parameter(weight) matrix. It is important to note that all θ get updated simultaneously as mentioned earlier in section 1.3.

Step4: Use the gradient descent algorithm (section1.3) to find θ that minimizes the cost function $J(\theta)$. Note that this should be calculated for each training example as mentioned at the beginning of Step3.

4.3 Learning curve and performance

The training data overfitted the model; however, performed well (Table2 and figure9). The performance could be better by choosing other hyper-parameters.

Training error	Testing error
0.999	0.976

Table 2: Comparing training error and testing error. *activation* = *relu*, *alpha(regularizer)* = 1.11, *learning_rate_init* = 0.000167, *solver* = *adam*, *hidden_layer_size* = 256

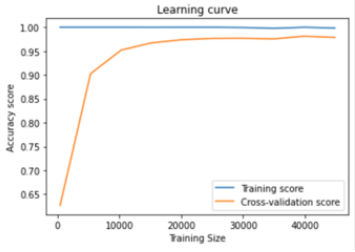

Figure 9: Training and validation curve depending on the training(sample) size

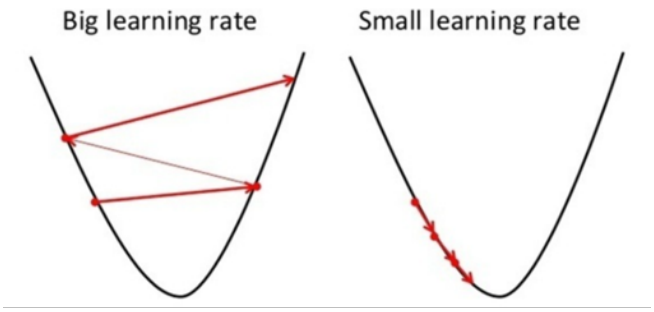

Figure 10: Both too big and too small learning rate would not reach to optimal value [1]

Learning rate	training set	testing set
0.0001	0.997	0.980
0.001	0.985	0.966
0.01	0.307	0.315
0.1	0.098	0.097

Table 3: training and testing accuracy depending on the learning rate

4.4 Hyper-parameters and performance

Learning rate: Learning rate and performance is closely related. Accuracy tends to decrease as the learning rate increases. Both too small or too big learning rate would not reach the optimal point or fall into local optima (figure10 and Table3).

Regularizer: It is possible to change regularization parameter by using *alpha* in *sklearn MLPClassifier*. The accuracy on training and testing set decreased as the alpha value was too big or too small. This is because appropriate alpha value was 1.11 acquired by *RadomizedSearchCV*. This can be clearly explained by the following graph:

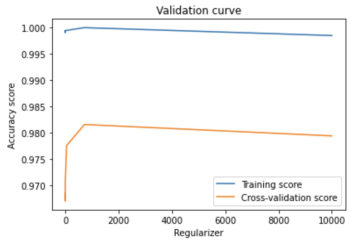

Figure 11: Accuracy score depending on regulariser α

Hidden layer size: The hidden layer size could be changed to another value such as 128 or 512. The other hyper-parameters such as the regulariser and the learning rate would be changed if hidden layer size changes. In fact, it is possible to put more than 1 hidden layer between input layer and output layer such as 5 hidden layers; however, it might lead to over-fitting the training data.

4.5 Limitations of ANN

One of the major limitation of ANN is that it greatly depends on the learning rate and the solver since ANN's cost function is not a convex function (which implies that there are many local optimas. This can also be observed by figure2). In addition, there are various ways to do perform gradient descent algorithm: Batch gradient descent, stochastic gradient descent (SGD), and Mini-batch gradient descent [2]. Therefore it is always important to the find optimal learning rate

and the appropriate solver (+ other hyper-parameters) when training the data with ANN.

5 Support Vector Machine (SVM)

SVM is a model that defines a decision boundary for classification; thus, when a new sample is observed, the algorithm could perform the classification task by checking which boundary(class) it belongs to. (If there are two classes it uses one vs one. If there are more than two classes it uses one vs rest) SVM is trained so that the direct decision function maximises the margin (the generalization ability) enabling the model to perform well on both the training and the testing data. The algorithm is also based on the statistical learning theory [11]. This section will deeply analyse how this is done in SVM (train:test = 8:2).

5.1 SVM algorithm [11]

1: Linearly separable case, linear decision boundary
2: Linearly non-separable case, linear decision boundary
3: Kernel method, non-linear decision boundary.

5.1.1 Linearly separable case

SVM tries to maximise the margin over the training set like figure12 where margin is the distance between $w^T \cdot x + b = 1$ and $w^T \cdot x + b = -1$. The margin could be calculated by the following equation:

$$\begin{aligned} margin &= distance(x^+, x^-) \\ &= \|x^+ - x^-\|_2 \\ &= \|(x^- + \lambda w) - x^-\|_2 \\ &= \lambda \|w\|_2 \end{aligned}$$

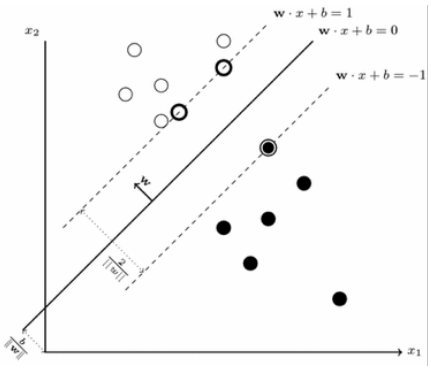

Figure 12: SVM linearly separable case [3]

where x^+ indicates the sample on the $w^T \cdot x + b = 1$ plane and x^- indicates the sample on the $w^T \cdot x + b = -1$ plane. And let λ the width of the parallel translation in the w direction then $x^+ = x^- + \lambda w$.
λ can be calculated by plugging $x^+ = x^- + \lambda w$ to $w^T \cdot x^+ + b = 1$, $w^T(x^- + \lambda w) + b = 1$. Therefore the margin can be expressed in an equation:

$$\lambda = \frac{2}{w^T w}$$
$$margin = \frac{2}{\sqrt{w^T w}}$$

The goal of the algorithm is to maximise the margin (find w); therefore, could be finally expressed like:

$$\max margin \propto \min \frac{1}{2} w^T \cdot w = \min \frac{1}{2} \|w\|_2^2$$

There is also a constraint which is

$$y_i(w^T \cdot x_i + b) \geq 1$$

where y_i is either ≥ 1 or $\leq -1$by observing figure 12. The objective function is quadratic and constraint is linear which implies that this is a quadratic programming. This indicates that it is a convex optimisation leading to one existing globally optimal solution.

$$\text{objective function} : \min_{w,b} \frac{1}{2} \|w\|_2^2$$
$$\text{constraint function} : y_i(w^T \cdot x_i + b) \geq 1$$

By using Lagrangian multiplier, the functions could be changed to Lagrangian primal and the equation could be expressed as the following:

$$\max_{\alpha} \min_{w,b} L(w, b, \alpha) = \frac{1}{2} \|w\|_2^2 - \sum_{i=1}^{56000} \alpha_i (y_i(w^T x_i + b) - 1)$$

$$\text{subject to } \alpha_i \geq 0, \text{ i=1,2,..}$$

By solving $\min_{w,b} L(w, b, \alpha)$

$$\frac{dL(w, b, \alpha)}{dw} = 0, w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

$$\frac{dL(w, b, \alpha)}{db} = 0, \sum_{i=1}^{n} \alpha_i y_i = 0$$

and the primal equation could be simplified into:

$$\max_{\alpha} \sum_{i=1}^{56000} \alpha_i - \frac{1}{2} \sum_{i=1}^{56000} \sum_{j=1}^{56000} \alpha_i \alpha_j y_i y_j {x_i}^T x_j$$
$$\text{subject to } \sum_{i=1}^{56000} \alpha_i y_i = 0$$

Finding α is called the dual problem. Again objective function is quadratic and constraint is linear; thus, this implies that there is a globally optimal solution. It is possible to get α by solving the dual problem and the following condition called KKT(Karush–Kuhn–Tucker).

1. stationary: $w = \sum_{i=1}^{56000} \alpha_i y_i x_i, \sum_{i=1}^{56000} \alpha_i y_i = 0$
2. primal feasibility: $y_i(w^T x_i + b) \geq 1$
3. dual feasibility: $\alpha \geq 0$, $i = 1, 2, ..$
4. complementary slackness: $\alpha_i(y_i(w^T \cdot x_i + b) - 1) = 0$

Also, by using complementary slackness, α could be divided into two cases:
1. $\alpha > 0$: $y_i(w^T \cdot x_i + b) = 1$. And x is called support vector that satisfies this equation. x_i lies in plus or minus plane ($w^T \cdot x + b = 1$ and $w^T \cdot x + b = -1$)
2. $\alpha = 0$: $y_i(w^T \cdot x_i + b) \neq 1$. And this indicates that x lies on $w^T \cdot x + b \geq 1$ and $w^T \cdot x + b \leq -1$

$\alpha > 0$ implies that the optimal hyper-plane to separate data only needs support vector x_i (this is import to note) and optimal hyper-plane could be calculated by the following:

$$w^* = \sum_{i=1}^{56000} {\alpha_i}^* y_i x_i = \sum_{i \in SV}^{56000} {\alpha_i}^* y_i x_i$$

Also b^* could be calculated by the following:

$$b^* = y_{sv} - w^{*T} x_{sv}$$

Therefore, by combining all these facts, the optimal hyper-plane is able to classify the data(also can be expanded to one vs rest):

$$\hat{y_{new}} = sign(w^{*T} x_{new} + b^*) = -1 \text{ or } 1$$

5.1.2 Linearly non-separable case

Linearly non-separable case's (also called as soft margin) optimal hyper-plane could be derived similar to linearly separable case (also called as hard margin, section 5.1.1) just by adding slack variable ξ_i.

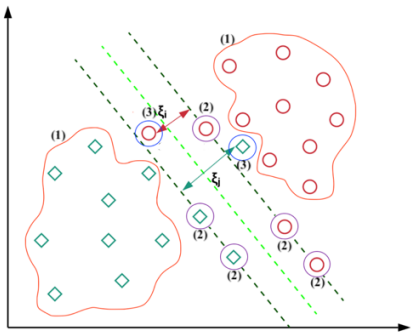

Figure 13: Apply slack variable. Non-linearly separable case, linear decision boundary [4]

Slack variable is just a variable that controls error since some data like above cannot be linearly separable. And this enables to the algorithm to find a optimal hyper-plane although the training data is not linearly separable. This situation can be expressed as an objective and constraint function.

$$\text{objective function} : \min_{w,b} \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{56000} \xi_i$$

$$\text{constraint function} : y_i(w^T \cdot x_i + b) \geq 1 - \xi_i$$

$$\xi_i \geq 0, i = 1, 2..$$

where C is a hyper-parameter that decide trade-off between margin w and training error ξ. Increasing C might incur over-fitting and decreasing C might incur under-fitting. By using the same technique in section5.1, these objective and constraint function could be changed to Lagrangian primal by using Lagrangian multiplier. The primal equation is:

$$\max_{\alpha,\gamma} \min_{w,b,\xi} L(w, b, \alpha, \xi, \gamma) =$$

$$\frac{1}{2}\|w\|_2^2 - \sum_{i=1}^{56000} \alpha_i(y_i(w^T x_i + b) - 1 + \xi_i) - \sum_{i=1}^{56000} \gamma_i \xi_i + C\sum_{i=1}^{56000} \xi_i$$

And solving $\min_{w,b,\xi} L(w, b, \alpha, \xi, \gamma)$

$$\frac{dL(w, b, \alpha, \xi, \gamma)}{dw} = 0, w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

$$\frac{dL(w, b, \alpha, \xi, \gamma)}{db} = 0, \sum_{i=1}^{n} \alpha_i y_i = 0$$

and additionally,

$$\frac{dL(w, b, \alpha, \xi, \gamma)}{d\xi} = 0, C - \gamma_i - \alpha_i = 0$$

the primal equation could be simplified into:

$$\max_{\alpha} \sum_{i=1}^{56000} \alpha_i - \frac{1}{2}\sum_{i=1}^{56000}\sum_{j=1}^{56000} \alpha_i \alpha_j y_i y_j {x_i}^T x_j$$

$$\text{subject to } \sum_{i=1}^{56000} \alpha_i y_i = 0,\ C - \gamma_i - \alpha_i = 0$$

which again transformed to a dual problem where finding α is the goal. α_i 's boundary is $0 \leq \alpha_i \leq C$ since $\alpha_i \geq 0$, $\gamma_i \geq 0$ and $C - \gamma_i - \alpha_i = 0$. Also by KKT condition, the following equation could be derive from complementary slackness:

$$\alpha_i(y_i(w^T \cdot x_i + b) - 1 + \xi_i) = 0,$$

$$\gamma_i \xi_i = 0, \gamma_i = C - \alpha_i$$

Unlike linearly separable case, The α's range could be separated into three cases:

$$\alpha_i = 0 : y_i(w^T \cdot x_i + b) \neq 1, \xi = 0 \ldots (1)$$

$$0 < \alpha_i < C : y_i(w^T \cdot x_i + b) = 1, \xi = 0 \ldots (2)$$

$$\alpha_i = C : \alpha_i(y_i(w^T \cdot x_i + b) - 1) = -\alpha_i \xi_i \neq 0, \xi_i > 0 \ldots (3)$$

and these facts could be visualised like figure13, (2) and (3)'s x is called the support vector.

5.1.3 Kernel SVM

Non-linear SVM which uses kernel method is to train SVM in the feature space, not in the original space where feature space dimension is higher than original space. For example, the original space is 2D and the feature space is 10D. The decision boundary becomes non-linear by finding a linear decision boundary in the feature space and projecting it into the original space. The kernel method is introduced to make this possible and can be expressed by dual formulation,

$$\max_{\alpha} \sum_{i=1}^{56000} \alpha_i - \frac{1}{2}\sum_{i=1}^{56000}\sum_{j=1}^{56000} \alpha_i \alpha_j y_i y_j \phi(x_i)\phi(x_j)$$

$$\text{subject to } \sum_{i=1}^{56000} \alpha_i y_i = 0$$

where ${x_i}^T x_j$ is replaced by $\phi(x_i)^T \phi(x_j) = K < x_i, x_j >$. The kernel uses the property of the inner product therefore it is possible to obtain the optimal hyper-plane without directly changing the dimension of the data (without having to know the explicit form of ϕ). This is called 'kernel trick'.
There are many kernels such as, linear kernel, polynomial kernel, sigmoid kernel, and (gaussian)RBF kernel [10].

$$\text{Linear kernel: } K < x_i, x_j > = x_i^T \cdot x_j$$

$$\text{Polynomial kernel: } K < x_i, x_j > = (1 + x_i^T \cdot x_j)^d$$

$$\text{sigmoid kernel: } K < x_i, x_j > = tanh(a \cdot x_i^T \cdot x_j + b)$$

$$\text{RBF kernel: } K < x_i, x_j > = exp(-\gamma \|x_i - x_j\|^2)$$

There is no standard way to choose a kernel because the different data has different features; thus, all kernels are given a try. The table below indicates that the model slightly over fits the training set with default hyper-parameters given by *sklearn*.

Kernel type/accuracy	training	testing
linear	0.922	0.906
polynomial	0.977	0.961
sigmoid	0.886	0.884
RBF(gaussian)	0.986	0.963

Table 4: training/testing accuracy depending on the kernel

5.2 Performance

Learning curve (figure14) is based on RBF kernel because RBF kernel is performed the best among the 4 kernels. RBF γ parameter defines how far the influence of a single training example reaches [10]. The performance could be better by choosing other hyper-parameters.

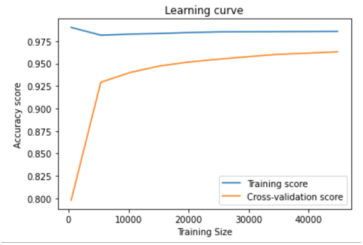

Figure 14: Learning curve of SVM with RBF Kernel

	training	testing
RBF	0.998	0.971

Table 5: The table shows that the model slightly overfitted although SVM has one optimal solution

5.3 Hyper-parameters of SVM

There are mainly two hyper-parameters when training data with SVM kernels: C and γ. Optimal value of γ and C could be obtained by running *sklearn.GridSearchCV*. γ was set to 0.001 and C=10. These two tables below describe the accuracy when changing C (and fix γ) and γ (and fix C) with other default hyper-parameters

C (RBF)	training	testing
10	0.999	0.970
100	1	0.9697
1000	1	0.9697

Table 6: training and testing accuracy depending on the kernel hyper-parameter C

γ (RBF)	training	testing
0.001	0.981	0.961
0.01	0.999	0.848
0.1	1	0.19

Table 7: training and testing accuracy depending on the kernel hyper-parameter gamma

C did not affect the model a lot however, γ value significantly affected the testing accuracy. Therefore, it is highly likely that that choosing the appropriate hyper-parameters is vital when running SVM. C might impact the model a lot on the other circumstance. Optimal hyper parameter could be obtained by using *GridSearchCV* or *RandomizedSearchCV* (quicker but sometimes does not work well)

5.4 Difference between ANN and SVM

1. ANN has many local optimas which indicates that its cost function is not convex whereas SVM is one global optima which indicates that its cost function is convex.
2. ANN's performance is affected from initial weights since there are many local optimas.
3. SVM usually requires less data than ANN because SVM uses only support vectors to make the decision boundary. SVM is also based on a strong statisical theory.
4. ANN needs many parameters: the number of parameter increases significantly by adding hidden layers.

6 Bayesian Linear Regression

Linear regression is frequentists' approach, but Bayesian linear regression is Bayesian's approach. That is, frequentists predict the target by obtaining the model parameters using only the given data, but Bayesian includes the prior distribution of the model parameters, calculates the likelihood from the data, and samples the posterior to obtain the model parameter which can be expressed like [13]:

$$posterior = \frac{likelihood \times prior}{Evidence}$$

where each component can be expressed also like:

$$p(\beta|y, X) = \frac{p(y, X|\beta) \times p(\beta)}{p(y, X)}$$

where LHS of the equation is the posterior of model parameter given the observation, $p(y, X|\beta)$ is the likelihood and $p(\beta)$ is the prior of the model parameters which can be chosen specifically with expert's knowledge or randomly with naively (non-informative). Prior distribution was chosen to be normal distribution because firstly, prior knowledge of California Housing data set is not obtained and secondly, this characteristic will be used in section8. By looking at the formula, it is possible to conclude that as the number of sample increases the model would be more robust which indicates that the uncertainty shrinks: overwhelms the prior distribution. And $p(y, X)$

$$p(y, X) \sim \int_{-\infty}^{\infty} p(y, X|\beta) \times p(\beta) d\beta$$

is quite tricky to acquire since it needs a lot of computations. Therefore, instead of directly solving the formula, Markov Chain Monte Carlo (MCMC) is usually used which enables the model to sample from posterior distribution and to reconstruct the distribution. This implies that the more samples, the more accurate the posterior distribution. No U Turn Sampler(NUTS) Hamiltonian Monte Carlo (HMC) was used for sampling the posterior which uses recursive algorithm to get an sample and stops automatically when it starts to double back and retrace (when the target acceptance is not satisfied) [5]. There are other sampling algorithm besides HMC such as Metropolis-Hastings algorithm. Considering all these facts, the distribution of y can be expressed like:

$$y \sim N(\beta^T X, \sigma^2 I)$$

where β is coefficients of the model and $\sigma^2 I$ is the variance. In fact, $\beta^T X$ is the general linear regression (LR) model but $\sigma^2 I$ has been added to the model prediction for Bayesian LR model. This formula implies that the Bayesian LR model considers the uncertainty of the data. Similarly to other machine learning algorithms, the data should be normalised (for

Bayesian LR, the normalisation should be conducted before MCMC sampling) since the different features have different ranges. For example, *Longitude* and *AveBedrms* cannot be compared directly without scaling because the range of each parameter is different.

6.1 LR vs Bayesian LR

In order to check the performance of the Bayesian LR, Root Mean Square Deviation (RMSE) and Mean Absolute Error (MAE) are used and compared with LR (training:testing=8:2). (variance of the Bayesian LR was not considered and only coefficients and an intercept are taken into account for calculation):

	MAE	RMSE
Linear Regression	0.5332	0.745581
Baysian Linear Regression	0.5331	0.745598

Table 8: MAE and RMSE of each model

MAE and RMSE of LR and bayesian LR model is very similar which is not surprising. This is because, as mentioned earlier, Bayesian model considers both likelihood $p(y, x|\beta)$ and prior $p(\beta)$. That is, if there are sufficient enough training data (16512 training data) it overwhelms the prior distribution $p(\beta)$.

6.2 Results

By running pymc3, it was possible to get model parameters (coefficients) indicating that, the House value can be predicted by the following equation:

$$MedHouseVal = Intercept(=1) \times 2.068 + MedInc \times 0.853 +$$

$$HouseAvg \times 0.122 - AveRooms \times 0.305 +$$

$$AveBedrms \times 0.371 - Population \times 0.002 -$$

$$AveOccup \times 0.037 - Latitude \times 0.896 -$$

$$Longitude \times 0.869 + sd \times 0.720$$

By looking at the equation above, MedInc affects the house value in proportion and population does not affect the house value a lot. The values of Latitude and Longitude implies that the south west region's house value is higher than other regions in California. And by using the formula above, the effect of a single variable (MedInc is shown as an example) and the uncertainty could be visualised since the pymc3 can get different set of parameters/coefficients from the trace multiple times to draw blue lines [6]. By observing figure15, the model is quite certain about MedInc cofficients as blue lines throng with other lines.

6.3 Possible improvements

Burn-in chain could be introduced to restrict the sampler to get samples from the beginning, eventually leading the sampler not to converge. However, this method is not used because the model has already found posterior distribution of model parameters well. And this could be observed by *traceplot*.

Hierarchical Model can be used which assumes the model parameters (coefficients) all share similarity. That is, consider the *blocks* not as completely different but having an underlying similarity [7]. This approach has been tried but it took too much time for sampling, also fails to converge often (target acceptance was too low). $glm()$ was used for prior

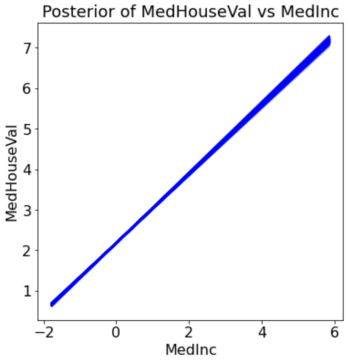

Figure 15: Relationship between MedInc and MedHouseVal [6]

distribution for the program but it says on the document, "Finally, readers of my blog will notice that we didn't use glm() here as it does not play nice with hierarchical models yet. [7]" which clearly supports the reason why Hierachical model did not work well. By not using $glm()$ it may possibly get a better result.

7 Random Forest (RT)

Decision tree is easy to build and to interpret by plotting as a tree. However, the model tends to over-fit easily [8]. Therefore, as a way to solve this, random forest algorithm will be introduced. (training: testing split = 7:3)

7.1 Random forest regression algorithm

Step1: Generate bootstrapped (allowing repetition of sample when selecting) sample set for training. (Note: it is possible to generate the sample without bootstrapping.)

Step2: Use bootstrapped sample set and build decision tree by using random subset of features. This can be adjusted by $max\_features$ which implies how many feature the model will consider when branching from the node. When branching, the model selects a feature and its threshold by finding the threshold that minimizes the model's square residual that is, yield largest information gain. This is called the CART (Classification And Regression Trees) algorithm. This can be expressed as an equation [9]:

$$Q_{left}(\theta) = (x, y)|x_j \leq t_m$$

$$Q_{right}(\theta) = Q - Q_{left}(\theta)$$

$$G(Q, \theta) = \frac{n_{left}}{N_m} H(Q_{left}(\theta)) + \frac{n_{right}}{N_m} H(Q_{right}(\theta))$$

And finally select the feature(θ) that minimises the impurity

$$\theta^* = argmin_\theta G(Q, \theta)$$

where x_i is the $i-th$ training example, Q is the data at node m, $\theta = (j, t_m)$ denoting the specific feature j of sample x and threshold t_m which segments the data into left $Q_{left}(\theta)$ and right $Q_{right}(\theta)$ hand side of tree, n_{left} and n_{right} is the number of samples in left and right hand side of branch, and finally N_m is existing training examples at note m. And the impurity function $H()$ which calculates the mean squared error (MSE) can be expressed by following formula:

$$\bar{y}_m = \frac{1}{N_m} \sum_{i \in N_m} y_i$$

$$H(X_m) = \frac{1}{N_m} \sum_{i \in N_m} (y_i - \bar{y}_m)^2$$

where X_m is the training data at node m.
Repeat the algorithm to choose the appropriate feature and its threshold for each node until it reaches max_depth considering min_sample_leaf and min_sample_split.
Step3: Repeat Step1 and Step2 to build trees until the number of trees reaches $n\_estimator$.
Step4: The generated decision trees can be aggregated to predict the target. This method is so called **Bagging** (Bootstrapping + Aggregating)

7.2 Hyper-parameters of RT

There are many hyper-parameters in RT as written in italics in italics in section 7. In this section, max_depth, $max\_features$, and $n\_estimator$ will be discussed also comparing the model's performance by changing these hyper-parameters [14, 15].
1. max_depth**:** defines how deep the decision tree is. The deeper the tree, the more splits the model can generate and the better the model captures information about the data, However, it might incur over-fitting. If the depth is set too low, the tree would be not flexible and might incur under-fitting.
2. $max\_features$**:** defines the number of features which is put in consideration when branching from the node. If a particular feature has a great influence on the model, $max\_features$ is needed to prevent the phenomena that all decision trees tend to be similar.
3. $n\_estimator$**:** defines the number of trees in the forest. It greatly affects other hyper-parameter values and vice-versa. It is likely that it prevents over-fitting the model by increasing its value.

max_depth	Training	Testing
5	0.677	0.652
10	0.877	0.783
20	0.998	0.814
50	1.0	0.82
100	1.0	0.819

Table 9: Changing max_depth and fix other hyper-parameters

$max\_features$	Training	Testing
all features(auto)	1	0.617
sqrt	1	0.821
log2	1	0.822

Table 10: Changing $max\_features$ and fix other hyper-parameters

$n\_estimator$	Training	Testing
10	1	0.794
50	1	0.824
100	1	0.822

Table 11: Changing $n\_estimator$ and fix other hyper-parameters

7.3 Optimising hyper-parameters

RandomizedSearchCV or GridSearchCV could be used to tune hyper-parameters on validation set. As can be seen by table12, the accuracy on the testing set did not improve a lot considering the time taken for optimising the hyper-parameter. Therefore, RandomizedSearchCV is recommended. Note that choosing hyper-parameter greatly depends on other hyper-parameters as well as depending on the data. The best hyper-parameters generated by $GridSearchCV$ are $n\_estimator = 282$, $min\_samples\_split = 2$, $min\_samples\_leaf = 1$, $max\_features = \log2$, $max\_depth = 50$, and $bootstrap =$ False.

	Time taken	Training	Testing
RandomizedSearchCV	6.2min	0.960	0.815
GridSearchCV	374min	1	0.821

Table 12: Time taken and accuracy for hyper-parameter tuning

7.4 Learning Curve

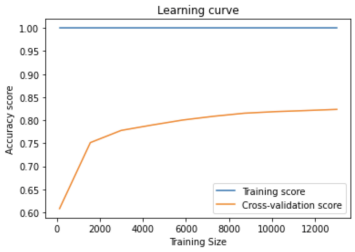

Figure 16: Training and Testing accuracy

Training accuracy stays at 1 because best hyper-parameters were plugged (calculated by $GridSearchCV$). The model tends to over-fit however got better result than LR and Bayesian LR.

	MAE	RMSE
Random Forest	0.312	0.4834
Linear Regression	0.5332	0.745581
Baysian Linear Regression	0.5331	0.745598

Table 13: MAE and RMSE of each model

7.5 Interpretation of RT

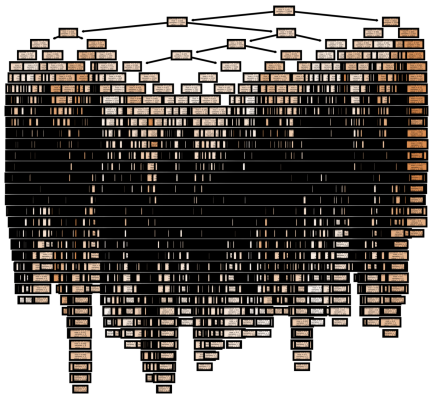

Figure 17: One of the decision tree in RT(282 DTs) [17]

It is technically *possible* to interpret figure17. However, considering there are 282 DTs, interpreting visualised RT is not recommended and nearly impossible.

8 Stacking

Stacking is a one of the technique among ensemble learning. Ensemble learning combines several machine learning model to create a single predictive model. There are a few examples

of ensemble learning such as Bagging, Boosting, and Stacking. Bagging is a technique to decrease the variance which tries to avoid over-fitting. In fact, random forest regression mentioned in section7 used Bagging to improve from single decision tree to multiple decision trees which consists a forest. Boosting is a technique used to decrease bias which aims to reduce under-fitting. Lastly, stacking is a technique to improve prediction stability for the unseen data [16]. Bayesian linear regression model and random forest regression were used for stacking. Note that there is only one layer for stacking but there could be multiple layers of stacking which could be worked similar to ANN.

8.1 Stacking algorithm

Step1: Train the model on two base models: M1, M2 and get estimations for each training sample. As usual, training data matrix shape is (M × N) where M is number of samples and N is features and target matrix (M × 1)
Step2: Use the previous estimation matrix to train the meta regressor (DecisionTreeRegressor) where the estimation matrix shape is (M × 2) and target matrix (M × 1).

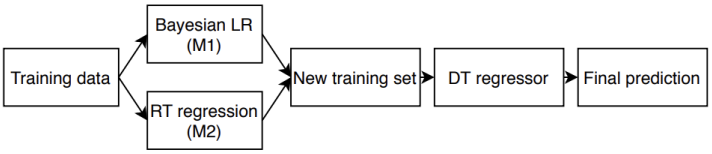

Figure 18: Stacking algorithm workflow. (DT Regressor or *RidgeCV* for meta regressor)

8.2 Bayesian ridge regression vs Bayesian LR

Bayesian ridge regression works in similar manner to Bayesian linear regression when prior distribution $p(\beta)$ ($p(\beta)$ mentioned in section 6) is normal distribution which indicates that its posterior distribution is also a normal distribution (conjugate prior). Ridge regression is a regression with regularization on coefficients (L2 norm). β can be expressed by following [12]:

$$\hat{\beta} = argmin_{\beta}(y - \beta^T X)^T(y - \beta^T X) + \lambda\|\beta\|_2^2$$

where $\|\beta\|_2{}^2 = {\beta_1}^2 + {\beta_2}^2 ...$ λ is a regularizer. And reminding that p(β) is chosen to be normal distribution in section 6, the posterior distribution of β given the data (y, X) can be expressed like following(assuming τ is a constant $\beta \sim N(0, \tau^2 I)$):

$$\begin{aligned} p(\beta|y, X) &\propto p(y, X|\beta) \times p(\beta) \\ &\propto exp[\frac{-1}{2}(y - \beta^T X)^T \frac{1}{\sigma^2}(y - \beta^T X)] \\ &\times exp[\frac{-1}{2}(\beta - 0)^T \frac{1}{\tau^2} I(\beta - 0)] \\ &= exp[\frac{-1}{2\sigma^2}(y - \beta^T X)^T(y - \beta^T X) - \frac{1}{2\tau^2}\|\beta\|_2^2] \end{aligned}$$

By acquiring β that maximises the above equation, also known as the Maximum a Posteriori (MAP) estimation,

$$\begin{aligned} \hat{\beta} &= argmax_{\beta} exp[\frac{-1}{2\sigma^2}(y - \beta^T X)^T(y - \beta^T X) - \frac{1}{2\tau^2}\|\beta\|_2^2] \\ &= argmax_{\beta}(y - \beta^T X)^T(y - \beta^T X) + \frac{\sigma^2}{\tau^2}\|\beta\|_2^2 \end{aligned}$$

this implies that it is the ridge regression estimate when $\lambda = \frac{\sigma^2}{\tau^2}$. This fact indicates that the Bayesian ridge regression works in similarly with Bayesian linear regression.

8.3 Results

By using the fact mentioned on section8.1, *sklearn.linear_model.BayesianRidge* was used for stacking. By looking at the table below it is clear that the stacking model worked well improving prediction on testing data. It outperforms over the individual models and the mean of individual models. The R2 score were calculated with random forest $max\_depth = 10$, $n\_estimator = 10$, and meta classifier *RidgeCV* (default). These hyper-parameters were tried since they were noted in the guideline. However, it is certain that the R2 score can be increased by choosing the other hyper-parameter values.

model	training	testing
Stacking	0.858	0.751
Bayesian LR	0.611	0.592
RT regression	0.863	0.750
RT regression + Bayesian LR	0.737	0.671

Table 14: R2 score of each model. RT regression + Bayesian LR is mean prediction from individual models (training:testing=7:3).

Instead of *RidgeCV*, *DecisionTreeRegressor* is used as a meta regressor to combine the base estimators (*DecisionTreeRegressor* was chosen because it was the one with the guideline). It turned out that the model performed poorly denoting that choosing appropriate meta regressor is vital to the stacking model.

model	training	testing
Stacking	0.609	0.509
Bayesian LR	0.611	0.592
RT regression	0.863	0.750

Table 15: R2 score of each model. *DecisionTreeRegressor* is used as a meta/combination regressor

9 Summary

A total of seven machine learning algorithms have been applied to MNIST and California housing data. Each section examined the concept of the algorithms and applied its algorithm to the data. Also, the limitations of each of the algorithm were investigated. All machine learning algorithms had to be written in 10 pages; thus, there are some concepts that are not properly detailed or could be further developed such as :

1. *solver* of ANN (gradient descent algorithm)
2. Ridge regression
3. Feature importance of RT
4. Applying kernel to the PCA
5. EM algorithm and GMM (Gaussian Mixture Model)

and more. Furthermore, the algorithm's running time was too long to find the best hyper-parameter values; therefore, some testing results were not good enough. However, the testing result could be increased further indeed.

10 References for report and code

[1] Gradient Descent for Linear Regression Explained, Albert Lai. URL: https://blog.goodaudience.com/gradient-descent-for-linear-regression-explained-7c60bc414bdd

[2] Machine Learning, Andrew Ng. URL: https://www.coursera.org/learn/machine-learning

[3] Spatial Choice Modeling Using the Support Vector Machine (SVM): Characterization and Prediction, Yong Yoon. URL: https://www.researchgate.net/publication/320452643_Spatial_Choice_Modeling_using_the_Support_Vector_Machine_SVM_Characterization_and_Prediction

[4] Support Vector Machines — Soft Margin Formulation and Kernel Trick, Rishabh Misra. URL: https://towardsdatascience.com/support-vector-machines-soft-margin-formulation-and-kernel-trick-4c9729dc8efe

[5] The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo. Matthew D. Hoffman, Andrew Gelman, URL: http://www.stat.columbia.edu/~gelman/research/published/nuts.pdf

[6] Data-Analysis, Will Koehrsen. URL: https://github.com/WillKoehrsen/Data-Analysis/

[7] GLM: Hierarchical Linear Regression. Danne Elbers, Thomas Wiecki, URL: https://docs.pymc.io/notebooks/GLM-hierarchical.html

[8] The Elements of Statistical Learning. Jerome H. Friedman, Robert Tibshirani, and Trevor Hastie, URL: https://web.stanford.edu/~hastie/Papers/ESLII.pdf

[9] 1.10. Decision Trees. scikit-learn developers, URL: https://scikit-learn.org/stable/modules/tree.html

[10] Optimizing the Prediction Accuracy of Concrete Compressive Strength Based on a Comparison of Data Mining Techniques. Jui-Sheng Chou, Chien-Kuo Chiu, Mahmoud Farfoura, Ismail Altaharwa. URL: https://www.researchgate.net/publication/239386696_Optimizing_the_Prediction_Accuracy_of_Concrete_Compressive_Strength_Based_on_a_Comparison_of_Data_Mining_Techniques

[11] Pattern Recognition and Machine Learning. Christopher Bishop, URL: http://users.isr.ist.utl.pt/~wurmd/Livros/school/Bishop%20-%20Pattern%20Recognition%20And%20Machine%20Learning%20-%20Springer%20%202006.pdf

[12] Bayesian interpretation of ridge regression. URL: https://statisticaloddsandends.wordpress.com/2018/12/29/bayesian-interpretation-of-ridge-regression/e

[13] 11d Machine Learning: Bayesian Linear Regression. Michael Pyrcz. URL: https://www.youtube.com/watch?v=LzZ5b3wdZQk

[14] Random forest. URL: https://en.wikipedia.org/wiki/Random_forest

[15] 3.2.4.3.1. sklearn.ensemble.RandomForestClassifier. scikit-learn developers. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[16] Ensemble methods: bagging, boosting and stacking. Joseph Rocca. URL: https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

[17] Visualizing Decision Trees with Python (Scikit-learn, Graphviz, Matplotlib). Michael Galarnyk. URL: https://towardsdatascience.com/visualizing-decision-trees-with-python-scikit-learn-graphviz-matplotlib-1c50b4aa68dc