MATH97131 - Machine Learning Coursework 2

Feburary 2020

1 Question 1

1.1 PCA

As PCA transforms the data into orthogonal linearly uncorrelated variable (The priciple components) we scale the data before applying PCA. We represent the covariance of the variables X as $\Sigma = COV(X)$, the eigenvectors of the matrix X as U in order of largest to smallest eigen value. We want to find a vector of constants a such that $Var(a^T X) = a^T \Sigma a$ is maximised but we enforce the constraint $a^T a = 1$ (a is unit length). Using Lagrange multipliers we maximise the function

$$f = a^T \Sigma a - \lambda(a^T a - 1)$$

Taking the derivative a w.r.t a we get

$$\Sigma a = \lambda a$$

This is the same as the definition of the eigenvalues and vectors of Σ with λ the eigenvalues and a the eigenvectors. As we want the largest value we chose the largest eigen value and its eigen vector. This forms the first principle component of X and the one to which the most variance is attributed. We want to find the other principle components as maximising the same function but with a new a and λ but uncorrelated with all other principle components. This can be shown to be the other eigenvalues and vectors of Σ similarly.

We transform X to be in the space of the principle components through the multiplication XU. To find the variance attributed to each principle component we calculate the variance across the columns of the transformed X. As each PC is orthogonal the sum of the variance of the transformed data in each principle component is the variance of all the data ie the covariance of PC is zero as they are orthogonal. We can then find the percentage of the variance attributed to each by scaling them by dividing by the total variance and getting the cumulative sum. This gives the results that 90% of the variance of data is 6 and 9 and 16 for 95 and 99% of the variance. To map the data to just the first two principle components we multiply by a U st it only contains the first two components. The plot of this map can be seen in fig. 1. We can see a clear separation between the cases in the first two components. We would expect this as they represent the most variation in the data. This method of reducing the dimensions of the data is useful in allowing us to train models easier and faster (as dimensionality increases often training time increases rapidly) whilst losing minimal information. However a major disadvantage of this is the principal components are just variables orthogonal to the data so reduce our ability to interpret the model. This may be important when looking at a diagnosis as you likely want to know the interactions of variables and how they effect the diagnosis.

1.2 Kmeans and Hierarchical Clustering

We use the scaled data for these two models as the differing scales would drastically reduce the predictive ability of the models otherwise. We can also use the principle components from before in these two models as the X if we wished. One disadvantage of both of these models is as it is unsupervised we can find the clusters but we cannot determine which cluster is each diagnosis.

1.2.1 Kmeans

We assume there are $k = 2$ clusters one for each diagnosis. We want to minimise the average distance of a point to the centre of its cluster. Commonly we use the squared L2 norm. Initially we assign each point at random to each of the $c_1, \ldots, c_k$ centroids. We define convergence criterion's the max number of iterations and the minimum change in the sum of the squared distances between the k centroids before and after the iteration. X_j represents the partition of X in centroid c_j and $dist(x, y)$ a distance function such as the square L2 norm $dist(x, y) = \|x - y\|^2$.

Algorithm 1 K-means

```
For a dataset $X = \{x_1, \ldots, x_n\}$ $(x_i \in \mathbb{R})$
**Initialise** Choose centroids $c_1, \ldots, c_k$ in $X$ at random
it = 1
**while** it $\leq$ iterMax & $\epsilon >$ converged **do**
    $cOld = c$
    **for** $i = 1, \ldots, n$ **do**
        $cluster(x_i) = \text{argmin}_{j \in \{1,\ldots,k\}} dist(x_i, c_j)$
    **end for**
    **for** $j = 1, \ldots, k$ **do**
        $n_j = \sum_{i=1}^{n} \mathbb{I}(x_i \in X_j)$
        $c_j = \frac{1}{n_j} \sum_{i=1}^{n} x_i \mathbb{I}(x_i \in X_j)$
    **end for**
    $\epsilon = \sum_{j=1}^{k} \|c_j - cOld_j\|^2)$
**end while**
```

The algorithm is guaranteed to converge after a finite number of steps as there are a finite number of partitions of X $\binom{n}{k}$. However there is no guarantee on how fast it will converge hence the convergence criterion. We can also apply K-means to a kernel function of X.

1.2.2 Hierarchical Clustering

We define a distance $dist(x, y)$ with points x,y and linkage $link(A, B)$ with clusters A,B function for use in the algorithm. We use the linkage functions the minimum distance between the points in two clusters, the maximum distance or the average distance between all points in each cluster.

We can use a divisive method assuming all points are in the same cluster and recursively dividing into the two least similar clusters until we have every point in a separate cluster. Alternatively we can use a agglomerative method assuming all points are in separate clusters and join the two most similar clusters recursively.

Algorithm 2 Agglomerative Hierarchical Clustering

```
For a dataset $X = \{x_1, \dots, x_n\}$ $(x_i \in \mathbb{R})$
Initialise Assign each point in $X$ to a separate cluster let the list of groups be $g = -1, \dots, -n \in \ltimes$.
$M \in \mathbb{R}^{n-1x2}$ to represent the merge order. $H \in \ltimes - \nVdash$ to represent the height at each merge.
for $i = 1, ..n$ do
    for $j = 1, ..n$ do
        $D_{i,j} = dist(x_i, x_j)$
    end for
end for
$diag(D) = \infty$                                    ▷ Calculate pairwise distances
for $j = 1, \dots, n-1$ do                            ▷ Find minimum distance
    $h_j = \min D$
    $i = which(D = h_j)_i$                            ▷ Find points indices with the minimum distance
    $m_{j,} = sort(g_i)$                              ▷ Add to merge matrix
    $group = (i, which(g = g_{i_{1,which(g_i>0)}})$   ▷ Find indices of the cluster its joining
    $g_{group} = j$                                   ▷ Update cluster assignments
    Let A represent the new cluster j and $B \in \mathbb{R}^s$ the clusters $\neq$ j $s = 1, \dots, l$
    for $s = 1, \dots, l$ do
        $r_s = link(A, B_s) \forall s$
    end for                ▷ We calculate the linkage between the new cluster and all other clusters
    $D_{\min(i),} = D_{,\min(i)} = r$                 ▷ Update distances according to linkage
    $D_{\min(i),\min(i)} = \infty$
    $D_{\max(i),} = D_{,\max(i)} = \infty$
end for
```

We use the symmetric property of the distance matrix when updating so we don't have to update as many positions with the linkage instead we assign them to ∞.

1.2.3 Comparison

We see the accuracy of the tested classifiers in fig. 2. We don't need to use a separate test set as they are unsupervised so can just evaluate their performance in light of the diagnosis. We see that kmeans performs the best with most of the Hierarchical clustering models performing poorly except in the case of the L1 distance function with the max as the linkage. This poor performance is likely due to the two clusters being very similar so separated into too many clusters by the hierarchical models but the L1 distance and maximal linkage allowing it to see the difference better. This problem is due to the hierarchical method assuming n different clusters and joining 1 by 1 so when evaluating the performance and cutting the denogram when there's 2 separate clusters can lead to it having one very large cluster and one very small as it identifies clusters within each class. K means makes the assumption of the number of classes before the algorithm starts which is better for this problem as we know k before.

2 Question 2

2.1 Dataset

We first explore the dataset and we see that in fig. 3a the histogram of the year of the song. This shows us that the number of songs before 1960 is small but after this it is pretty uniformly distributed. Thus showing there aren't long tails in the distribution of y. The heatmap of the correlations between the columns of the dataset in fig. 3b suggests that most of the parameters have a low correlation with the song year on their own but some parameters have a high correlation with each other but the vast majority are uncorrelated with one another. This is good as we don't want highly correlated variables (unless thyre highly correlated with y) otherwise we are unlikely to gain more information from having both. This shows that it is unlikely all the features will be important in the prediction of song year. We see in fig. 4 that the scale of the features varies greatly suggesting the need for scaling if the model is sensitive to that which neural networks are but regression trees are not.

2.2 Regression Trees

The algorithm for growing a regression tree with the loss function as the sum of squared error.

Algorithm 3 Regression Tree

```
Start with root node $X = \mathbb{R}^p$ and set $I = \{1, \dots, N\}$ add to the queue $Q$
while $Q \neq \emptyset$ do
    $i \in I$
    for $j = 1, \dots, p$ do
        for $s \in [\min x_j^{(i)}, \max x_j^{(i)}]$ do
            $I_1(j,s) = \{i \in I | x_j^{(i)} \leq s\}$
            $I_2(j,s) = \{i \in I | x_j^{(i)} > s\}$
            $q_{j,s} = \sum_{i \in I_1(j,s)} (y^i - \hat{c}_1)^2 + \sum_{i \in I_2(j,s)} (y^i - \hat{c}_2)^2$
            Where $\hat{c_m} = \frac{1}{N_m} \sum_{i \in I_m(j,s)} y^{(i)}$
        end for
    end for
    Choose the split $(j^*, s^*)$ with $\min q = q_{j^*,s^*}$
    Add the children $(x_{(i)}, y_{(i)})_{i \in I_1(j^*,s^*)}$ and $(x_{(i)}, y_{(i)})_{i \in I_2(j^*,s^*)}$ unless they have reached a stopping condition
end while
```

To predict from the tree we propagate down the tree until we reach a leaf node and we take the c calculated at that node as the output.

2.3 Random Forests

Using the algorithm to grow a tree we form an ensemble of trees taking the average prediction of the bagged trees. We train each tree on a bootstrap sample with a proportion of the parameters selected at random (mtry). This gives us two tuning parameters mtry and the number of trees in the forest.

We define a bootstrap sample as a sample of the training data taken with replacement. Those not in the bootstrap are the out of the bag samples.

Algorithm 4 Random Forest

```
B the number of trees and mntry
for $b = 1, \dots, B$ do
    Take a random sample of the training data set with replacement the boot. With the out of the bag samples those in the training dataset but not in the boot
    Grow a Regression Tree with the boot sample and only on mntry features selected at random
end for
```

To get a prediction from the Random Forest you take the mean prediction of the B trees.

2.4 Multilayer Perceptron

A multilayer perceptron is built up of layers of neurons. A neuron has inputs $X \in \mathbb{R}^n$ which can be the data or from another neuron, it has weights $W \in \mathbb{R}^n$ and a bias $\in \mathbb{R}$ It computes the activation as a function to the inputs, weights and bias $W^T X + b$. Then it applies an activation function $\sigma(\cdot)$ to the activation to give the output of the neuron as

$$y = \sigma(W^T X + b)$$

.

We can build multiple layers of multiple neurons that feed the output of the previous layer to the next with differing activations and sizes until we reach the output layer. We can define this neural network with $L + 1$ layers $z^{(0)}, \dots, z^{(L)}$ where $z^{(i)} \in \mathbb{R}^{n_i}$ with n_i the number of neurons in layer i. Input $x = z^{(0)} \in \mathbb{R}^d$ and output $\hat{y} = z^{(L)}$ We can then define for $l = 0, \dots, L - 1$ the forward propagation

$$z^{(l+1)} = \sigma\left(W^{(l)} z^{(l)} + b^l\right)$$

Where $W^{(l)} \in \mathbb{R}^{n_{l+1} x n_l}$, $b^{(l)} \in \mathbb{R}^{n_{l+1}}$ and the activation function $\sigma : \mathbb{R} \to \mathbb{R}$ applied elementwise. We want to find the weights and biases such that we maximise the log-likelihood. To do this we must define a loss function as this is a regression problem we select the mean squared error with p the dimensions of the output.

$$MSE = \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{p} (y_j^{(i)} - \hat{y}_j^{(i)})^2$$

2.4.1 Gradient Descent

As we assume the samples from the data are independent and identically distributed the loss is a summed across each sample for the model parameters θ for N samples and loss function L.

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} L(x^{(i)}, y_{(i)}, \theta)$$
$$\implies \nabla_\theta J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta L(x^{(i)}, y_{(i)}, \theta)$$

This can be expensive for large datasets so instead we use Stochastic Gradient Descent. We approximate the expected gradient by using a random minibatch of the data $(x^{n_i}, y^{n_i})_{i=1}^{N'}, N' \ll N$ This means the estimated gradient is

$$g = \frac{1}{N'} \sum_{i=1}^{N'} \nabla_\theta L(x^{(n_i)}, y_{(n_i)}, \theta)$$

We the adjust the parameter with learning rate $1 \leq \epsilon \leq 0$

$$\theta \leftarrow \theta - \epsilon g$$

This does introduce two more training parameters the size of the minibatch and the learning rate.

For our case with the loss function as the MSE and $\sigma_{(l)}$ representing the activation function of layer l.

We can define the loss

$$L = \frac{1}{2N'} \sum_{i=1}^{N'} (z_i^{(L)} - y_i)^2$$

We then take the derivative with the respect to the output of layer L $z^{(L)}$

$$\frac{\partial L}{\partial z^{(L)}} = \frac{1}{N'} \sum_{i=1}^{N'} (y_i - z_i^{(L-1)}) \sigma_L'(z_i^{(L)})$$

We can can then apply the chain rule to get

$$\frac{\partial L}{\partial z^{(l)}} = \prod_{i=l}^{L-1} \frac{\partial L}{\partial z^{(i+1)}} \frac{\partial z^{i+1}}{\partial z^{(i)}}^T$$

Then we can calculate the gradients

$$\begin{aligned} g(w^{(l)}) &= \frac{\partial L}{\partial z^{(l)}} \frac{\partial z_{(l)}}{\partial w^{(l)}}^T \\ &= \frac{\partial L}{\partial z^{(l)}} (z^{(l-1)})^T \\ g(b^{(l)}) &= \frac{\partial L}{\partial z^{(l)}} \frac{\partial z_{(l)}}{\partial b^{(l)}}^T \\ &= \frac{\partial L}{\partial z^{(l)}} \end{aligned}$$

Using that $\frac{\partial z_{(l)}}{\partial b^{(l)}}$ is the identity and $\frac{\partial z_{(l)}}{\partial w^{(l)}} = z_{(l-1)}$. We then use the gradients to update each of the parameters as explained above. Repeating this for each batch. We then loop over new random samples of batches of the data for each training epoch updating the weights.

2.4.2 Activation functions

Activation functions I used include the identity function as the only output activation. A multilayer perceptron with only identity functions is equivalent to a linear model. It is also easily differentiable.

$$identity(x) = x$$

A activation function I did not use but is the basis for others is the ReLU function. The main reason in not using ReLU is as it is zero and has zero gradient for negative values it cannot backpropagate on these values. It is easily differentiable. It is not commonly used in an output layer.

$$ReLU(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$$

Another activation function used is the Leaky ReLU function this overcomes the problems of the ReLU function having zero gradient for negative values but it adds an extra parameter to optimise. It is easily differentiable.

$$lReLU(x, \epsilon) = \begin{cases} x & x \geq 0 \\ \epsilon x & x < 0 \end{cases}$$

Another activation function is ELU which also has an extra parameter to optimise. It is easily differnetiable and it less sharply smooths negative values.

$$ELU(x, \alpha) = \begin{cases} x & x \geq 0 \\ \alpha(\exp x - 1)x & x < 0 \end{cases}$$

2.5 Comparison

We use a 80:20 split between the train data and the test data. We dont apply k-folds due to the training time being too high for each model. The random forest is uneffected by scaling the data however for the perceptron we use scaled data. This will help stop the weight matrices diverging and should decrease training times. We use 5 trees. The more we use the lower our MSE should become. We could use more but we are limited by the time taken to train. The reduction in MSE as we add trees is shown in fig. 6b. We see this reduction as trees are added but for 5 trees it increases though it could decrease further with more trees with 5 trees it had a test mse of 667.8. One major disadvantage of the random forest is it is not good at predicting out of sample variables unlike the neural network. It does however give us a better understanding of the data as we can see how the dataset is partitioned and which features are used so are therefore useful. We test different hyper parameters for the neural net. We don't test using different activation functions for each layer or different sizes for each hidden layer as this would greatly increase the number of parameters to test. Taking the test MSE of the best performing model out of all the epochs to avoid overfitting. The mse over training of the best model is shown in fig. 6a. We see it starts very high for the random weights but quickly reduces.

The number of hyper parameters to tune make using a neural network more difficult. You are also not guaranteed that the training will not cause the weights to diverge and thus cause the model to be unusable. This is more likely with a larger learning rate but too small a learning rate will cause the model to train slowly and increase the risk you'll converge at a local minima. However it is a much better performing model in this case with a MSE for the best model of 0.68 on the test set.The parameters of this model were 3 hidden layers of 128 neurons with the identity activation function for every layer. The difference in MSE is shown in fig. 7. We could also put a group of independently trained neural nets of differing or the same structure into an ensemble to potentially get a better model. This would also reduce the risk of overfitting. We do to reduce the risk of overfitting take the model that performs best on the test set out of all training epochs.

2.6 Feature Importance

I chose approach 2 in the notes as it uses out of the bag estimates unlike approach 1 so it is not prone to ever fitting. We use the MSE as our loss. So for $b = 1, \dots, B$ where B is the number of trees in the forest.

$$\hat{e}^{(b)} = L(y, \hat{y})$$

Algorithm 5 Feature Importance

```
$\hat{e}^{(b)} = L(y, \hat{y})$ for the out of bag sample with $p$ the number of variables.
for $j = 1, \ldots, p$ do
    For a random permutation $\tau(\cdot)$ generate new samples $(\bar{x}^{(1)}, y^{(1)}), \ldots, (\bar{x}^{(n)}, y^{(n)})$ where the k-th predictor is permuted $\bar{x}^{(i)} = x^{(\tau(i))}$
    Compute predictions $\hat{y}$ with the new dataset and get $\hat{e}_j^{(b)} = L(y, \hat{y})$
end for
$\frac{1}{B}\sum_{b=1}^{B} \hat{e}_b^{(b)} - \hat{e}^{(b)}$
```

for loss $L(y, \hat{y}) = mean((y - \hat{y})^2)$

The reasoning behind this method is that by permuting just the kth predictor we remove its relationship with y. We then see its effect on the loss. It the feature is important it will have a high $\hat{e}^{(b)}$ and thus have a high importance.

We see in fig. 5 that a few values are much more important than the others due to their much higher relative importance. The parameter with the highest importance is 80 followed by 19 and 41. The majority of the features have low feature importance. This was suggested by the correlation matrix.

3 Question 3

3.1 Bayesian Linear Regression

For a design matrix with or without basis functions applied $X \in \mathbb{R}^{nxp}$ and $y \in \mathbb{R}^n$. We assume a prior independent Gaussian distribution on the weights θ and independent noise σ. Applying Bayes rule the posterior distribution over the parameters is

$$p(\theta|X, y, \sigma) = \frac{p(y|\theta, X, \sigma)p(\theta)}{p(y|X, \sigma)}$$

with the marginal likelihood written as

$$p(y|X, \sigma) = \int p(y|X, \theta, \sigma)p(\theta)d\theta$$

As we have defined the prior on the weights as independent Gaussian

$$p(\theta) = \mathcal{N}_\theta(0, \alpha I)$$

and

$$p(y|\theta, X, \sigma) = \mathcal{N}_y(x\theta, \sigma I)$$

This leads to (we can remove terms not involving θ as they are removed by evaluating the marginal likelihood

$$\begin{aligned} p(\theta|X, y, \sigma) &= \frac{\mathcal{N}_y(x\theta, \sigma I)\mathcal{N}_\theta(0, \alpha I)}{\int \mathcal{N}_y(x\theta, \sigma I)\mathcal{N}_\theta(0, \alpha I)d\theta} \\ &\propto \exp\left(-\frac{1}{2\sigma^2}(y - X\theta)^T(y - X\theta)\right)\exp\left(-\frac{1}{2\alpha^2}\theta^T\theta\right) \\ &= \exp\left(-\frac{1}{2\sigma^2}\left(\theta^T\left(X^TX + \frac{\sigma^2}{\alpha^2}I\right)\theta - 2\theta^TX^TY\right)\right) \end{aligned}$$

This gives us as our posterior a Gaussian with

$$\mu = \left(X^TX + \frac{\sigma^2}{\alpha}I\right)^{-1}X^TY$$

$$\Sigma = \sigma^2\left(X^TX + \frac{\sigma^2}{\alpha}I\right)^{-1}$$

3.2 Gaussian Process Proof

If we introduce a basis function $\phi(x) = \phi$ that maps the p-dimensional x to a N dimensional feature space with Φ the aggregation of rows $\phi(x)$ for the training dataset. For simplicity let $\gamma = \frac{\sigma^2}{\alpha} I$ the predictive posterior becomes

$$p(\phi|\Phi, y, \sigma) = \mathcal{N}_\phi(\phi \left(\Phi^T\Phi + \gamma\right)^{-1} \Phi^T Y, \sigma^2 \phi \left(\Phi^T\Phi + \gamma\right)^{-1} \phi^T)$$

Now looking at μ and rearranging we get

$$\begin{aligned} \mu &= \phi \left(\Phi^T\Phi + \gamma\right)^{-1} \Phi Y \\ &= \phi\alpha\Phi^T \left(\Phi^T\alpha\Phi + \sigma^2\right)^{-1} Y \end{aligned}$$

and for Σ we get

$$\begin{aligned} \Sigma &= \sigma^2\phi \left(\Phi^T\Phi + \gamma\right)^{-1} \phi^T \\ &= \phi I\alpha\phi^T - \phi I\alpha\Phi^T \left(\Phi^T I\alpha\Phi + I\sigma^2\right)^{-1} \Phi I\alpha\phi^T \end{aligned}$$

If we define a kernel function

$$K(x_1, x_2) = x_1\alpha x_2^T$$

which is a valid kernel function as

$$\begin{aligned} \sum_{i=1}^{n}\sum_{j=1}^{n} k(x_i, x_j)c_i c_j &= cX\alpha X^T c^T \\ &= \langle \alpha^{1/2}Xc, \alpha^{1/2}Xc \rangle \\ &= \|\alpha^{1/2}Xc\|^2 \geq 0 \end{aligned}$$

so is positive semi definite and therefore a valid kernel function. We can find $\alpha^{1/2}$ as the single value decomposition. If we paramaterise the above in terms of the kernel we get

$$\Sigma = k(\phi, \phi^T) - k(\phi, \Phi^T)(k(\Phi^T, \Phi) + \sigma^2)k(\Phi, \phi^T)$$

$$\mu = k(\phi, \Phi^T)(k(\Phi^T, \Phi) + \sigma)y$$

This is a Gaussian Process for this kernel function k.

3.3 Gaussian Processes

We make the assumptions the posterior distribution is Gaussian and the observations in X are independent and the noise is constant. We can change the Kernel function to produce other gaussian processes not equivalent to the bayesian linear model. We can also add and multiply kernels together to form new kernels as the product and sum of semidefinite matrices is semidefinite.

We define a Gaussian Process for a kernel k a train set X, y,noise σ and test set x' and $\hat{y}$ the predictions we define the

$$\begin{aligned} L &= k(x', X)(k(X, X) + \sigma^2)^{-1} \\ \mu &= Ly \\ \sigma &= k(x', x') - Lk(x', X)^T \\ p(\hat{Y}|k, X, y, \sigma) &= \mathcal{N}(\mu, \sigma) \end{aligned}$$

We use the kernel for the bayesian linear approach as described above to create 1 model. We use the squared exponetial kernel to create the second model defined as

$$k_{SE}(x, x') = \sigma^2 \exp\left(\frac{(x - x')^2}{2l^2}\right)$$

σ^2 and l are scaling parameters with σ controlling amplitude and l periodicity usually over long periods such as yearly variation in the case of this data.

For our final model we use as a kernel

$$k(x,x') = c_1 x^T \sigma x' + c_2 \exp\left(\frac{(x-x')^2}{2l^2}\right) \exp\left(\frac{2\sin(\frac{\pi|x-x'|}{p})}{l^2}\right)$$

with c_1, c_2 linear scaling constants and a new parameter p controlling small periodic variations

We use k fold cross validation to determine the best model and each models best parameters. We then us these to predict the temperature in March 2011. The best performing model with the lowest MSE on the test set was the third model with the first model the bayesian linear regression model performing worst on average. Show in fig. 9. Model 1 having $\alpha = 6$, model 2 $l = 13$ and fixed $\sigma^2 = 1$ and model 3 $l = 3$ and $p = 13$ with fixed $c_1 = 1$,$c_2 = \alpha = 2$. We fix the noise at 2 for all models. We fix some values as otherwise the grid to search would be very large.

We see that if we look at fig. 8a why the bayesian model fits so poorly as it fails to fit the rapid variations in temperature. Models 1 and 3 predicting 19.8c and 20.1c for March 2011 but Model 2 predicts a temperature of 6.9c fig. 8b. All these estimates are much lower than 36c. Either a temperature of 36c is very unusual which could be the case as its much higher than any previously recorded temperature or as March 2011 is 15 months beyond the last training point it is hard for the models to predict this far into the future. This is shown by the 2nd model which uses a l lower than the forecast is ahead of the last point so has no data less than l away. This is also why we see model 3 prediction move towards that of model 1. This also explains why the 95% CI for both models expands. This is why the performance of these models within the sample space is so much better than it seems to be for forecasting a large number of months ahead.

4 Appendix

4.1 Figures

4.1.1 Question 1

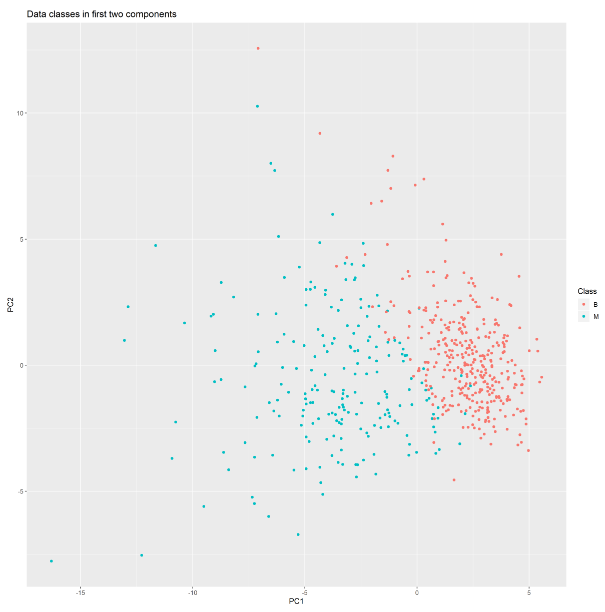


Figure 1: Scatter plot of classes in the two first PCs

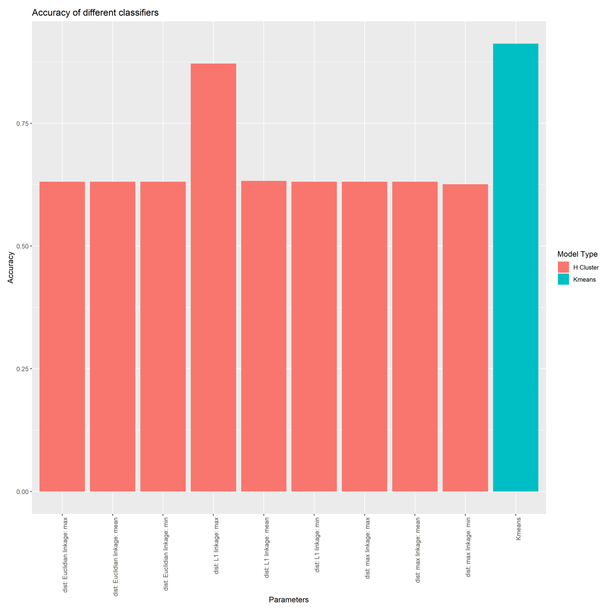
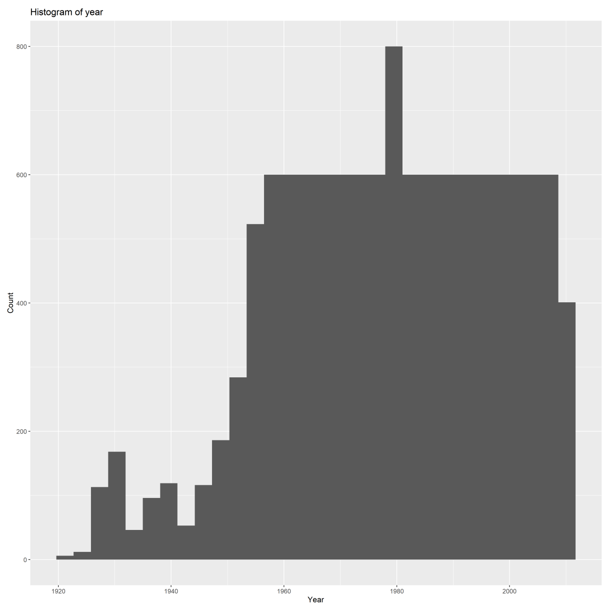
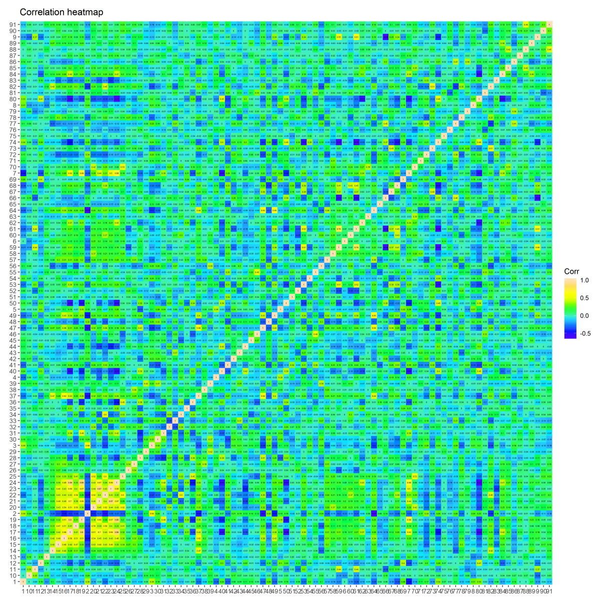


Figure 2: Accuracy of kmeans and hierarchical clustering algorithms

4.1.2 Question 2



(a) Histogram of years



(b) Correlation heatmap of dataset

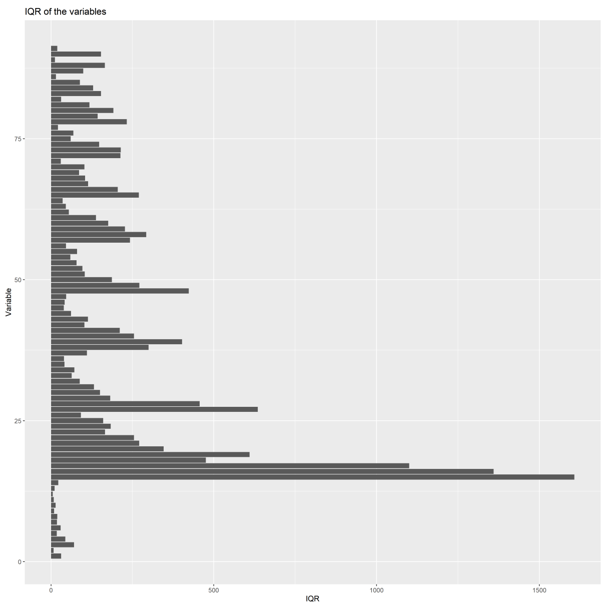


Figure 4: IQR of features

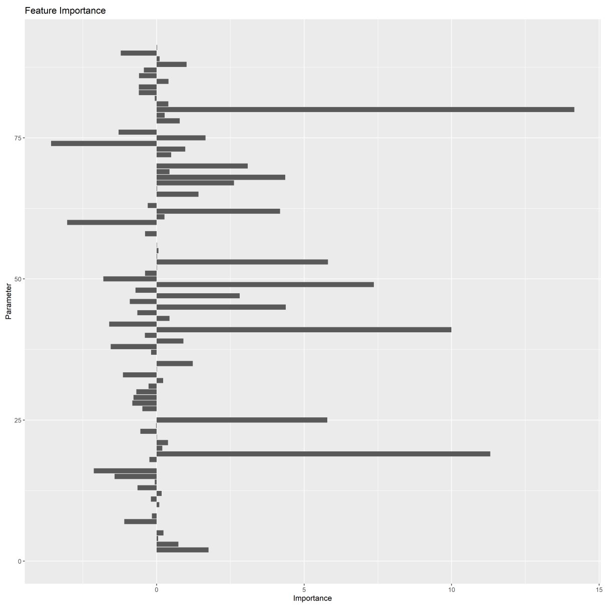
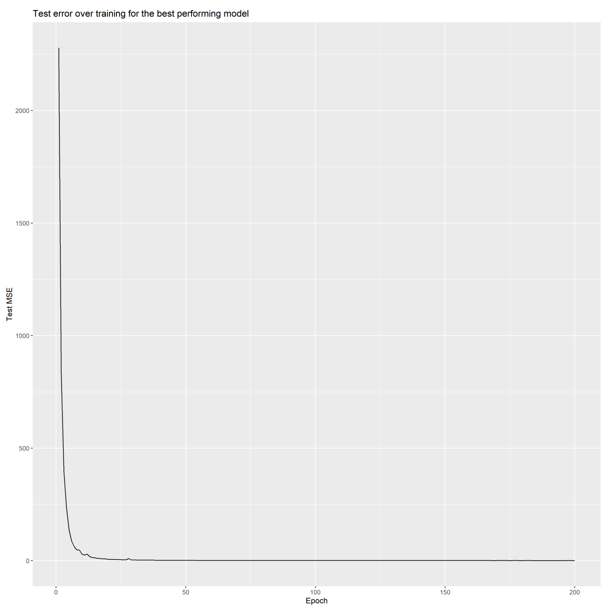
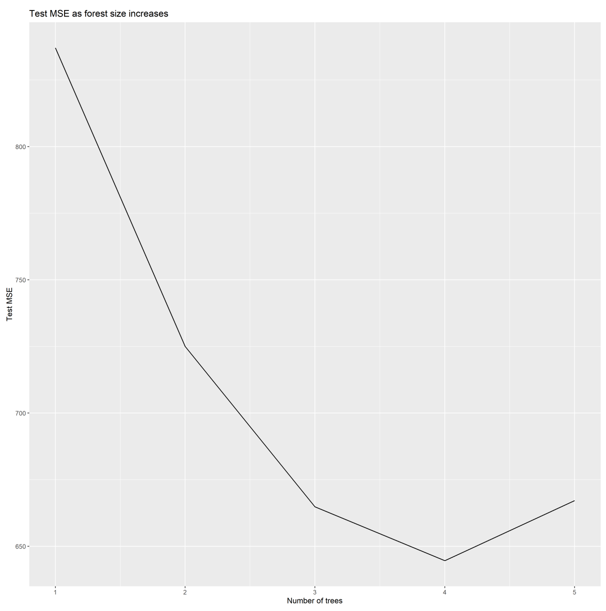


Figure 5: Importance of features



(a) MSE of best model over epochs



(b) MSE of random Forest as trees are added

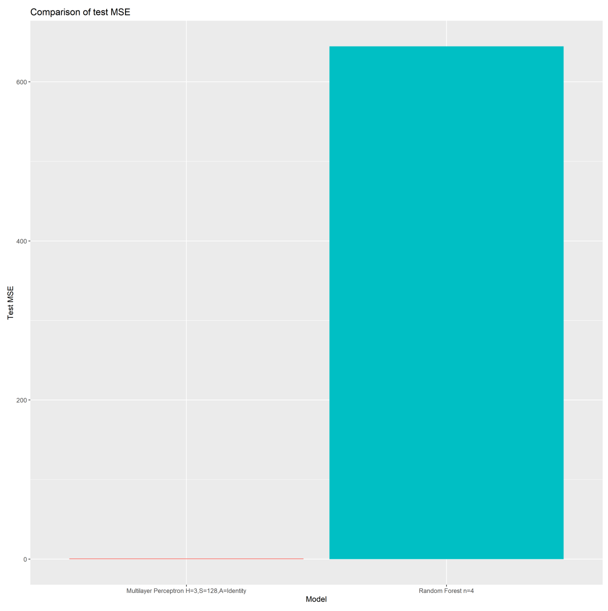
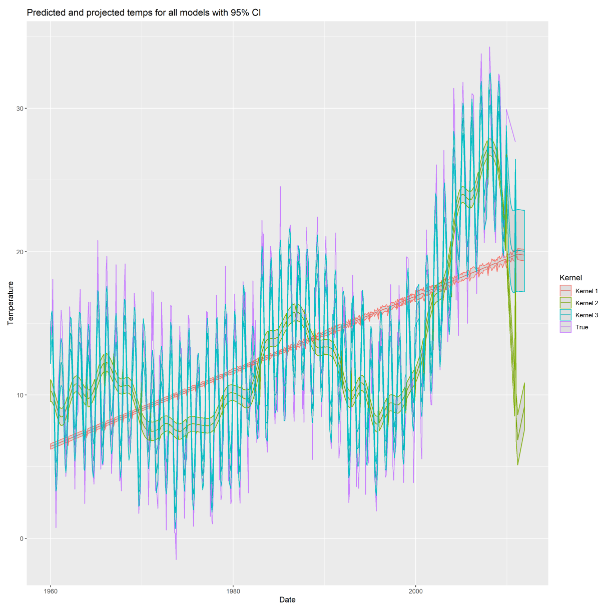
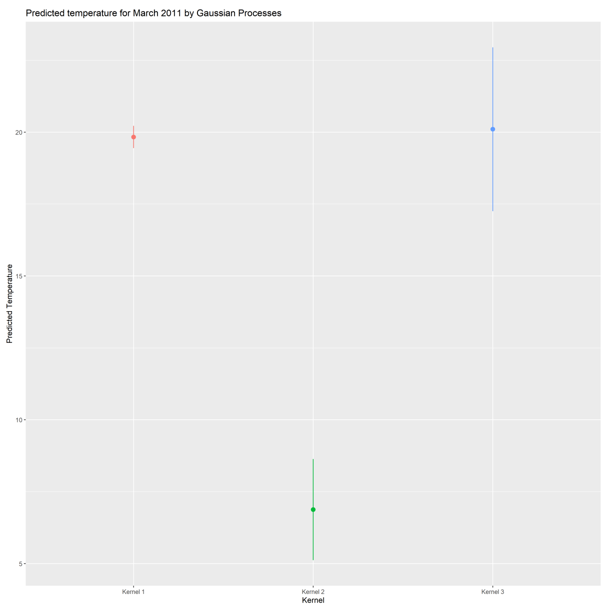


Figure 7: Comparison of random forest and best performing NN

4.1.3 Question 3



(a) Predicted and real temperatures from January 1960 to March 2011



(b) Predicted March 2011 temps

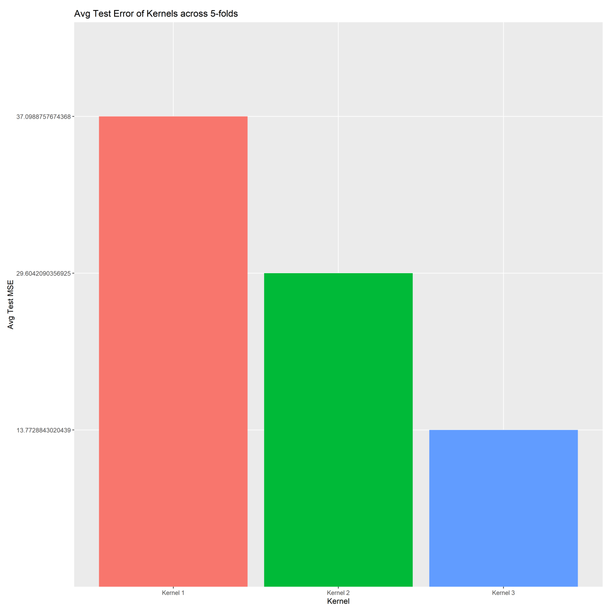


Figure 9: MSE of kernels on 5-folds

4.2 Code

```r
library("ggplot2") # plotting
# reshape2::melt is also used later for convenience

# defaults
path <- "~/Imperial Projects/Machine Learning/Coursework 2/"

defaultPlotSettings <- list(path=path,height = 12,width = 12,units = "in")

# question 1

# load data
q1Data <- read.csv(paste0(path,"dataQ1.csv"))
q1DataS <- q1Data
```

```r
q1DataS[,2:ncol(q1DataS)] <- scale(q1DataS[,2:ncol(q1DataS)])


# PCA
x <- as.matrix(q1DataS[,2:ncol(q1DataS)])
co <- cov(x) # get covariance matrix

eig <- eigen(co) # eigen vectors and values
y <- x %*% eig$vectors # transform data to pc space
vars_pca <- apply(y, 2, var) # calculate variance in each pc
vars_pca <- vars_pca/sum(vars_pca) # standardise

# cumsum
cumVar <- cumsum(vars_pca)

# number of components to account for perc of total variance
var90 <- sum(cumVar<=0.9)
var95 <- sum(cumVar<=0.95)
var99 <- sum(cumVar<=0.99)

y <- x %*% eig$vectors[,c(1,2)] # transform into first two pc
y <- as.data.frame(y)
colnames(y) <- c("PC1","PC2")
y$Class <-q1Data[,1] # add classifications

# plot
ggplot(data = y) + geom_point(aes(x=PC1,y=PC2,color=Class,group=Class)) +
  ggtitle("Data classes in first two components") +
  do.call(ggsave,c("q1PCA.png",defaultPlotSettings))

kmeans <- function(X,k=2,conv=10e-6,iterMax=100){
  # initial values
  p <- NCOL(X)
  n <- NROW(X)
  centroids <- X[sample.int(NROW(X),k),]
  epsilon <- Inf
  it <- 0
  while(epsilon>conv & it<=iterMax){
    oldCentroids <- centroids # keep old centroids
    dst <- sapply(1:k, function(ce){sapply(1:n,function(i){sum((centroids[ce,]-X[i,])^2)
    })}) # calculate distances
    cluster <- apply(dst, 1, which.min) # find which each point is closest to

    centroids <- t(sapply(1:k,function(ce){apply(X[cluster==ce,],2,mean)})) # calculate
    new centroids

    epsilon<- sum((centroids-oldCentroids)^2) # get change in centroids
    it <- it+1

    }

  return(list(centroids=centroids,cluster=cluster))

}


clusterCut <- function(m,k){
  n <- nrow(m) + 1
  res <- rep(NA,n)
  m_nr <- rep(0,n) # store  merge at each step



  for (i in 1:(n-1)){
    # get merge at step i
    m1 <- m[i,1]
    m2 <- m[i,2]
    # assign new groups
    if (m1<0&m2<0){
      m_nr[-m1] = m_nr[-m2] <- i

    }else if (m1<0 | m2 <0){
      if (m1<0) {j=-m1;m1=m2} else {j=-m2}

      l <- which(m_nr==m1)
      m_nr[l]<-i
      m_nr[j] <- i
    }else{

      m_nr[which(m_nr == m1| m_nr == m2)] <- i
    }
    # when k clusters stop and return clusters
    vals <- unique(m_nr)
    if (length(vals)==k){
      clusts <- seq(1,k)
      for (s in 1:k){
        l <- which(m_nr==vals[s])
        res[l] <- s
      }

    }


  }
  return(res)
}

hcluster <- function(X,k,dist_method=c("Euclidian","L1","max"),
                     linkage=c("max","min","mean")){

  dist_fn <- switch (match.arg(dist_method),
                      Euclidian = function(x){dist(x,method="euclidean")},
                     L1 = function(x){dist(x,method="manhattan")},
                     max = function(x){dist(x,method="maximum")}
  )

  d <- as.matrix(dist_fn(X)) # get distance matrix



  linkage_fn <- switch(match.arg(linkage),
                      max = max,
                      min = min,
                      mean = mean)


  N <- nrow(d)

  diag(d) <- Inf # set distance to self to infinity

  grpMem <- -(1:N) #remember group assignment
  m <- matrix(0,nrow=N-1,ncol = 2) # merge at each step
  h <- rep(0,N-1) # min dist at each step
  for (j in 1:(N-1)){
    h[j] <- min(d) # get min dist
    # get its location
    i <- which(d==h[j],arr.ind=TRUE)[1,,drop=FALSE]
    p <- grpMem[i]
    p <- p[order(p)]
    m[j,] <- p # add to merge matrix

    grp <- c(i,which(grpMem%in%grpMem[i[1,grpMem[i]>0]]))
    grpMem[grp] <- j # assign all same group mem




    r <- apply(d[i,], 2, linkage_fn) # apply linkage

    # update dist matrix
    d[min(i),] = d[,min(i)] <- r
    d[min(i),min(i)]        <- Inf
    d[max(i),] = d[,max(i)] <- Inf
  }
  return(clusterCut(m,k)) # return clusters

```

```r
}


res<-kmeans(q1DataS[,2:31])

# dont know which cluster is which so get which ever has highest acc
acc1 <- sum(res$cluster==((q1Data[,1]=="B")+1))/nrow(q1Data)

acc2 <- sum(res$cluster==((q1Data[,1]=="M")+1))/nrow(q1Data)
if (acc2<acc1){
  acck <-acc1
}else{
  acck <-acc2
}
sprintf("kmeans accuracy: %s",acck)
# test different dist and linkage funcs
acc <- data.frame()
it<-1
for (i in c("Euclidian","L1","max")){
  for (s in c("max","min","mean")){
    res<-hcluster(q1DataS[,2:31],2,i,s)
    acc1 <- sum(res==((q1Data[,1]=="B")+1))/nrow(q1Data)
    acc2 <- sum(res==((q1Data[,1]=="M")+1))/nrow(q1Data)
    if (acc1<acc2){
      acc[it,c("Accuracy","Parameters","Model Type")] <- list(accuracy=acc2,x=paste("
    dist:",i,"linkage:",s),Model="H Cluster")
    }else{
      acc[it,c("Accuracy","Parameters","Model Type")] <- list(accuracy=acc1,x=paste("
    dist:",i,"linkage:",s),Model="H Cluster")
    }

    it <- it+1
  }
}
sprintf("Hcluster accuracy: %s",max(acc$Accuracy))

acc[nrow(acc)+1,c("Accuracy","Parameters","Model Type")] <- list(acck,"Kmeans","Kmeans")
# plot
ggplot(data = acc,mapping = aes(y=Accuracy,x=Parameters,group=`Model Type`,fill=`Model
    Type`)) + geom_col() +
  ggtitle("Accuracy of different classifiers") + theme(axis.text.x = element_text(angle
    =90,hjust = 1)) +
  do.call(ggsave,c("q1Class.png",defaultPlotSettings))




set.seed(0)

# get data and shuffle and scale
q2Data <- read.csv(paste0(path,"dataQ2.csv"))
n<-nrow(q2Data)
q2Data <- q2Data[sample.int(n),]
q2DataS <- scale(q2Data)

ggplot(data = q2Data,aes(x=V1)) +
  geom_histogram(stat="bin") + xlab("Year") +
  ylab("Count") + ggtitle("Histogram of year") +
  do.call(ggsave,c("q2Hist.png",defaultPlotSettings))



# get correlations
co <- cor(q2Data)

df <- reshape2::melt(co)

df$Var1 <- sapply(df$Var1,substring,2)
df$Var2 <- sapply(df$Var2,substring,2)

ggplot(data=df,aes(x=Var1,y=Var2,fill=value))+
  geom_tile() + geom_text(aes(Var2, Var1, label = round(df$value,2)), color = "black",
    size = 1) +
  guides(fill = guide_colorbar(title = "Corr")) + xlab("") + ylab("") + ggtitle("Correlation heatmap") +
  scale_fill_gradientn(colours = topo.colors(10)) + do.call(ggsave,c("q2Cov.png",defaultPlotSettings))

# IQR

i <- cbind(1:ncol(q2Data),apply(q2Data, 2, IQR))
colnames(i) <- c("Variable","IQR")
ggplot(data=as.data.frame(i),aes(x=Variable,y=IQR))+
  geom_col() + coord_flip()+ ggtitle("IQR of the variables") +
  do.call(ggsave,c("q2Iqr.png",defaultPlotSettings))


regTreeRec<-function(data,ind,feMap=NULL,minMSE=0){

  if(is.null(feMap)){ # mappng of features
    fMap_fn<-function(x){x}
  }else{
    fMap_fn<-function(x){feMap[x]}
  }


  data <- data[ind,] # get portion of data
  y <- data[,1] # get y
  N <- nrow(data)

  n <- ncol(data)
  data <- data[,2:n]
  n <- n-1


  # search all s for all ps and get mses
  gridSearch <- apply(data,2,function(x){unique(x)})
  mses <- matrix(NA,nrow=2,ncol=n)
  for (x in 1:n){
                                      dataT <- data[,x]


                                      err <- Inf
                                      for (i in gridSearch[[x]]){


                                        b <- which(dataT<=i)
                                        l <- length(b)



                                        yL <-y[b]
                                        yG <-y[setdiff(1:N,b)]
                                        if (l==N){
                                          res <- sum((yL-sum(yL)/l)^2)
                                        }else{
                                          res <- sum((yG-sum(yG)/(N-l))^2)+sum((yL-sum(yL)/l)^2)
                                        }

                                        if (res<=minMSE){
                                          return(list(res=list(p=NA,s=NA),i1=list(),i2=list(),mseV=0,y=sum(y)/N))
                                        }


                                        if (res<err){
                                          err <- res
                                          s<-i
                                        }


                                      }
                                      mses[,x] <- c(s,err)
  }
  # select one with lowest mse
  splitParam <- which.min(mses[2,])[1]

  splitParamS <- mses[[1,splitParam]]


  res <- list(p=fMap_fn(splitParam),s=splitParamS)
  # get child indicies
  i1 <- which(data[,splitParam]<=splitParamS)
  i2 <- which(data[,splitParam]>splitParamS)

  return(list(res=res,i1=i1,i2=i2,mseV=mses[2,splitParam]/N,y=NA))




}

regTree<- function(data,fMap=NULL){
  it <- 1
  # create queues
  queueL <- list(FALSE)
  queueP <- list(0)
  queue <- list(1:nrow(data))

  res <- data.frame()

  N <- ncol(data)-1

  # initial mse
  initialMSE <- sum((data[,1]-sum(data[,1])/nrow(data))^2)/nrow(data)

  nodes <- 0

  while (length(queue)!=0){
    # iterate till none in queue


    l <- queueL[[1]]
    p <- queueP[[1]]
    cu <- max(nodes)+1
    nodes <- c(nodes,cu)
    temp <- regTreeRec(data,queue[[1]],fMap)

    res[it,c("p","s","less","prev","current","mse","y")] <- list(temp$res$p,temp$res$s,l
    ,p,cu,temp$mseV,temp$y)




    if(is.na(temp$y)){
      if (length(temp$i1!=0)){
        qlen <- length(queue) + 1
        queue[[qlen]] <- temp$i1
        queueL[[qlen]] <- TRUE
        queueP[[qlen]] <- cu
      }
      if (length(temp$i2!=0)){
        qlen <- length(queue) + 1
        queue[[qlen]] <- temp$i2
        queueL[[qlen]] <- FALSE
        queueP[[qlen]] <- cu
      }

    }



    queue[[1]] <- NULL
    queueL[[1]] <- NULL
    queueP[[1]] <- NULL
    it <- it + 1

  }
```

```r

  return(list(res=res,imse=initialMSE))


}


testTree <- function(data,tree){
  # get predictions from a tree
  pred <- rep(NA,nrow(data))

  for (i in 1:nrow(data)){
    x <- data[i,]
    currentStep <- tree[1,]
    while(is.na(currentStep$y)){
      val<-x[currentStep$p]
      if(val<=currentStep$s){
        left <-TRUE
      }else{
        left<-FALSE
      }
      currentStep <- tree[which(tree$prev==currentStep$current&tree$less==left),]
    }
    pred[i] <-currentStep$y
  }
  return(pred)



}

featureImp<- function(data,res,features){
  # get feature imp
  pred <- testTree(data,res)

  y <- data[,1]

  err <- mean((y-pred)^2)

  permutation <- sample.int(length(y))


  N <- ncol(data)-1
  errP <- rep(err,N)
  for (i in features){ # permutate each column

    permData <- data
    permData[,i] <- permData[permutation,i]
    temp <- testTree(permData,res)
    errP[i] <- mean((y-temp)^2)
  }

  return(errP-err)
}




ranForest <- function(dataTrain,dataTest,ntrees=5,mtry=NULL){

  varP <- ncol(dataTrain)-1
  if (is.null(mtry)){
    mtry <- floor(varP/3)
  }

  res <- c()
  fi<-c()
  testMSE<-c()
  set.seed(0)

  for (i in 1:ntrees){ # fit each tree storing mses and calculating fi
    print(sprintf("Tree %s/%s",i,ntrees))
    # bagging
    bag <- sample.int(nrow(dataTrain),replace = TRUE)
    outBag <- setdiff(1:nrow(dataTrain),bag)
    boot <- dataTrain[bag,]
    features <- sample.int(varP,mtry)

    temp <- regTree(boot[,c(1,features+1)],features+1)

    res[[i]] <- temp$res
    pred <- sapply(sapply(res, function(x){testTree(dataTest,x)}), mean)
    testMSE[[i]] <- mean((dataTest[,1]-pred)^2)

    fi[[i]] <- featureImp(dataTrain[outBag,],temp$res,features+1) # calculate feature
    importance

  }
  fi <- Reduce(cbind,fi)
  fi <- apply(fi,1,mean)


  return(list(mseSeries=testMSE,fi=fi))
}
# split test and train data
n <- nrow(q2Data)
trainInd <- 1:floor(n*0.8)
testInd <- setdiff(1:n,trainInd)

trainData <- q2Data[trainInd,]
testData <- q2Data[testInd,]

res<-ranForest(trainData,testData)

fi <- res$fi
ind <- order(fi,decreasing = T)
fi <- fi[ind]
fi <- as.data.frame(cbind(ind,fi))
colnames(fi) <- c("Parameter","Importance")
# plot fi
ggplot(data = fi,aes(x=Parameter,y=Importance)) +
  geom_col() + ggtitle("Feature Importance") + coord_flip() +
  do.call(ggsave,c("fi.png",defaultPlotSettings))

# get best forest
mses <- unlist(res$mseSeries)

bestRF <- min(mses)
nT <- which(mses==bestRF)

testMSEs <- list(Model=sprintf("Random Forest n=%s",nT),"Test MSE"=bestRF)


df <- as.data.frame(cbind(seq(1,length(mses),1),mses))
names(df)<-c("Number of trees","Test MSE")
# plot mse for increaing forest size
ggplot(data = df,aes(x=`Number of trees`,y=`Test MSE`)) +
  geom_line() + ggtitle("Test MSE as forest size increases") +
  do.call(ggsave,c("forestSize.png",defaultPlotSettings))


# activation functions
relu <- function(x,d=F){
  ret <- x
  l0 <-which(x<0,arr.ind = T)
  if (!d){
    ret[l0] <-0

    ret

  }else{
    ret[l0] <- 0
    ret[which(x>=0,arr.ind = T)] <- 1

    ret

  }

}

lRelu<- function(x,ep=0.01,d=F){
  ret <- x
  l0<-which(x<0,arr.ind = T)
  if (!d){
    ret[l0] <-ep*x[which(x<0)]
    ret
  }else{
    ret[l0] <- ep
    ret[which(x>=0,arr.ind = T)] <- 1

    ret

  }

}


elu <- function(x,a=1,d=F){

  ret <- x
  l0<-which(x<0,arr.ind = T)

  if (!d){
    ret[l0] <- a*(exp(x[l0])-1)
    ret


  }else{
    ret[l0] <- a*exp(x[which(x<0)])*x[l0]
    ret[which(x>=0,arr.ind = T)] <- 1
    ret

  }

}

idF<- function(x,d=F){
  if(!d){
    x
  }else{
    matrix(1,nrow = nrow(x),ncol=ncol(x))
  }

}



# get output of layer
layer_output <- function(X,w,b,activation){
  input <- X %*% w + b
  activation(input)

}


# back propigate
backProp <- function(xy,X,y,ws,bs,activ,lr=0.1){

  n <- length(ws)



  err <- list()
  slope <- list()

  for (i in rev(1:n)){ # get losses
    if (i==n){
      err[[i]] <- (y-xy[[i]])/length(y)
      slope[[i]] <- err[[i]]*activ[[i]](xy[[i]],d=T)
```

```r
    }else{


      err[[i]] <- ws[[i+1]]%*%t(slope[[i+1]])
      slope[[i]] <- t(err[[i]])*activ[[i]](xy[[i]],d=T)
    }}
  for (i in 1:n){ # update weights
    if (i==1){
      s<-X
    }else{
      s<-xy[[i-1]]
    }
    ws[[i]] <- ws[[i]] + lr * (t(s)%*%slope[[i]])
    bs[[i]] <- bs[[i]] + lr*slope[[i]]
  }




  return(list(w=ws,b=bs))

}




multiPerc <- function(X,y,testX,testY,hidden_layers,activation_fns,batch_size=64,epochs
    =200,lr=10e-5){
  y <- as.matrix(y,byrow=F)
  testY<- as.matrix(testY,byrow=F)
  input_dim <- ncol(X)
  hidden_layers <- c(hidden_layers,1)
  n <- length(hidden_layers)
  w <- c()
  b <- c()

  # initialise
  for (i in 1:n){
    neurons <- hidden_layers[[i]]
    w[[i]]<- matrix(runif(input_dim*neurons,-1,1),input_dim,neurons)
    bias_in <- runif(neurons)
    bias_in_temp <- rep(bias_in,batch_size)
    b[[i]]<-matrix(bias_in_temp, nrow = batch_size, byrow = FALSE)
    input_dim <- hidden_layers[[i]]

  }

  N <- nrow(X)
  N2 <- nrow(testX)

  maxB<- floor(N / batch_size)
  maxBT <- floor(N2/batch_size)
  N2 <- maxBT*batch_size

  testMSE <- rep(NA,epochs)
  best <- Inf

  for (i in 1:epochs){ # train loop
    print(sprintf("EPOCH: %s/%s",i,epochs))
    ind <- sample.int(N) # random batches

    for (s in 1:maxB) { # for each batch
      i1 <- (s-1)*batch_size +1
      i2 <- i1 + batch_size - 1
      xy <- list()
      for (l in 1:n){ # forward
        if(l==1) {
          input <- as.matrix(X[ind[i1:i2],])
        }else{
            input <- xy[[l-1]]
}
        xy[[l]] <- layer_output(input,w[[l]],b[[l]],activation_fns[[l]])
      }
      # back propagate
      temp <- backProp(xy,as.matrix(X[ind[i1:i2],]),y[ind[i1:i2]],w,b,activation_fns,lr/
    maxB)
      # update
      w <- temp$w
      b <- temp$b
    }
    temp <- rep(NA,N2)
    for (s in 1:maxBT) { # get test loss for current epoch
      i1 <- (s-1)*batch_size +1
      i2 <- i1 + batch_size - 1
      xy <- list()
      for (l in 1:n){
        if(l==1) {
          input <- as.matrix(X[ind[i1:i2],])
        }else{
          input <- xy[[l-1]]
        }
        xy[[l]] <- layer_output(input,w[[l]],b[[l]],activation_fns[[l]])
      }
      temp[i1:i2] <- (y[ind[i1:i2]]-xy[[length(xy)]])^2
    }
    testMSE[i] <- mean(temp)
    print(sprintf("Test loss: %s",testMSE[i]))

    if (testMSE[i] < best){ # store best model
      bestModel <- list(w=w,b=b,epoch=i)
    }




  }


  return(list(testMSE=testMSE,bestModel=bestModel))


}

trainData <- q2DataS[trainInd,]
testData <- q2DataS[trainInd,]

X<- trainData[,2:ncol(trainData)]
y<- trainData[,1]
testX <- testData[,2:ncol(testData)]
testY <- testData[,1]

hidden_layer_sizes <- list(128,64)
number_hidden_layers <- c(1,2,3)
activ_fns <- list(idF,lRelu,elu)
output_fns <- list(idF)



# search performance of different hyper parameters
param_grid<-expand.grid(hidden_layer_sizes,number_hidden_layers,activ_fns,output_fns)
act <- function (i) c("Identity","lRelu","ELU")[sapply(activ_fns, identical,param_grid[[
    i,3]])]

n <- ncol(q2Data)
modRes <- data.frame()
best <- Inf
for (i in 1:nrow(param_grid)){
  number_hidden_layers <- param_grid[i,2]
  hidden_layer_sizes <- rep(param_grid[i,1],number_hidden_layers)
  act_fns <- c(rep(param_grid[i,3],number_hidden_layers),param_grid[i,4])
  res<-multiPerc(X,y,testX,testY,hidden_layer_sizes,act_fns)
  mmse <- min(res$testMSE)
  modRes[i,c("nH","sH","aF","mMSE")] <- c(param_grid[i,2:1],act(i),mmse)
  if (mmse<best){
    best<-mmse
    bestModel<-res$bestModel
    bestTestSeries <- res$testMSE
    bestModelParam <- modRes[i,c("nH","sH","aF","mMSE")]
  }
}
# plot loss over epochs
df <- as.data.frame(cbind(seq(1,200,1),bestTestSeries))
colnames(df) <- c("Epoch","Test MSE")
ggplot(data=df,aes(x=Epoch,y=`Test MSE`)) +
  geom_line() + ggtitle("Test error over training for the best performing model")+
  do.call(ggsave,c("nnEpoch.png",defaultPlotSettings))


df <- data.frame("Model"=c(testMSEs$Model,sprintf("Multilayer Perceptron H=%s,S=%s,A=%s"
    ,
                                                      bestModelParam[[1]],bestModelParam
    [[2]],bestModelParam[[3]])),
                 "Test MSE"=c(testMSEs$`Test MSE`,bestModelParam[[4]]))


# plot vs random forest
ggplot(data = df,aes(x=Model,y=Test.MSE,fill=Model))+
  geom_col() + ggtitle("Comparison of test MSE") +
  theme(legend.position = "none") + ylab("Test MSE") +
  do.call(ggsave,c("q2Comp.png",defaultPlotSettings))


q3Data <- read.csv(paste0(path,"dataQ3.csv"))
q3DataS <- q3Data

set.seed(0)

n <- nrow(q3Data)
sam <- sample.int(n)
d <- q3Data[sam,]
b <- floor(n*0.2)

# kernels
kern1 <- function(x1,x2,k){
  temp<-x1* as.double(k)
  temp %*%t(x2)
}
kern2 <-function(x1,x2,k){

  m  <- nrow(x1); n <- nrow(x2)
  x1x2 <- x1 %*% t(x2)    # (m,n)-matrix
  x1x1 <- matrix( rep(apply(x1*x1, 1, sum), n), m, n, byrow=F )
  x2x2 <- matrix( rep(apply(x2*x2, 1, sum), m), m, n, byrow=T )


  s <- abs(x1x1 + x2x2 - 2*x1x2)

  exp(-s/(2*as.double(k)^2))
}

kern3<-function(x1,x2,k){

  as.double(k[5])*kern1(x1,x2,k[3])+as.double(k[4])*kern2(x1,x2,k[1])*kern4(x1,x2,k[1],k
    [2])
}


kern4 <- function(x1,x2,k,p=4){
  m  <- nrow(x1); n <- nrow(x2)

  x1x1 <- matrix( rep(apply(x1, 1, sum), n), m, n, byrow=F )
  x2x2 <- matrix( rep(apply(x2, 1, sum), m), m, n, byrow=T )
  s<-x1x1-x2x2

  exp(-(2*sin(pi*s/as.double(p))^2)/(as.double(k)^2))
}


# gaussian process
```

```r
gaussianProc <- function(x1,x2,y,ker,k,sigma_n){
  x1 <- cbind(1,x1)
  x2 <- cbind(1,x2)
  y <- as.matrix(y,by.row=F)

  k1 <- ker(x1,x1,k)
  k2 <- ker(x2,x1,k)
  k3 <- ker(x2,x2,k)
  temp <- k2%*%solve(k1+sigma_n^2*diag(nrow(x1)))
  posteriorMean <- temp %*% y
  posteriorCov <- k3 - temp %*% t(k2)

  posteriorSD <- sqrt(abs(diag(posteriorCov)))

  return(list(m=posteriorMean,sd=posteriorSD))


}
y <- q3Data[,2]
x1 <- q3Data[,1]
lastDate <- max(x1)

s<- as.matrix(seq(2,13,1),by.row=F)
k <- list(s,s,expand.grid(s,s,2,2,1))

it <- 1
# test grid of parameters
mses <- list()
fullM <- rep(list(data.frame()),3)
allKernels <- list(kern1,kern2,kern3)
for (i in allKernels){
  best <- Inf
  for (l in 1:nrow(k[[it]])){
    m <- rep(NA,5)
    for (s in 1:5){
      i1 <- (s-1) * b + 1
      i2 <- s * b
      temp <- setdiff(1:n,i1:i2)
      pred <- gaussianProc(d[temp,1],d[i1:i2,1],d[temp,2],i,k[[it]][l,],2)
      m[s] <- mean((d[i1:i2,2]-pred$m)^2)
    }
    s <- which.min(m)
    m <- mean(m)
    if (m<best){
      best <- m
      mses[[it]] <- list(m=m,k=k[[it]][l,],s=s)
    }
    if (it!=3){
      fullM[[it]][l,c("mse","k")] <- list(m,k[[it]][l])
    }else{
      p<-k[[it]][l,]
      fullM[[it]][l,c("mse","k1","k2","k3","k4")] <- list(m,p[[1]],p[[2]],p[[3]],p[[4]])
    }

  }
  it <- it + 1
}
# plot
df <- sapply(mses, function(x) x$m)
df <- cbind(c("Kernel 1","Kernel 2","Kernel 3"),df)
colnames(df) <- c("Kernel", "MSE")

ggplot(data=as.data.frame(df),aes(x=Kernel,y=MSE,fill=Kernel))+
  geom_col() + ylab("Avg Test MSE") + ggtitle("Avg Test Error of Kernels across 5-folds"
    ) +
  theme(legend.position = "none") +
  do.call(ggsave,c("kernel.png",defaultPlotSettings))

param <- lapply(1:length(mses), function(x) list(s=mses[[x]]$s,k=mses[[x]]$k))
x2 <- c(q3Data[,1],seq(lastDate+1,lastDate+15,1)) # predict full series and up to march
    2011 for plotting
pred <- rep(list(data.frame(t=x2)),3)
for (i in 1:3){
  s <- param[[i]]$s
  i1 <- (s-1) * b + 1
  i2 <- s * b
  temp <- setdiff(1:n,i1:i2)
  pred[[i]][,c("pred","sd")] <- gaussianProc(d[temp,1],x2,d[temp,2],allKernels[[i]],param[[i]]$k,1)
  pred[[i]][,c("Kernel")] <- paste("Kernel",i)
}

# convert months to date objects
convertToTimes <- function(x){
  month <- x %% 12
  year <- floor(x/12) + 1960
  month[which(month==0)] <- 12
  as.Date(paste0("01/",month,"/",year),format = "%d/%m/%Y")
}


allSeries <- Reduce(rbind,pred)
# get 95% ci
allSeries$hi <- allSeries$pred + 1.96*allSeries$sd
allSeries$lo <- allSeries$pred - 1.96*allSeries$sd

df <- q3Data
colnames(df) <- c("t","pred")
df$Kernel <- "True"
for (i in colnames(allSeries)[which(!colnames(allSeries)%in%colnames(df))]){
  df[[i]] <- NA
}

allSeries <- rbind(df,allSeries)

allSeries$t <- convertToTimes(allSeries$t)

ggplot(data=allSeries,aes(x=t,y=pred,group=Kernel,color=Kernel,ymin=lo,ymax=hi)) +
  geom_line() + geom_ribbon(alpha=0.1) + ggtitle("Predicted and projected temps for all models with 95% CI") +
  xlab("Date") + ylab("Temperature") + do.call(ggsave,c("allTS.png",defaultPlotSettings))


ind <- sapply(c("Kernel 1","Kernel 2", "Kernel 3"), function(x){
  which(allSeries$Kernel==x&allSeries$t=="2011-03-01")

})


pred<-allSeries[ind,]


ggplot(data = pred,aes(x=Kernel,y=pred,ymin=lo,ymax=hi,color=Kernel)) +
  geom_pointrange()+ theme(legend.position = "none")+
  ylab("Predicted Temperature") +
  ggtitle("Predicted temperature for March 2011 by Gaussian Processes") +
  do.call(ggsave,c("q3pred.png",defaultPlotSettings))
```