COMP 652 Machine Learning Coursework 1 : Logistic Regression for Automatic Assessment of Rehabilitative Speech Treatment in Parkinson's Disease

Riashat Islam
McGill University
Reasoning and Learning Lab
riashat.islam@cs.mcgill.ca

February 1, 2017

1 Introduction

In this work, we investigate the potential of using a logistic regression machine learning classifier to automatically evaluate the sustained vowel phonations as being acceptable or unacceptable, such as to improve the effectiveness of rehabilitative speech treatment for LSVT Companion system. In the sections below, we analyse the effectiveness of a logistic regression classifier.

2 Section 1

2.1 Question A and B

We have split the data randomly using the MATLAB dividerand function, for 60% as training set, 20% as validation set and the rest 20% for test set. In order to include bias, the a feature constant input of 1 is also added to the data.

The dataset consists of 126 samples in total, with 311 maximally informative features. The original dataset consists of 314 dimensions, and at first we investigate the correlations between some of the features in the dataset as per the diagrams below.
We then investigate how some of the features look like in the 2D dimension. We find that even for two of the features among 313, there is no linear separation between the features, which suggests that a linear classifier would perform poorly. In other words, we might not be able to find a single plane or class boundary for this dataset that would linearly separate the data points with two class labels.

Cross-Validation : For our experiments below, we use a 5-fold cross validation method in order to find suitable parameters for the classifier. The 5-fold CV method ensures that all the data points are selected at every interval of CV, such that from the list of training points, the validation points are selected uniformly. The rest set of 20% of the data is always kept separated.

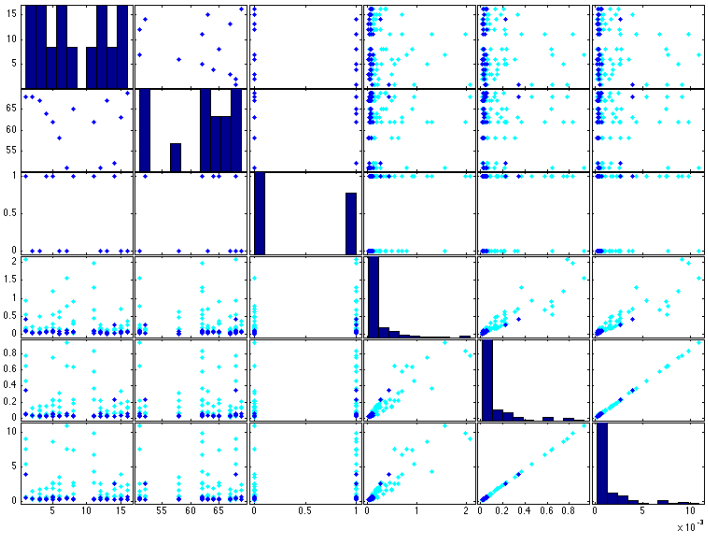

Figure 1: Data correlations of First Six Features

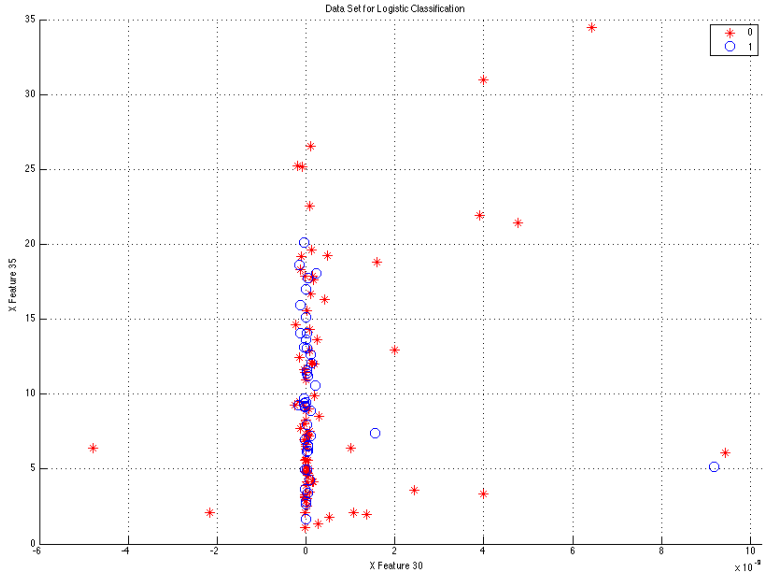

Figure 2: Visualisation of Data Features 30 and 35

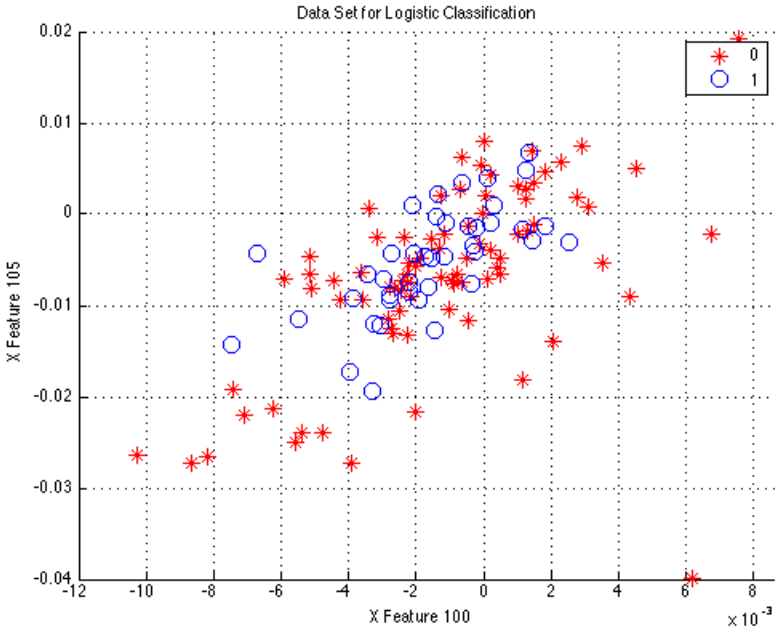

Figure 3: Visualisation of Data Features 100 and 105

2.2 Question C

Logistic Regression with L2 Regularision:

We add a L2 regulariser to the logistic regression classifier. Derivations are as shown below. We also show the gradient ascent update rule of the logistic regression classifier.

Given that the likelihood of the datapoints is $P(\mathbf{y}|X, w)$, we denote the log likelihood as **L(w)**.

$$L(w) = \log P(y|x, w) = \sum_{i=1}^{N} \log P(y|x, w) \tag{1}$$

Hence the cross entropy function with L2 regularision can be written as :

$$L(w) = -(\sum_{i=1}^{m} y_i \log h(x_i) + (1 - y_i) \log(1 - h(x_i))) + \lambda \|w\|^2 \tag{2}$$

where $h(w) = \frac{1}{1+exp(-w^T x)}$ is the sigmoid hypothesis function. By taking derivatives of $L(w)$, to get the gradient of the log likelihood $\frac{d(L(w))}{dw}$ as follows:

$$\frac{dL}{dw} = -\sum_{m} [\mathbf{y} h_w(\mathbf{x})\mathbf{x} - (1 - \mathbf{y}) h_w(\mathbf{x})\mathbf{x}] \tag{3}$$

The gradient ascent update rule can be written as

$$w \leftarrow w + \alpha[\sum_{i=1}^{m} (y_i - h_w(x_i)) * x_i] - \alpha\lambda * w \tag{4}$$

where α is a step size parameter or learning rate of the gradient ascent update rule. **NOTE :** For our work, we fine-tune the α parameter using a trial and error method by analysing the training and validation plots of convergence, rather than following any adaptive learning rate rule to fine-tune the α parameter.

2.3 Question D

Experiments with Logistic Regression

For our experiments, we use a 5-fold cross validation method and for each training, validation and test set, the results are averaged over 3 trials. By careful fine-tuning of the α step size parameter, the learning rate was set at a low value of 0.0004 and we test for convergence with gradient ascent update rule for 15000 runs.

Dimensionality Reduction : Since in our dataset, we have only 126 samples and 313 features, in order for logistic regression classifier for the high dimensional inputs, we apply a dimensionality reduction technique. We applied the PCA (Principal Component Analysis) technique for dimensionality reduction, and lowered the input dimensions from 313 to 126 (same as the number of training samples). Experimental results below are therefore carried with the lower dimensional data compared to the original dataset. In later sections, we analyse how using basis functions and kernels,

we can avoid this dimensionality reduction technique and apply the kernelised logistic regression classifier.

The logistic regression weight parameters are updated based on the gradient descent, and by evaluating $J(w)$ for the learnt weights, we can evaluate how the objective or cost function changes with the number of iterations for convergence.

L2 Regularisation:
We add a L2 regulariser to the objective function we are trying to minimize. This is because regularisation can reduce overfitting by adding a complexity penalty to the loss function. Even though regularisaion helps reduce overfitting, but we would still need to pick the λ parameter, which is why we perform cross validation. Regularisaion penalizes hypothesis complexity since L2 regularisation leads to small weights. The λ parameter controls the importance of the regularisation term.

Plot 1: Average Cross Entropy as function of λ:

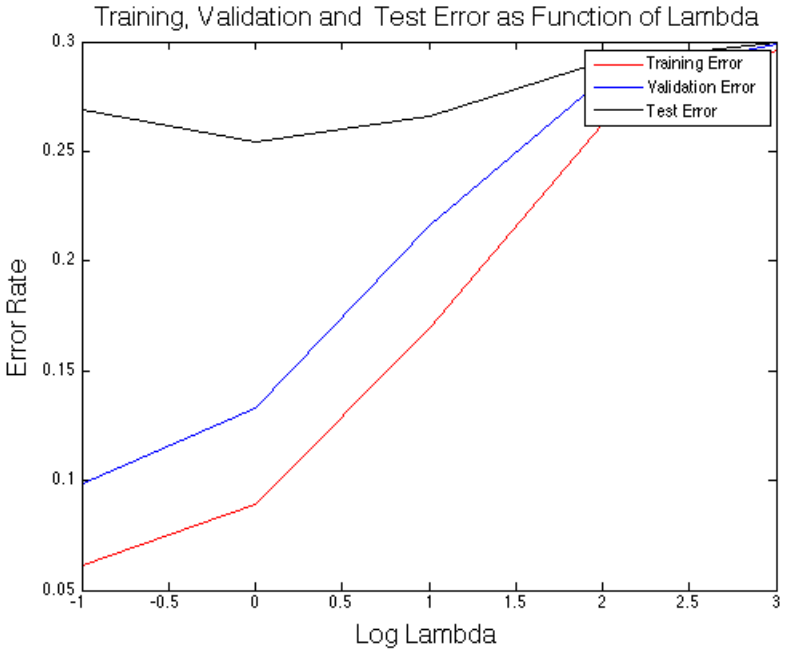

Figure 4: Error Rate vs Lambda Parameters of L2 regularisation

Figure 4 shows the training, validation and test errors with changes in the λ parameters. After analysing further plots below, we will investigate how to choose the λ parameter using a cross validation method for $L2$ regularisation.
Note that, by analysing the training and validation results, we will fine-tune the λ parameter, and select which λ fits best for L2 regularisaion.

Figure 4 shows that as λ increases, the error rates usually increase for all training, validation and

test sets. This is because as λ increases, $w^\intercal w$ decreases. We would therefore need to choose a λ parameter which can minimize the loss function.

Overfitting and Bias/Variance Trade-Off:

From experimental results in figure 4 we analyse for the bias/variance trade-off, and whether the model is overfitting or not. From figure 4 we can say that $\lambda = 1$ (corresponds to 0 on the log scale) is just the right value which balances vias and variance.

A large value of λ corresponds to high bias and the model is likely to underfit. This is because large λ adds a larger complexity penalty term thereby enforcing simpler models to be chosen. Similarly, when the λ parameter is small, this means we are allowing for complex models by adding smaller penalty to complex models. Small values of λ therefore leads to high variance and the model is likely to overfit. Therefore, as shown in figure 4, when λ is large, the high bias leads to small differences in the errors, whereas small λ values corresponds to larger variance and there is a significant differnece in the training and test errors.

Plot 2 : L2 Norm of Weight Vector :

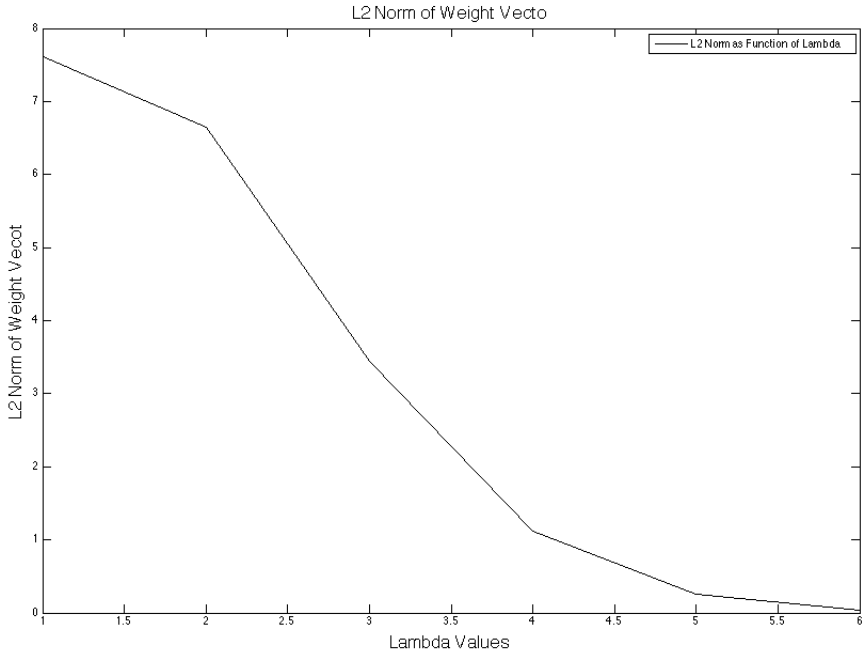

Figure 5: L2 Norm of Weight Vector

Results in figure 5 show that as the lambda parameter is increased, the L2 norm of weights are decreasing. This is mainly because a higher λ paramter regularises the weights more, or in other words disallows the weights to shoot up way too much. A higher λ parameter indicates a greater constrain on the weight vectors from becoming too large.

The large values of λ makes the weights and hence the l2 norm to decrease. This again suggests that high λ allows for underfitting to occur which is why the weight norm values are smaller (since in underfitting these weights are fitting a simple model). Similarly, when λ value was small, we are likely to overfit since we are trying to fit a complex model to the dataset. This makes the weight values to be larger (since fitting a complex hypothesis) which is why the L2 norm value is high for smaller λ values.

Plot 3 : Actual Values of Weight Vector :

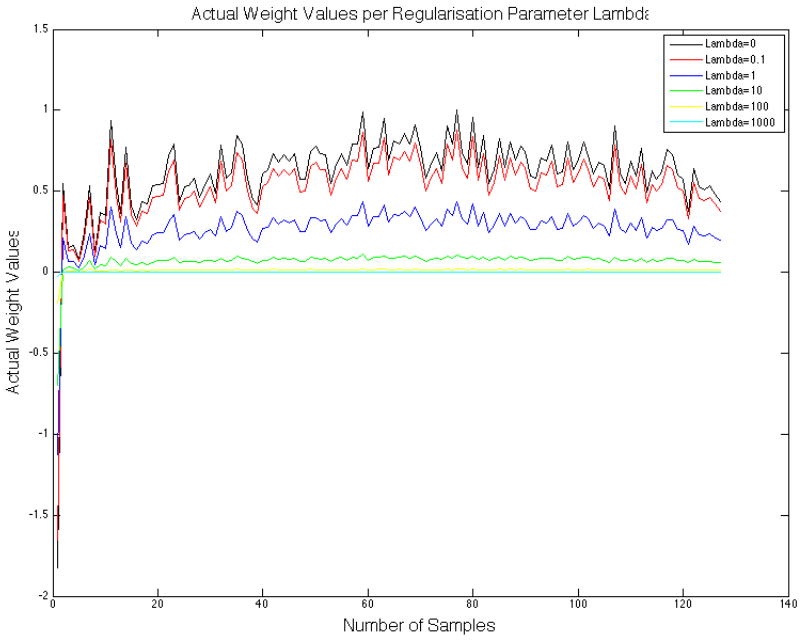

Figure 6: Actual Values of Weight Vector

Figure 6 is in line with figure 5 where it shows that with higher values of λ paramter, the actual values of weights are not allowed to vary too much, whereas for low values of regulariser, the weights are allowed to vary more.

From figure 6, we can analyse for bias and variance. Small λ values corresponds to high variance (overfitting) which is why the actual values of weights have higher variance. The high variance in weights corresponds to the higher variance in error rate (overfitting). Similarly, for small λ values, there is lower variance and higher bias, which is again denoted by the low variance in the actual values of the weights.

Figure 6 also aligns with our discussion as above. For larger λ, the model is likely to underfit which is why there is less variation in the fitted weights (since a simple model can be fit very easily). Again, when λ value is small, we have high variance and this is again suggested by the higher variation in the actual values of the weights.

Plot 4 : Accuracy on Training, Validation and Test Set :

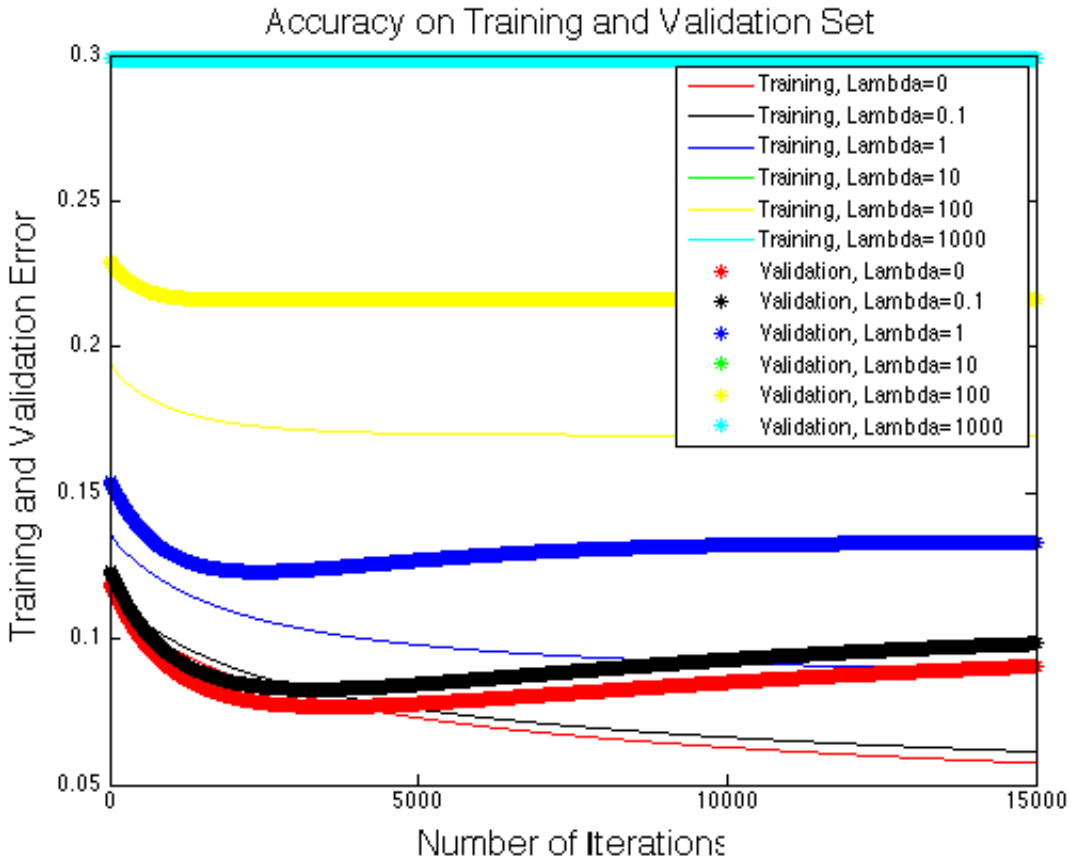

Figure 7: Error on Training and Validation Set

In figure 7 we plot the accuracy (error rate) on the training and validation sets. This is in order to analyse for overfitting and fine-tune the regularisation parameter for l2 regularisation. Figure 7 shows that for $\lambda = 0$ and $\lambda = 0.1$, overfitting occurs. This is because even though with lower iterations, the test error rate decreases, but as the training objective function converges more with larger iterations, the error on the validation set actually increases. This suggests overfitting is occurring with $\lambda = 0$ and $\lambda = 0.1$. Also, as in figure 7, with $\lambda = 10$, $\lambda = 100$, and $\lambda = 1000$, we can see that underfitting is occuring due to higher constraints on the regularisation paramter which results in greater training and validation errors as shown in the plot.

From our results in figure 7, we therefore conclude that the L2 regularisation λ parameter should be set at $\lambda = 1$ where both the training and validation error is low, and there is less increase in test error rate as the training objective function converges more. Our results in this graph are averaged over 3 trials.

Error Rate or Accuracy on Training and Test Set
Figure 7 further justifies for overfitting for complex models. At small λ values, even though the

training and validation errors are low, but with the number of iterations, the validation error actually increases (overfitting). Similarly, when λ parameters are too large, it suggests for a very simple model which leads to high training and validation errors (underfitting).

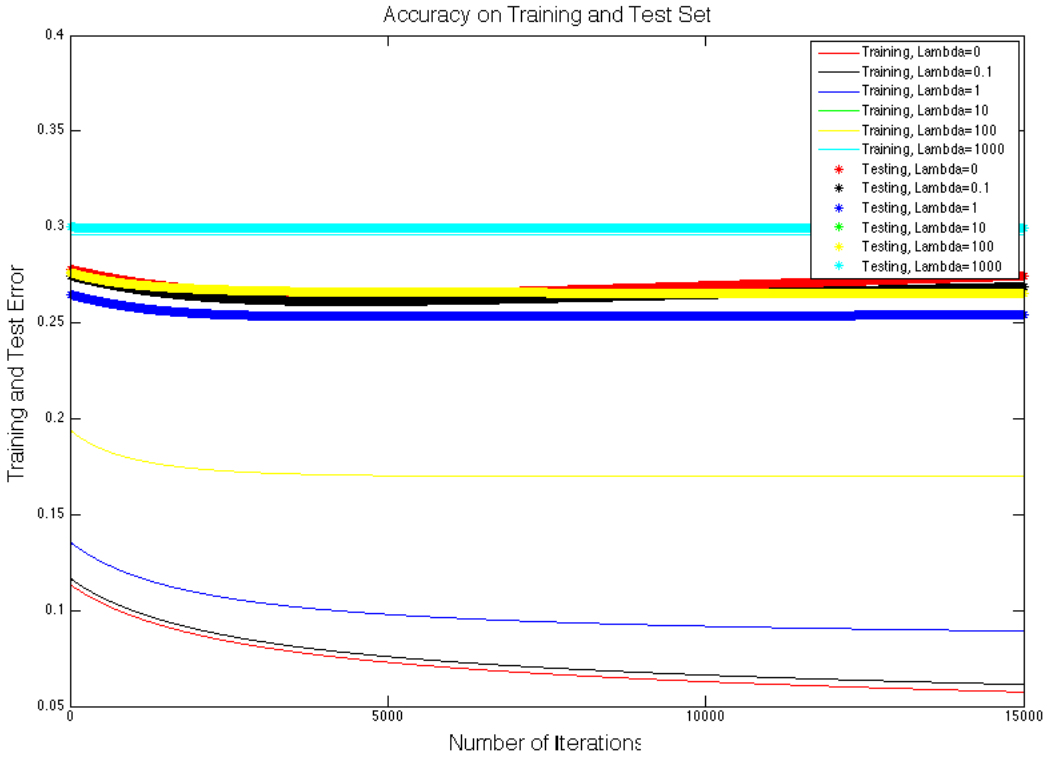

Figure 8: Error on Training and Test Set

Figure 8 shows the difference in training and test errors, again for different values of λ. **NOTE :** We are not choosing the regularisation parameter λ based on the generalisation error (difference between training and testing error). Our results clearly show that with $= 1$ the error rate on the test set is lowest compared to all other parameters. The smallest two values with $\lambda = 0$ and $\lambda = 0.1$ and largest three values of λ at $\lambda = 10$, $\lambda = 100$ and $\lambda = 1000$ results in overfitting and underfitting to occur respectively. The error rate on the rest set is lowest for $\lambda = 1$ which we have chosen based on figure 7.

Final Note: Finally, as shown in figure 9, we show the difference in training and test error for the fine-tuned regularisation parameter. We note that the generalisation error is lowest when the objective function is carefully regularised. We note that the test error is low even with L2 regularisation. This is mainly because the data set is highly non-linear and the inputs are also largely high dimensional. We applied a linear logistic regression classifier on the high dimensional non-linear dataset which results in the low test set error. In later sections, we will analyse how a non-linear kernelised logistic regression classifier may further improve the generalisation performance.

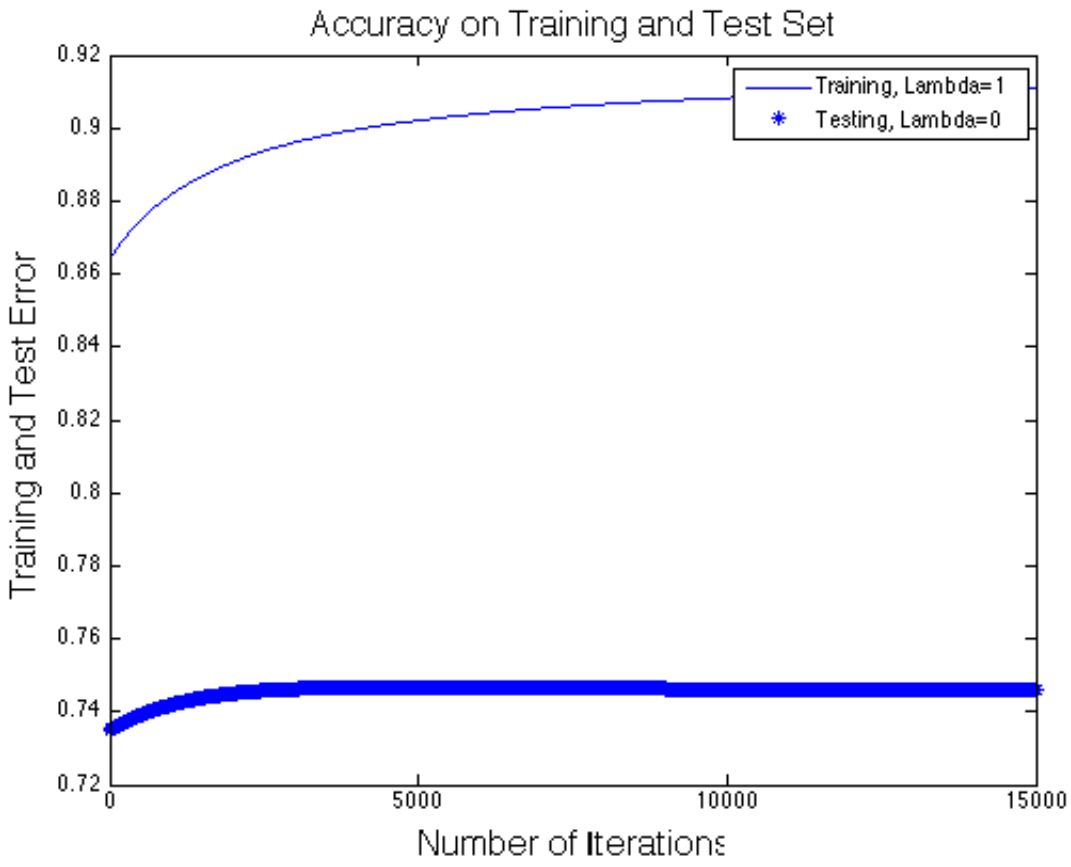

Figure 9: Accuracy on Training and Test Set with Chosen $\lambda = 1$ for L2 Regularisation

2.4 Question E

Gaussian Basis Functions :

In this section, we apply a Gaussian basis function on each of the input variables, with means ranging from -10 to 10. Note that in this section, we **no longer apply a dimensionality reduction technique**, but instead transform the original input features by using a set of Gaussian basis functions denoted as $\Phi(x)$. This can be denoted as $\Phi(x)$ which is a Gaussian function with mean μ. For each mean μ which corresponds to a new feature map input, we also change the variance paramter σ as in the experimental results below. We again apply a 5 fold cross validation method, and take our results for each train, validation and test set averaged over 3 trials.

Each value of μ and σ corresponds to a Gaussian basis function which is used to transform each of the input variables to a feature space. The reason we are transforming data using Gaussian basis functions are as follows:

Modelling Data with Basis Functions:

We want to fit a classifier based on our input data distribution, bur we are not trying to model the distribution of the input data. The basis function corresponds to a non-linear transformation for the input data. The Gaussian basis functions corresponds to local basis functions which are often more appropriate. The μ and σ parameters in Gaussian basis function controls the location and scale of the functions. Since we do not know the true mean of the input distribution, we first experiment with different values of μ which determines where in the feature space the input data will be located. Similarly, we fine tune with different σ values which corresponds to the width of the input data distribution in the feature space.

2.5 Question F

Experiments with Gaussian Basis Functions

For notational simplicity, first we show the training error plots as a function of σ as shown in figure 10 below. By only looking at the training errors as in figure 10 we find that it seems more difficult to fit an objective function and make it converge when the width parameters are high. In other words, less convergence can be achieved of the training objective function when the width of the basis functions is small (corresponds to the input data being concentrated in a particular region in the feature space rather than being distributed over the entire feature space).

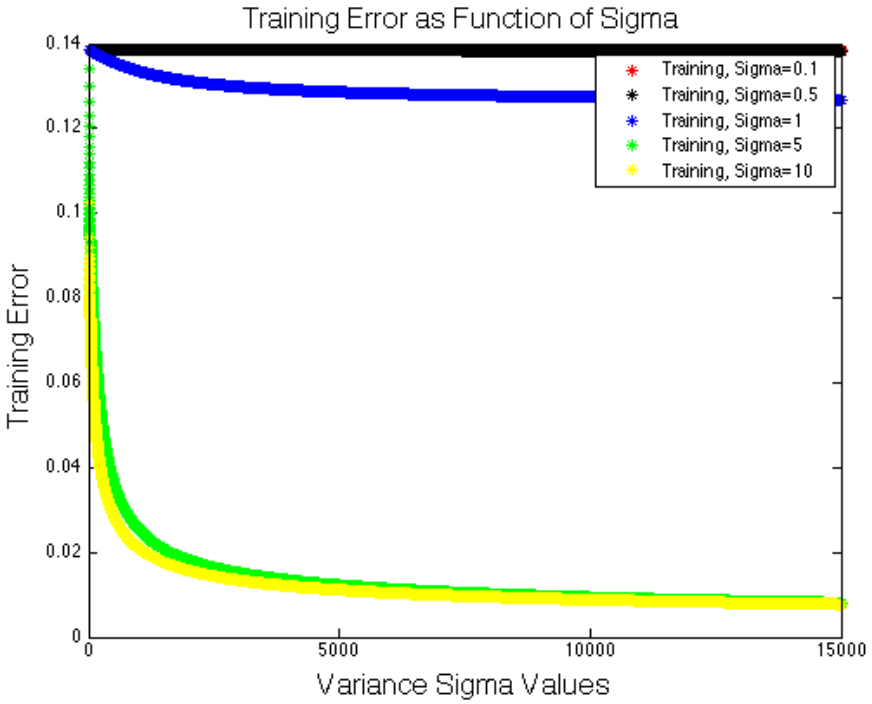

Figure 10: Training Error as a Function of σ (x axis should be number of iterations)

In figure 11 below, we then show the training and test error as a function of σ.

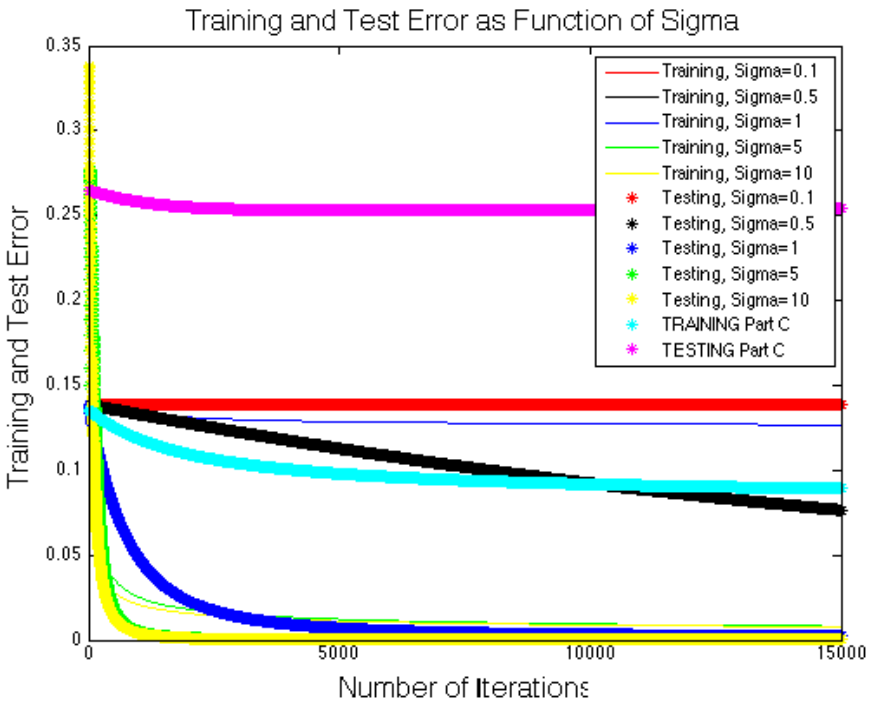

Figure 11: Training and Test Error, compared to Linear Logistic Regression Results from part d

Our results in this section, as in figure 11 show the effect of the variance parameter σ on the basis functions. Results show that compared to the test error from part D based on linear logistic regression, the accuracy is much higher when using basis functions for the input features. This suggests that by transforming the inputs using a feature map, we are able to better find a linear classifier in the feature space.

Based on figure 11, with increasing values of σ, the test error seems to be decreasing. In other words, the more widely distributed the input data is in the feature space, the more easier it seems for us to optimize our training objective function. Therefore, for widely distributed input data in feature space, it seems more easier to find a classifier in feature space and this corresponds to the increasing test accuracy (lower test errors) with increasing values of σ. The generalisation performance improves as the input data is more widely distributed in the feature space, since it is easier to train and optimize the training objective function.

Effect of Sigma on Overfitting and Bias/Variance Trade-off:

Our results as in figure 11 shows that as σ increases in the basis functions, the tendency to overfit decreases. As shown in figure 11, for $\sigma = 0.1$ and $\sigma = 0.5$, the test error is quite high compared to the other values of σ. As we increase the σ parameter, the training error reaches almost close to zero. Also, with relatively small values of sigma such as $\sigma = 0.1$ and $\sigma = 0.5$, the test error is close to what we got for a linear logistic regression, which may also suggest overfitting issues with smaller values of σ.

As σ value is low, the width of basis functions is smaller, and therefore a more complex model is required to fit the distribution (overfitting). Similarly, when the σ values are high, the test error decreases (improved performance) since overfitting decreases as a less complex model would now be required to fit the input data distribution. **Relation with Bias/Variance :** As σ increases, there is therefore lower variance (less overfitting). The lower variance can be interpreted as follows : As width increases, it is much easier to optimise the learning parameters for the objective function so as to learn the input data distribution. Hence due to lower norm values of weights, or less variations of the actual weight values, the variance decreases as σ increases.

We can conclude that even though as σ increases, the test error rate lowers down and performs better than a linear classifier,and the tendency to overfit also decreases. This is in line by analysing the difference between train and test errors for high values of σ, where the test errors are relatively low, and close to the training errors (no sign of test error increasing). However, our suggestion is that for the basis functions, a relatively higher value of σ should be used such that by fine-tuning to avoid overfitting by using cross-validation, we can improve the test set performance of the classifier.

Bias-Variance Tradeoff:

High bias can lead to the model close to underfitting, whereas higher variance may also lead to overfitting. In order to analyse the bias-variance tradeoff, we experiment to see how the error rates changes as a function of σ on a different plot.
Figure 12 shows that as the sigma values increases to almost 5 or 10, the difference between the train and test error significantly decreases. This suggests that the generalisation performance of the classifier improves with increasing values of σ. This may suggest that higher σ values, there is high bias and low variance (reducing overfitting). Then again, with lower values of σ, since the difference in the error rates is much higher, this may also suggest that overfitting is occuring. This is perhaps in line with the previous results as in figure 11, where the test error rate was much higher for smaller values of σ. The low values of σ again corresponds to overfitting because we would need to fit a complex model in the feature space if the input data distribution is concentrated in a single region rather than being spread out.

We can therefore conclude that σ values of 1 or 5 may be suitable for the means of the basis functions. A high value of $\sigma = 10$ makes the model to have high bias but lower variance. Then again, with very small values of sigma, overfitting may occur (as shown in the larger values of test errors), and there may be higher variance and lower bias in the model predictions.

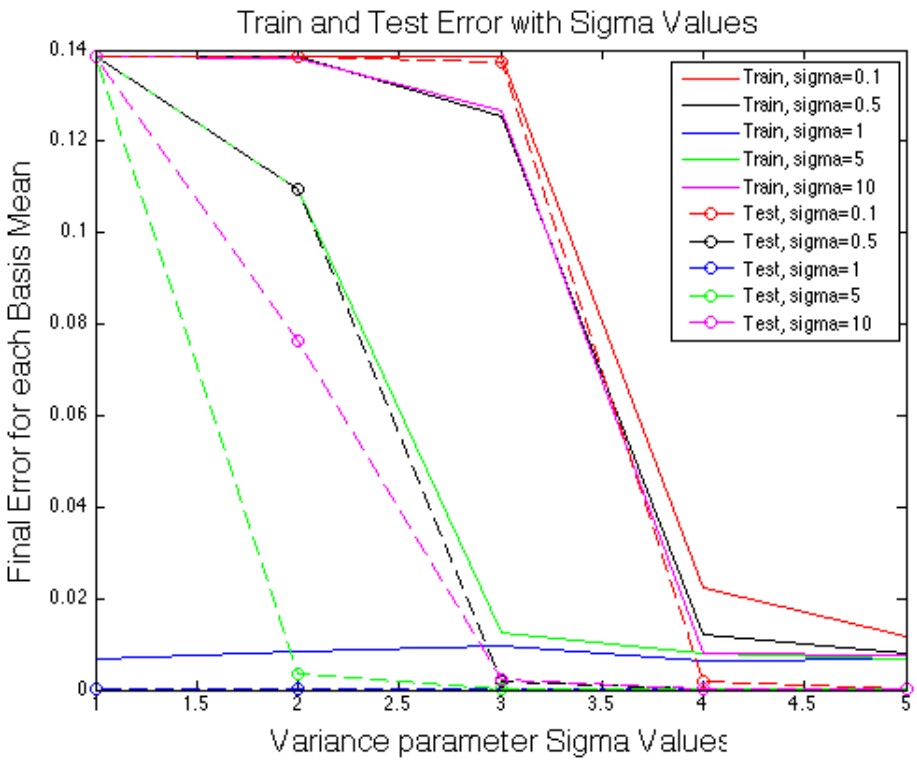

Figure 12: Final value of train and test error as a function of σ in basis function for each μ

2.6 Question G

From the results in Question F, we argued that our best performing value of σ for the basis functions is $\sigma = 5$ or $\sigma = 10$ which balances the bias variance trade-off. We suggested that a larger value of σ would reduce overfitting for us since the error rates (training,test) decreased as σ parameter was increased.

In this section, we first experiment with basis functions with μ and analyse the effect of training and test error when a further L2 regulariser is added to the objective function.

We plot the training, validation and test error as a function of the regularisation parameter λ, in order to first evalate the best performing regularisation parameter, for non-linear logistic regression with all basis functions added together for $\sigma = 10$. Our basis function now is for all the mean values considered (since we do not know the ideal location that the input data distribution should be in feature space), keeping $\sigma = 10$ constant first when adding the basis functions.

Here again, in order to fine tune for regularisation parameter , we would need to analyse for overfitting and bias/variance tradeoff from figure 13.

Figure 13 shows the results of the experiment with changing values of regularisation parameter λ

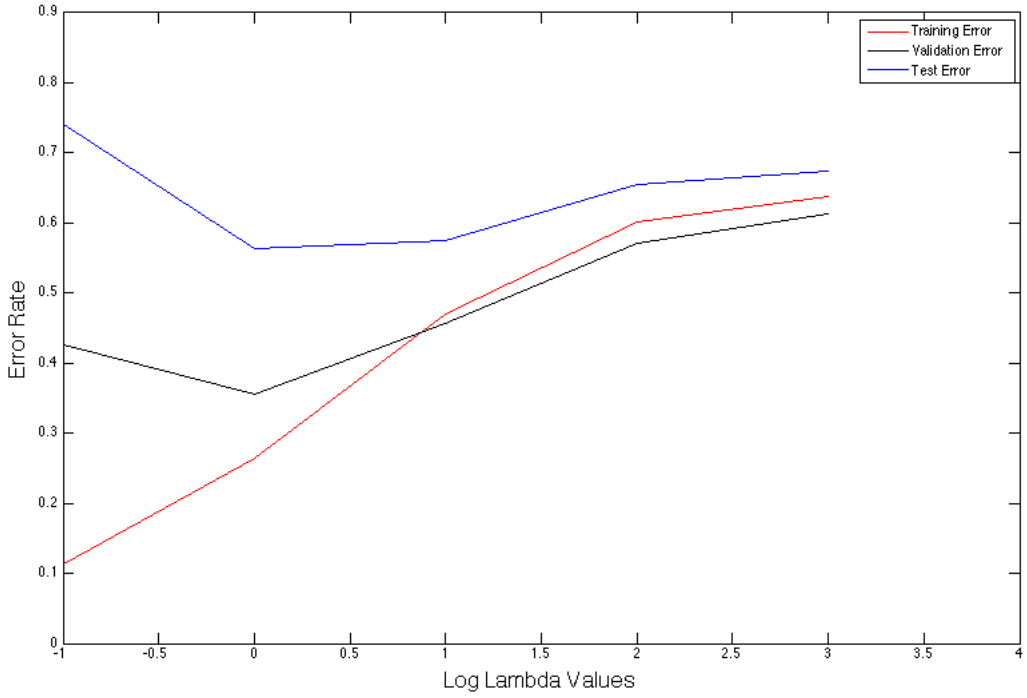

Figure 13: Training, Validation and Test Error as a Function of λ

and it's influence on the error rate. We find that higher values of λ parameter actually corresponds to increasing values of the error rate of the training objective function, and the test error first decreases and then starts increasing as λ value is very high. This can be explained as follows:

Higher values of regularisation parameter in case of basis functions corresponds to tendency to underfit. The best performing result is when the λ parameter is 1, corresponds to the 0 on the figure (since log scale is taken). It also balances the bias variance tradeoff, since with higher λ values, underfitting occurs there is lower variance in the model estimates with higher bias. Similarly, figure 13 also shows that when λ values are low, the model overfits, and has low bias but higher variance.

Therefore, similar to the linear case, and comparing to the case when we have basis functions as inputs, the similar relation of bias variance tradeoff holds. When we increase the λ parameters, the complexity penalty term penalises the model more, and therefore a simple model is more likely to be chosen (corresponds to underfitting, high bias and low variance), which is why there is a small difference between training and test errors, but the error values are generally high. Again, as λ values are lowered, we are adding less penalty and complexity term which makes the model close to choosing a complex hypothesis, and therefore our model is more likely to overfit (high variance, low bias). Overfitting for small values of λ is again justified since when log lambda value is -1, there is a large difference between the training and test errors.

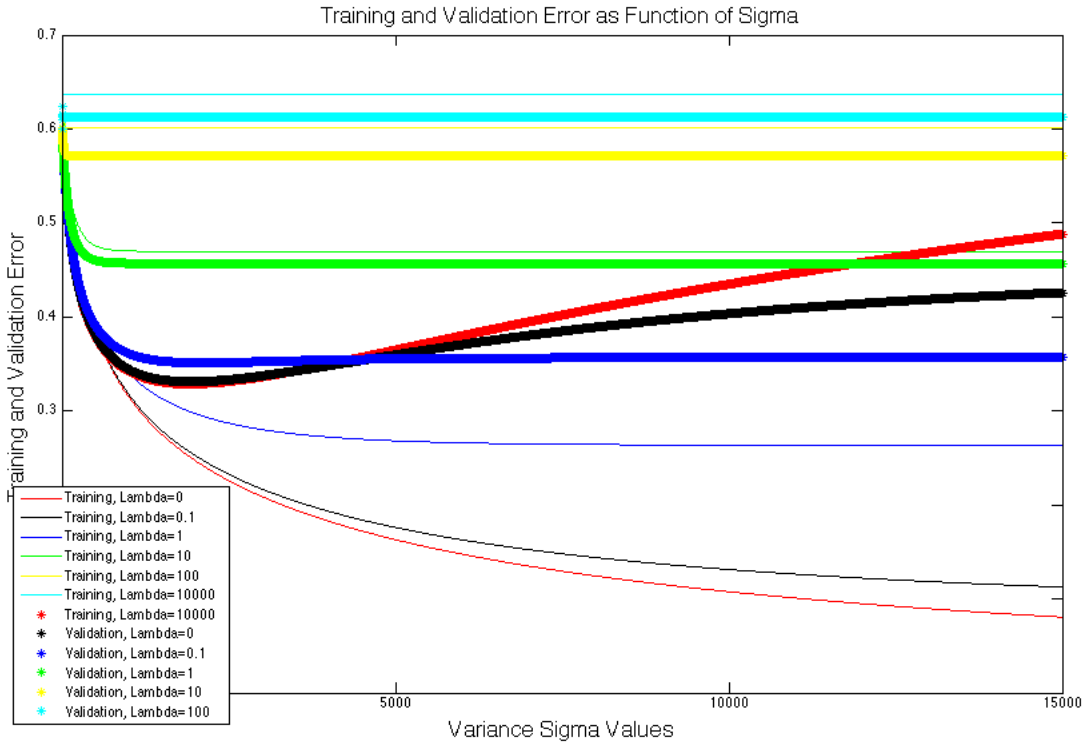

Figure 14: Training and Validation Error as a Function of λ

Figure 14 shows that for lower values of $\lambda = 0$ and $\lambda = 0.1$, the model overfits, and there is largely underfitting when the λ values are chosen to be too high.

As in figure 15, we then show the L2 norm of the weight vectors obtained. As the λ values are low, the model chooses a complex hypothesis (overfitting more likely) which is why the actual and l2 norm values of weights are higher. During overfitting, and fitting a complex model, it is more difficult to fit the learning parameters to minimize the objective function, and hence there is more variations in the weight estimates (which therefore leads to higher variance in the model prediction estimates). As λ value is increased a lot, underfitting is more likely to occur since the weights can be learnt easily for the simple model, but then this leads to poor generalisation performance. Our analysis therefore suggests that the λ parameter should again be chosen to be close to 1.

We then analyse how the L2 norm of the weight vectors changes for each of the corresponding basis functions, having different values of σ. Figure 16 below shows changes in L2 norm values, for each corresponding value of basis function means μ, against the values of the regularisation parameter. Figure 16 is to analyse how λ and σ parameter fine tuning can balance for the overfitting and bias/-variance tradeoff. Previously, we discussed that smaller values of λ leads to less overfitting to occur,

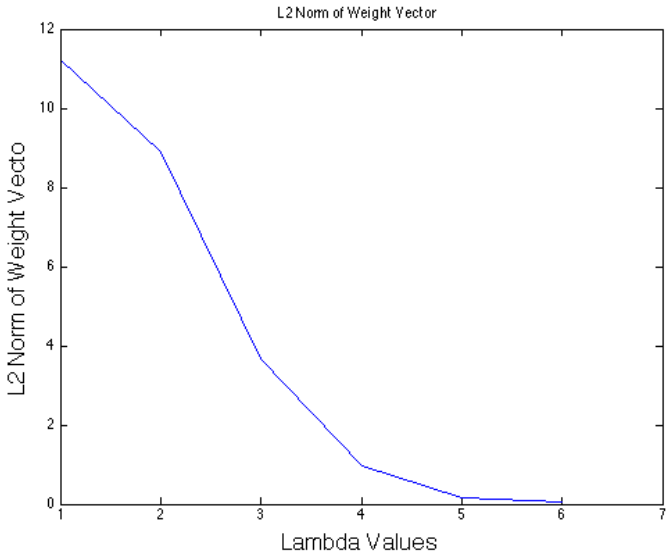

Figure 15: L2 Norm of the obtained weight vectors

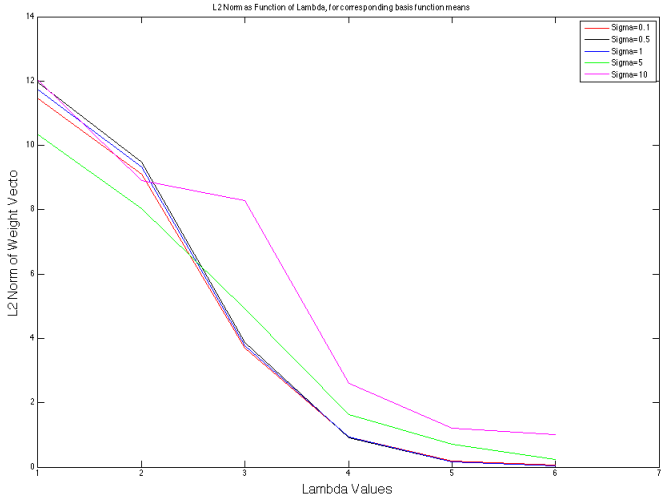

Figure 16: L2 Norm of the obtained weight vectors, for each corresponding values of σ

and similarly, higher values of σ can also lead to lower likelihood of overfitting. Results in figure 16 shows that as σ values are high, for a particular λ the l2 norm value is slighly higher. This is to suggest that since higher λ values leads to underfitting to occur, σ values for larger λ should be higher so that it can reduce for overfitting, thereby bring balance to the overall bias/variance analysis.

2.7 Question H

Design of Gaussian Basis Functions:

The design of suitable Gaussian basis functions essentially depends on the optimal choice of the mean μ and variance σ parameters. The Gaussian basis function would compute the Euclidean distance between an input feature vector and mean μ, or alternatively, in the case of Gaussian kernels, it would compute Euclidean distance between two feature vectors. Since the Gaussian kernel descreases with distance, it is often interpreted as a similarity measure.

The μ parameter in designing Gaussian basis functions controls the location of the input data distribution in the feature space. The σ values corresponds to width, which means how much distributed should the input data be in the feature space.

In order to choose appropriate μ and σ parameters, we can do the following, similarly to what we have done for our experiments as above. We can consider these as parameters we need to fine-tune by using a cross validation set, where the optimal parameters μ and σ would be the combination which has a good generalisation performance and balances the bias variance tradoeff.

Effect on Bias Variance Tradeoff
In order to design for basis functions, we should choose a μ parameter which is close to the true mean of input data distribution. We should choose μ parameter by cross validating, with different values of μ and compare generalisation performance. Alternatively, we can add all the basis functions together for different μ values. For choosing the σ parameter, we should ensure that the σ is such that the input data distribution in the feature space is widely distribution such that we do not have to fit too complex or too simple a mode. The σ parameter essentially controls for overfitting and bias variance trade-off when designing Gaussian basis functions. High σ would make the model unlikely to overfit, and therefore there would be less variance in the model prediction estimates.

Our analysis is that, when designing Gaussian basis functions, the σ parameter perhaps plays a more important role in balancing the bias variance trade-off compared to the μ which essentially controls the location of the data distribution.

2.8 Question I

Learning Algorithm to compute placement of basis function:

Here, we show theoretical proofs of how we can derive a learning algorithm to find the optimal paramters μ and w. In order to find the best placement of basis functions, we can consider the μ parameter to be a learnable parameter which we can find by gradient descent, by finding the gradient of the objective function w.r.t to μ.

The parameters of our model are now μ and w. Given that the objective function now is $J(\mu, w)$, we can derive a learning algorithm where we can simulateneously update both the paramters μ and

w. First, we show the derivation of $\nabla_\mu J(\mu)$ as below:

$$J(\mu) = -[\sum_{i=1}^{m} y_i \log(h_\mu(x_i)) + (1 - y_i) \log(1 - h_\mu(x_i))] + \lambda \|w\|^2 \tag{5}$$

where $h_\mu(x) = \frac{1}{1+e^{-w^\intercal \phi(x)}}$, and here $\phi(x) = e^{\frac{(x-\mu)^2}{2\sigma^2}}$ which is the basis function with paramters μ. Therefore, the gradient can be written as:

$$\nabla_\mu J(\mu) = -[\sum_{i=1}^{m} y_i \nabla_\mu \log(h_\mu(x)) + (1 - y_i) \nabla_\mu \log(1 - h_\mu(x_i))] + \nabla_\mu \lambda \|w\|^2 \tag{6}$$

We can therefore separately find the gradients w.r.t μ for each of the terms above. For convenience, we do not show the full derivation, but only show some of the important derivations.

We can write $\nabla_\mu \log(h_\mu(x))$ as below, after deriving the expression

$$\nabla_\mu \log(h_\mu(x)) = \frac{w^\intercal \phi(x)(\frac{x-\mu}{\sigma^2}) \exp(-w^\intercal \phi(x))}{1 + \exp(-w^\intercal \phi(x))} \tag{7}$$

Similarly, we can write the expression of $\nabla_\mu \log(1 - h_\mu(x))$ as follows. Again, we do not show the entire derivation of finding this gradient, but only show the final result which is required to derive the full expression:

$$\nabla_\mu \log(1 - h_\mu(x)) = \frac{-w^\intercal \phi(x)(\frac{x-\mu}{2})}{1 + \exp(-w^\intercal \phi(x))} \tag{8}$$

Therefore, we can $\nabla_\mu J(\mu)$ as follows:

$$\nabla_\mu J(\mu) = w^\intercal \phi(x)(\frac{x - \mu}{\sigma^2})(h_\mu(x) - y) \tag{9}$$

Note that this is similar to the original derivative of $\nabla_w J(w)$ that we before found for the objective function of logistic regression.

Therefore, we can update the parameters μ of $J(\mu)$ by following gradient descent update rule:

$$\mu \leftarrow \mu - \alpha \nabla_\mu J(\mu) \tag{10}$$

Learning Algorithm

Our learning algorithm will now therefore consist of two update equations of the learnable paramters μ and w. Given that the sigmoid function is

$h_{\mu,w}(x) = \frac{1}{1+\exp(w^\intercal)\phi(x))}$, we will therefore iteratively update both μ and w using gradient descent. This will therefore be an iterative algorithm, where we iterartively update both μ and w using gradient descent. Note that, we may use different learning rates α_μ and α_w for the two gradient descent update rules.

2.9 Question J

For this section, we have not implemented the learning algorithm as in (i), but only have included theoretical analysis of how we can adapt the placement of the basis functions by learning μ parameter.

Our hypothesis is that this learning algorithm, where we iteratively update both μ and w is likely to converge. This algorithm is similar to the EM algorith, where the update of one parameter is initially done by setting the other parameter fixed, and once it is updated, it is then held fixed to further update the other parameter. Just as EM algorithm, this iterative algorithm is also likely to converge, especially because we can find a closed form derivative for $\nabla_\mu J(\mu)$ as shown above, and for $\nabla_w J(w)$. The convergence of this algorithm furhter depends on the learning rates of the two gradient descent update rules α_μ and α_w, and just as any gradient descent, this is likely to converge based on fine tuning the learning rates.

However, the iterative algorithm is likely to converge to a locally optimal solution. Just as EM algorithm, this algorithm also seems to be finding a lower bound solution to the original loss function. This is mainly because the updates of one parameter depends on the updates of the other parameter. Also, since we now have $J(\mu, w)$, this function is a non-convex objective function and therefore the gradient descent iterative update rules does not guarantee convergence to the globally optimal solution.

3 Section 2 : Kernelized Logistic Regression

3.1 Question A

Dual View of Logistic Regression :

We consider inputs in the feature space $\phi(x)$ for the dual view of logistic regression. As given in notes, the gradient $\nabla_w J(w)$ can be written as, even in the case of using feature maps as inputs, as :

$$\nabla_w J(w) = \phi(x)[y - h_w(x)] + \lambda w \tag{11}$$

To find the dual view, firstly set $\nabla_w J(w) = 0$. Therefore, we can write:

$$\phi(x)[y - h_w(x)] + \lambda w = 0 \tag{12}$$

Therefore, w can be written as:

$$w = -\frac{1}{\lambda}[\phi(x)(y - h_w(x))] = \Phi a \tag{13}$$

where:

$$a = -\frac{1}{\lambda}(y - h_w(x)) \tag{14}$$

Putting back the expression of **a** in the original equation, we therefore get the following, ignoring the regularisaion parameter λ as follows, since $w = \Phi a$:

$$J(a) = -[y \log(\frac{1}{1+\exp(-\Phi^\intercal \Phi a)}) + (1-y)\log(1 - \frac{1}{1+\exp(-\Phi^\intercal \Phi a)})] \tag{15}$$

The dot product of $\Phi^\intercal \Phi$ can therefore be written as a kernel since we know that $K = \Phi^\intercal \Phi$. Therefore, $J(a)$ can be written in terms of kernels:

$$J(a) = -[y \log(\frac{1}{1+\exp(-Ka)}) + (1-y)\log(1 - \frac{1}{1+\exp(-Ka)})] + \frac{\lambda}{2} a^\intercal K a \tag{16}$$

Let us call the expression of the sigmoid function in terms of a kernel as $H(x)$:

$$H(x) = \frac{1}{1+\exp(-Ka)} \tag{17}$$

Since for the gradient descent update rule, we need to find $\nabla_a J(a)$, this expression will be similar to the original expression of the gradient, where x is replaced by K and w is simply replaced by a. Therefore, $\nabla_a J(a)$ can be written as:

$$\nabla_a J(a) = K(y - H(x)) + \lambda a K \tag{18}$$

where note that the regularisaiton term will now have a dependence with the kernel matrix K.

Learning Algorithm:

The gradient based update rule for the dual form of logistic regression can therefore be written as :

$$a \leftarrow a - \alpha \nabla_a J(a) \tag{19}$$

$$a \leftarrow a - \alpha K(y - H(x)) + \lambda a K \tag{20}$$

where the sigmoid function is $H(x) = \frac{1}{1+\exp(-Ka)}$.

3.2 Question B

In this section, we use a kernelized logistic regression, and provide an implementation of it with the polynomial kernel. Note that, we do not add any L2 regulariser in this section, as the hope is that with the kernel feature mapping, we will find a linear classification in the high dimensional feature mapped state. We will eventually compare how the kernelised version performs better than the linear logistic regression classifier. As before, we again perform 5-fold cross validation.
Below, we include an implementation with the kernel $K(x, z) = (x * z + 1).^d$ where d denotes the order of the polynomial.

MATLAB Code Implementation : : Implementation of the dual view of logistic regression using the polynomial kernel.

```
clear all
clc

load('hw1data.mat')
X = hw1x;
y = hw1y;
[m, n] = size(X);

[trainInd,valInd,testInd] = dividerand(size(X,1),0.6,0.2,0.2);

data = [X,y];

training = data(trainInd', :);
validation = data(valInd', :);
test = data(testInd', :);

x_train = training(:, 1:end-1);
y_train = training(:, end);

x_valid = validation(:, 1:end-1);
y_valid = validation(:, end);

x_test = test(:, 1:end-1);
y_test = test(:, end);

iterations = 15000;
alpha = 0.009;


kernel_dim = [1, 2, 3]';


Training_log_likelihood = zeros(iterations,length(kernel_dim));
Testing_log_likelihood = zeros(iterations,length(kernel_dim));
Validation_log_likelihood = zeros(iterations,length(kernel_dim));

val = 5;
Validation_Cost_Train = zeros(iterations, val);
Validation_Cost_Validation = zeros(iterations, val);
Validation_Cost_Test = zeros(iterations, val);


    for j = 1:length(kernel_dim)

        Kernel = zeros(size(X, 1), size(X,1));
        Kernel = (X * X' + 1).^kernel_dim(j);

        data = [Kernel, y];

        training = data(trainInd', :);
        validation = data(valInd', :);
        test = data(testInd', :);

        train_validation_data = [training; validation];
        test = [test; train_validation_data(end, :)];
        train_validation_data = train_validation_data(1:100, :);

        x_test = test(:, 1:end-1);
        y_test = test(:, end);
```

```
61
62      [crossValidation_Training, crossValidation_Validation] = cross_val(
    train_validation_data);
63
64      for v = 1:5
65          if v ==1
66              training = crossValidation_Training(1:80, :);
67              validation = crossValidation_Validation(1:20, :);
68          elseif v==2
69              training = crossValidation_Training(81:160, :);
70              validation = crossValidation_Validation(21:40, :);
71          elseif v==3
72              training = crossValidation_Training(161:240, :);
73              validation = crossValidation_Validation(41:60, :);
74          elseif v==4
75              training = crossValidation_Training(241:320, :);
76              validation = crossValidation_Validation(61:80, :);
77          elseif v==5
78              training = crossValidation_Training(321:400, :);
79              validation = crossValidation_Validation(81:100, :);
80          end
81
82      x_train = training(:, 1:end-1);
83      y_train = training(:, end);
84      x_valid = validation(:, 1:end-1);
85      y_valid = validation(:, end);
86
87
88      [m, n] = size(Kernel);
89     w = zeros(size(Kernel,2), 1);
90
91      for i = 1:iterations
92
93         sigmoid = (1 ./ (1 + exp(- (x_train * w))));
94
95          %update rule of gradient descent
96          grad_d = x_train' * (sigmoid - y_train);
97          w = w - alpha*grad_d;
98
99          Cost_Train =  (-1/length(y_train)) *  sum(   (y_train .* log10(1 ./
    (1 + exp(- x_train * w))))     +   (1-y_train) .* log10(1 - (1 ./ (1 + exp(-
    x_train * w)))))     ;
100         Validation_Cost_Train(i, v) = Cost_Train;
101         Training_log_likelihood(i,j) = mean(Validation_Cost_Train(i, :));
102
103         Cost_Valid =  (-1/length(y_valid)) *    sum(   (y_valid .* log10(1
    ./ (1 + exp(- x_valid * w))))     +   (1-y_valid) .* log10(1 - (1 ./ (1 + exp
    (- x_valid * w))))    );
104         Validation_Cost_Validation(i, v) = Cost_Valid;
105         Validation_log_likelihood(i,j) = mean(Validation_Cost_Validation(i, :)
    );
106
107         Cost_Test =  (-1/length(y_test)) *  sum(   (y_test .* log10(1 ./ (1 +
     exp(- x_test * w))))     +   (1-y_test) .* log10(1 - (1 ./ (1 + exp(- x_test
    * w)))))     ;
108         Validation_Cost_Test(i, v) = Cost_Test;
109         Testing_log_likelihood(i,j) = mean(Validation_Cost_Test(i, :));
110
111     end
112
```

```
        end
    end
```

Listing 1: Implementation of Polynomial Kernelised Logistic Regression

3.3 Question C

Experimentation with Polynomial Kernelised Logistic Regression

We perform 5-fold cross validation and average the results over 3 trials. First, we present the results of training and validation for each order of the kernel as shown below:

First, we investigate the effectiveness of the **width of kernel** by analysing training and validation error rates. Figure 17 shows how the training and validation errors changes with changing dimensions of the polynomial kernel. From this, we can infer that with $d = 1$ and $d = 3$, the model is perhaps more close to overfitting, and underfitting respectively, and the width of 2 seems to be performing the best. From this, we infer by performing 5-fold cross validation, and taking averaged trials over 3 runs, that the width of the polynomial kernel be chosen to be 2.

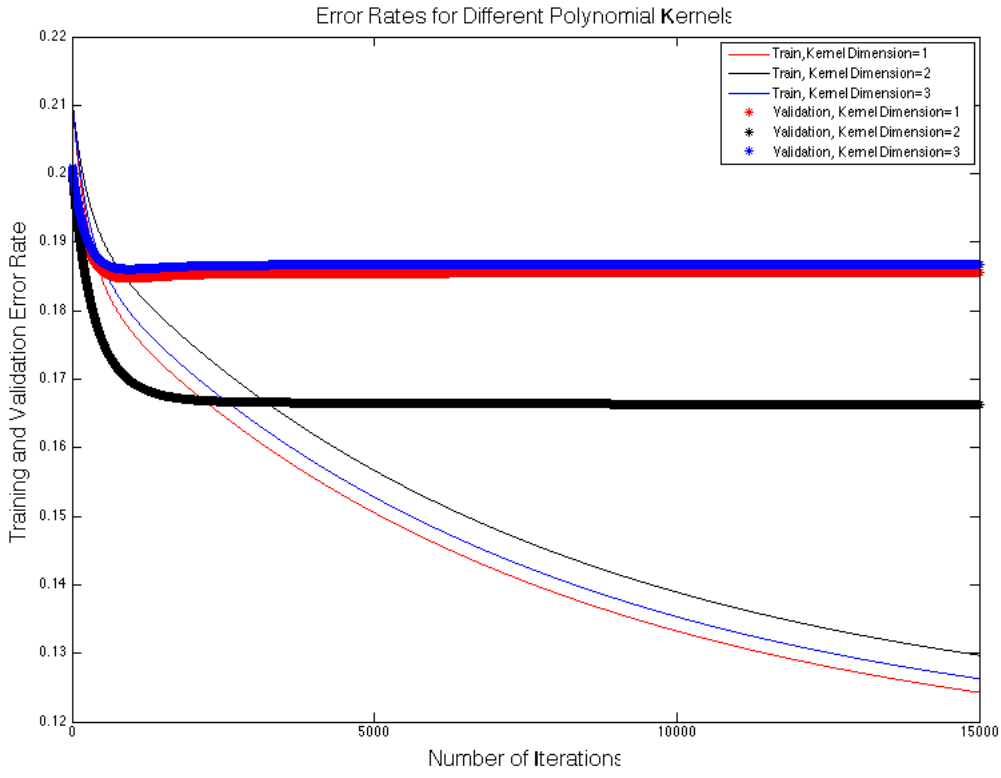

Figure 17: Training and Validation Errors with Polynomial Kernelised Logistic Regression

By further plotting the train and test errors, we find that with the width of 2, the model performs relatively well, even though the test error is lowest with polynomial kernel dimension of 3. We again chose the α learning rate parameter to be relatively small, and ran 15000 iterations of gradient descent to optimise the kernelised objective functions. Figure 18 furhter summarises the results.

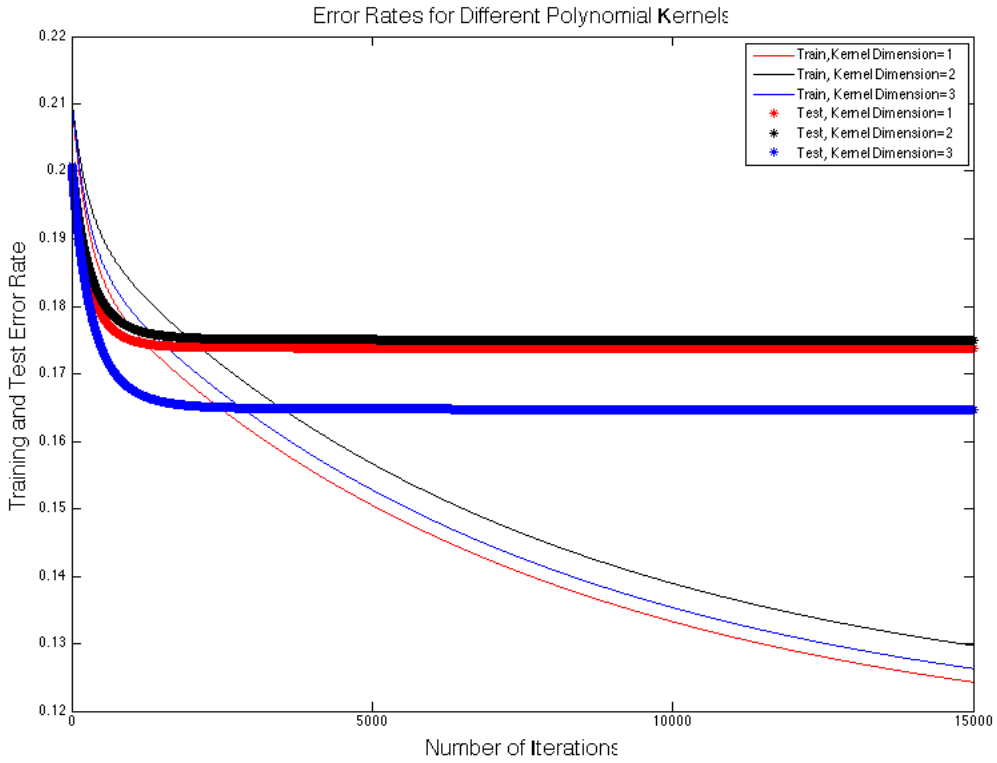

Figure 18: Training and Testing Errors with Polynomial Kernelised Logistic Regression

Effect of Width of Approximator : Based on the validation set results, it seems that the width should not be either too high or too low. Even though we are investigating with only three dimensions for the polynomial kernel, it seems that the model overfits if the width is either 1 or 3. This is reasonable since a higher width of the approximator corresponds to a more complex model which is more likely to overfit.

Comparison of Linear and Non-Linear Logistic Regression:

We analyse the differences in training and test results for the linear logistic regression and the kernelised (non-linear) logistic regression classifier. Note that since the data set is high dimensional and has very few samples, and based on our initial investigation we found that there exists no linear separating hyperplane, the linear logistic classifier was not expected to perform well. By applying the kernel feature maps, and taking inputs to a high dimensional space where a linear hyperplane

could be found, our results as shown below shows that there is a significant improvement in classifier generalisation performance when kernel feature maps are used.

We further demonstrate the results in the figures 19 and 20.

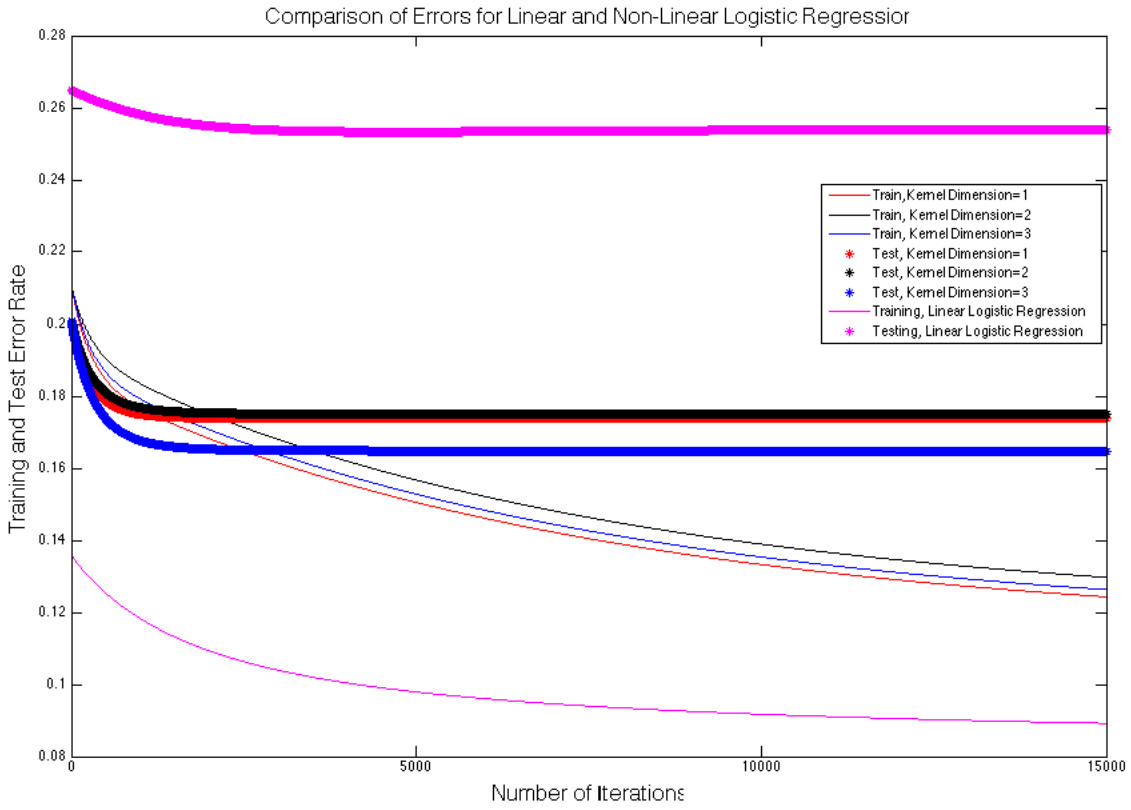

Figure 19: Comparing Linear and Non-Linear (Kernelised) Logistic Regression Classifiers

Figure 19 specifically shows that the linear classifier highly overfits and performs really poorly on the test set compared to the kernelised version. This can be demonstrated from the large difference in generalisation performance. Even though we can optimise the training objective function for the linear case, the model overifts and performs poorly on test set. Figure 20 further shows a clear difference when we have chosen the appropriate width for the polynomial kernel.

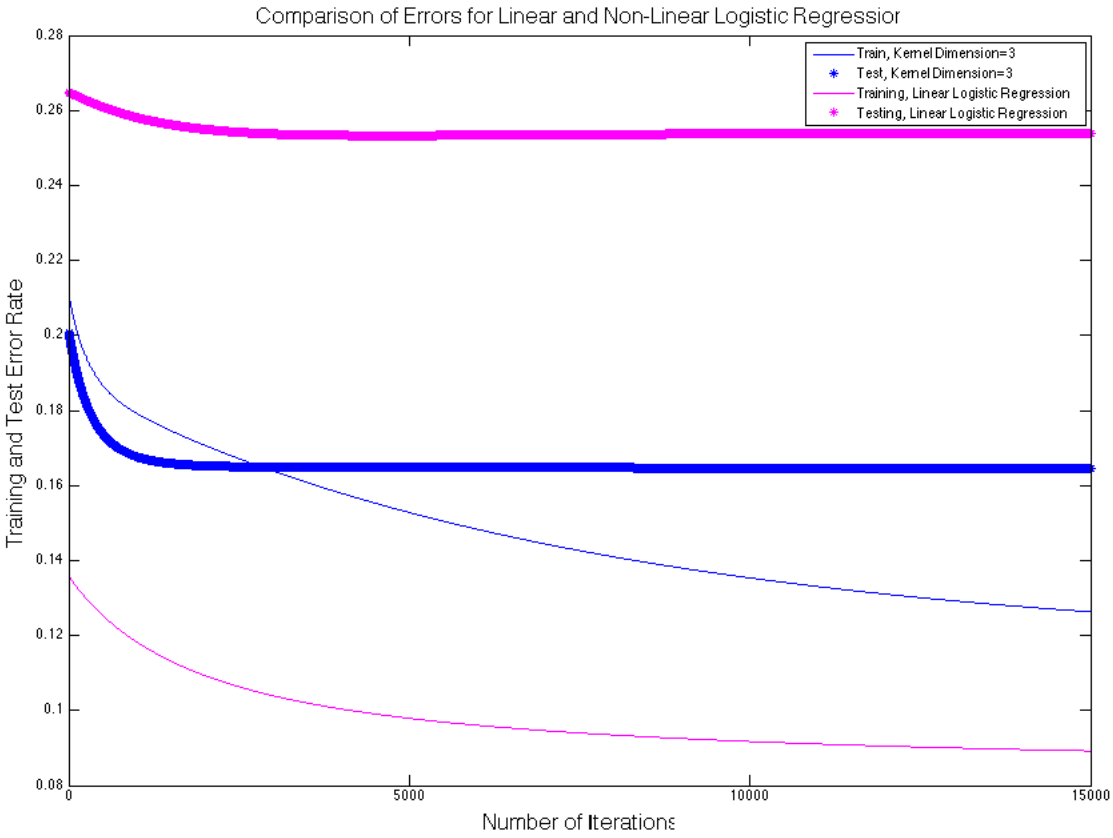

Figure 20: Comparing Linear and Non-Linear (Kernelised) Logistic Regression Classifiers, with the chosen polynomial kernel with width 2

3.4 Question D

Advantages and Disadvantages of Kernelised Logistic Regression :

Advantages:
One of the major advantages of kernelised logistic regression is that it can deal with high dimensional inputs. Our results above suggests that even though the linear classifier could not perform well on this very high dimensional dataset, the kernelised version was able to significantly improve on the generalisation performance. Our results suggest that the kernelised logistic classifier, by transforming the inputs into a high dimensional feature space, can find a linear separating hyperplane even though the dataset in the original space is highly non-linear.

Unlike the feature based versions which maps the examples into feature space, the kernel versions uses similarly between one example with other examples, and in cases such as ours where there are few training exmaples and many features, the kernel versions are more efficient. When the number

of features in the dataset are low, the kernel versions computes the feature space, and makes it easy to compute over enormous feature spaces. Furthermore, we can also use infinite dimensional feature spaces and still obtain results with same computational complexity. Using kernels and the kernel trick, non-inear patters in the data can be easily found without having to increase the computational cost. The complexity of kernel methods is therefore a function of the number of training instances, rather than the number of input dimensions.
In our experiments, we analysed with a linear kernel which is good when there is a lot of features. Solving an optimisation problem for a linear kernel is often much faster than compared to a Gaussian kernel or using a set of basis functions. However, the most important advantage of linear kernels is that it performs well in cases when the number of features is significantly higher than the number of samples (observations). A Gaussian kernel is generally used when the number of samples is larger than the number of features.

Disadvantages:
However, one of the drawbacks of kernelised logistic regression is the need for computation time. Since the original inputs are high dimensional, it takes significant computation time to calculate the kernels even for the polynomial case. If we were to consider Gaussian kernels, it would have been computationally more expensive. Furthermore, the choice of the kernel is always a design issue. There are no fixed kernels that we can use, and it is upto the designer to choose the kernel for the particular problem. There are no fixed rules to justify in advance whether a polynomial kernel would perform better than a Gaussian kernel or not.

In cases when the number of features is already large, it may also be disadvantageous to use a kernel method, since it will take the input space to an even higher dimensional feature space. Furthermore, for low dimensional problems with many training examples, linear kernel methods may yield a poor predictive accuracy.

4 Section 3 : Kernels

4.1 Question A

Here we show that the summation of kernels is also a valid kernel:

$$K(x, z) = \alpha K_1(x, z) + \beta K_2(x, z) \tag{21}$$

For this, we first need to show that if K is a kernel, then for $\alpha > 0$ αK is also a valid kernel. For the feature map ϕ for K, let us define that $\phi_2(x) = \sqrt{\alpha}\phi_1(x)$. Therefore, if we write $\alpha K(x, z)$ as

$$\alpha K(x, z) = \phi_2(x) \cdot \phi_2(z) \tag{22}$$

Since $\alpha K(x, z)$ can be written as a dot product of feature vectors, it is also a valid kernel for $\alpha > 0$. Next, we show that the sum of valid kernels is also a valid kernel as follows:

Assuming ϕ_1 is a feature map of $\mathrm{K}_1 and \phi_2$ is a feature map of K_2, then we can define a third feature map ϕ_3 for the kernel $\mathrm{K}_3 as follows. Given we write \phi_1(x)$ as

$$\phi_1(x) = f_1(x), f_2(x), f_3(x)... \tag{23}$$

and writing $\phi_2(x)$ as

$$\phi_2(x) = g_1(x), g_2(x), g_3(x)... \tag{24}$$

We can therefore write the feature map ϕ_3 as:

$$\phi_3(x) = f_1(x), g_1(x), f_2(x), g_2(x), f_3(x_3), g_3(x)... \tag{25}$$

Therefore, we can write that $\phi_3(x) \cdot \phi_3(z)$ can be written as $\phi_1(x) \cdot \phi_1(z) + \phi_2(x) \cdot \phi_2(z)$. Therefore, ϕ_3 is a feature map for the sum of kernels $K_1 + K_2$. Since, the feature maps can be written as dot product and corresponds to the sum of dot product features for ϕ_1 and ϕ_2, we can write the following:

$$K_3(x, z) = \alpha K_1(x, z) + \beta K_2(x, z)$$

(26)

where dot product of ϕ_3 can be written as dot product of ϕ_1 summed with the dot product of ϕ_2.

4.2 Question B

Here we show that the following is a valid kernel:

$$K(x, z) = K_1(x, z) K_2(x, z) \tag{27}$$

Let us first assume that the feature map of $K_1 be \phi_1$ and the feature map for kernel $K_2 be \phi_2$. Since we know that $K_1(x, z)$ is a valid kernel, let us represent the kernel K_1 as

$$K_1(x, z) = f(x) g(z) \tag{28}$$

and simiarly $K_2(x, z) = f(x)g(z)$. Therefore, considering feature maps ϕ_1 and ϕ_2, we can write the product of the two valid kernels as follows. For notational simplicty, let's take $x_1 = x$ and $x_2 = z$

$$\begin{aligned} K_1(x_1, z_2) K_2(x_1, z_2) &= (\phi_1(x_1) \cdot \phi_1(z_2))(\phi_2(x_1) \cdot \phi_2(z_2)) \\ &= (\sum_{i=1}^{\infty} f_i(x_1) f_i(z_2))(\sum_{j=1}^{\infty} g_j(x_1) g_j(x_2)) \\ &= \sum_{i,j} f_i(x_1) f_i(x_2) g_j(x_1) g_j(x_2) \\ &= \sum_{i,j} (f_i(x_1) g_j(x_1))(f_i(x_2) g_j(x_2)) \end{aligned} \tag{29}$$

For the kernel $K_3(x_1, x_2)$ to be a valid kernel, let us define a feature map ϕ_3 with a feature of $h_{i,j}(x)$. We can define the feature map for the kernel K_3 as follows:

$$h_{i,j} = f_i(x) g_j(x) \tag{30}$$

Therefore, from above, we can write $K_1(x_1, x_2)K_2(x_1, x_2)$ as $\phi_3(x_1) \cdot \phi_3(x_2)$ where the inner product will sum over all the pairs of $< i, j >$.

Therefore, we have shown that since $K_3(x_1, x_2) = K_1(x_1, x_2)K_2(x_1, x_2)$ can be written as the dot product $< (f(x_1)g_(x_1), f(x_2)g(x_2)) >$, so the product of kernels is also a valid kernel. Therefore, the kernel $K_3(x_1, x_2)$ will have the feature map $\phi_3(x) = f(x)g(x)$.