Machine Learning and Pattern Recognition 2011/2012

Stephen McGruer (s0840449)

1.

a.

The main problems with using maximum likelihood to choose the best predictive classifier are that it ignores any uncertainty in the data and it ignores any prior beliefs that we may have about the classifiers. In maximum likelihood model selection, the maximum likelihood estimation for the parameters would be used to determine the likelihood of each model (i.e. the maximum likelihood of $P(D|M = m)$ is calculated). This approach ignores any uncertainty associated with the data, which may be quite high since we have a small data-set. Additionally, we may hold some prior belief (for whatever reason) that one of the models is more or less likely to be the correct one - maximum likelihood model choosing ignores this.

Instead, I would use a Bayesian (belief-based) method to choose a model. Bayesian model selection calculates the posterior probability for each model using Bayes' theorem:

$$P(M = m|D) = \frac{P(D|M = m)P(M = m)}{P(D)}$$

As the value of $P(D)$ does not rely on the model and is just a scaling factor, it is often omitted for simplicity. The Bayesian method takes care of both the uncertainty about the parameter values, as $P(D|M = m)$ is calculated by marginalising over the parameter values, and also any beliefs we have about the models, by including the prior term $P(M = m)$.

b.

Radial basis functions are defined by two parameters - $\mathbf{m}$, a vector of centres in the input space, and α, which defines the width of each radial 'bump'. In an adaptive radial basis function network, these values are determined by the hidden layer(s) of the network, and so are optimized online using the training data. Therefore, if such a network is trained on the set of normal inputs, the radial basis centres and weights will be set such that they capture the 'normal' space of the inputs. When a new and extreme input is encountered, none of the basis functions will have much weight in the region of space where the extreme input is located, and as such the classifier will struggle to make a decisive or correct prediction.

This problem is difficult to solve due to the combination of it being non-linear and because we are attempting to train a model on different data from what it then may be asked to solve. One solution would be to transform the problem into a linear one if possible, and to then use a standard hyperplane-fitting linear regressor. Such a classifier would fit an infinite hyperplane to the problem, and so as long as the extreme inputs followed the same relative form as the 'normal' training data (i.e. as long as the relationship between the input features and the output value was roughly the same), it should be able to predict on the new inputs as well.

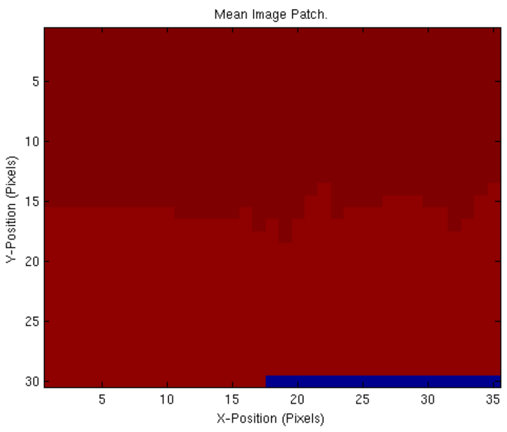

Figure 1: The image patches for the mean image. The darker a colour is, the higher the intensity value.

c.

Although multi-layer perceptrons are able to approximate any non-linear function, and thus could be trained on and applied to an unknown problem, there are a number of downsides. Firstly, it is likely that the multi-layer perceptron will over-fit the training data, meaning that it will then perform poorly on real inputs. Additionally, choosing a multi-layer perceptron means that we do not learn anything about our problem; we are essentially classifying 'blind'. This means that we cannot know whether or not there might be a simpler or more suitable classifier for our problem that would be superior to a multi-layer perceptron.

Instead of immediately training a multi-layer perceptron, I would prefer to try and discover the nature of the problem, to determine if it could be better approached with a specific classifier. If this were not possible, I would try to tackle the over-fitting problem present in multi-layer perceptrons. This can be done by either penalising large weight values or by using a Bayesian approach to control the model complexity. Bayesian inference in multi-layer perceptrons can be implemented using Monte Carlo sampling or Gaussian approximations.

2.

a.

The mean image patch can be seen in Figure 1. It is a mostly uniform patch, with slightly higher intensity in the upper half of the patch. The first three principal vectors can be seen in Figure 2. In each image blue signifies a negative value, and red a positive one. The darker a pixel is, the higher the absolute value at that pixel. The first principal vector therefore allows the expression of positive values in the centre of an image patch, with circular bands of stronger intensities towards the middle. The second principal vector has a positive area at the left hand side of the image, and a 'mirror' negative area at the right hand side. The third principal vector is similar to the second, but with a positive area at the top of the image, and a negative one at the bottom.

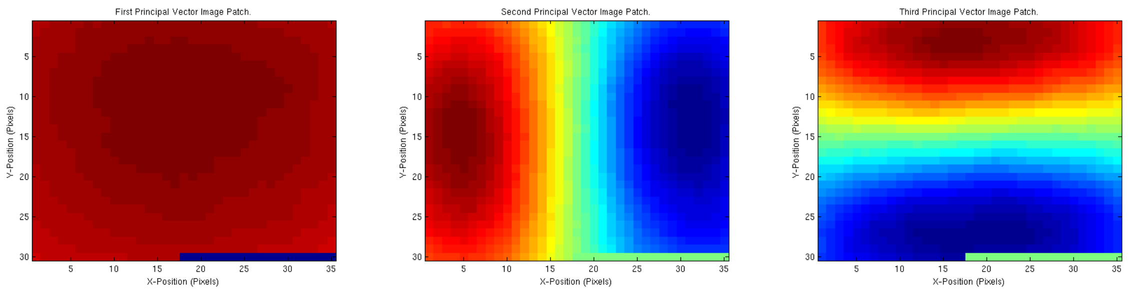

Figure 2: The image patches for the first three principal vectors. The values range from negative (dark blue) to positive (dark red).

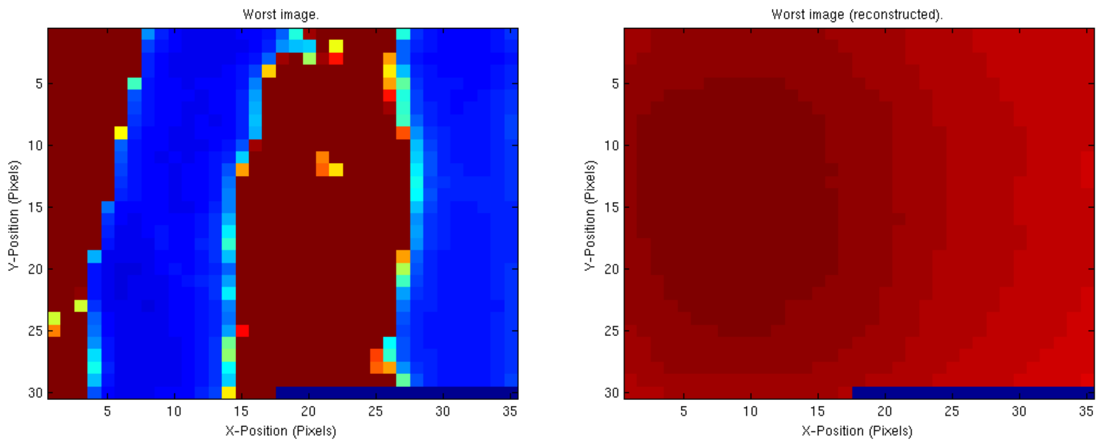

Figure 3: Image patches for the worst described image (left) and it's reconstructed form (right). The values range from 0 (dark blue) to high positive (dark red).

b.

The worst described image is shown in Figure 3, along with it's reconstruction. This image (index 34729, $i = 2787$) has multiple vertical stripes of high (63) and low (0) intensity. Since none of the three principal component vectors are striped in this way, it makes it very difficult to reconstruct the image. As can be seen in right image of Figure 3, the best reconstruction that could be managed by three-dimensional PCA is a left-aligned, high intensity patch - very unsimilar to the actual data.

For interest, the best described image and it's reconstruction are shown in Figure 4. Unsurprisingly it is very similar to the mean image, having a uniform intensity of around 7.86 (compared to the mean image's intensity of 9.10.) As such the reconstructed version is almost identical.

c.

The histogram of the target values can be seen in Figure 5, on the left side. The distribution could considered to be heavy tailed, and roughly matches either the Poisson or Inverse-Gamma distribution. Unlike either of these distributions, however, the histogram is slightly bi-modal,

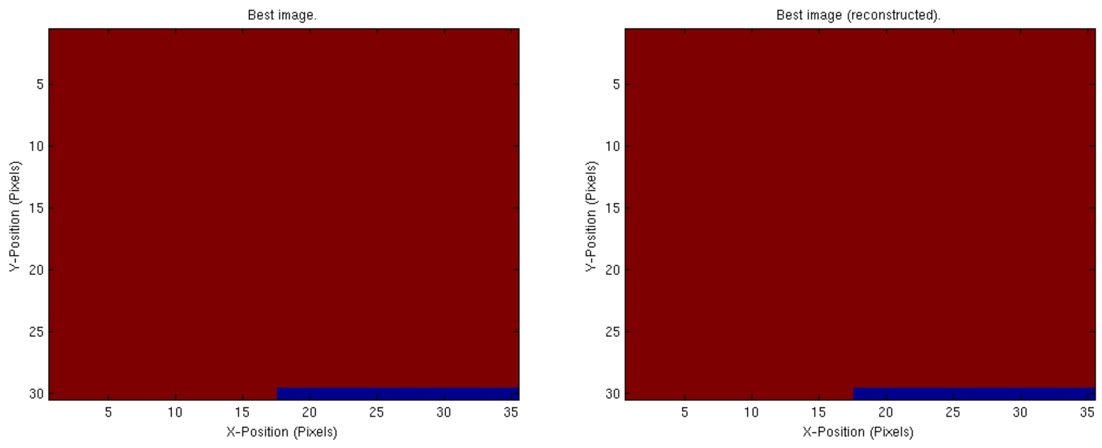

Figure 4: Image patches for the best described image (left) and it's reconstructed form (right). The values range from 0 (dark blue) to high positive (dark red).

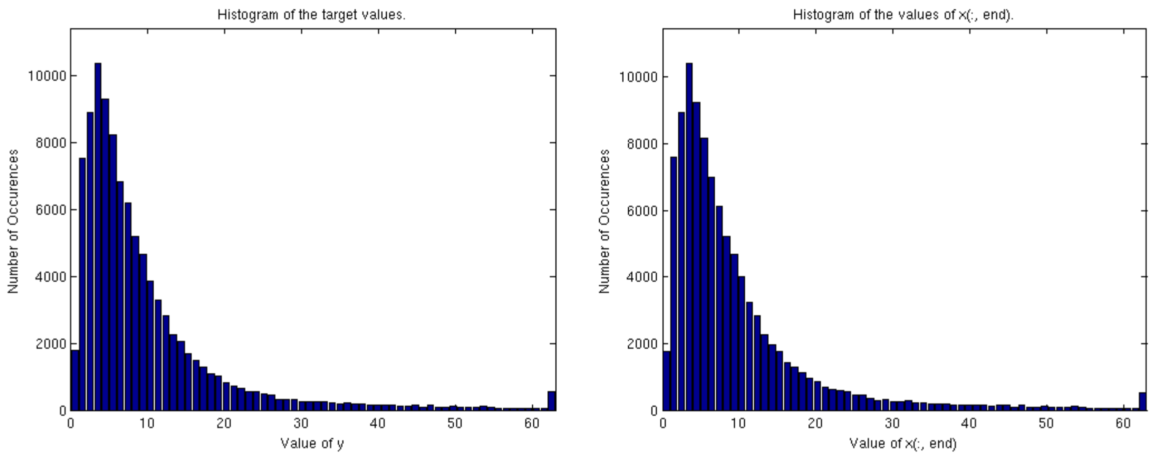

Figure 5: Histograms for the target values, and for x(:, end).

with a small second peak at an intensity value of 63. In general this shows that there is a large preference for low values of y, with some preference for a high value, but little preference for values between 25 and 62.

d.

The histogram of the difference between the target values (y) and the last element of the input values ($x(:, end)$) is given in Figure 6. This histogram is a fairly steep Gaussian, with most of it's weight centred around 0. This indicates that the distribution of values for x(:, end) is very similar to that of y, as can be seen in Figure 5 on the right side.

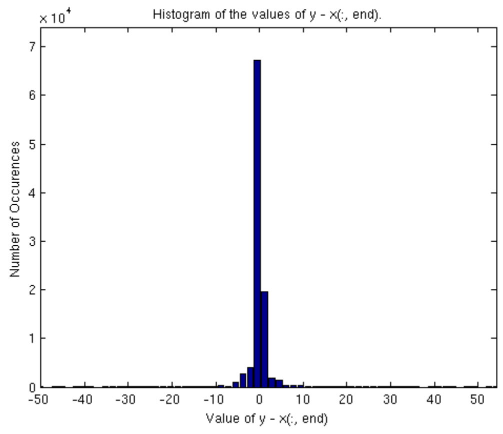

Figure 6: Histograms for the values of y - x(:, end).

3.

a.

The problem with using maximum likelihood to do parameter estimation for Naive Bayes is that it is unable to deal with uncertainty in the data. This shows itself in the case of Naive Bayes as the *zero probability problem*. The zero probability problem can occur in two cases. Firstly, consider the maximum likelihood estimation of the priors $P(c)$ when training Naive Bayes. This is given as:

$$P(c) = \frac{N_c}{N}$$

where N_c is the number of training items with class c, and N is the total number of training items. In the case where $N_c = 0$ (i.e. we do not have any training items of class c), the prior $P(c)$ is 0, and as such the probability of any new input point x being classified as class c becomes 0:

$$P(c|x) \propto P(x|c)P(c) = P(x|c) \times 0 = 0$$

The second case occurs when calculating the maximum likelihood estimation of $P(x_i|c)$ for feature values x_i. This is given as:

$$P(x_i|c) = \frac{N_{ci}}{\sum_{i=1}^{|F|} N_{ci}}$$

where N_{ci} is the number of times that the feature i appears in training items with class c, and $\sum_{i=1}^{|F|} N_{ci}$ is the total count of all features for class c. If we have an input x with a feature x_i which never appears in a training item for class c, N_{ci} is 0 and so $P(x_i|c) = 0$. As Naive Bayes computes the product of feature probabilities, we then have that $P(x|c) = 0$, and so the probability of x being classified as class c is 0:

$$P(c|x) \propto P(x|c)P(c) = 0 \times P(c) = 0$$

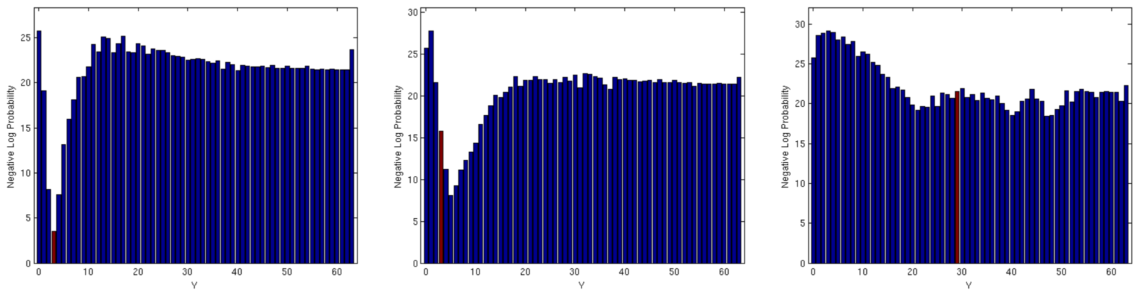

Figure 7: Histograms for the probabilities assigned to each class by Naive Bayes for three example inputs. On each histogram, the correct class is given in red. Note the use of negative log probabilities means that the 'highest' probability is the minimum.

In this coursework there is never a zero probability problem with regards to the class priors, as even after the training data is split into four (for cross-validation purposes), each class appears at least once[1]. However, the feature values do suffer from the zero probability problem. On average I found that the training examples for a class c had counts for only 16 out of the 64 possible values for each feature. If we examine the histogram of the difference between the class and the feature values for one feature $(x(:, end))$ in Figure 6 we can see that the majority of feature values are very close to their class.

Therefore, if a Naive Bayes classifier that had been trained on the data provided using maximum likelihood estimation came across an input which had features which were not one of the 'allowed' values for it's correct (and unknown) class c, it would have zero probability of correctly classifying the input as being of class c.

b.

i.

The 4-fold cross-validation perplexity for my Naive Bayes classifier was 2868.99.

ii.

By examining the range of probabilities $P(Y|x)$ calculated by Naive Bayes for a number of inputs x, I was able to spot three common cases. These cases are given in Figure 7 and are, from left to right: the case where Naive Bayes predicts the correct class, with a high probability; the case where Naive Bayes predicts a class that is close but not correct, and gives the correct class a far lower probability; and the case where Naive Bayes is unable to definitely determine a correct class, and so assigns each class a generally uniform probability.

These results show a disadvantage of Naive Bayes in this situation - it is either over-confident in it's prediction, or is not confident about any class. This means that if it selects the wrong answer, it will assign a very low score to the correct class, even if it was only off by a small amount. This may explain the poor perplexity score that the Naive Bayes classifier obtains, despite it's reasonable accuracy. In general this over-confidence may be ok, as Naive Bayes is usually applied to distinct classes, but here we have a range of real-valued intensities

[1]Actually, at least 9 times.

mapped to discrete classes. As such it is possible to consider the 'distance' between classes - for example, if an input has a true class of 3, a classification of the input as 4 is more correct than if it was classified as 10, despite neither being the correct answer.

iii.

The strong assumption made by Naive Bayes is that of *conditional independence* on the feature values given the class. That is, Naive Bayes assumes that the value of a feature is independent of the value of any other feature, as long as the class c is taken into account:

$$P(x_i, x_j|c) = P(x_i|c)P(x_j|c)$$

This means that the value of $P(x|c)$ in the Naive Bayes equation can be calculated as a simple product of the features $P(x_i|c)$:

$$P(x|c) = \prod_{i=1}^{N} P(x_i|c)$$

In the case of our problem the conditional independence assumption is not true. The class y does not necessarily define the values of the pixels around it - for example, if we have the value of $y(i)$, the values of $x(i, end)$ and $x(i, end - 35)$ do still provide information as to the likely value of $x(i, end - 34)$. Essentially, the problem is that the value we are predicting is really just another feature here, and as such does not define the input features used.

iv.

Submitting the results of applying my Naive Bayes classifier on `imtestdata.mat` to Kaggle achieved a perplexity score of 4.63981. This value is much lower than the value I achieved in *3.b.i*, and as such may hint towards an error in my perplexity calculations (although they are, of course, calculated for different data.) My Kaggle username is *smcgruer2*.

4.

a.

As linear regression is an exact predictor, it's output is a single numerical value (the answer). This means that it is not possible to directly calculate a perplexity score for traditional linear regression, as it does not estimate probabilities for any value or class. In order to add this ability I considered the case of linear regression with Gaussian noise:

$$y = \mathbf{w}^T\phi + \eta, \qquad \eta \sim N(0, \sigma^2)$$

This approach means that the probability for an intensity value (class) y_i given an input value x_i is determined by evaluating $N(y_i; \mu, \sigma^2) = N(y_i; \mathbf{w}^T\phi(x_i), \sigma^2)$. The question still remains as to how the value of σ^2 should be estimated. The variance of a Gaussian using maximum likelihood estimation is given by:

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i' - \mu)^2$$

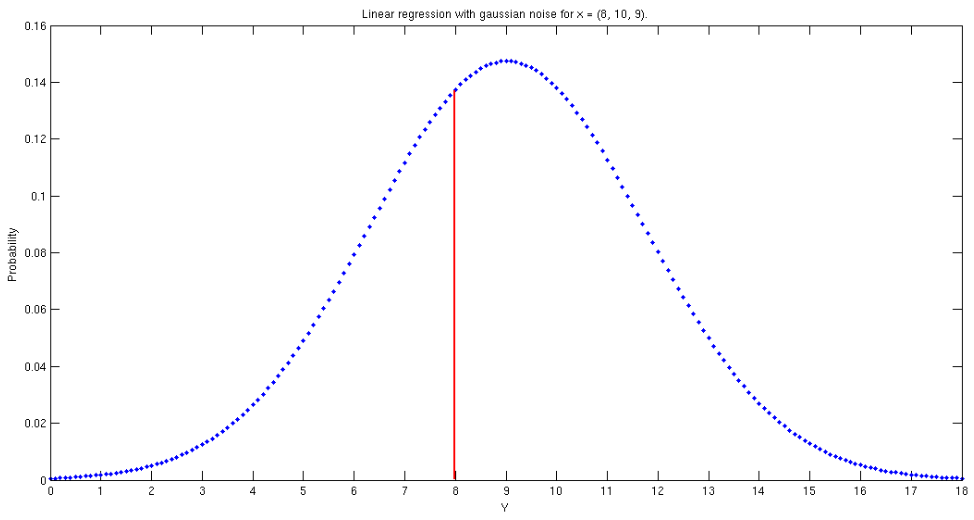

Figure 8: The Gaussian created by my linear regression for input x(49811). Note that although the Gaussian is centred at the wrong answer (9.0160), the probability of the correct class (y = 8, shown by a red line) is still quite high.

In our case we have $\mu = 0$, and the x'_i is determined by the expected error of the model; $x' = y - \mathbf{w}^T\phi(x)$. This gives a final calculation of σ^2 as:

$$\sigma^2 = \frac{1}{N}(y - \mathbf{w}^T\phi(x))^2$$

Note that this is just the *Mean Squared Error*, which is not unexpected as that is what we are using the Gaussian to estimate. An example of the Gaussian centred around an input value of x is shown in Figure 8.

A final point to consider is how to convert from the real-valued output of the linear regression to the discrete classes $0, ..., 63$. For simplicity I simply evaluated the Gaussian at each discrete value of y, however this approach ignores the shape of the Gaussian around each class. For example, the probability value of $y = 4$ in Figure 8 is around 0.025, but the Gaussian increases sharply between $y = 4$ and $y = 5$ - as such it might be more accurate to consider $y = 4$ to have a higher probability. One approach would be to estimate the probability for a value $y = c$ by averaging the values of the Gaussian between $y = c - 0.5$ and $y = c + 0.5$.

The four-fold cross validation perplexity for the linear regression on was 6.5375.

b.

As in question *4.a*, Gaussian noise was used to interpret the results in a probabilistic manner to allow a perplexity to be calculated. The four-fold cross validation perplexity for the linear regression on the reconstructed data was 14.0958.

c.

Of the three approaches considered in questions *3* and *4*, Naive Bayes has a far higher perplexity value than either linear regression. As theorised in question *3.b.ii*, this is likely due to it's over-confidence in it's potentially incorrect answer, and it's lack of confidence in any answer if it is unsure. The much lower perplexity of the linear regression models is due to a combination of the fact that it is able to treat the problem as a continuous one, and thus come up with 'halfway' answers (e.g. 6.7), and that the Gaussian that is placed over each answer is much more accepting of similar answers. The linear regression model based on the reconstructed training data had a higher perplexity than the one based on the true training data, most likely because the reconstruction reduced the quality of the data.

In terms of classification accuracy rather than perplexity, none of the models were particularly accurate. The Naive Bayes classifier achieved an accuracy of 51.15%, the first linear regression model achieved an accuracy of 55.12%, and the second linear regression model achieved an accuracy of 33.32%. However, it is important to remember the fact that this problem is a continuous one modelled as a discrete case. Extending the accuracy test to allow for class values ± 1 from the correct value increases the three classifier's accuracy to 81.85%, 88.06%, and 64.88% respectively. A more correct measure of classifier performance would have to consider the weighting that each classifier would give to the values on either side of the chosen one.

5.

I started my experiments by investigating parts of the data that had not yet been used. So far I had not made use of the image numbers (stored in the variable i), so I began by adding this as one of the features used in my linear regression. Unfortunately, however, it did not seem to affect the classifier at all, as I still obtained a perplexity score of 6.5375. I did not attempt to add the image value to my Naive Bayes classifier due to it's lack of effect on linear regression[2].

The next approach that I investigated was using a full neighbourhood for the pixel y that we are attempting to classify. During questions *3* and *4*, we were instructed to use the values $x(:, end)$, $x(:, end - 34)$, and $x(:, end - 35)$. These features correspond to the pixels to the left of, above-and-left of, and above the y pixel. As the pixel above-and-right of the y pixel is also part of it's neighbourhood, and thus most likely influences the y pixel just as much as the other features, I added this feature ($x(:, end - 33)$) to both my Naive Bayes classifier and linear regression. This had a detrimental performance on the perplexity of the Naive Bayes classifier, increasing it to 16176.5, but improved the linear regression's perplexity to 6.4022.

Reasoning that increasing the dimensionality to include the next layer of the neighbourhood around y would cost more than it would improve anything, I then tried to consider if other classifiers might perform better. Throughout the coursework it had seemed like the problem is definitely better represented as a continuous one rather than discrete (given the problems with Naive Bayes and the better performance of linear regression), and so I attempted to see if a *radial basis function* based regressor would outperform the standard linear regression. As with the linear regression I placed a Gaussian over the output values in order to create a probabilistic output. Although the radial basis function network did not

[2]Plus, I will admit, a poor code structure that assumed there would be 64 values for a feature.

outperform the linear regressor, it did achieve a perplexity score of 7.6960 which was close.

Finally, mostly for interest, I also trained a multi-layer perceptron classifier on the data. The perceptron performed fairly poorly, with an average perplexity score of 81.2730 (obtained over 5 runs of the classifier.) I expect that some of this is due to the problem of treating the problem as discrete as I explained earlier, but I did not do a full analysis into using a multi-layer perceptron here.