Perceptron Rule Hyperplane Learning and Kernel PCA

Matthew Sun

1 Abstract

The first purpose of this assignment was to implement the Perceptron rule algorithm to find a hyperplane of separation for a given data set of US colleges and compare the computed hyperplane with one found using logistic regression. It was found that the data was not linearly separable thus the Perceptron rule failed to perfectly separate it and produced an accuracy of 87.64% and AUC of 0.9495. It was also found that the Perceptron rule hyperplane did not perform as well as the linear regression hyperplane in both metrics (accuracy of 91.12% and an AUC of 0.9600). The second purpose of this assignment was to perform kernel PCA on Fisher's Iris data set to separate data based on species. It was found that the two-dimensional kernel PCA score for the data revealed linear separability and when k-means clustering was performed on the score, the cluster indices matched with the original data labels.

2 Introduction

There are two parts to this assignment. The first objective was to use the Perceptron rule to find a hyperplane of separation for the two-dimensional reduction of a data set containing information on US colleges, including a column of labels that indicated whether they were private or not. The hyperplane found from the Perceptron rule algorithm was assessed using its receiver operator characteristic (ROC) curve and its classification accuracy, then compared these metrics with those of the hyperplane found using logistic regression. The second objective was to perform Gaussian kernel PCA on the Fisher's Iris data set which contains a column of labels indicating whether a flower is of the species "setosa" or not. This process maps four-dimensional Iris data to two dimensions. The performance of kernel PCA on the given data set was evaluated by performing k-means clustering on the 2D kernel PCA scores, then plotting the reduction along with the cluster labels provided by k-means. This was compared to another plot that presents the same reduction but with the original labels.

2.1 Perceptron Rule Learning

The Perceptron rule is an algorithm for finding a hyperplane which is a weight vector $\vec{w}$ and a bias value b. These two items are gathered into an augmented weight vector $\hat{w}$. In plain language, the idea behind the Perceptron rule is to update $\hat{w}$ if the classification does not match the label and leave $\hat{w}$ unchanged otherwise. To make these calculations, let each observation have a label $y_i \in \{0, 1\}$ and let the quantization (predicted classification) for each observation be $q_i \in \{0, 1\}$. The residual error between y_i and q_i is $e_i := y_i - q_i$. It follows that that $e_i \in \{-1, 0, 1\}$. Let $\hat{x}_i$ be an observation with label y_i and quantization q_i that is zero if $\hat{x}_i^T \hat{w} < 0$ and one if $\hat{x}_i^T \hat{w} >= 0$. The linear weighted sum $\hat{x}_i^T \hat{w}$ indicates how likely x_i is in the positive or negative class of $\hat{w}$, a larger magnitude of this value indicates a higher probability. Each adjustment of $\hat{w}$ is based on the value of e_i. An e_i of zero indicates a true positive or negative, in either case, the label matches classification so no changes are required for $\hat{w}$. An e_i of -1 indicates a false positive, in this case, meaning that $\hat{w}$ should be adjusted so that $\hat{x}_i$ is closer to the negative class. An e_i of 1 indicates a false negative, in this case, $\hat{w}$ should be adjusted so that $\hat{x}_i$ is closer to the negative class. The adjustments of $\hat{w}$ can be written as the update $\hat{w} \leftarrow \hat{w} + e_i \hat{x}_i$. There is also a learning rate value $r \in$ that determines the magnitude of each update so the general update for the Perceptron rule can be written as $\hat{w} \leftarrow \hat{w} + r e_i \hat{x}_i$.

The Perceptron rule algorithm performs these updates iteratively until there are no misclassifications. For a given data matrix $X \in \mathbb{R}^{m \times n}$, batch learning can be applied by gathering the observations into an augmented design matrix $\hat{X} \in \mathbb{R}^{m \times (n+1)}$, gathering the labels into the vector $\vec{y}$, and gathering quantizations into the vector $\vec{q}$. Doing so generates a more concise expression which, for each iteration k, can be written as

$$\hat{w}_k = \hat{w}_{k-1} + r \hat{X}^T (\vec{y} - \vec{q}) \tag{1}$$

If the data is linearly separable, it is guaranteed that the Perceptron algorithm will converge on some $\hat{w}$ and the algorithm can be initialized with a random starting $\hat{w}_0$. For general data, the algorithm may not converge, resulting in an infinite loop. Furthermore, the computed final $\hat{w}$ might only be a local optimum, not a global one.

The scientific question to be explored is: Can the given data set be separated using the Perceptron Rule algorithm and regardless of whether it can be, how does the hyperplane found by this algorithm compare to the one found using logistic regression? The effectiveness of each hyperplane was assessed by calculating each of their accuracy values and through visual observation of the plotted hyperplane on the two-dimensional reduction of the given data.

2.2 Kernel PCA

Kernel principle component analysis (PCA) is an extension of the PCA and Like PCA, it is used for dimensionality reduction. In the context of binary classification problems, a given data with m data points is often not linearly separable in $d < m$ dimensions but often is in $d \geq m$ dimensions. Kernel PCA is the process of embedding a data matrix to higher dimensions and then applying PCA to the embedded matrix to uncover separability in the data. Let $A \in \mathbb{R}^{m\times n}$ be a data matrix with rows $\underline{a}_i \in \mathbb{R}^n$. The goal is to embed each $\underline{a}_i \in \mathbb{R}^n$ to $\hat{a}_i \in \mathbb{R}^p$ using a kernel function where p is a higher dimension using a kernel function. A kernel function $k(\underline{u}_i, \underline{u}_j)$ is a symmetric function with row vector arguments. There are many different types of kernel functions but for this assignment, the Gaussian kernel function was chosen. This kernel function is defined as $k(\underline{u}_i, \underline{u}_j := \exp(\frac{-||u_i - \underline{u}_j||}{2\sigma^2})$ and is the square of the norm between each argument in the unscale Gaussian distribution $e^{\frac{-x^2}{2}}$. The variance σ^2 is a hyperparameter where a small variance causes the Gram matrix to approach the identity matrix while a large variance causes the Gram matrix to approach some constant value. For finite data $u_i \in$, a Gram matrix $\hat{W}$ can be computed with entries $\hat{W}_{ij} = k(\underline{u}_i, \underline{u}_j)$ and $\hat{W}$ will be positive semi-definite.

Recall that for PCA, the zero-mean matrix M is found for A and can be written as $[I - \frac{1}{m}\vec{1}\vec{1}^T]A$, this will be important later on to ensure that an embedded matrix is zero-mean. Thus centering matrix $G_m = [I - \frac{1}{m}\vec{1}\vec{1}^T]$. The principle components are the eigenvalues of the variable-style scatter matrix $S_V = M^T M$. By expanding M using its singular value decomposition (SVD) $M = U\Sigma V^T$, S_V can be written as the eigendecomposition $S_V = V\Lambda_V V^T$. The PCA score is $Z_V = MV$ and when M is expanded using its SVD, the score can be written as $Z_V = U\Sigma$. Kernel PCA operates with the observation-style scatter matrix $S_U = MM^T = U\Lambda_U U^T$, and since $\Lambda = \Sigma\Sigma^T$, the first r eigenvalues of Λ_V and Λ_U are equal, where r is the rank of M. Similarly, the score of S_U is $Z_U = U\Sigma$. By substituting $M = G_m A$, the observation-style scatter matrix can be written as $S_U = G_m AA^T G_m^T$, whose eigenvectors are the principal components. Observe that each entry (i,j) of AA^T is $\underline{a}_i \underline{a}_j^T = \underline{a}_i \cdot \underline{a}_j$. When performing kernel PCA, A needs to be embedded to $\hat{A}$. Each entry (i,j) of $\hat{A}\hat{A}^T$ is $\phi(\hat{a}_i)\cdot\phi(\hat{a}_j)$ where ϕ is some embedding function that maps $\mathbb{R}^n$ to $\mathbb{R}^p$. However, performing an embedding function for every row is computationally expensive. This can be avoided by computing the Gram matrix instead where each entry (i,j) of $\hat{A}\hat{A}^T$ is $\hat{W}_{ij} = k(\underline{a}_i, \underline{a}_j)$. Thus, the observation-style scatter matrix for kernel PCA can be written as $\hat{S}_U = G_m \hat{W} G_m^T$. Since $\hat{W} \in \mathbb{R}^{m\times m}$ then $\hat{S}_U \in \mathbb{R}^{m\times m}$, so it can be decomposed to find the eigenvectors $\hat{U}$

$$\hat{S}_U = \hat{U}\hat{\Lambda}\hat{U}^T \tag{2}$$

which has score $\hat{Z} = \hat{U}\hat{\Sigma}$.

The scientific question to be explored is: Can the Gaussian kernel PCA scores of the data be separated by k-means clustering and how do the labels provided by the clustering compare with the original data labels? The results were evaluated through visual observation of two plots with the kernel PCA scores, one is labelled by the original data labels while the other is labelled by the cluster indices found by the k-means clustering algorithm.

3 Methods

3.1 Perceptron Rule Algorithm

The given data set concerning US colleges was first processed into a label vector $\vec{y}$ which is the first column of the data containing only +1, -1 and a data matrix $A \in \mathbb{R}^{m \times n}$ that contains all information except the first column. Principle component analysis (PCA) is then performed on the standardized form of A using the MATLAB function "pca". This generated a PCA score that is the two-dimensional reduction of the A which will be denoted as X. Next, the Perceptron rule algorithm was performed on X to estimate a weight vector $\hat{w}$ that is a hyperplane separating the labels.

The algorithm begins with transforming X to an augmented matrix $\hat{X} \in \mathbb{R}^{m \times (n+1)}$ by adding an extra column of ones beside the rightmost column of X. This is done because there is a bias value b associated with $\hat{w}$. The Perceptron rule finds $\hat{w}$ by iteratively updating an estimate of $\hat{w}$. There are many ways to choose an initial estimate but for the sake of simplicity and consistency, the initial estimate was chosen to be an augmented weight vector of all ones. Before the loop begins $\vec{y}$ was converted to a logical array containing labels of zeros and ones representing the negative and positive classes respectively. For each iteration, a new prediction vector is calculated by taking the product between $\hat{X}$ and the current estimate. Each element of the vector is assigned to one if the element is greater than or equal to zero, else it is assigned to zero (this is equivalent to applying a Heaviside function). The residual error was then calculated by taking the difference between $\vec{y}$ and the prediction. The new estimate is computed based on the residual error using batch learning; this is a more concise method of implementing the Perceptron rule as opposed to going through each sample. The new estimate is $new\_est = cur\_est + eta \cdot \hat{X}^T \cdot res\_err$ where the variable eta is a scalar value that determines the learning rate of the algorithm, in this case eta was chosen to be 0.001. This process is repeated until the estimate converges. Since the desired result of this problem is to correctly classify the data, convergence is achieved when there are no misclassified data remaining. This is done in code by taking the maximum absolute column sum norm and checking if it is greater than zero, in other words, the residual error vector is searched for any non-zero values. If this comparison results in a false value then the loop breaks. To avoid performing an impractical number of iterations, the maximum number of iterations was chosen to be 10000. The final estimate is also divided by its Euclidean norm to convert it to a unit vector.

The score of the hyperplane is computed by taking $\hat{X}\hat{w}$. The $\hat{w}$ generated by the Perceptron rule was then compared to the hyperplane generated using logistic regression. Logistic regression was performed on X with assistance from the MATLAB function "glmfit" and its score was found as well. The ROC curve was plotted for both hyperplane scores by finding the true positive (TPR) and false positive (FPR) rates using the MATLAB function "perfcurve". The accuracy of each hyperplane was found by first assigning each element in the score a one if it is greater than or equal to zero, otherwise, it is assigned a zero. The accuracy is the number of occurrences where the assigned value is equal to the corresponding label in $\vec{y}$ divided by the number of observations. Two plots were also created, both displaying X where one contains the hyperplane found by the Perceptron rule and the other with the hyperplane found using logistic regression. The effectiveness of the classifications was evaluated by comparing the accuracy values, the area under the ROC curve and visual observation of the plotted hyperplane. Additionally, the optimal threshold for each score was computed by finding the element in the score that produces the highest accuracy based on the following formula: $acc = \frac{TPR \cdot P + (1-FPR) \cdot N}{m}$, where P and N are the number of actual positives and negatives, respectively and $1 - FPR$ is the true negative rate. The accuracy is calculated this way since only the true positive and false positive rates are known.

3.2 Kernel PCA

The given data set, Fisher's Iris data set, was first processed into a label vector of $\vec{y}$ containing labels of zeros and ones which indicates whether a flower is a member of the "setosa" species or not and a standardized data matrix $X \in \mathbb{R}^{m \times n}$. The observation-style scatter matrix S_U of X was found using the kernel trick and the Gaussian Gram matrix $\hat{W} \in \mathbb{R}^{m \times m}$. The kernel trick is performed by taking $G_m \hat{W} G_m^T$, where G_m is the centring matrix described previously that is used to ensure the Gram matrix is zero-mean. The kernel trick is done to improve the efficiency of this process by avoiding embedding every row one by one. This is possible by taking advantage of the fact that only the dot products of the embedded rows are needed for the entries of the Gram matrix. This decreases the number of operations that need to be carried out and thus improving the computational complexity of the calculation.

$\hat{W}$ was calculated using the definition of Gaussian kernel function with rows of X as arguments $k(\vec{x}_i, \vec{x}_j) := \hat{W}_{ij} := \exp\left(\frac{-||\vec{x}_i - \vec{x}_j||^2}{2\sigma^2}\right)$ where σ^2 is a hyperparameter that in the context of a normal distribution, is the variance. This is done in MATLAB by first taking the square Euclidean distance between each observation using the "pdist2" function then dividing each entry of that $m \times m$ matrix by $2\sigma^2$ and taking e^x of each entry using the MATLAB function "exp". In this problem, σ^2 was chosen to be m as per recommendation. As described previously, the score of a centred Gram matrix is equivalent to $\hat{W}$ multiplied with the first i eigenvectors of $\hat{W}$. In this case, i was chosen to be two for ease of visualization. The score of $\hat{W}$ was then clustered into two sets using the MATLAB function "kmeans" with cluster indices of zeros and ones to match the original labels $\vec{y}$. Two plots were then made, both showing the score of $\hat{W}$, one is labelled with $\vec{y}$ and the other with the cluster indices from the k-means clustering algorithm. The effectiveness of kernel PCA on the given data set was assessed for differences between the two labels and visual confirmation of separability.

4 Results

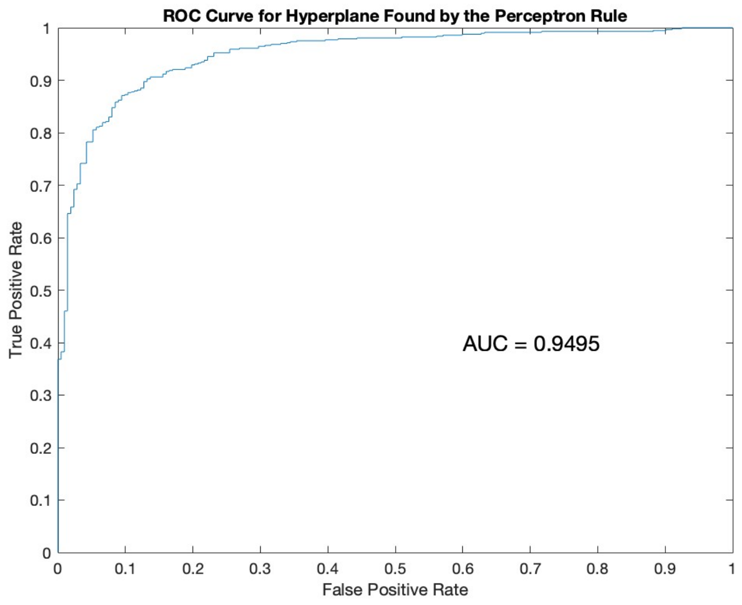

Figure 1: ROC curve for the binary classification produced by the hyperplane found by the Perceptron rule algorithm. The area under the curve is 0.9495.

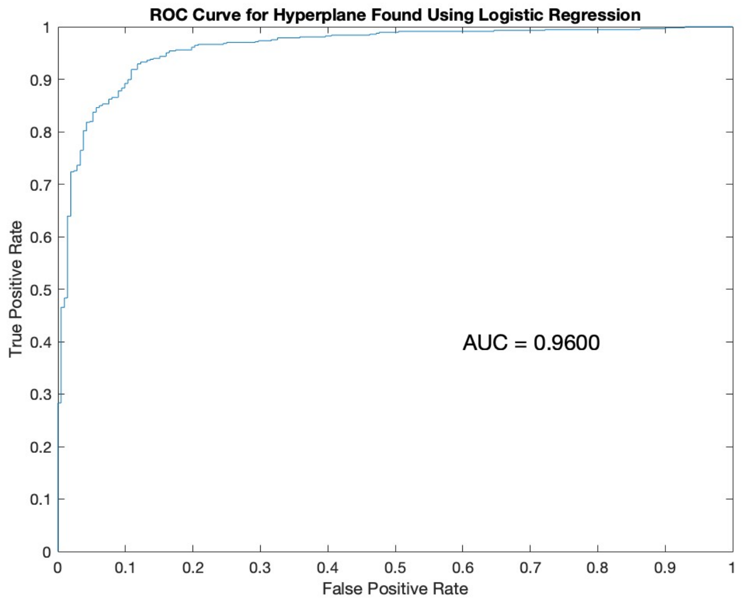

Figure 2: ROC curve for the binary classification produced by the hyperplane found using logistic regression. The area under the curve is 0.9600.

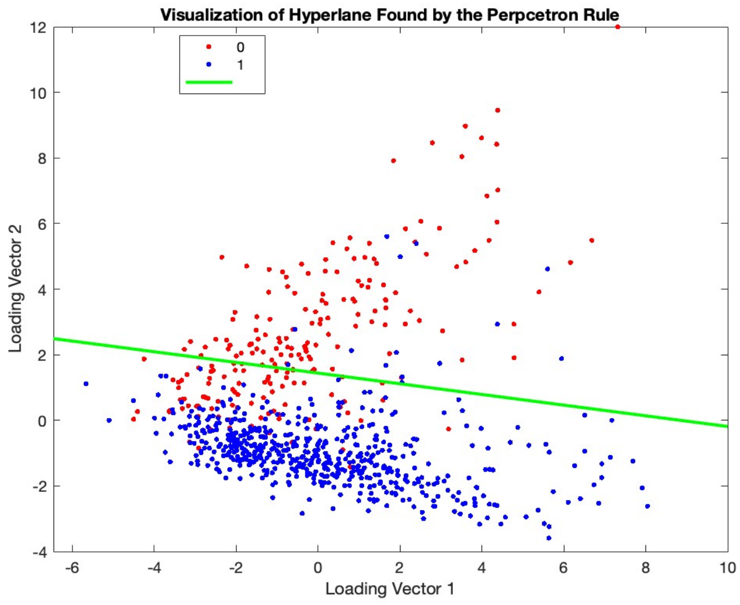

Figure 3: Visualization of the hyperplane found by the Perceptron rule in green and the two-dimensional reduction from PCA. Positive group (1) in blue, negative group (0) in red.

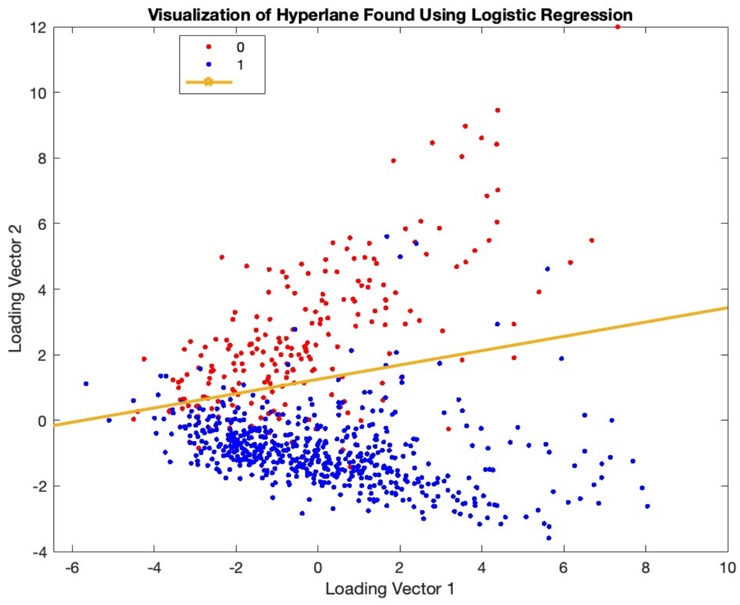

Figure 4: Visualization of the hyperplane found using logistic regression in yellow and the two-dimensional reduction from PCA. Positive group (1) in blue, negative group (0) in red.

Table 1: The two methods used to find the hyperplane of best separation and the accuracy of each rounded to four significant digits.

	Perceptron Rule	**Logistic Regression**
Accuracy	0.8764	0.9112

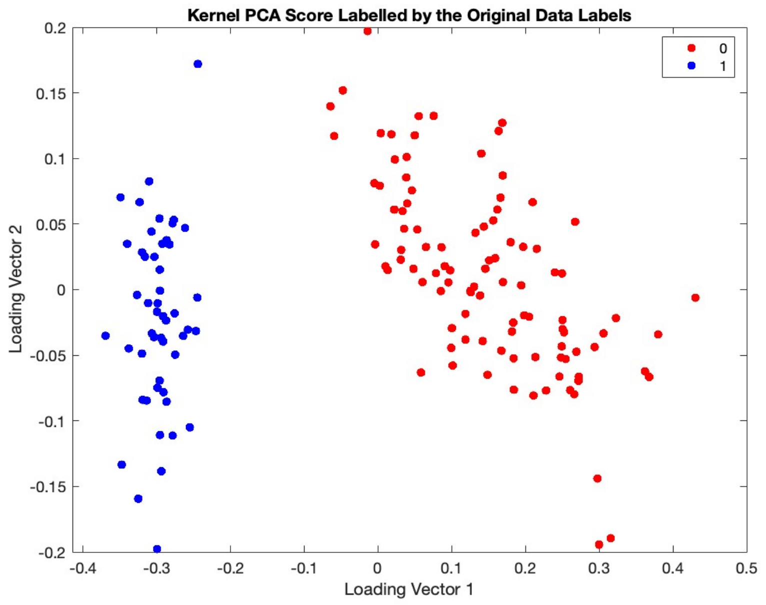

Figure 5: Kernel PCA score labelled with the original data labels. Setosa species (1) in blue and other species (0) in red.

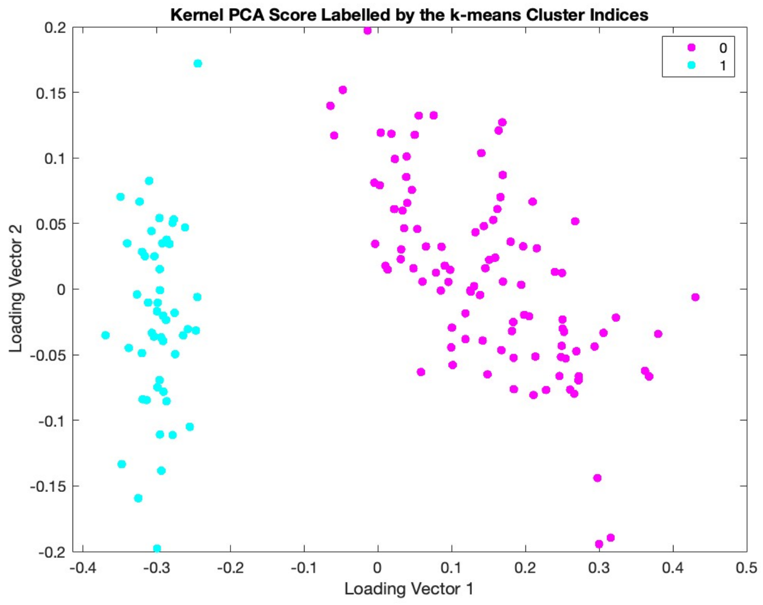

Figure 6: Kernel PCA score labelled with the cluster indices provided by k-means clustering. Setosa species (1) in cyan and other species (0) in magenta.

5 Discussion

5.1 Perceptron Rule Hyperplane

For the given data set of US colleges, it was found that the Perceptron rule algorithm produced a hyperplane with a classification accuracy of 87.64% and the hyperplane produced by logistic regression had a classification accuracy of 91.12%. The ROC curve for the Perceptron rule hyperplane had an AUC of 0.9495 while the AUC of the ROC curve for the logistic regression hyperplane was found to be 0.9600. Note that the AUC as a metric differs from the accuracy as it describes the general effectiveness of the scores of the hyperplane as thresholds for classification rather than measuring the hyperplane directly. It was concluded that the two-dimensional reduction of the data could not be separated by the hyperplane found by the Perceptron rule algorithm and based on the accuracy and AUC values, the Perceptron rule hyperplane did not perform as well as the hyperplane found using logistic regression. Visual observation of the regions near the hyperplane in Figures 3 and 4 would agree that the logistic regression hyperplane appears to separate the data better than the Perceptron rule hyperplane. It is reasonable to conclude from visual observation of Figures 3 and 4 that the two-dimensional reduction is not linearly separable. This is further supported by the fact that the hyperplane of the Perceptron had not converged after 10000 iterations.

These results highlight the shortcomings of the Perceptron rule. Firstly, its inability to converge in general data that is not linearly separable thus causing it to oscillate between values. In addition, the algorithm can only find the local optimum for general data rather than the desired global optimum. Even though it cannot be stated conclusively that the logistic regression hyperplane is the global optimum, it is at least evidence that the Perceptron rule algorithm has not produced the most optimal hyperplane of separation for the given data set. Although the differences between the accuracy ($\sim$ 6.5%) and the AUC ($\sim$ 0.01) were not drastically different, in a context where these metrics are critical performances, logistic regression clearly performed better, at least for this data set. Of course, this one problem is not enough to demonstrate a clear difference but an advantage of logistic regression to consider is that it evaluates data vectors based on the probability of them being in the positive or negative class. On the other hand, the Perceptron rule algorithm implemented in this assignment performs a Heaviside function to determine whether a data vector is on the positive or negative side of the hyperplane and thus does not heavily penalize outliers that are "badly" misclassified, i.e. misclassified and very far away from the hyperplane. This is not the case with logistic regression, where the unrestricted values of the score between the hyperplane and the data vectors are used thus heavily penalizing misclassified outliers. When observing the upper region of Figures 3 and 4, it could be said that there are points susceptible to being misclassified outliers and logistic regression may have done a better job penalizing those occurrences of misclassification.

Perceptron rule learning can be improved upon with additional calculations in conjunction with other models. In a 2016 study [Wib+16], researchers used a Fast Incremental Model Trees with Drift Detection (FIMT-DD) model to create a model for a traffic data stream. This model used a tree structure to train perceptrons in the leaves of the tree. They trained the perceptrons using the hyperbolic tangent function which is beneficial because the values are normalized from -1 to 1 and as they put it, the "hyperbolic tangent (tanh) function is monotonic and enables efficient training when the weights are initialized in a smaller random variable than the sigmoid function. For those reasons, tanh is preferred as the activation function to update the weight because its range transitions within a very small span."

For the given data set, the Perceptron rule algorithm was not able to perfectly separate the data and it was found that the linear regression hyperplane performed better in terms of the accuracy of the classification and the AUC of its ROC curve. Visual observation of the visualization of the hyperplanes would agree. A logistic regression characteristic of considering the probability of classification may be a reason that it performed better in this binary classification problem.

5.2 Kernel PCA

For Fisher's Iris data set, the kernel PCA score provided a linearly separable two-dimensional reduction of the data and it was found that performing k-means clustering on the kernel PCA score generated a clustering that matched the original data labels. From visual observation of Figures 5 and 6, it is clear that there are two regions that could be separated by drawing a line through them. What is more interesting is that when k-means clustering was performed on the kernel PCA score, the clusters not only matched with the expected visual separation but also matched exactly with the original data labels. These are indicators that the kernel PCA was very effective since it was able to find separability and the separability matched the original data labels.

One limitation that holds back kernel PCA (and PCA) is variance inflation. Variance inflation occurs when describes how much the behaviour of one independent variable is influenced by other independent variables. This effect is especially pronounced in smaller data sets of higher dimensions and leads to the model being over-fitted. The authors of a 2011 study [AH11] proposed to include a correction factor when calculating the Gaussian kernel matrix and to execute a leave-one-out procedure to isolate each variable to check for dependencies. In addition to this, kernel PCA is overall a computationally expensive process. Despite tools such as the kernel trick being available, having to compute a Gram matrix with the number of observations squared entries is a computationally demanding task and since in real-life data the number of observations often far exceeds the number of variables, kernel PCA is slower, relative to PCA for example.

For the given data set, it was found that kernel PCA was successful in separating the data and it provided a score of good cluster quality that matched the original data labels. Even without the labels provided by k-means clustering, visual observation of the kernel PCA score would agree that the process performed well in this binary classification problem.

References

[AH11] Trine Julie Abrahamsen and Lars Kai Hansen. *A cure for variance inflation in high dimensional kernel principal component analysis.* Jan. 2011. URL: `https://jmlr.csail.mit.edu/papers/v12/abrahamsen11a.html`.

[Wib+16] Ari Wibisono et al. “Perceptron rule improvement on FIMT-DD for large traffic data stream”. In: *2016 International Joint Conference on Neural Networks (IJCNN)*. 2016, pp. 5161–5167. DOI: 10.1109/IJCNN.2016.7727881.