Rohan Samuel

Multi-Class Classification with Kernel-based Latent Factor Models and Perceptions

Abstract

This report aims to discuss the linear separability of the Iris Dataset that is provided by using three main methods to determine if perfect classification can be produced: PCA Decomposition, Multi-class Perceptron Classification, and High-Dimension Projection through a Kernel Matrix. Each of these three methods has been implemented with Jupyter Notebook - the process and results of each implementation will be discussed and an underlying conclusion of 100% accurate classification is justified through the method of high-dimensional projection.

Libraries required to run the notebook:

- import pandas as pd
- import numpy as np
- import matplotlib.pyplot as plt
- import matplotlib.patches as mpatches
- import seaborn as sns
- from sklearn.metrics import mean_squared_error
- from sklearn.metrics import accuracy_score
- import random
- import math
- import time

A: Data Visualisation

For PCA decomposition, the data set X is entered into the PCADecomposition() function, which computes the SVD **(1)** of the data set; this returns matrices U, S and V where V holds the principle components (the top row being the top PC). Depending on the number of top PCs to select, the top k rows are selected where k is the number of PCs. The new basis Z is computed through matrix multiplication of the weight matrix V and the data set X which is centered - this creates the Z matrix which is a representation of the X matrix in a k-dimensional form.

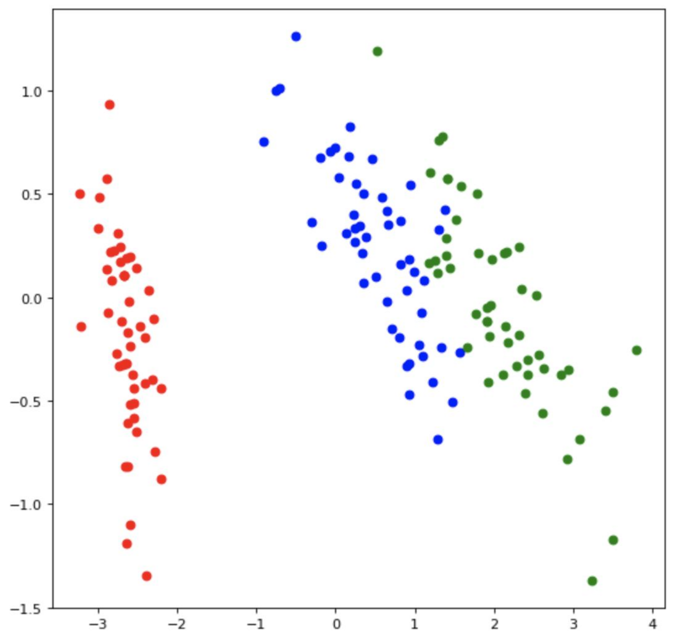


***Figure 1**: 2D Subspace after PCA Decomposition.*

From the observation when plotting the top 2 PCs of the Z basis, as seen in Figure 1, we can see class Iris Setosa is linearly separable from classes Iris Virginica and Iris Versicolour - there is an intersection of data points between Virginica and Versicolour so they cannot be separated with a straight line, but a linear function can be created to separate Setosa from the rest.

B: Multi-Class Perceptron

The multi-class perceptron is implemented using the “one vs all” technique **(2)**; this involves having k classifiers where k = the number of class labels. For each classifier, the perceptron algorithm will run to determine the weight vector for each class - these weights are held in the matrix W which holds k weight vectors for each class. The weights will be the classifiers that will be applied to the training examples Xi to get a score for each class c (Z = WX where Z is a matrix holding all the scores of each class for each row). The weights are updated such that if a prediction is made using weights set to a particular value (initialised to 0), the weight of the correct class is increased by Xi and the weight of the incorrect class is decreased by Xi - this helps promote the correct class and demotes the incorrect class, directing the predictions to the correct values. This update rule also helps to update all values simultaneously rather than individually to increase the classification speed and still maintain the same level of accuracy. In addition to this update rule, and the extra rule is implemented where all the weight vectors that produce a higher prediction than the truth will be reduced and the weight of the true class increases - this updates all classes rather than just 2 classes in the previous rule but it is in addition to the previous class for better classification **(3).** Because the training examples are used in random order and the dataset is mainly non-linearly separable, the perceptron will attempt to get a perfect classifier but it will not always be the case. The weight values will fluctuate due to the random order.

Upon observation of accuracy scores, the multi-class perceptron gives an accuracy as close to 93-95% with each run. Data projected via PCA onto 2 or 4 dimensions give a higher accuracy percentage. On average the original data is 93%, the PCA of 2-d subspace is 95%, and the PCA of 4-d subspace is 97%. The average is found by calculating the mean of the accuracy scores computed when retrieving the 50th iteration weights - the perceptron algorithm is run 50 times and an average is computed.

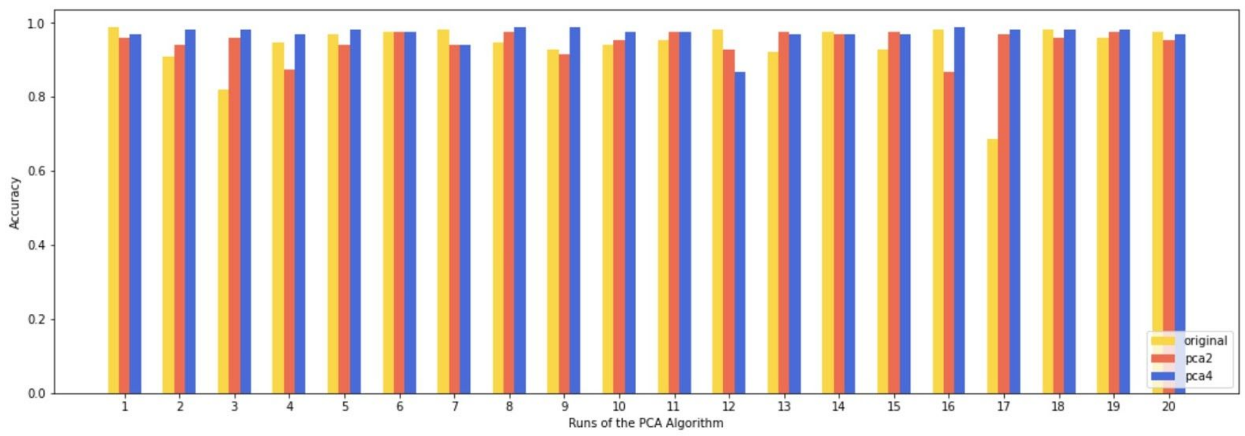


Figure 2: *Bar chart showing the accuracy values for each dataset passed to the multi-class perceptron 20 times.*

Figure 2 above displays how consistent the data in the 4d subspace is in having high accuracy of classification, with the 2d subspace slightly lower with each run and the original data fluctuating but mainly having a lower accuracy.

The cause of the difference in accuracies is because PCA decomposition aims to remove redundant information/features but still summarise the data as accurately as possible. The resulting basis matrix given by the PCA decomposition method is a matrix that provides better classification. Having more principal components like 4d will have improved classification because of the extra parameters used to summarise the original data in X **(4)**; hence, the accuracy value for a PCA 4d subspace has the highest accuracy.

C: Projection onto a high-dimensional subspace

For non-linear classification through the construction of a Kernel matrix, the RBF method is implemented by calculating the range of the hyper-parameter γ (gamma) through the variance of the features in X. The range is calculated by finding the minimum and maximum variance and calculating the formula for γ_{min} and γ_{max} (**add formula)**. The resulting values for both are 0.15911 and 3.55911. Once the range was found, the grid search algorithm is performed implemented such that a combination of different γ values and several top principal components can be used to find a normalised Kernel matrix and the ideal (smallest possible) number of principal components to create a k dimensional project subspace for the multi-class perceptron to perfectly predict the class labels.

Figure 3 below shows the output of the grid search through a heat map diagram provided by the Seaborn library, where the lighter colours represent a higher accuracy value.

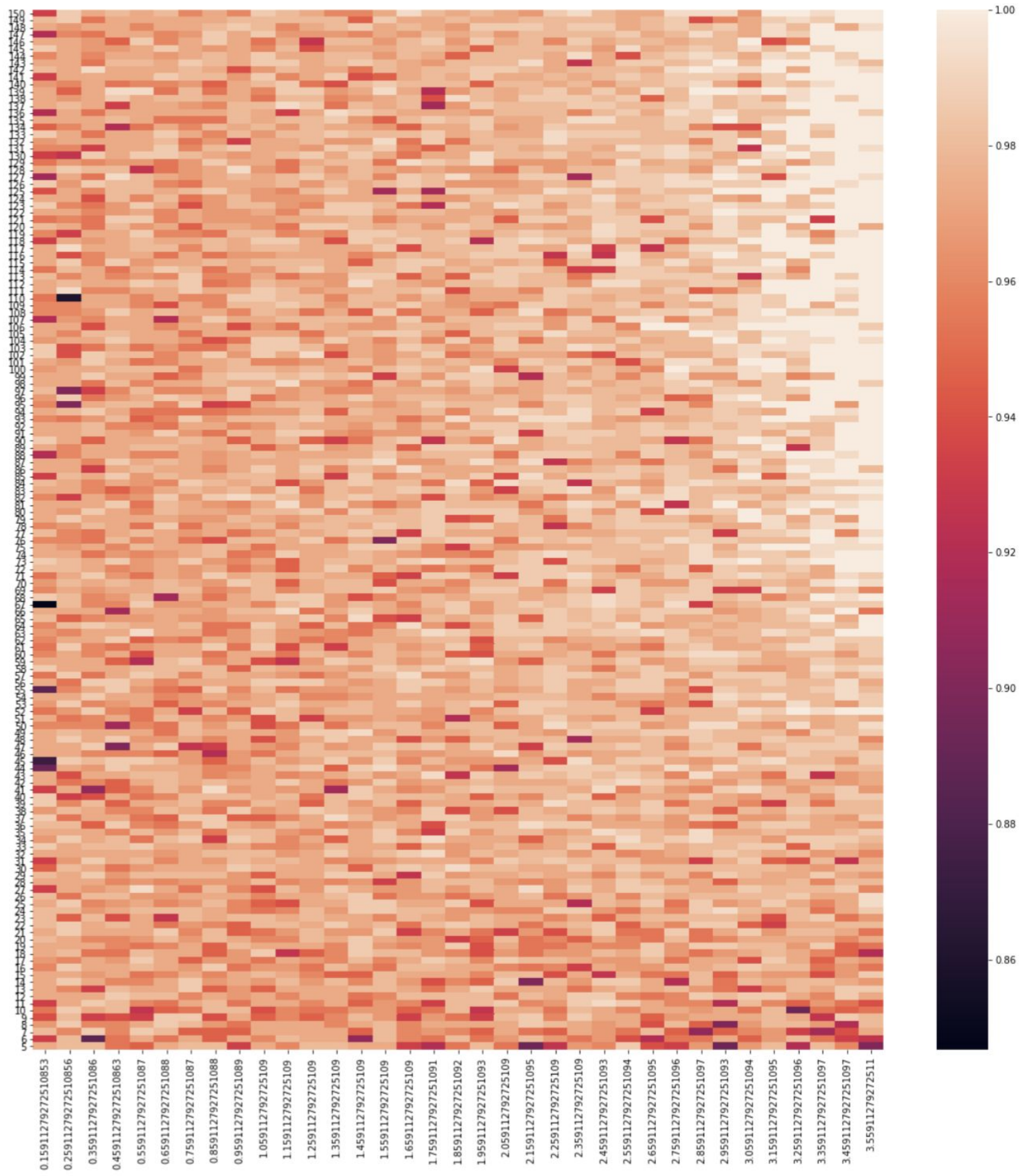


Figure 3: *Grid Search results on a Heatmap*

From the heatmap output produced by the grid search, perfect accuracy tends towards the right of the graph where γ approaches the value, γ_{max}. From γ values of 3.35611, 3.45911 and 3.55911, combinations consisting of almost half of the top principal components from 5 to 150 provide perfect classification. Figure 4 below shows further grid search analysis where the γ range is reduced to between 2.85911 and 3.55911, and also top PCs from 67 to 150. From this further testing, the best value for γ is the max variance, 3.55911 because more than 50% of the principal components that use the Kernel matrix with γ_{max} will produce a perfect classification.

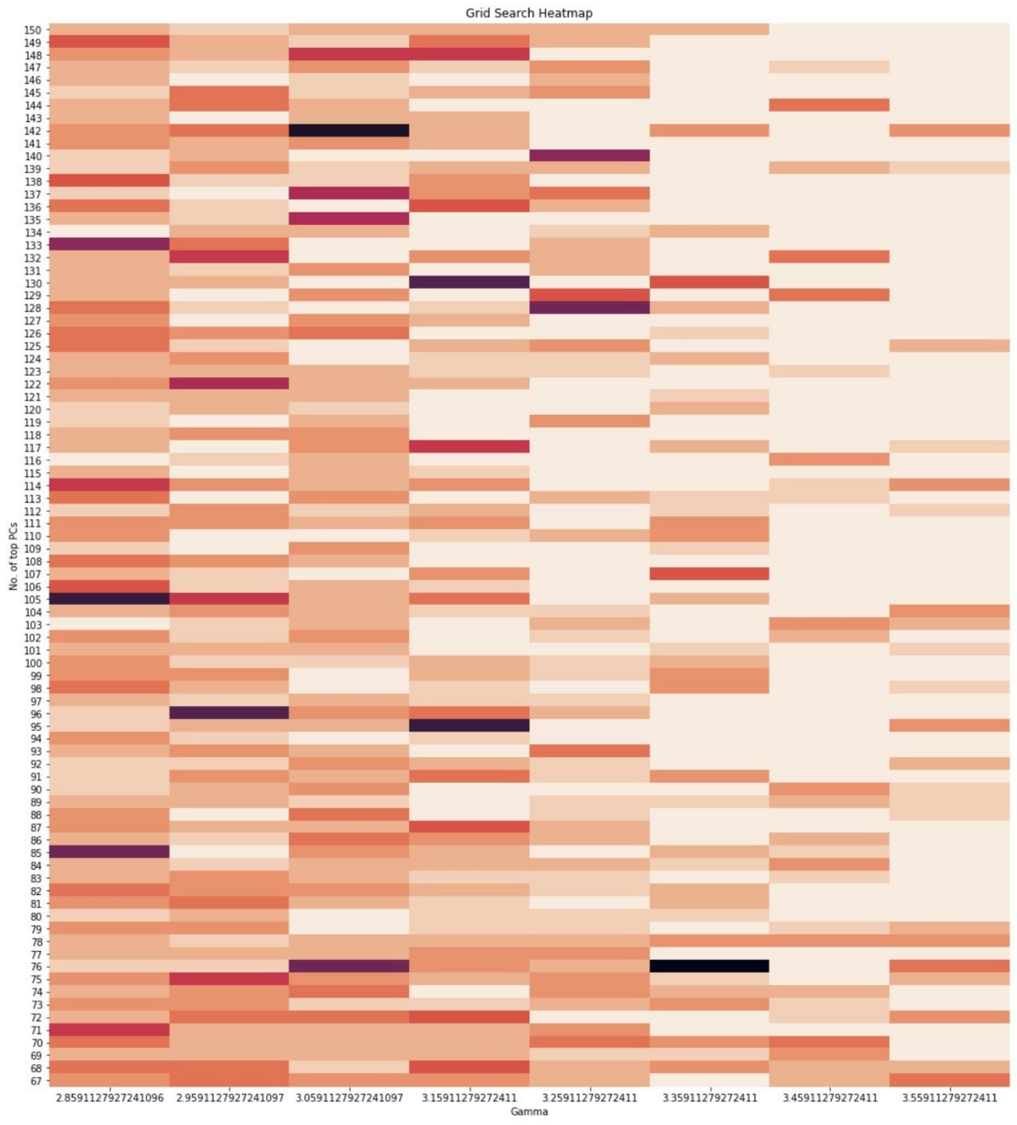
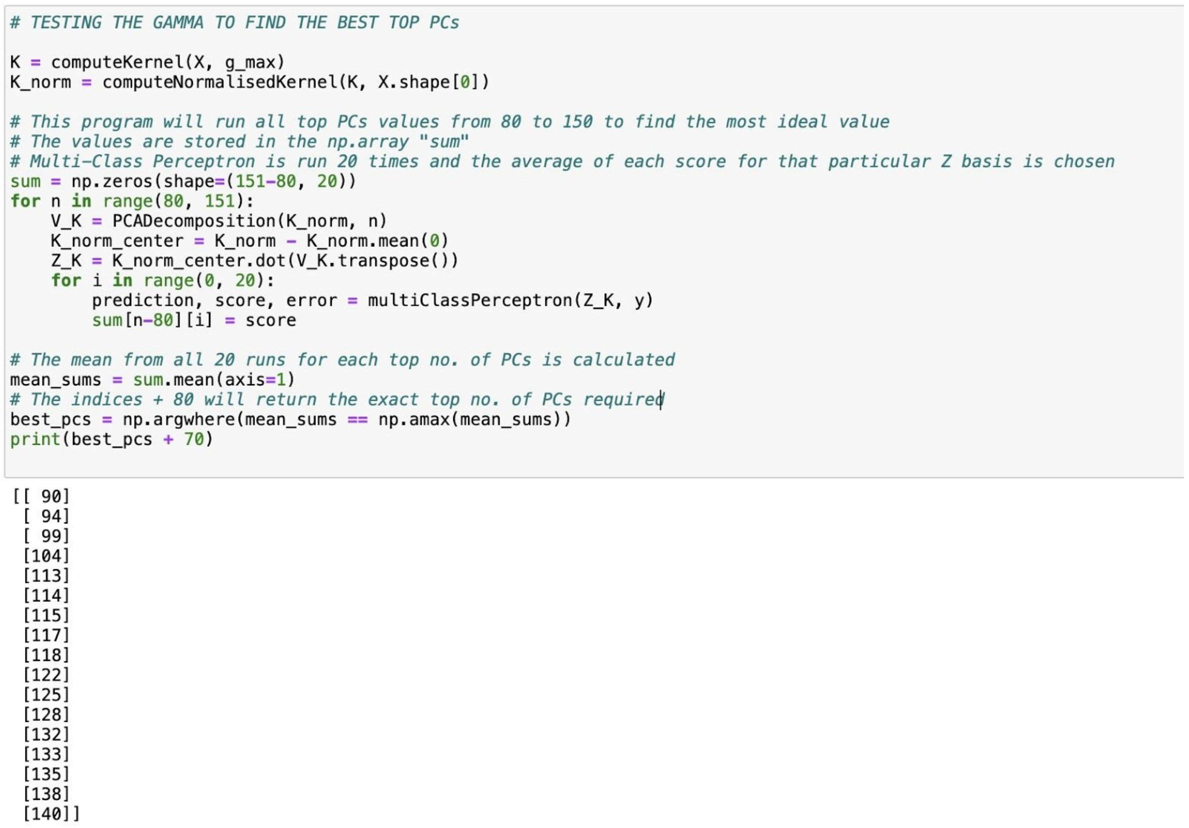


***Figure 4**: Grid Search of a specific γ range and number of top PCs based on Figure 3*

Using γ=3.55911, the grid search identifies the number of top PCs from 94 to 150 to have 100% accuracy. Ideally, the best number of top PCs around 80 onwards because the average classification score when running the perceptron 20 times is 99.9%-100%. For specificity, the multi-class perceptron was run 20 times to compute an average accuracy for a Kernel with the previously specified γ, and a range of top PCs from 80 to 150 since that was the best range examined in Figure 4.



```
# TESTING THE GAMMA TO FIND THE BEST TOP PCs

K = computeKernel(X, g_max)
K_norm = computeNormalisedKernel(K, X.shape[0])

# This program will run all top PCs values from 80 to 150 to find the most ideal value
# The values are stored in the np.array "sum"
# Multi-Class Perceptron is run 20 times and the average of each score for that particular Z basis is chosen
sum = np.zeros(shape=(151-80, 20))
for n in range(80, 151):
    V_K = PCADecomposition(K_norm, n)
    K_norm_center = K_norm - K_norm.mean(0)
    Z_K = K_norm_center.dot(V_K.transpose())
    for i in range(0, 20):
        prediction, score, error = multiClassPerceptron(Z_K, y)
        sum[n-80][i] = score

# The mean from all 20 runs for each top no. of PCs is calculated
mean_sums = sum.mean(axis=1)
# The indices + 80 will return the exact top no. of PCs required
best_pcs = np.argwhere(mean_sums == np.amax(mean_sums))
print(best_pcs + 70)
```

```
[[ 90]
 [ 94]
 [ 99]
 [104]
 [113]
 [114]
 [115]
 [117]
 [118]
 [122]
 [125]
 [128]
 [132]
 [133]
 [135]
 [138]
 [140]]
```

Figure 5: *Program code to test for the best no. of top PCs*

From further testing using the program created in Figure 5, it was discovered that the best top number of PCs to use is 94. This is an ideal number since fewer parameters are used to achieve perfect classification, hence the computational cost for classification is smaller than having a higher number of top PCs. In conclusion, the observations of the heatmap given by Grid Search show that the Iris dataset can be linearly classified in a high dimensional subspace.

References

1. The SciPy Community. Numpylinalgsvd - NumPy v119 Manual. [Online]. Available from: https://numpy.org/doc/stable/reference/generated/numpy.linalg.svd.html [Accessed 28 November 2020].
2. Sanchez, V. Part II: ML Linear Classifiers. [Online]. Available from: https://warwick.ac.uk/fac/sci/dcs/teaching/material/cs342/lecture14_linearclassifiers_amended.pdf [Accessed 7 December 2020].
3. Keshet, Y. Multiclass Classification. [Online]. Available from: https://u.cs.biu.ac.il/~jkeshet/teaching/aml2016/multiclass.pdf [Accessed 1 December 2020].
4. Lopes, M. Dimensionality Reduction — Does PCA really improve classification outcome?. [Online]. Available from: https://towardsdatascience.com/dimensionality-reduction-does-pca-really-improve-classification-outcome-6e9ba21f0a32 [Accessed 30 November 2020].