Maynooth University
National University of Ireland Maynooth

Department of Computer Science
CS401 Machine Learning & Neural Networks
HW2 Report

NAME: **ROSS HALPIN**

STUDENT NUMBER: **14558373**

E-MAIL: **ROSS.HALPIN.2015@MUMAIL.IE**

1. Finding Hidden Structure

In order to visualise any structure within a high dimensional dataset, I must reduce the dimensions to a format which can be intuitively visualised such as 3D or preferably 2D. I explored a few different linear and non-linear dimensionality reducing algorithms such as Principal Component Analysis(PCA) and Independent Component Analysis(ICA). I reduced the dimensions down from 8 to 3, I then plotted the points in 3D but did not see any obvious structure. I then reduce the dimensions down from 8 to 2 and still could not see any obvious structure. In searching and testing many different possible solutions I came across Contrastive Principle Component Analysis. The cPCA algorithm “takes as input datasets X and Y and efficiently identifies lower-dimensional subspaces that capture structure specific to X”, it’s goal being to “identify directions v which account for large variances in the target and small variances in the background” (Abid et al., 2018).

I didn’t need to delve deep into how the CPCA algorithm works, as an abstracted library is available freely to use. My initial exploration with this algorithm led to revealing there was possible contrastive structure within the dataset. This can be seen in Figure 1.

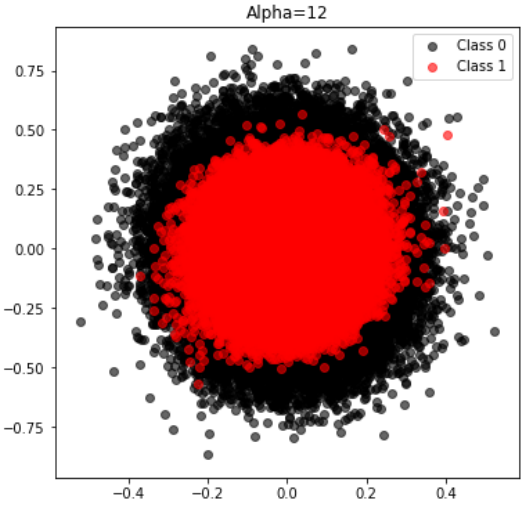

Figure 1.0. cPCA performed on training dataset, alpha being a contrast hyperparameter.

Next I tried looking at the densities within the classes. First I performed cPCA, contrasting Class 0 against Class 1 and vice versa. by plotting them on 2D histograms, along with a plot of the Kernel Density Estimate(KDE).

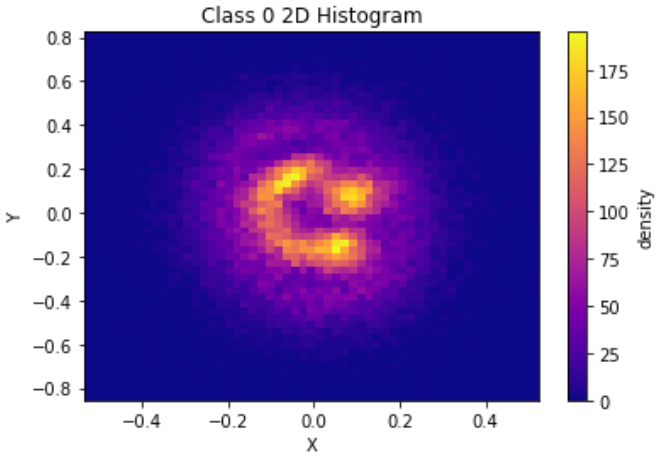

Figure 1.1. 2D histogram of Class 0 contrasted with Class 1.

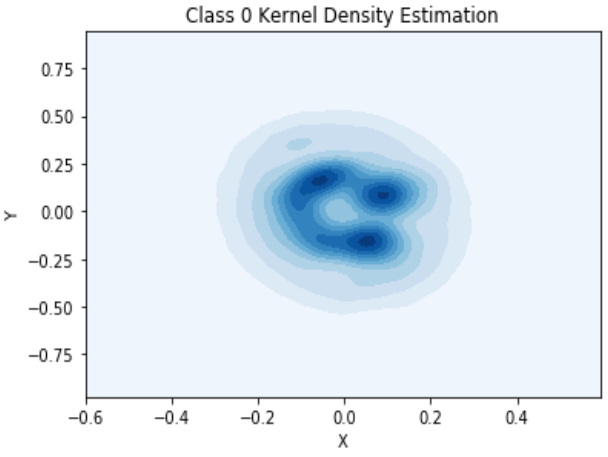

Figure 1.2. KDE plot of Class 0 contrasted with Class 1.

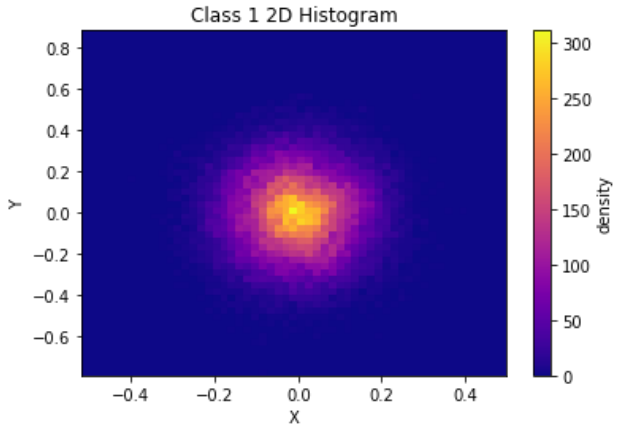

Figure 1.3. 2D histogram of Class 1 contrasted with Class 0.

It was clear from looking at Figure 1.1 and 1.2 that there appeared to be an interesting structure within the dataset. Initially it appeared that there was no interesting structure within Class 1, but in experimenting with the cPCA algorithm I contrasted Class 1 against itself. This produced the following histogram:

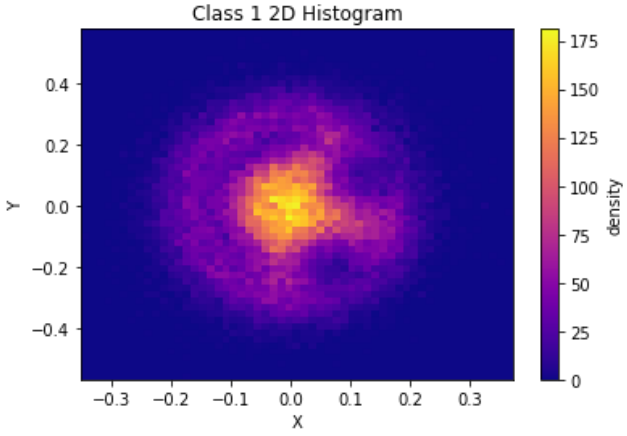

Figure 1.4 2D histogram of Class 1 contrasted with itself.

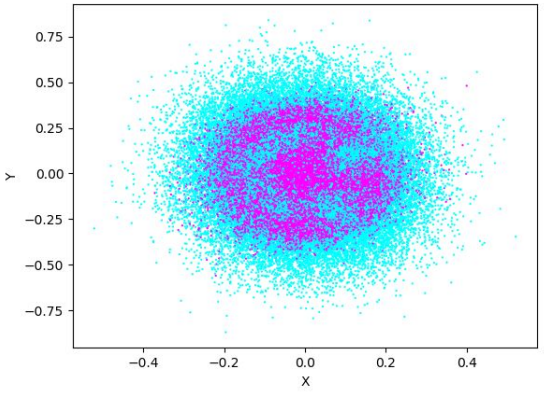

Figure 1.5 Scatter Plot of entire Training set contrasted with Class 1.

As can be seen in Figure 1.4, there is without a doubt a smiley face shaped structure within this dataset, which can be found in both classes. Class 0 appears to feature the structure in its the more dense points, while Class 1 features within the less dense points. This leads me to believe the structure was possibly independently added to Class 0 and subtracted from Class 1. Figure 1.5 shows a scatter plot displaying the structure.

2. Data Preprocessing & Dimensionality Reduction

As finding the hidden structure gave me an avenue to classify the dataset, I simply repeated some of the process in preprocessing the data. I loaded the dataset, but I specifically chose to perform cPCA with the training set as the target, and Class 1 as the background data. This is because doing so revealed the smiley face structure when the data was plotted on a scatter plot as can be seen in Figure 1.5. This structure is what I want my ML algorithm to identify and learn from. I reduced the features dimensionality to 2 features. I then calculated the Kernel Density Estimation(KDE) of these 2D points. The KDE algorithm, will smoothly fit the 2D points to a probability density function. From this smoothed distribution we get the estimated density values.

I then compiled all the data I had created, the cPCA reduced 2D points and their corresponding KDE values, along with the given feature set, into a new feature set for training. I then split my data into training, validation and test sets. The ratio being 80:10:10. The algorithm will be trained and validated, then tested.

3. Training Deep Sequential Neural Network

As I had initially had poor results with some sklearn machine learning algorithms such as the multi layer perceptron algorithm, I switched to using a simple “Deep Sequential Neural Network” using both Keras and TensorFlow. The model featured two hidden layers, each with 100 hidden units. At first the model was overfitting by a large margin, so I also used two dropout layers and kernel regularizers. The dropout layer “consists in randomly setting a fraction rate of input units to 0 at each update during training time” (Keras.io, 2018a), while kernel regularizers “apply penalties on layer parameters or layer activity during optimization. These penalties are incorporated in the loss function that the network optimizes” (Keras.io, 2018b). These will work to combat overfitting.

As I had never used deep learning before, I didn’t get too tied down in the hyperparameters. I merely applied a very simple example to my extracted features. I ran the algorithm over 100 epochs in order to achieve a satisfactory accuracy value. I also tweaked the regularizer and dropout values until overfitting had reduced to almost none.

```
# Sequentail Model https://github.com/Msanjayds/Keras/blob/master/Classification_Model_using_Keras.ipynb
model = Sequential()
# Hidden Layer-1
model.add(Dense(100,activation='relu',input_dim=11,kernel_regularizer=l2(0.0001)))
model.add(Dropout(0.05, noise_shape=None, seed=None))
# Hidden Layer-2
model.add(Dense(100,activation='relu',kernel_regularizer=l2(0.0001)))
model.add(Dropout(0.05, noise_shape=None, seed=None))
# Output Layer
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])

model_output = model.fit(X_train,y_train,epochs=100,batch_size=1000,verbose=1,validation_data=(X_valid,y_valid),)
```

Figure 3.0. Model structure.

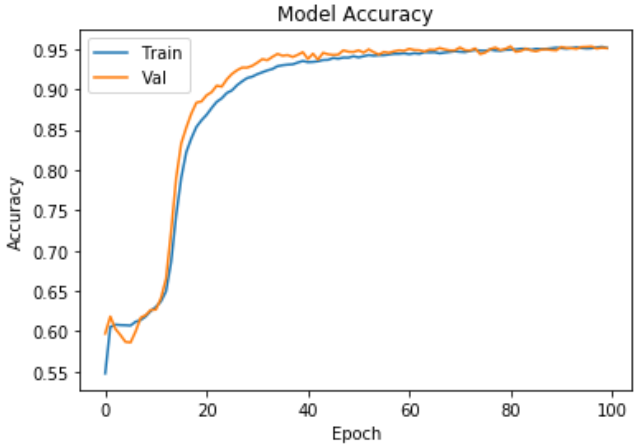

Figure 3.1. Model Accuracy of training and validation sets over 100 epochs

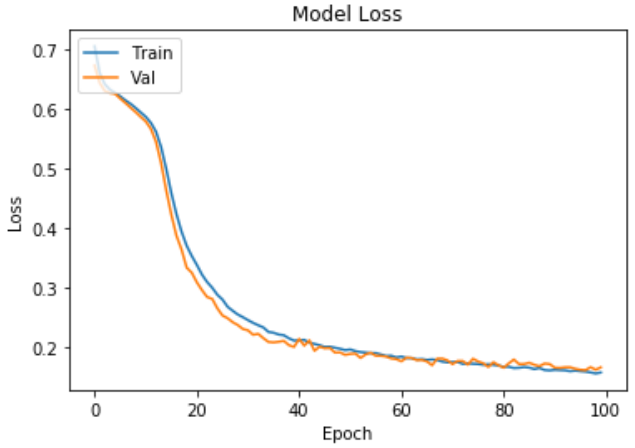

Figure 3.2. Model Loss of training and validation sets over 100 epochs.

As can be seen in Figure 3.1. The training accuracy and validation accuracy increases significantly over 100 epochs. The model attempts to maximise the accuracy while minimising the loss function. This can be seen in Figure 3.2.

```
Epoch 96/100
80000/80000 [==============================] - 1s 10us/step - loss: 0.1559 - acc: 0.9526 - val_loss: 0.1669 - val_acc: 0.9494
Epoch 97/100
80000/80000 [==============================] - 1s 10us/step - loss: 0.1579 - acc: 0.9517 - val_loss: 0.1699 - val_acc: 0.9477
Epoch 98/100
80000/80000 [==============================] - 1s 10us/step - loss: 0.1550 - acc: 0.9527 - val_loss: 0.1615 - val_acc: 0.9540
Epoch 99/100
80000/80000 [==============================] - 1s 9us/step - loss: 0.1545 - acc: 0.9534 - val_loss: 0.1618 - val_acc: 0.9502
Epoch 100/100
```

Figure 3.3. How the model improved over 100 epochs.

As can be seen in Figure 3.3. The final ‘Epoch 100’ features an accuracy of 0.9534 and a validation accuracy of 0.9502. As the accuracy and validation are very close this means the model should be generalising well. To ensure the algorithm was working I correctly I then used the model to predict labels for my test set. K fold cross validation is generally avoided due to having to run different models for each K fold, this would be expensive computationally and take a significant amount of time.

	precision	recall	f1-score	support
0.0	0.93	0.96	0.95	4872
1.0	0.96	0.94	0.95	5128
micro avg	0.95	0.95	0.95	10000
macro avg	0.95	0.95	0.95	10000
weighted avg	0.95	0.95	0.95	10000

Figure 3.4. Test set precision, recall and f1-score.

As the model appears to be achieving very good results across the board, as can be seen in Figure 3.4., I then reproduced the process on the “test.txt” dataset. I first reduced the data set to 2 dimensions using cPCA, contrasting the dataset with Class 1 from the training set in order to presumably reveal the smiley face structure. I then calculated the KDE of the resulting 2D set. Once again I compiled the original feature set, the 2D reduced set and the KDE set into one new feature set. I then used my trained model to predict the labels of this set.

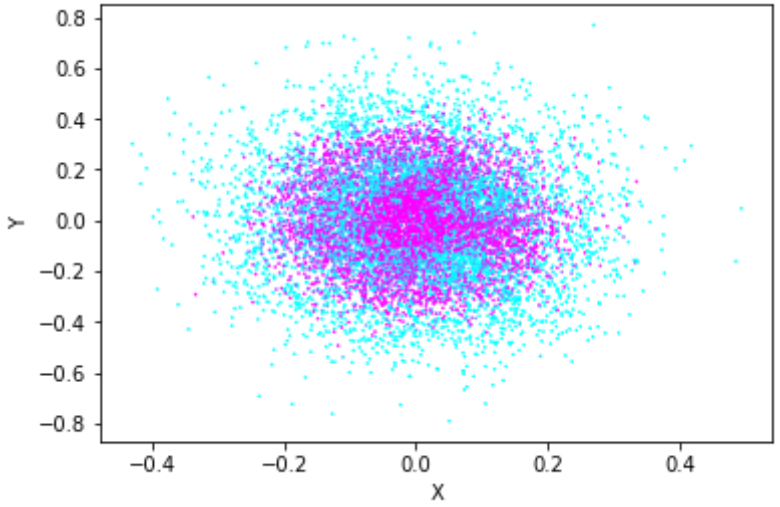

Figure 3.5.Scatter plot of test.txt dataset after classification

I was able to check my performance on the “test.txt” data by plotting it after predicting the labels. As the smiley face structure was revealed as seen in Figure 3.5., I am confident that my model is performing well on the “test.txt” data and should achieve similar results to test and validation scores of around 95% accuracy.

Citations

A. Abid, A., Bagaria, V., Zhang, M. and Zou, J. (2017). Contrastive Principal Component Analysis. [online] Available at: https://arxiv.org/abs/1709.06716 [Accessed 2 Dec. 2018].

B. Keras.io. (2018a). *Core Layers - Keras Documentation*. [online] Available at: https://keras.io/layers/core/ [Accessed 2 Dec. 2018].

C. Keras.io. (2018b). *Regularizers - Keras Documentation*. [online] Available at: https://keras.io/regularizers/ [Accessed 2 Dec. 2018].