Machine Learning for Visual Computing: Assignment 1

Bahramali Mohammadali – 1329789

Sofía Pérez Pérez – 12121328

Inés Martínez Merino - 12121300

PART 1: Binary classifications and the perceptron

1.1.1 The MNIST data set

The aim of this part is to plot two chosen classes of the MNIST data set to see if they are linearly separable. The two classes we chose are *t-shirts* (label = 0) and *sandals* (label = 5). Then, we obtained a subset of the training and test sets containing only those instances with these associated labels.

In order to, plot two relevant features, we decided to implement **principal component analysis** (PCA), whose aim is to reduce the dimensionality of a data set containing a large number of variables. The original variables are transformed into a new set of variables (principal components) that can be ordered according to the variance they explain of the original variables. Hence, after transforming our original data set and obtaining its principal components, we first plotted the explained variance of all features and observed that the first two contained more than 50% of the original variance:

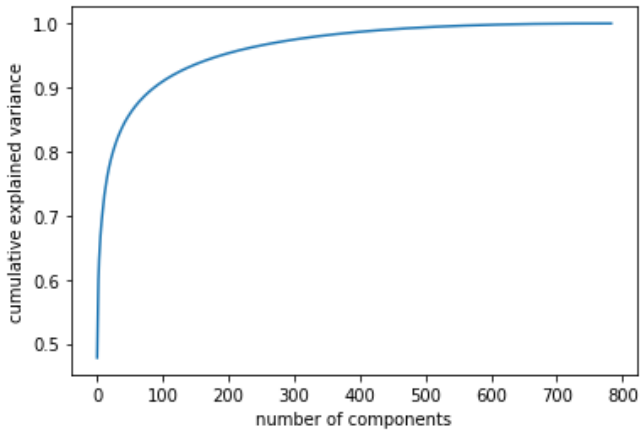

Figure1:

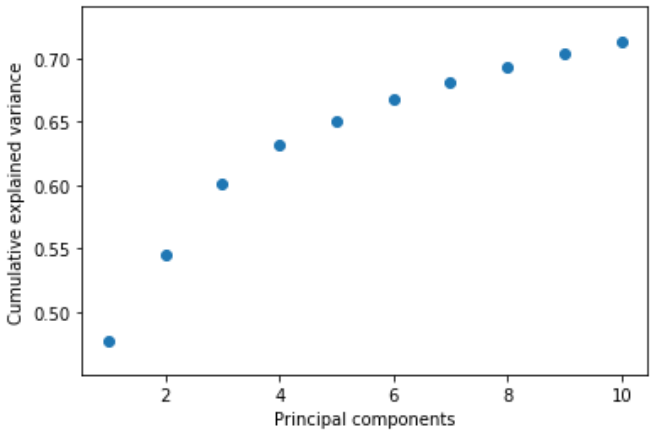

Figure 2:

After selecting these features, we plotted our data set with labels *t-shirts* and *sandals* and obtained the following plot:

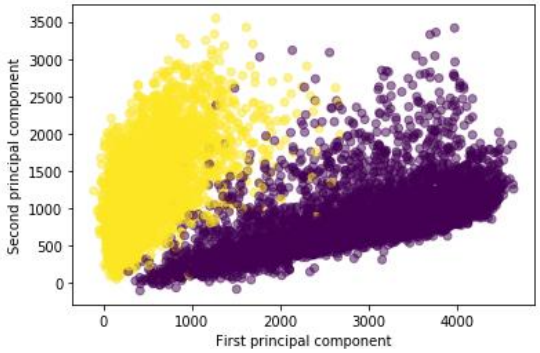

Figure3: plot of the data set with labels t-shirts and sandals in two different colours.

As we see in figure3, despite existing some outliers (noise), the data is linearly separable. It means, we can draw a line to separate both classes.

1.1.2 Perceptron training algorithm

1. Implementing both functions:

First of all, as the perceptron algorithm works with labels {-1, 1}, we changed those observations with labels 0 to -1 and 5 to 1. Then, we changed our data such that it was represented by homogeneous coordinates, that is, it has an additional first feature that has value 1. Finally, we implemented the functions **accuracy** and **error rate**, which receive as parameters the **w** obtained after training the perceptron, the training data, and their labels.

The function **perc Train** takes as parameters the training data, its parameters, the maximum number of iterations it should run through and whether it should run the online version or the batch-version. The **w** vector is initialized as a Gaussian random vector, and it will be updated until it reaches the maximum number of iterations. One iteration is achieved after updating the **w** vector when a randomly selected observation has been wrongly predicted using the **perc** function in the online- version. For the batch-version, one iteration means updating the **w** vector after predicting the labels of all observations and using those predictions at the same time to update **w**.

2. Two-feature training data

After training the perceptron with the data obtained after implementing PCA, we get the following decision boundary:

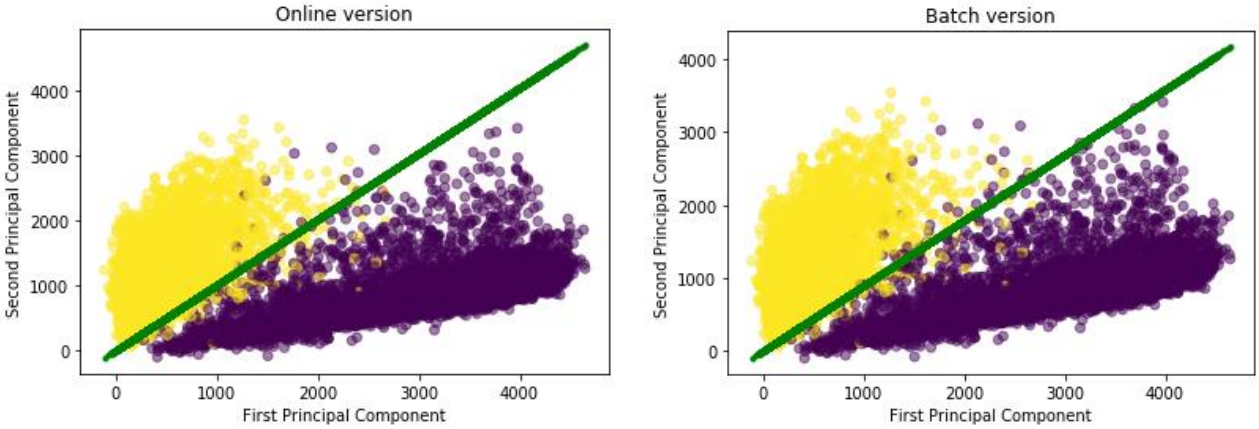

Figure 4: Decision boundary in online version and Batch version after training the perceptron with the data.

As stated before, the data is linearly separable, and the perceptron should converge into a weight vector that separates the data points. Both plots show the expected output: a decision boundary that fully separates both classes. Both online and batch version gave similar accuracies for the training set: 0.9858 for the former and 0.9894 for the latter.

The perceptron training algorithm will always converge if the data is linearly separable. Thus, there is always a fixed maximum number of iterations the algorithm will reach depending on the

data. If the algorithm has not converged once it has reached the maximum number of iterations, this means that the data is not linearly separable. Thus, we could say that the perceptron training algorithm is capable of detecting linearly non-separable data.

3. Five-feature training data

To crate this new training data, we applied the required transformation to each observation in the data set where we implemented PCA. Then, we trained two weight vectors using this data set: one using the online version and the other one using the batch version.

To plot the result obtained, a subset of the grid was selected such that it is enough to visualize the decision boundary. Then, the previous transformation implemented in the training data set was applied to all data points in the selected subset. Afterwards, we multiplied the trained weight vector (batch-version) with all these data points to obtain their predicted class. Finally, we plotted the contour function and obtained the following:

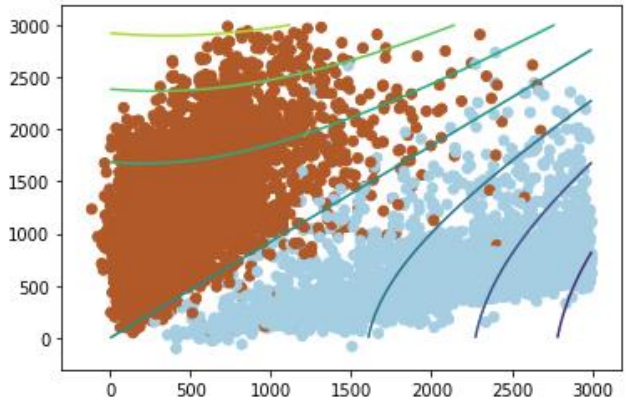

Figure 5.

We can see that both classes are being separated: the contour lines over the data points that belong to one class are bluer as they are further from the decision boundary, and the ones over the points that belong to the other class are yellower as they are further from the decision boundary.

4. All-pixels training data

To create this new training data, we used the initial training set with all pixels but adding homogeneous coordinates (784+1 features). Then, we trained the weight vector using the online version of the perceptron algorithm and this new data set. Finally, we plotted the obtained decision boundary by visualizing it as a Gray-scale image:

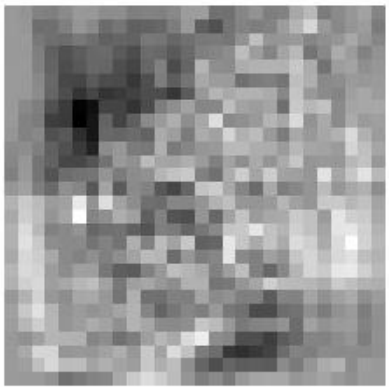

Figure 6: plot obtained decision boundary by visualizing it as a Gray-scale image

Although the error-rate on the training data set is very small (0.00325), when we plot the weight vector obtained, we are not able to distinguish between both classes. That is, we cannot distinguish

a sandal or a t-shirt. This may be because we trained it using the online-version algorithm instead of the batch-version. Another reason could be that we did not have enough images on the training data set.

5. Error rate

Before computing the error rate on each test set, we need to apply the same transformations we implemented in the training sets. First, we change its labels to 1 or -1 in the same way as we did for the training sets. Finally, we create three different test sets by transforming the initial test set (all of them with homogeneous coordinates): one with 2 features, another one with 5 features and the last one containing all pixels.

The error rates obtained on each test set using the online version obtained weight vectors are the following:

- PCA test set: 0.502
- 5-feature test set: 0.5015000000000001
- All-feature test set: 0.49950000000000006

As we can observe, these error rates are slightly decreasing while increasing the number of features of the training set the weight vector has been trained with. This is due to the fact that the more features the training set has, that is, the more information it has, the more accurate the weight vector will be.

PART 2: Linear basis function models for regression

1.2.1 Experimental setup

We have generated the data by subsampling the interval [0,5] in step size of 0.1. Regarding the target values, they have been obtained from the function $f(x) = 2x^2 - 11x + 1$ plus an added normal noise.

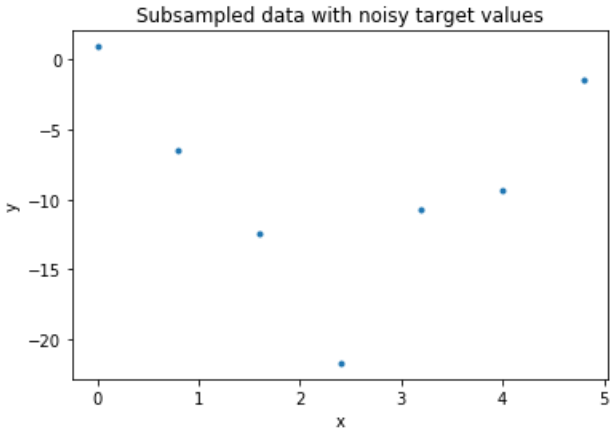

Figure 2.1. subsampled data including 7 points with noisy target values

As we can see in the plot, since the normal noise comes from a distribution where the standard deviation is 4, the target values differ a lot from the original ones.

1.2.2 Optimization: LMS-learning rule vs. closed form

The exercise asks to obtain the coefficients of a polynomial such that the error obtained when predicting the training points with those results, compared with the original target values is

minimized. For this purpose, we will first compute w by using the online LMS learning rule and, afterwards, with the closed form formula.

On the one hand, in order to compute the vector w* by using the online LMS rule, we have used the updating formula $w_{k+1} = w_k + [t_{nk}(x_{nk})^T w_k]x_k$. Since the online version is used, for each observation w is updated. One of the main problems encountered when trying to find the optimal w, was that when training the algorithm with ordered data, it did not learn properly. Therefore, one of the pre-processing steps that we included was to shuffle the data, so the order of the observations is random. This improves the predictive performance (w was more accurate and more similar to the original polynomial coefficients). Afterwards, another issue regarding the quality of the solution, was to determine the learning rate and the maximum number of iterations. We came up with the conclusion that a small value of the learning rate (0.00005) led to better results, since otherwise the w diverges. In addition, a lot of iterations (150000), are needed so w converges. Furthermore, the first coefficient of w was the most difficult to converge since for its learning it only used the first components of the training points, and they are always 1.

On the other hand, when computing w* in closed form, we used the Moore-Penrose-inverse formula.

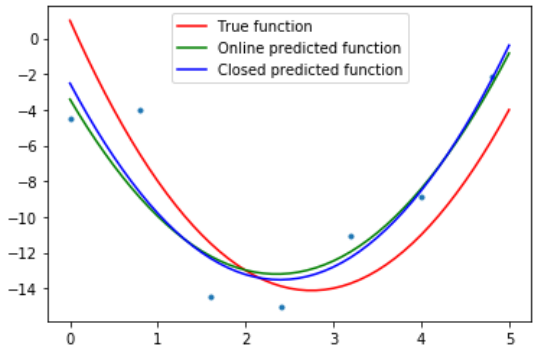

Figure 2.2. true function (red) in comparison to online predicted function (green) and closed prediction function(blue). Dots are the target value

As we can observe in Figure 2.2, the blue and green polynomials are both a good approximation of the true function (red). There is not a better solution since, depending on the point, the optimal w* may differ. For example, for the point x = 0.8, the optimal w* is obtained with the closed form. However, for the point x = 0.0, the optimal w* is the corresponding solution of the LMS rule.

1.2.3 Image data

We followed the process described in the statement to create the image data.

When computing the w* in closed form, Python raised an error which said that the $\phi(x)\phi(x)^T$ was non-invertible. Therefore, we had to add the term λI, where lambda takes a small value (0.2). We then have the following expression for w in closed form:

$$w = (\phi\phi^T + \lambda I)^{-1}\phi t^T$$

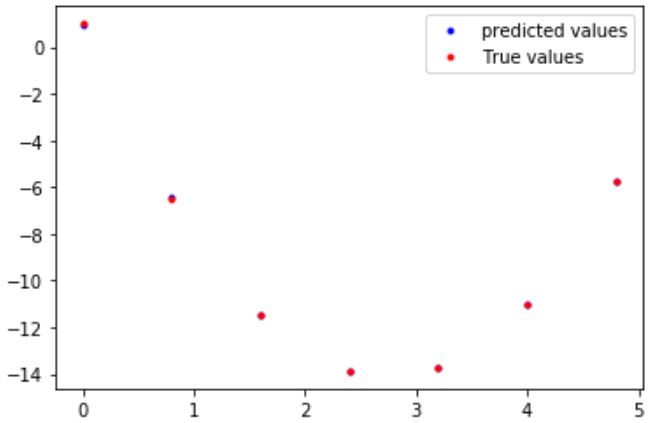

Figure 2.3. predicted points (blue) vs true values

As it can be seen in Figure2.3 the predicted points (blue) are almost overlapping the true values. This makes sense since these points are the ones used for the training process.

Therefore, the obtained coefficients for the polynomial are accurate.

The MSE of the experiment is 0.0019, which is really small (almost perfect interpolation).

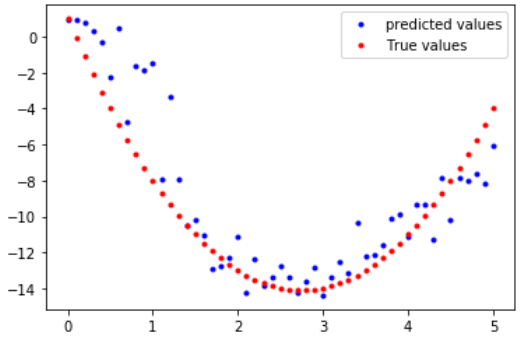

Figure 2.4. predicted values vs true values when we use the whole data set including 51 pictures.

For this last experiment, instead of using 7 points, we used the whole data set. As it can be seen, the predictions in this part are not as accurate as the ones obtained using only 7 points of the training set, which is reasonable since the algorithm did not use these points for the training process.

However, the overall behaviour of w* when predicting unseen points is good. The predicted values simulate the shape of the original function.

The MSE for this experiment is 4.8925, which is larger than the one obtained when predicted training instances. However, this is logic since w* is predicting unseen observations.

If we increase the noise variance of the centres, the resulting w* is not capable of predicting the points that well, since the training labels differ a lot from the original target values, making the learning process more difficult.

As an example, if we modify sigma to 15 (a lot of distortion), the predicted values now do not overlap the original ones, as it happened when sigma is 8. For this case, both the training and test error increase significantly.

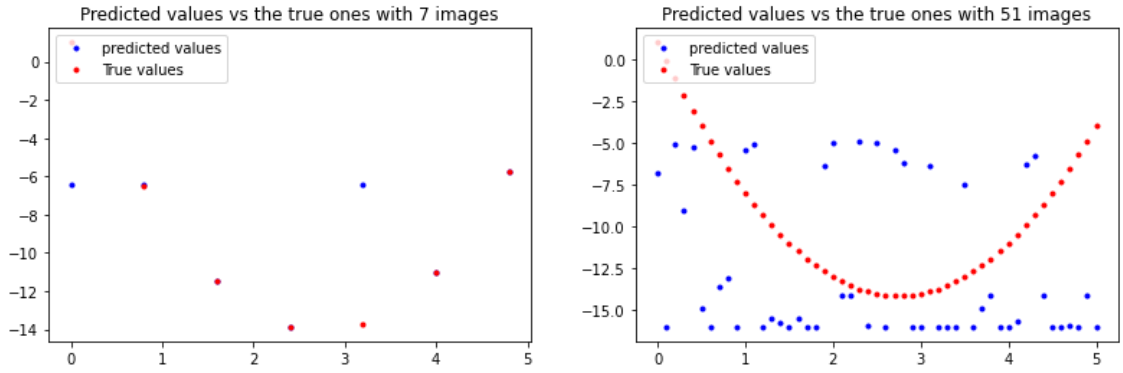

Figure 2.5. After increasing the noise variance from 2 to 15. The left side predicted values vs true values just for 7 training pictures. Right side predicted values vs true values for whole data set,51 pictures respectively.

As a matter of fact, we presented the pictures of the circles for seven training values. As we can see, by increasing the noise sigma, the centre of the circle is sometimes going out of 29*29 pixels picture. It means that the values of training data are not as before. Therefore, the model will not predict correctly as it was showed before.

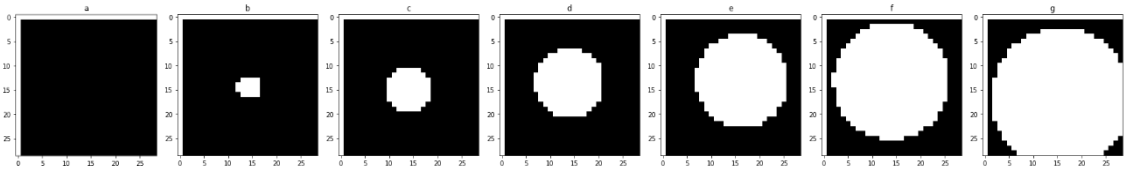

Figure 2.6. Representation of values 0, 0.8, 1.6, . . ., 4.8 by binary images with circular regions with corresponding radius and random centre. When Sigma of the noise is equal to 2. (Above)

When Sigma of the noise is equal to 15. (Below)