Chapter - 3
Linear Models

Aviral Janveja

1 Linear Classification

Let us consider a real world data-set of hand-written digits collected from postal-stamps. It is always better to test your models on real data, to get a better understanding of how your system would actually perform in the real world. Here is a sample from the data-set :

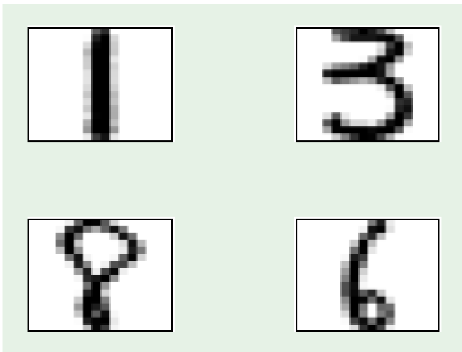

Figure 1: Real Data Example

As shown above, we have a bunch of hand-written digits, collected from postal stamps. We would like to design a model, that can learn to decipher these digits from the given images. People can sometimes write digits in weird ways, making it difficult to understand even for a human operator. Indeed, the error rate for human operators is found to be around 2.5% and we would like to see if our machine learning model can at least equal that or maybe do better.

1.1 Input Representation

Let us begin by looking at the given input data more closely. We are given a set of grayscale images containing hand-written digits, as shown below :

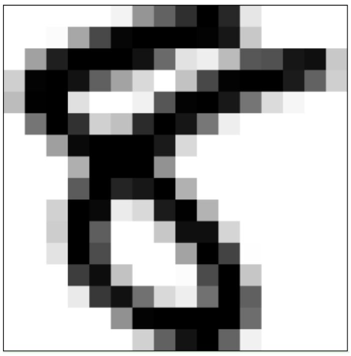

Figure 2: Hand-Written Digit Example

Now, each of these images is 16×16 pixels. Meaning, each digit is represented by 256 real-number attributes. The raw-input $\mathbf{x}$, would therefore look like :

$$\mathbf{x} = (x_1, x_2, x_3 ... x_{256})$$

That is a very long input to represent such a simple object. If we take this raw-input and try the perceptron directly on it, we get too many weights :

$$\mathbf{w} = (w_1, w_2, w_3 ... w_{256})$$

The idea of input representation is to simplify the algorithm's life. We know that it is not about the individual pixel values, when trying to recognize a digit. We can instead extract some relevant features from the raw-input and then give those to the learning model and let it figure out the pattern.

1.2 Feature Engineering

Features are basically useful information that can be extracted from the given raw-input. For example, average pixel intensity, symmetry-score and curve-score. The digit 1 for instance, will score higher on the symmetry measure as compared to a 5, whereas 5 will score higher on the intensity-score. Using such features instead of the raw-input, significantly simplifies our input representation :

$$\mathbf{x} = (x_1, x_2, x_3)$$

Admittedly, we are losing some information in this process of converting the raw-input to features. But chances are, most of it is irrelevant information anyway. From a generalization point of view as well, going from 256 to 3 parameters is a pretty good situation. Plotting a scatter diagram for digits 1 and 5 alongside just two features - symmetry and intensity. We get the following graph :

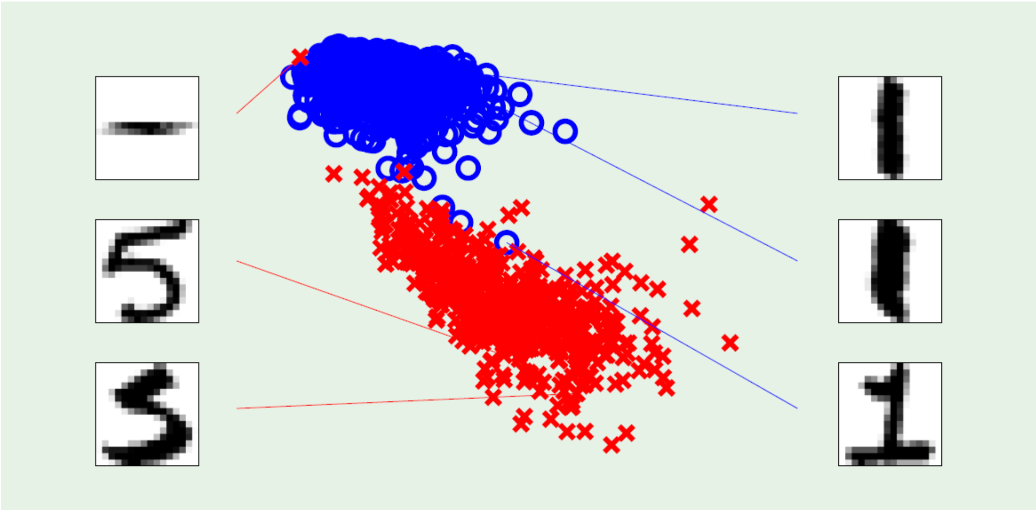

Figure 3: Illustration of Features

The blue points represent 1 and the red points represent 5. The horizontal-axis represents intensity and the vertical-axis represents symmetry. Indeed the red-fives are tilted more towards the right on the horizontal-axis, corresponding to their higher intensity-score. Meanwhile the blue-ones are higher on the vertical-axis owing to their higher symmetry-score. Just by using these two features, we see that the above data is already classified correctly, for the most part.

1.3 Pocket Perceptron

As seen in chapter 1, the perceptron model implements the following formula :

$$\text{sign}(\mathbf{w}^{\mathbf{T}}\mathbf{x})$$

We iterate over the misclassified points, trying to nudge the weight vector $\mathbf{w} = (w_0, w_1, w_2)$ such that all points are eventually classified correctly. However, the above digits data-set is not linearly separable, as visible in the graph above. There is a red point, deep in the blue region and other similar outliers that cannot be classified using a straight line.

This means that the perceptron learning algorithm will never stop and keep looping from one misclassified point to another. So, how do we solve this?

Well, we present a pretty simple solution here. We can set a limit on the number of iterations, let us say one thousand. During those iterations, we keep track of which hypothesis reports the lowest in-sample error E_{in} and report it as our final hypothesis g at the end. This is why, it is called the pocket perceptron. We pick the best candidate so far and put it in our "pocket".

2 Linear Regression

3 References

1. CalTech Machine Learning Course - CS156, Lecture 2.