Detection of Pedestrians in Images using SVM Project Report

Gopal Menon
Machine Learning
Fall 2016, University of Utah

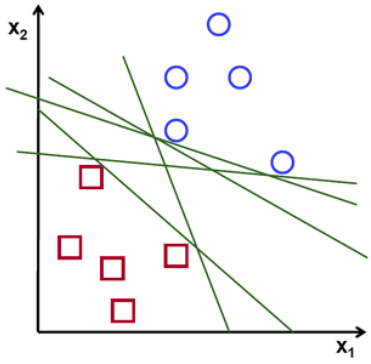

Fig. 1. Many possible separating boundaries[1]

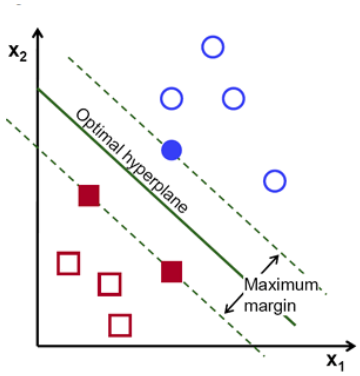

Fig. 2. Separation with maximum margin[1]

I. Project Scope

The scope was to construct a classifier using a Support Vector Machine (SVM) that can classify a street image as one containing or not containing a pedestrian. The main reason why it is interesting to me is that it is one of those problems that can be easily explained to a human, while being extremely difficult to program using traditional methods.

An SVM is a linear classifier that finds the separating boundary between instances of two classes or labels. Although there may be many separating boundaries as shown in figure 1, the linear separation is achieved by selecting a boundary that maximizes the separation between the classes. The vectors that lie on the separating band are called support vectors. These are shown as the solid squares and circle in figure 2. By finding a such a band that maximizes the separation between classes, the classifier has better generalization for examples that have not been seen yet.

A. Initial Assumptions

When I started with the project, I went forward with the assumption that through use of a suitable kernel function, images can be linearly separated into ones that contain and do not contain a human. My understanding was that even if the images were not linearly separable in the input space, they would be linearly separable in some higher dimension space through the use of Kernels. Another reason for using an SVM with a Kernel was the belief that since the SVM loss function with respect to the weight vector was convex, it would be possible to find the global minima as opposed to the case for a neural network where I would only be able to find a local minima.

Since then, I learnt during office hours after class that my understanding about the perceived superiority of SVMs when compared to a Neural Network was not correct any more. Neural Networks have surpassed SVMs in image recognition.Very Deep Convolutional Networks [5] are currently the state of the art in image recognition. The suggestion I got during office hours was to use the input to the final layer of a Visual Geometry Group (VGG) Very Deep Convolutional Net (VGG NET) classifier as features for an SVM to do linear separation.

II. Ideas Explored

A. SVMs using Non-Linear Kernels

While looking for project ideas, I had gone through the syllabus and saw that SVMs were going to be covered and decided to do a project using an SVM for pedestrian image classification. What interested me most was the idea of using a linear classifier in a higher dimensional space in order to achieve a non-linear separating boundary in the input space. This is illustrated in figure 3 where the curved line separating the classes is the decision boundary and the examples marked with the squares are the support vectors that lie on the margin.

I had read up on the use of Kernels and understood how the Kernel trick could be used to achieve a dot product in a higher dimension space without actually converting the inputs to the function to the higher dimension. I also understood how hard-margin and soft-margin SVMs worked. I understood the

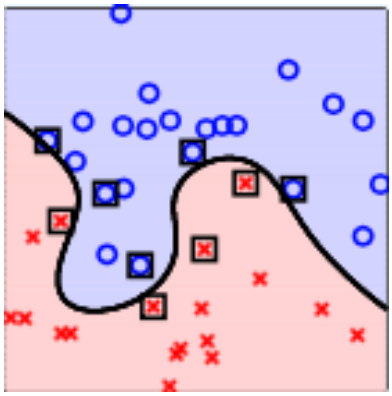

Fig. 3. Non-Linear linear decision boundary in an SVM achieved through use of a Kernel [2]

mathematical part of the dual form of an SVM where the learnt weight is the sum of products of the m training inputs, labels and an α [2].

$$\mathbf{x} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i$$

Since the prediction is done using $sgn(\mathbf{w}^T\mathbf{x})$, it could be represented by

$$\begin{aligned} sgn(\mathbf{w}^T\mathbf{x}) &= \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i^T \mathbf{x} \\ &= \sum_{i=1}^{m} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) \end{aligned}$$

where K is a kernel function. I could not map this theory into an algorithm for using a Kernel with an SVM. Also I got feedback on the interim project report that a non-linear kernel SVM was too complex to be done as a class project. This along with the realization that a Neural Net would outperform an SVM with a non-linear kernel, made me drop the idea of using a Kernel.

B. Feature Extraction and Input size reduction

Since I was going to transform the images to be classified into long pixel intensity vectors, I wanted to do some sort of feature extraction that would make the training task faster and also would allow the feature set to fit into main memory during training. I investigated OpenCV and instead decided to simply scale the images so that the feature vectors would be reduced in size. In addition to scaling the images, I also converted them to grayscale so that I would have one integer per pixel as my features. I used standard Java library functions for the scaling and grayscale conversion.

C. Literature used

1) Since I needed to start work on the project before SVMs were covered in class, I used the class recordings from last year to get familiarized with the subject.

Fig. 4. Sample of image without person

Fig. 5. Sample of image with person

2) In addition to the class recordings, I also used SVM class recordings from a Caltech course on machine learning available at http://amlbook.com and based on the book *Learning from Data: A Short Course* [2].
3) Once I got comfortable with the material, I referred to the SVM book by Christiani et. al. 2000 [3] for an understanding of the dual form of linear learning machines and kernel functions.

D. Resources used

The INRIA labelled dataset [4] for humans present and absent was used for training and testing. All the images were of the same size and due to this, no padding of training data vectors with zeroes needed to be done as all the pixel array vectors had the same length. Figures 4 and 5 are samples of images from the INRIA dataset that were used in the project.

III. Ideas used from class

Since I started the project before SVMs were taught in class, I was still able to use class resources by watching recordings of the class from last year. My SVM was not predicting simple cases correctly and I used feedback from discussions during office hours on how to troubleshoot by using simple inputs with predictable results, setting the seed for the random number generator (used for generating simple test data) so that

inputs could be recreated for easier debugging and the very useful suggestion of monitoring the objective so that it reduces. I was able to find and fix a few errors and my SVM started to work. I had coded some kernels (linear, polynomial and Gaussian) but ended up not using them.

IV. Concepts Learnt

I got a good understanding of SVMs and linear classifiers especially the concept of maximizing the margin for better generalization. I understood the importance of cross validation for selecting hyper-parameters, which was especially useful in building a linear classifier where the inputs were not necessarily full linearly separable.

I downloaded and used a classifier based on a Deep Convolution Network [6] and was able to get it to classify the images that were used by the SVM classifier. The accuracy of this classifier far exceeded that for the SVM linear classifier, but this was expected. I however did not get a complete understanding of these classifiers although I think I do have a good understanding of Neural Net classifiers. I am still not sure how I would go about troubleshooting a Neural Network that does not work (due to a bug in back propagation implementation for example).

V. Design choices

1) I converted the color images to grayscale.
2) Images were scaled to 20%, 25% and 30% of the original size so as to make computational load smaller. Also, because the classifier was failing with memory allocation error when run on the original images.
3) All the images were of the same size and did not need to pad the feature vectors with zeroes in order to make them of the same size.
4) I used 20 epochs, 5-fold cross validation, learning rates from $10^1, 10^0, 10^{-1}, 10^{-2}, \ldots, 10^{-10}$ and tradeoff values of $2^1, 2^0, 2^{-1}, 2^{-2}, \ldots, 2^{-10}$ for cross validation.

VI. Results

TABLE I
Testing Data Metrics for SVM Classifier

Scaling	Accuracy	Precision	Recall	F1 Score
20%	0.6370	0.7143	0.7065	0.7104
25%	0.6027	0.7073	0.6304	0.6667
30%	0.6164	0.6800	0.7391	0.7083

Table I has the metrics for the SVM classifier on the testing data. The accuracy measures are not very high although I was surprised that I was able to get an F1 Score of around 70%. I was expecting numbers that were lower than chance as I did not expect the image data to be linearly separable. I expected the numbers to improve when the scaling was reduced (scaling percentage was increased), however the accuracy got worse when scaling was changed from 20% to 25% and then increased from 25% to 30%. I am not sure how to explain this. With scaling of 20% the total run time on my laptop was slightly more than 1 hour and with 30%, it increased to around 4 hours. Due to the higher run time at 30%, I did not try going beyond 30%. Initially I had tried to run the classifier without any scaling and this resulted in a memory allocation error.

TABLE II
Training Data Metrics for SVM Classifier

Scaling	Accuracy	Precision	Recall	F1 Score
20%	0.7432	0.7489	0.8913	0.8139
25%	0.7568	0.7868	0.8424	0.8136
30%	0.7603	0.7545	0.9185	0.8284

Table II has the metrics for the SVM classifier on the training data and as expected, the accuracy was higher when compared to the test data.

TABLE III
Metrics for Convolution Net Classifier

Dataset	Accuracy	Precision	Recall	F1 Score
Testing	1.0	1.0	1.0	1.0
Training	0.9444	0.95	0.9444	0.9472

Table III has the metrics for the Convolution Net Classifier and the accuracy of this classifier far exceeded the one using an SVM. This classifier was run using source code that was downloaded [6] and modified so as to use the same dataset as the SVM classifier.

Although my plan was to use the inputs to the last layer of the Convolution Net Classifier as the inputs to the SVM classifier, I was not able to do so as internals were not accessible.

VII. If I had more time

If I had more time or if I knew what I know now at the beginning of the project selection, I would have chosen to use a regular Neural Network or one based on a Convolutional Network as these seem to work best with non-linear classification. I would also have liked to use ensemble classifiers as the idea seems very elegant to me.

References

[1] "Introduction to Support Vector Machines." *Introduction to Support Vector Machines* - OpenCV 2.4.13.1 Documentation. N.p., n.d. Web. 27 Sept. 2016.

[2] Abu-Mostafa, Yaser S., Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning from Data: A Short Course*. United States: AMLBook.com, 2012. Print.

[3] Cristianini, Nello, and John Shawe-Taylor. *An Introduction to Support Vector Machines: And Other Kernel-based Learning Methods*. Cambridge: Cambridge UP, 2000. Print.

[4] "INRIA Person Dataset." *INRIA Person Dataset*. N.p., n.d. Web. 08 Nov. 2016.

[5] "Visual Geometry Group Home Page." *Visual Geometry Group Home Page*. N.p., n.d. Web. 08 Nov. 2016.

[6] Gibson, Chris Nicholson Adam. "Deep Learning for Java." *Deeplearning4j: Open-source, Distributed Deep Learning for the JVM*. N.p., n.d. Web. 15 Dec. 2016.