Hyperspectral Image Classification Using SVM

December 12, 2017

Introduction

A hyperspectral image is an image where each pixel contains the light intensity over many wavelengths of the electromagnetic spectrum. Usually, such image contains enough information that many different materials are associated with a distinct spectral. This property allows automatic material identification.

The goal of this project was to implement a hyperspectral image classification system with a Support Vector Machine (SVM). The use of SVM is popular for this task as the pixels tend to be densely populated in a high-dimensional space, thus maximizing the margin tends to reduce the generalizing error. In this report, we discuss different parameters of SVM and its effect on the system. We also compare the prediction by SVM with that by LS. Finally, we discuss some noise-reducing techniques to improve the system.

Theoretical Background

Binary Support Vector Machine (SVM)

Suppose there are N data points $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^D$ with labels $t_1, \ldots, t_N = \pm 1$. The goal of SVM is to separate the points into two classes in accordance with the labels using a generalized linear model $\mathbf{w}^{\mathrm{T}}\mathbf{z} + b = 0$ such that the resulting hyperplane has the largest margin (distance to the closest data point). Suppose further that the mapping into high-dimensional space is fixed as ϕ and that the data is linearly separable by the model, then the maximum margin problem is formulated as $\max_{\mathbf{w},b} \left\{ \frac{1}{\|\mathbf{w}\|} \min_n \left[\mathbf{w}^{\mathrm{T}} \phi(\mathbf{x}_n) + b) \right] \right\}$.

Taking advantage of the fact that scaling $\mathbf{w}, b$ by the same factor does not change the solution, we may assume that $\mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n) + b \geq 1$ and that the bound is tight for at least one n. Thus our problem is equivalent to maximize $\frac{1}{2}\|w\|^2$ with the constraint $\mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n) + b \geq 1$ for all n.

Since the problem is convex, the above problem is equivalent to its dual problem given by

$$\begin{aligned} \underset{a}{\text{maximize}} \quad & -\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m) + \sum_{n=1}^{N} a_n \\ \text{subject to} \quad & a_n \geq 0 \quad \forall n \text{ and } \sum_{n=1}^{N} a_n t_n = 0 \end{aligned} \tag{1}$$

which is a convex quadratic program that can be solved efficiently.

With the dual formulation, the discriminant function of SVM can be written in the form $y(x) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}_n, \mathbf{x}_m) + b$. Using the complementary slackness condition $a_n(t_n y(x_n) - 1) = 0$, we note that the discriminant function is equivalent to $y(x) = \sum_{n \in \mathcal{S}} a_n t_n k(\mathbf{x}_n, \mathbf{x}_m) + b$ where $\mathcal{S} = \{n : a_n > 0\}$ corresponds to the set of 11support vectors" that maximizes the margin. Using the fact that $t_n y(\mathbf{x}_n) = 1$, we obtain $b = \frac{1}{N_S} \sum_{n \in S} \left(t_n - \sum_{m \in S} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m) \right)$. Thus the class of a new point $\mathbf{x}$ can be estimated as $\text{sign}(y(\mathbf{x}))$.

However, in general the the training data is not linearly separable. Consequently, we modify the problem formulation by introducing slack variables to account for misclassification in the training phase. Ultimately,

the optimization program in (1) remains almost unchanged except that this introduces an additional constraint $0 \leq a_n \leq C$, where C is a parameter called the box constraint and determines how much violation SVM tolerates in the training phase. Generally greater C corresponds to less margin violation.

Extension to Multi-Class Classification

To extend binary classification to general K-class classification ($K \geq 2$), we use the one-versus-rest technique to train K classifiers, where the k'th classifier is used to separate the k'th class from the rest of the data. This gives us K discriminant functions $y_1, \ldots, y_K$ and the classifier assigns $\mathbf{x}$ to class $\hat{k} = \arg\max_k y_k(x)$.

System Implementation

Overview

In this report, we describe a hyperspectral image classification system implemented in Matlab (the script is in `main.m`). The classification system takes in a hyperspectral image with some training pixels and assigns each pixel in the image to one of the pre-defined classes. The overall flow of our system is outlined in Figure 1.

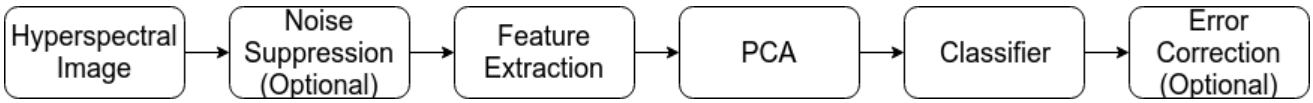


Figure 1: Flow diagram of the classifier system.

Noise Suppression

One way of dealing with noises from the image is with noise-suppressing filters. Because filtering can distort the original image, depending on the Signal Noise Ratio (SNR), noise suppression might not necessarily help with classification. The system leaves the use of noise-reducing filter as an optional parameter to the user. Three filters are available, Wiener, Median, and Box filter, which are available as Matlab's built-in functions.

Feature Extraction

Currently our system simply uses the pixels' spectral values as the features. The system also standardizes each feature to have zero mean and unit variance to avoid numerical overflow/underflow (for example, if $\|x - y\|^2$ is large, then $\exp(\|x - y\|^2)$ would evaluate to zero due to underflow).

PCA

PCA is useful for feature compression and data visualization. Furthermore, if the features are confined to a low-dimensional subspace, projecting the data down can improve the SNR. The number of principal components is left as a parameter for the user input. In the system, PCA is performed by Matlab's built-in `pca` function.

Classifiers

The classifier module has two modes available – Kernel SVM and Kernel LS, both of which are custom-built for this project. The kernel type is available as an input parameter from the user. The classifiers are available as custom-written functions `multi_svm` and `multi_ls` which perform classification using one-versus-rest strategy. In particular, `multi_svm` calls `binarysvm` to create K two-class classifiers. Due to the fact that `binarysvm` trains the SVM by solving a quadratic program with Matlab's built-in function `quadprog`, training can be slow.

Error Correction

Due to physical geometric constraints, a single pixel surrounded by pixels of class 1 is likely to be in class 1. Thus one could impose a continuity constraint on the output labels. This geometric intuition is used correct potential errors made by the classifier. Currently the system does this with mode filtering. That is, the system takes a 3×3 mask around each pixel and assigns the pixel to the majority class in the mask. A visualization of this process in shown in Figure 2.

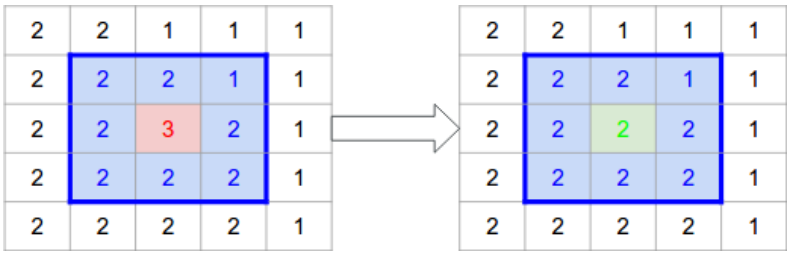
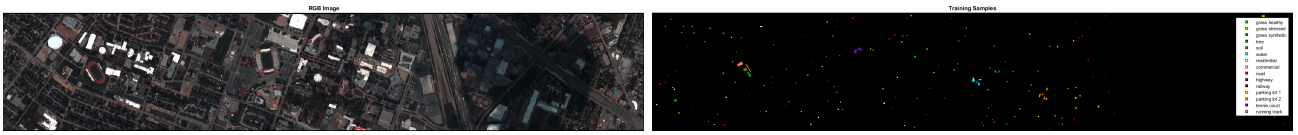


Figure 2: A visualization of mode filter. Suppose a pixel is corrupted by noise (shown in red), the filter takes a 3×3 mask around it (shown in blue) and reassign it to class 2 (shown in green).

Experimental Results

Dataset

The experimental data is a 144-channel hyperspectral image with 349×1905 pixels (Figure 3a). We assume each pixel is associated with one of the 15 material classes: 'grass healthy', 'grass stressed', 'grass synthetic', 'tree','soil','water', 'residential', 'commercial', 'road', 'highway', 'railway', 'parking lot 1', 'parking lot 2', 'tennis court', and 'running track'. The data comes with 2832 pre-labeled pixels (about 180 pixels for each class) shown in Figure 3b for training. Note that in Figure 3a, there is a region shaded by a cloud on the right. The lack of training data in that region will lead high error rate in that region (shown in a later section).



(a) The raw hyperspectral image plotted using RGB channels.

(b) Training pixels with labels.

Figure 3: Hyperspectral image used for the experiment.

Linear Kernel

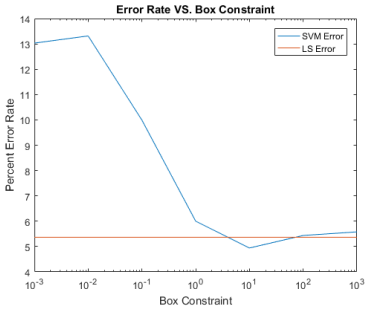
We first test the system on the dataset with linear kernel without the optional modules nor PCA. To determine the optimal value for the parameter box constraint C, we perform 5-fold cross validation. The box constraint is chosen from $\{10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}\}$. Figure 4a plots the error rate with different box constraints, which indicates a value of 10 gives the lowest error rate. While not shown here, we find $C = 10$ to be optimal for our kernel under consideration, so thus we assume a constraint value of 10 for the remainder of this report.

Figure 4a also shows that the error rate of SVM is at 13.4520% whereas LS is at 17.6169%. Thus SVM outperforms LS in this case. From Figure 7, we see that the labels predicted by SVM is significantly less noisy than that by LS. Also note that the classifier's outputs are almost all incorrect in the shaded region on the right.

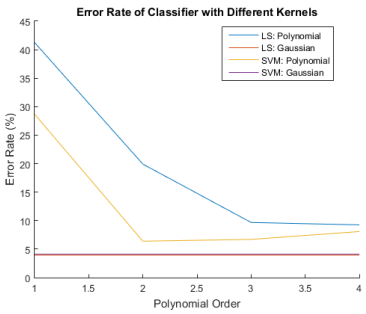
Nonlinear Kernels and PCA

Next, we experiment with Gaussian kernel and polynomial kernel. Figure 4b shows the cross validation error rate of these kernels. Here Gaussian kernel consistently outperforms polynomial kernel and SVM consistently outperforms LS. Error rate of Gaussian SVM sits at 5.0140%, whereas that of LS sits at 5.9674%. We see from Figure 7 the labels predicted with Gaussian kernel appears smoother than that with linear kernel. However, there is no significant performance gain in the shaded region. The rest of the report will assume a Gaussian kernel.

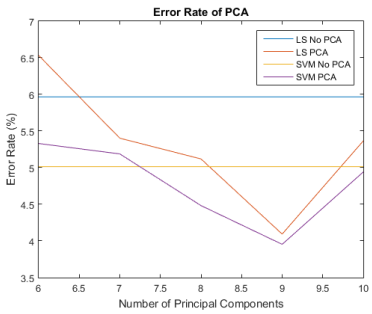
To explore the effect of PCA on the system, we run it in with Gaussian kernel. Figure 5a shows that only 4 components is needed to explain 99% of the data variance. Thus we expect PCA to help with noise reduction. Indeed, as Figure 4c shows, the optimal number of components is 9 components, giving us an SVM error rate of 3.9541% and a LS error rate of about 4.0940%. PCA gives us about 1% error rate reduction.



(a) Error rate with different C (linear kernel, no pca).

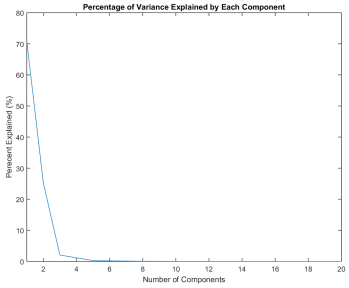


(b) Error rate with different kernels ($C = 10$, no pca).

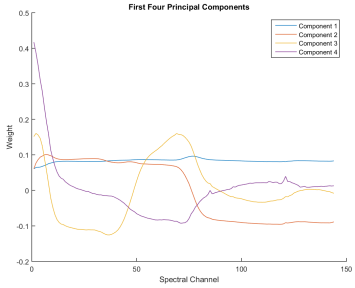


(c) Error rate with different pca ($C = 10$, gaussian kernel).

Figure 4: Classifier's 5-fold cross validation error rate with different parameters.



(a) Variance explained by each pc.



(b) First 4 principal components.

(c) Data in the first 3 pca coordinates.

Figure 5: Visualization of PCA on the data.

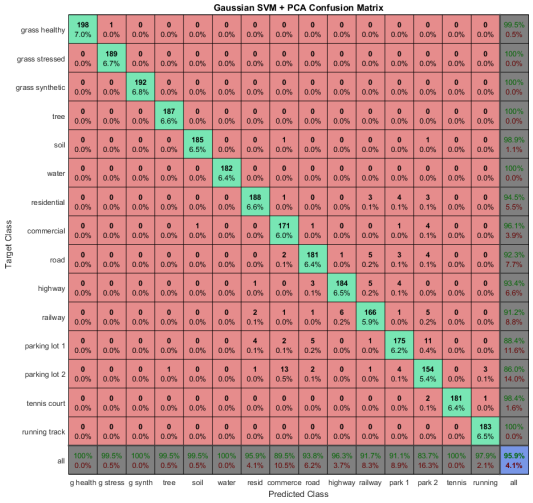


Figure 6: Confusion matrix with box constraint 10, Gaussian kernel, and 9 principal components.

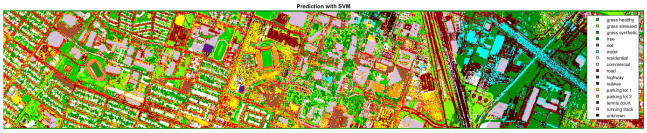
Error Analysis

To understand the classifier's behavior, it is illustrative to examine the confusion matrix in Figure 6. Here we see that most of the errors happen when the classifier cannot distinguish between 'parking lot 1' and 'parking lot 2'. This is reasonable as the parking lots do share similar characteristics. Interestingly, the classifier also tends to misclassify 'residential' as 'parking lot 2'. The reason for this is unclear and requires further study.

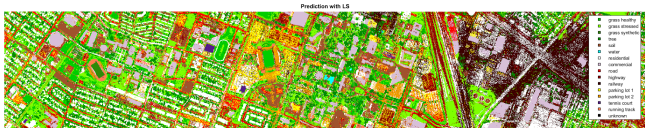
Preprocessing and Postprocesing Filters

This report considers three types of preprocessing filters for noise suppression – Wiener filter, median filter, and box filter. Figure 7e shows the result with median filter. Unfortunately, application of such filter incurs heavy distortion around the edges and do not lead to better performance, thus we do not suggest applying it for classification.

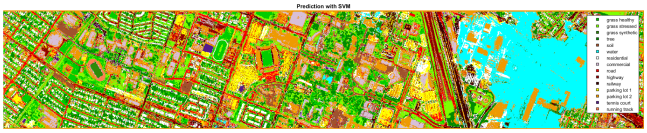
Figure 7e illustrates the effect of error-correcting mode filtering on the final labels. Note that this label significantly reduces the salt-and-pepper noise while incurring only a medium amount of distortion around the edges. Thus we suggest it be used with caution.



(a) Labels predicted by SVM with linear kernel.

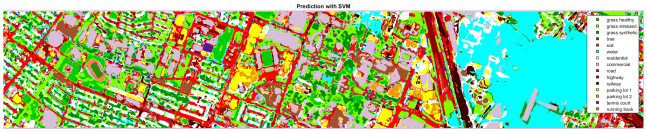


(b) Labels predicted by LS with linear kernel.



(c) Labels predicted by SVM with Gaussian Kernel.

(d) Labels predicted by SVM with Gaussian Kernel and PCA.



(e) Labels predicted by SVM with Gaussian Kernel and PCA and preprocessing median filter.

(f) Labels predicted by SVM with Gaussian Kernel and PCA and postprocessing median filter.

Figure 7: Labels predicted by the classifier system with various parameters.

Conclusion

Summary

In our experiment, we found that the optimal box constraint is $C = 10$ which gives the lowest error rate across different kernels. Furthermore, SVM consistently performs better than LS classifier – the error rate is lower and the label prediction appears cleaner. Gaussian kernel also outperforms polynomial kernel in our all our tests. The use of PCA sped up the training and increased the system performance. With 9 principal components and Gaussian kernel SVM, we obtained our best performance with an cross validation error rate of 3.9541% (Figure 7d). However, the shadow presented in the image leads to the classifier's consistent failure in that region. The confusion matrix indicates that the greatest source of error is from a reasonable confusion between parking lots 1 and 2. Preprocessing filters works well at denoising, but it significantly distorts the label image (Figure 7e), thus its use is not recommended. Postprocessing filter on labels does significantly reduce the salt-and-pepper noise with a modicum amount of distortion.

Future Directions

There are three main directions to improve the system. First, it would be interesting to consider features that take spatial information into account. For example, gradient and local entropy are interesting features to consider. Second, ideally the system should perform outlier detection. More precisely, the current system confidently outputs erroneous labels in the shaded region, but a better system would label them as unknown/uncertain. Finally, it would be good to consider other filters in the preprocessing and postprocessing modules. One possibility would be using a filter based on mean-shift clustering in the postprocessing step.