Classification for Spatial Data

Introduction

In this exercise we will be classifying a pair of satellite images into land cover classes on a per-pixel basis.

Background

We will learn how to use an R implementation of the Random Forest classifier. As the name suggests, a Random Forest is formed by random decision trees. Each tree is random in that the training samples are a random subset of the whole data and the descriptors (also called "features") used in each split of the tree are randomly chosen. This makes each tree a quite weak classifier. However, if you average the results over many weak trees, then it becomes robust and works really well. This is the magic of ensemble learning! Actually, Random Forest is one of the most used and versatile classification algorithms.
However, Random Forests are just one puzzle piece of a larger setup that is required for classification. In detail, a traditional machine learning-pipeline for classification as a couple of pre-defined steps:

1. Feature generation
2. Model (hyper-) parameter fine-tuning
3. Feature selection
4. Predictor selection (if available)
5. Model training
6. Prediction
7. Accuracy assessment

This sequence of steps is valid for virtually all classification tasks, but since we have remote sensing imagery as underlying data, we will alter a few of the steps to better address the spatial data.
In the book (James, G., Witten, D., Hastie, T., Tibshirani, R., 2017: "An Introduction to Statistical Learning, with Applications in R." Vol. 112, Springer, New York) the main chapters of interest are: 4.1 (we are doing classification) and 8.3 (using a tree-based model, a Random Forest). We further talk about general learning concepts like high-dimensional feature space, the "curse of dimensionality", model selection, and more.

Instructions

As stated above, we will be classifying a satellite image into a pre-defined set of land cover classes. "Land cover", as the name suggests, describes

the type of materials that are present at a specific spatial location on the earth. For example, *grass* or *road* are typical land cover classes, so is *water* (because it is visible in remote sensing images). Something like *football field*, however, is not, since it does not describe the type of material used. The latter is called a land *use* class and is much harder to identify from above (but models exist to do so; ask us if you're interested!).
We provide you with an R script file that is halfway filled-in, as well as with some explanations. However, if you are unsure about e.g. a function, we highly recommend you make use of the various helps you can obtain:

- R's built-in Help function. For example, click the "Help" menu in the top right of the screen or type `help(plotRGB)` to get documentation for a specific command (`plotRGB` in this case).
- Online help: StackOverflow is the best-known resource for programmers, also available for R. Search engines are your friend in general.
- Ask a colleague: feel free to discuss with your neighbours!
- Ask us: if everything fails, we may come and help you out.

Finally, a quick explanation for this PDF:

- Whenever there is a grey box, you have to provide something.
- An item in the box starting with a bold **R** requires you to fill in some code in the provided script.
- The letter **Q** marks a question that you have to answer in your report.

Without further ado, let's get started!

Tasks

1 Setup

1.1 Launch your R software environment and copy the data provided to a folder on your local disk.
1.2 Open the provided script. Examine and run the "1. Setup" section (highlight it and hit alt+enter). This loads a list of packages that had been pre-installed for you.

R One command sets the current working directory. Be sure to provide the right path here.

2 Data Preparation

We have two images: *Image 1* will serve as a data input to train a Random Forest; we will then apply it to predict a land cover map for *Image 2*.

2.1 Load the two images and the ground truth for *Image 1* and visualize them (code provided).

Q The image is clearly in false colour. What exactly is shown here, and why does it look like this?

Try to identify the land cover classes from the ground truth in the satellite image. You will see that with a bit of training it will be quite doable for you. At this stage we unfortunately cannot tell the same for the machine. Remember: you have both background knowledge (you know what a water body is supposed to look like), and you have *spatial context*: since neighbouring pixels are likely to contain the same class, you can trace individual regions very quickly. Our machine, however, does not see this right now; it just gets the individual pixels' colour (or spectral) values, which may not be enough (a blue pixel may be a water body if it is surrounded by other blue pixels, but on its own it could be a metal roof tile). To compensate for this, we will be calculating more expressive features than just pixel intensities in the next part.

3 Feature Generation

3.1 For your first coding assignment we will be calculating a new feature, based on the per-pixel spectral values. This is called the "Normalised Difference Vegetation Index" (NDVI) and is the best-known index in remote sensing. It is particularly suited to distinguish between different vegetation covers, and especially between vegetation and the rest. The formula is as follows:

$$NDVI = \frac{NIR - Red}{NIR + Red} \tag{1}$$

Since we have to compute each feature for both images, we will be putting them into re-usable functions.

R Calculate the NDVI by completing the function.
Q Have a look at the visualisation for image 1. Are there some classes you can now separate better by naked eye?

3.2 NDVI is a useful feature, but still acts only on a per-pixel basis. For such high resolutions, we need spatial features as well. Research has proposed hundreds of spatial features over the time, such as "Histogram of Colours", "Bag-of-Visual-Words", "Grey-Level Co-occurrence Matrix", and more. In this exercise we will use some simple local average and local standard deviation values, calculated over the pixel and its eight neighbours. The neighbours included can be defined via the *kernel size* of the moving window. We set it to 5 as a default (i.e., we use a 5×5 moving window), but feel free to experiment with it (adjust parameter `KS`).

R Complete the code inside the functions to do so.

3.3 To improve the model accuracy, we have pre-calculated some additional features for you. In this step we will load and append them to both images (code provided).
You don't need to know this, but if you are curious, the features provided belong to the family of "mathematical morphology" operators and are: opening, closing, opening by reconstruction, closing by reconstruction; each with filter sizes 3×3 and 5×5 on the NIR band (so 8 features in total). Here's some information on the subject: Volpi, M., Tuia, D., Bovolo, F., Kanevski, M. and Bruzzone, L., 2013. Supervised change detection in VHR images using contextual information and support vector machines. International Journal of Applied Earth Observation and Geoinformation, 20, pp.77-85.

3.4 A very important step in feature generation is data *standardisation* (or normalisation). This brings all variables into similar numerical ranges and prevents those with large values to dominate over the others (all variables are treated about equally). There are multiple ways of standardising data; we will be using the so-called "whitening" technique, whose formula goes as follows:

$$\mathbf{X}_{norm} = \frac{\mathbf{X} - \mu(\mathbf{X})}{\sigma(\mathbf{X})} \tag{2}$$

where $\mu(\mathbf{X})$ and $\sigma(\mathbf{X})$ are the column-wise mean and standard deviations of our feature vector X, respectively.
There is one catch to this: we cannot standardise the data for images 1 and 2 separately, because this would shift the two distributions incorrectly. Instead, we have to bring them to a common ground. A typical practice to this end is to normalise the test set data with the same mean and standard deviation values as the training set.

R The normalisation function is given, but the code to call it isn't. Examine the function and complete the code so that both data matrices are standardised the correct way.

4 Training / Validation Set Splits

As explained above we will need to find good parameters for our Random Forest, for which we need to split our data.

4.1 Define the training and validation set splits (code provided). Visualise the split data and answer the following questions:

Q Why do we *spatially* divide our data into training and validation splits and don't just randomly choose points?
Q What kind of problems could we get by doing so?
Q Why don't we include Image 2 or do the same there?

5 Model Creation

After all this preparation it is finally time to meet the Random Forest! We have seen how it works, so it should be clear which kind of (hyper-) parameters it may have. We need to fine-tune a couple of those, so in the following we will use the training data and our freshly created train/val splits to do that.

5.1 Cross-validate the two parameters (minimum leaf size and number of decision trees). Most of the code is provided, except for the Random Forest training call.

R Complete the line to train the Random Forest on the given set of parameter combinations.
Q Visualise the result: what can you see? Did you expect a behaviour in performance as shown? Why?
Q Also plotted is the time required to train the Random Forest. What kind of pattern can you see with respect to the hyperparameters? Does it correspond to what you expected? In which scenarios would you consider the training time?
R Choose your favourite parameter combination; complete the two lines of code to do so.

6 Predictor Importance

In the previous steps we have created a few features (NDVI, local statistics, etc.), all there to facilitate the task of the classifier: to separate groups of pixels according to their high-level semantic land cover classes. However, not every new feature is automatically useful. For example, if the number of data points is limited, one can quickly get the so-called *curse of dimensionality* (page 242 in the book). Even worse, some features may actually be harmful, in the way that including them into the model eventually *degrades* the final model performance! This notion of variable worthiness is called "Predictor Importance" or "Variable Importance" (page 319 in the book).

Luckily, Random Forests have some useful properties that help us evaluating whether a feature (or predictor) is actually of use or not. In this exercise we will be using the Random Forests' ability to assess predictor importance to select which variables to keep. Remember: in this case we treat each variable equally, meaning that it does not matter if we e.g. remove the local average for the red band, but keep the local averages for the others. Our main goal is to maximise the model performance on a given set of pixels.

6.1 Assess the predictor importance for the variables we have, based on the training set we created above.

R Complete the line to train the Random Forest with the correct flag enabled.
Q Plot the result (code provided). What is shown here?
Q Do you see some clear patterns (e.g. a certain type of variable that is particularly suited or not)? If yes: do you have an intuition on why this is the case?
R Choose which variables to keep by completing the line of code.

7 Train Final Model & Predict on Test Set

We now have everything ready to construct, train, and test our model, so let's do it!

7.1 Train your final Random Forest.

R Complete the line of code for that.
Q Which part of the data are you going to use for that? Why?

7.2 Predict the classes of the second image and visualise the result (code provided).

Q Are you satisfied with the prediction? Why resp. why not?
Q Do you see different kind of errors (random blunders, systematic confusions, etc.)? Which ones and where?
Q How do you think these could be avoided?

8 Accuracy Assessment

We have had a visual look at our prediction result, which is always a good idea and very important to do (and to document), regardless of the data at hand. However, a statistical evaluation is just as important, so this is what we are going to do here.

8.1 Run the provided code to calculate a confusion matrix and have a look at the result.

Q Include the confusion matrix and other provided statistics in your report.
Q As you can see the function printed a lot of different statistics. Can you describe in your own words what the following figures mean?

- (Overall) Accuracy
- Kappa coefficient
- Sensitivity and Specificity

Of course feel free to use online help or the literature for assistance.
Q Compare the overall accuracy with the individual, per-class accuracies. Do they correspond to what the overall value tells you, i.e., do all classes have roughly the same accuracy as the total one? If there is a lot of differences, can you explain why this happens?

For your information: there are a lot more ways to assess the accuracy of predictions, depending on the task and field of research. It is very important to know when to use which set of statistics, which you can find out

by checking well-cited publications of your research field. If in doubt it's better provide a more complete set than to omit measures, and to do it correctly. Remember: especially in statistics complete documentation of your work is paramount!
Otherwise, that's all for this exercise. We hope you had fun testing the water of learning semantic classes from geospatial imagery. If you are hungry for more, feel free to talk to us!