CEGEG076: Spatio-temporal Analysis and Data Mining

Tutorial 5: Support Vector Machines for Classification

11am, DMS Watson G15 - Public Cluster, 22nd February 2016

TAs: Dawn Williams and Juntao Lai

1. Introduction

In this tutorial you will learn how to use *Support Vector Machines* for *classification* in the R environment. To clarify the idea of kernel methods, the tutorial begins with the example of kernel ridge regression (KRR), which is a modified version of linear regression that enables kernels to be used. This is followed by an overview of the R packages that can be used to train SVMs. To demonstrate how to train SVMs, two datasets are used: corn yield data from Argentina and landslide susceptibility in Italy. Throughout the tutorial we will use a variety of visualisation methods from both *ggplot2* and the base *graphics* package.

2. A simple kernel method: Kernel Ridge Regression

To motivate the idea of kernel methods, we will briefly digress from classification and use the example of kernel ridge regression (KRR). KRR is the nonlinear extension of a linear algorithm, ridge regression (RR). RR is a penalised/regularised version of ordinary least squares (OLS). The basic algorithm is:

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}_p)^{-1}\mathbf{X}^T\mathbf{y} \quad (1)$$

Where:

$\mathbf{X}$ is an $n * p$ matrix of observations of the independent variables.
$\mathbf{y}$ is an $n * 1$ vector of the dependent variable.
n is the number of observations.
p is the number of variables.
$\mathbf{w}$ is a $p * 1$ parameter vector to be estimated
$\lambda \geq 0$ is the scalar ridge parameter
$\mathbf{I}_p$ is a $p * p$ identity matrix (matrix of zeros with one on the diagonal).

Equation 1 is known as the *primal form*. This algorithm can be expressed in *dual form* as follows:

$$\boldsymbol{\alpha} = \left(\mathbf{X}\mathbf{X}^{\boldsymbol{T}} + \lambda\mathbf{I}_n\right)^{-1}\mathbf{y} \quad (2)$$

Where:

$\boldsymbol{\alpha}$ is an $n * 1$ vector of parameters

In the dual form, $\mathbf{X}^T\mathbf{X}$ has been replaced by $\mathbf{X}\mathbf{X}^T$ (called the Gram matrix and often denoted as **G**) and $\boldsymbol{\alpha}$ is $n * 1$ whereas $\mathbf{w}$ was $p * 1$. This means that rather than having *one parameter per variable*, we now have *one parameter per observation*. Note that in either case, if $\lambda = 0$ the algorithm becomes the OLS estimator. To demonstrate these two algorithms we will use an example nonlinear regression dataset called Gauss1, which is two separated Gaussian functions on an exponentially

decaying baseline with added noise (details here: http://www.itl.nist.gov/div898/strd/nls/nls_main.shtml). Figure 1 is a plot of the data.

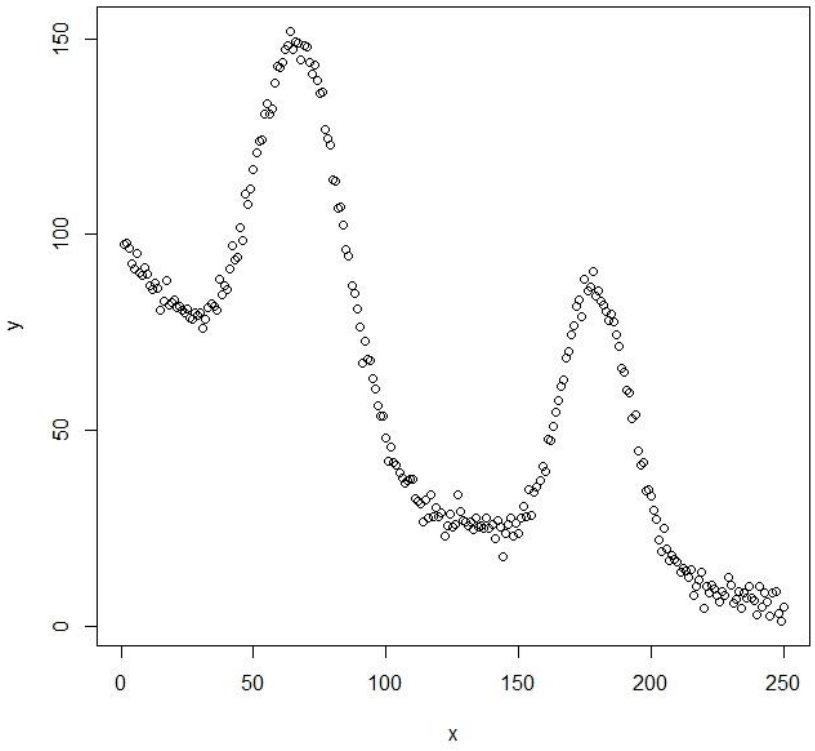

Figure 1 – Gauss1 dataset

First, load the workspace called *classificationWS* (file -> load workspace -> navigate to the file in standard R). This loads the data you will be using in this tutorial. Type `ls()` in the console to see the objects it contains. Type `head(gauss1)` to examine the first few rows of the data. The function is clearly nonlinear but we will first model it using primal and dual linear RR to illustrate the example. First, divide the data into $\mathbf{X}$ and $\mathbf{y}$, and choose a value of lambda:

```
# install.packages("kernlab") if not installed
library(kernlab)          # load the required library

X <- cbind(1,gauss1[,2]) # Create X matrix, including an intercept
y <- gauss1[,1]
n <- nrow(gauss1)         # number of observations
p <- ncol(gauss1)         # number of parameters
lambda <- 0.001
```

Estimate the parameters using standard RR with $\lambda = 0.001$ (equation 1):

```
w <- solve(t(X)%*%X + lambda*diag(p))%*%(t(X)%*%y)
```

Here, `solve` is used to invert a matrix, `t` transposes a matrix and `%*%` does matrix multiplication. `diag(p)` creates an identity matrix with p rows and columns. We can now use $\mathbf{w}$ to predict the values of $\mathbf{y}$ and plot them:

```
plot(gauss1[,c("x","y")], xlab="x", ylab="y")
yHatPr <- X[,2]*w[2,]+w[1,]
lines(yHatPr, col="blue", lwd=6)
```

Now we can fit the same model using the dual RR formulation (equation 2):

```
alpha <- t(y)%*%solve(X%*%t(X) + lambda*diag(n))
```

Next we predict the values of $\mathbf{y}$ and add them to the plot:

```
yHatDu <- alpha%*%(X%*%t(X))
lines(t(yHatDu), col="green", lwd=2)
```

You should see the solutions of the primal and dual RR algorithms are the same. However, the linear fit to the data is not much use because the function is clearly nonlinear. Therefore, we can replace the matrix $\mathbf{X}\mathbf{X}^T$ with a kernel matrix $\mathbf{K}$. We will use a Gaussian RBF kernel from the package *kernlab* using the `rbfdot` function*:*

```
rbfK <- rbfdot(sigma=1/100)
K <- kernelMatrix(kernel=rbfK, x=X)
```

The argument `sigma` specifies the inverse kernel bandwidth (i.e. 100). The function `kernelMatrix` uses the kernel you defined and $\mathbf{X}$ to create a kernel matrix. Now, we can fit the model and predict the $\mathbf{y}$ values (equation 2 with `X%*%t(X)` replaced with `K`):

```
alphaK <- t(y)%*%solve(K + lambda*diag(n))
yHatK <- alphaK%*%(K)
lines(t(yHatK), col="red")
```

You should see that by replacing $\mathbf{X}\mathbf{X}^T$ with $\mathbf{K}$ you get a nonlinear fit to the data. The value of sigma determines the complexity of the model fit. If you increase it (decrease the inverse) then you will get a simpler model, and vice versa. Try fitting the model with `sigma=10` and `sigma=1/100000` to see how it affects the outcome. You should end up with something like figure 2.

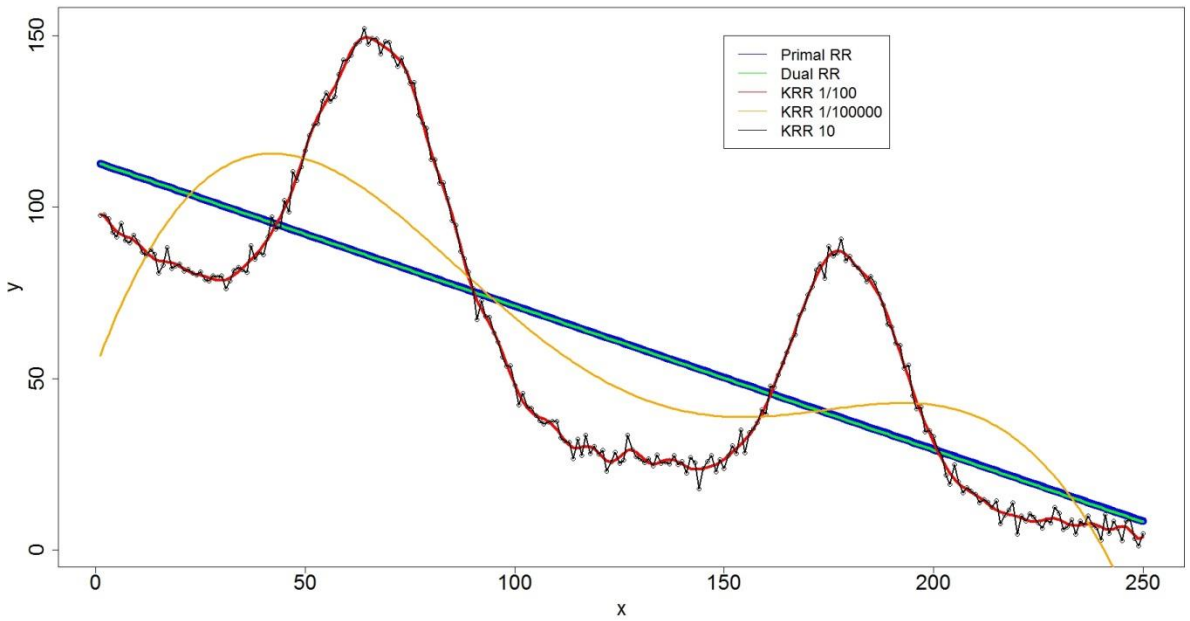

Figure 2 – Linear and nonlinear fits to the Gauss1 dataset

Choosing `sigma=10` (0.1) causes the model to overfit the data, and the regression function passes through each point. Choosing `sigma=1/100000` (100000) causes underfitting, and the regression function is too smooth. This illustrates the importance of selecting an appropriate value of σ. Now try varying the parameter lambda with a value of 1/10. You should see that selecting a larger value of lambda causes a smoother solution in the KRR models (figure 3).

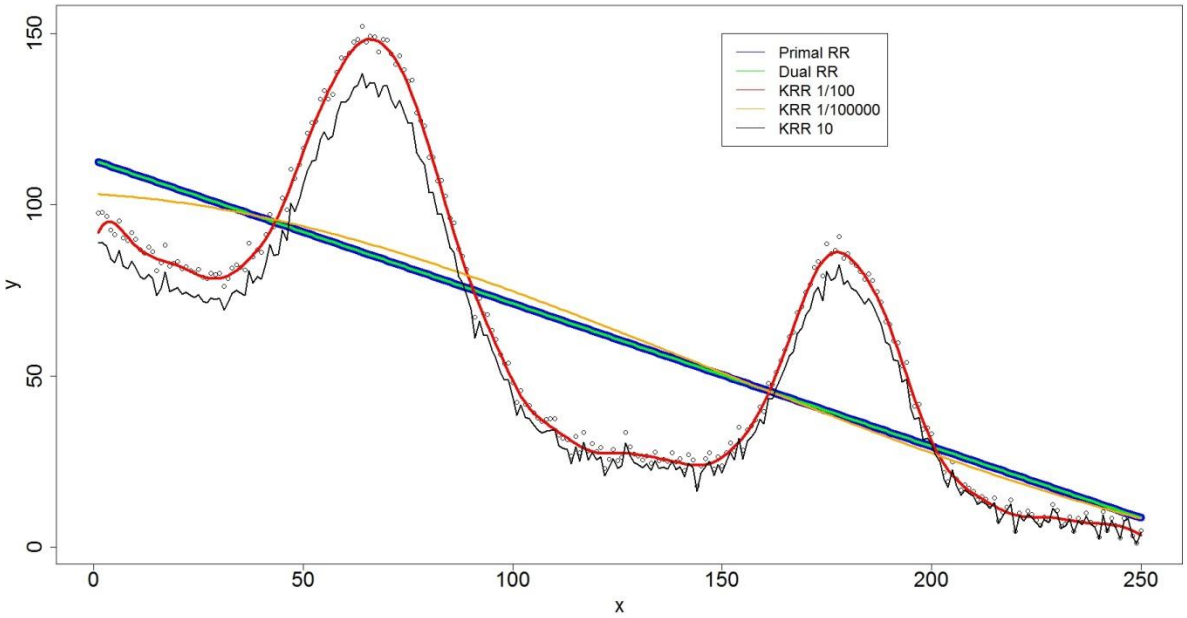

Figure 3 – Effect of increasing lambda on fitted models

Finding appropriate values of the regularisation parameter (lambda here, C in support vector machines) and the kernel parameter(s) (sigma here) is crucial to the success of kernel methods.

3. SVM Classification

The previous section introduced the idea of kernel methods through the example of KRR. Now we will look at SVMs, which use similar ideas to carry out nonlinear classification. There are a number of R packages that can be used to build SVMs, which include:

***e1701*:** A library for various statistical and machine learning algorithms developed at the Technical University of Vienna that includes an interface to *libsvm*, a C++ implementation of SVMs.

kernlab: A comprehensive library of kernel algorithms, including SVMs for classification and regression, kernel principal components analysis and Gaussian processes.

***klaR*:** A library of classification and visualisation tools developed by the Statistics Faculty of the Technical University of Dortmund, which includes an interface to SVMlight (only for two class classification).

***caret*:** An incredibly useful library containing a set of functions for streamlining the process of creating predictive models. *caret* inherits from a number of other R libraries including *kernlab, e1701* and klaR, allowing their functions to be used in a common framework.

In this tutorial we will use *kernlab*, as it has the most flexible and extensive framework for implementing kernel methods. In the second tutorial, we will use *caret* to demonstrate, more generally, the procedure of training machine learning models using R.

This part of the tutorial makes use of two example datasets: corn crop yield from Las Rosas, Argentina, and landslide susceptibility data from Piemonte, Italy. These examples are introduced in turn in the following subsections.

3.1. Example 1: Predicting high/low crop yield

The first dataset is corn yield (quintals per hectare), observed in Las Rosas, Cordoba Province, Argentina in 2001, which are shown in figure 4a. The data are available to download from the GeoDa Center: https://geodacenter.asu.edu/sdata. Table 1 gives a summary of the variables contained in the datastet.

```
# Plot the raw corn yield data

library(maptools)
library(sp)
brks <- quantile(lasRosas$YIELD, probs=seq(0,1,0.1))
lbls <- findInterval(lasRosas$YIELD, brks)
cols <- colorRampPalette(c("red", "green"))(11)
plot(lasRosas, col=cols[lbls])
legend("bottomleft",      legend=leglabs(brks),      fill=cols,      cex=0.5,
title="Yield")
title(main="Corn yield (quintals per hectare)")
```

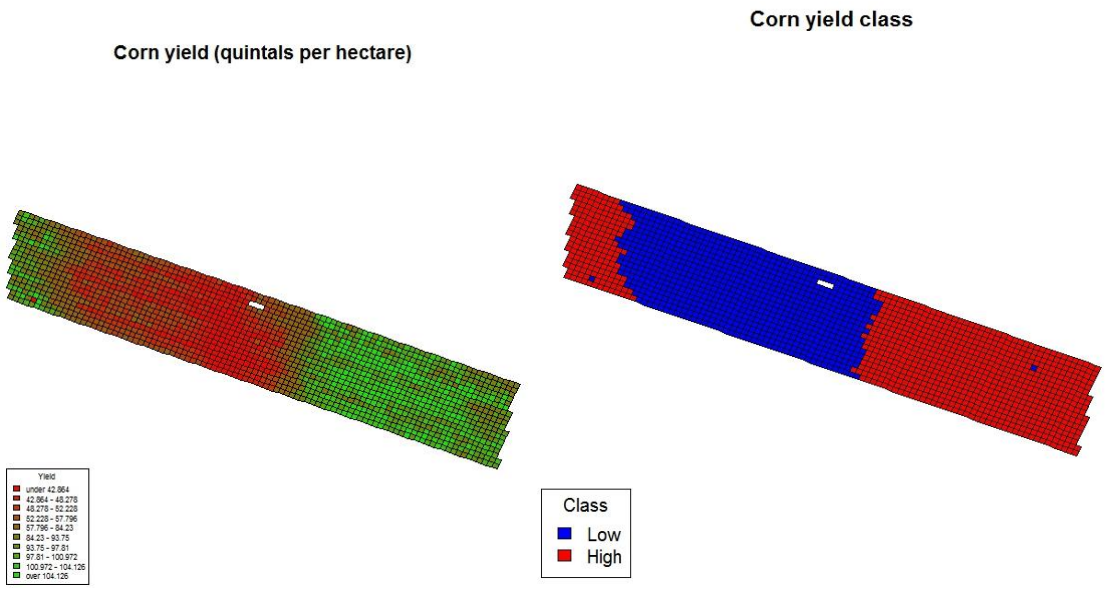

Figure 4 – Las Rosas dataset (a) Raw corn yield (b) corn yield class

Table 1 – Summary of variables in the Las Rosas dataset

Variable	Description
Id (01)	identifier
Top2	topography dummy, Slope E
Top3	topography dummy, Hilltop
Top4	topography dummy, Slope W
Nxtop2 to Nxtop4	interaction Nitrogen - topography zone
Longitude	longitude
Latitude	latitude
Obs	observation number
Yield	corn yield (quintals per hectare)
N	Nitrogen fertilizer application (kg per hectare)
N2	Nitrogen squared
Topo	zone: Low E (1), Slope E (2), Hilltop (3), Slope W (4)
Bv	brightness value (proxy for low organic matter content)
Bv2	brightness squared
Nxbv	interaction Nitrogen - brightness
Bvxt2 to Bvxt4	interaction brightness topographic zone
Bv2xt2 to Bv2xt4	interaction squared brightness topographic zone
Sat (99)	Red to NIR ratio (proxy for low organic matter content)
Sat2 (99)	Sat squared
Nxsat (99)	interaction Nitrogen - Red to NIR ratio
Satxt2 to Satxt4 (99)	interaction Sat - topographic zone
Sat2xt2 to Sat2xt4 (99)	interaction Sat squared - topographic zone

The variable that we will classify is yield, which is continuous. We will first transform it into two classes, high yield and low yield. Figure 5 shows the distribution of the yield values given by the following command:

```
hist(CornYield$YIELD)
```

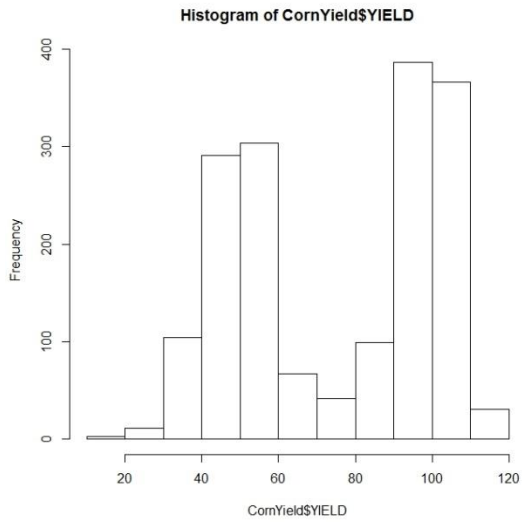

Figure 5 – Histogram of corn yield

It can be seen that the distribution has two distinct peaks. Since the mean corn yield (75.2) lies between these two peaks we will use it as the threshold between low and high yield (shown in figure 4b):

```
y <- CornYield$YIELD
yMean <- mean(y)
yClass <- y
yClass[which(y>=mean(y))] <- 1
yClass[which(y<mean(y))] <- -1
# Plot the corn yield high low classes (figure 4b)
brks <- c(-1,1)
lbls <- findInterval(yClass, brks)
cols <- c("blue", "red")
plot(lasRosas, col=cols[lbls])
legend("bottomleft", legend=c("Low", "High"), fill=cols, title="Class")
title(main="Corn yield class")
```

You can see from figure 4b that the area of low yield is located in the centre of the field between two areas of high yield. As covariates we will use nitrogen fertilizer application, topology zone and latitude and longitude:

```
X <- CornYield[,c(
"N"
,"LONGITUDE"
,"LATITUDE"
,"TOPO"
)]
```

If you have time, try visualising these covariates by modifying the code used to produce Figure 4. To ensure that the model generalises well to unseen data samples, we will divide it into training and testing sets. To start with, we will use 80% of the data to train the model and 20% to test it. The samples are selected at random:

```
n <- nrow(X)
trainInd <- sort(sample(1:nrow(X), n*.8))# Select 80% of samples at random
XTrain <- X[trainInd,]
XTest <- X[-trainInd,]                   # Select the remainder for testing

yClassTrain <- yClass[trainInd]
yClassTest <- yClass[-trainInd]
```

Now we can build a model using the training data. We will start with the default parameters:

```
modelClass <- ksvm(
x = as.matrix(XTrain),
y = as.matrix(yClassTrain),
scaled=TRUE,
type="C-svc",
kernel="rbfdot",
kpar="automatic",
C=1,
cross=0
)
```

The parameters are:

1. **x**: the matrix of independent variables.
2. **y**: the class labels.

3. **scaled**: whether or not to scale the data, set to TRUE by default. x and y are scaled to have zero mean and unit variance.
4. **type:** the SVM method to be used. It is set as C-svc by default, which is the standard version of SVM for classification.
5. **kernel:** the type of kernel we wish to use. "rbfdot" is a radial basis function (Gaussian) kernel. Other types include polynomial ("polydot") and linear ("vanilladot").
6. **kpar:** a list of kernel parameters that is specific to the kernel being used. This is set to "automatic" by default.
7. **C:** this is the parameter that controls the level of allowed error in the classification. It is a trade-off parameter between model complexity and generalisation ability and is set to 1 by default.
8. **cross:** this is an optional parameter, set to zero by default, that specifies whether k-fold cross validation is to be used in the training process. An integer >1 indicates the number of folds to be used.

The object `modelClass` is an S4 object. A list of its attributes can be accessed by `attributes(modelClass)`. Individual attributes can be accessed with the `@` operator. We will now check the number of support vectors required and check the training error:

```
modelClass@nSV
modelClass@error
```

Both values may vary slightly depending on the random selection of the training and testing data. The error should be around 5%. Divide `modelClass@nSV` by the number of training samples to obtain the proportion of the data that are support vectors. We can visualise the fitted model to see which data points are selected as SVs:

```
pltClass <- modelClass@fitted
pltClass[modelClass@SVindex] <- 2
brks <- c(-1,1,2)
lbls <- findInterval(pltClass, brks)
cols <- c("blue", "red", "black")
plot(lasRosas[trainInd,], col=cols[lbls])
legend(x=bbox(lasRosas)[1,1],     y=bbox(lasRosas)[2,1],     legend=c("Low",
"High", "Support Vectors"), fill=cols, title="Class")
title(main="SVM Classification Result")
```

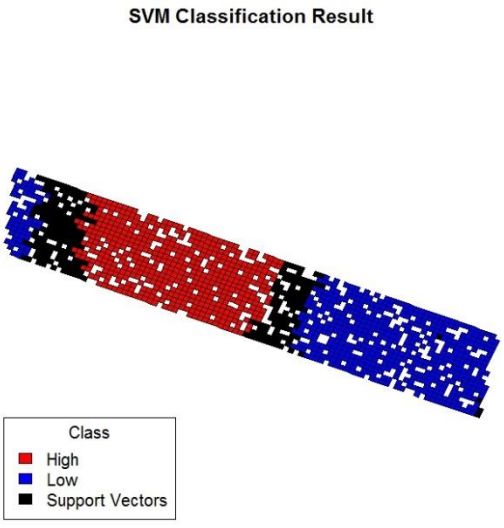

Figure 6 – Fitted values of classification model and position of SVs (white points are testing data).

As you can see from figure 6, the SVs lie on the margin between the two classes. We'll now use the model to predict the testing set that we left out and calculate the error:

```
predClass <- predict(modelClass, XTest)
predErr <- length(which((predClass-yClassTest)>0))/length(predClass)
```

How does the prediction error compare with the training error? We will now visualise the errors:

```
predRes <- yClassTest-predClass
brks <- c(-2,0,2)
lbls <- findInterval(predRes, brks)
cols <- c("blue", "grey", "red")
plot(lasRosas[-trainInd,], col=cols[lbls])
legend(x=bbox(lasRosas)[1,1], y=bbox(lasRosas)[2,1], legend=c("-ve classed
+ve", "Correct", "+ve classed -ve"), fill=cols, title="Class")
title(main="SVM Classification Errors")
```

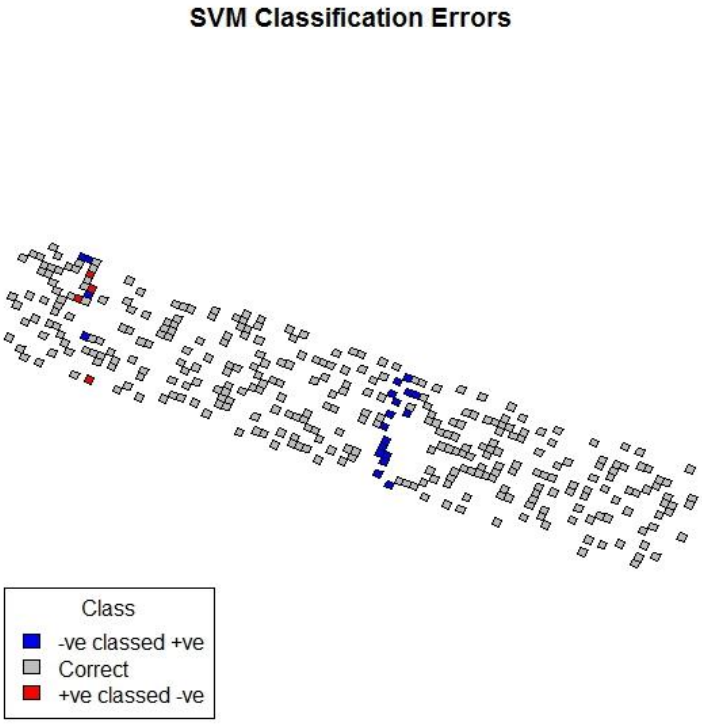

Figure 7 – Predicted classes of testing data

It can be seen from figure 7 that all of the misclassified points lie near to the decision boundary. This is where there is most uncertainty in the model. However, overall the model classifies very well.

3.2. A note on parameter tuning

Up to this point we have used the default values of the parameters to train the SVM model. We could almost certainly get a better classification if we try to optimise the parameters in some way. There are two parameters to train: the kernel parameter `sigma` and the constant `C`. `sigma` is set to `"automatic"` by default. This uses a function called `sigest` to estimate an appropriate value of `sigma` for a Gaussian kernel (see `?sigest` for details). Selecting `sigma` using `sigest` will usually produce good results. The value of `C` is not estimated and is arbitrarily set to 1. Therefore, it is worthwhile testing some more values of `C` to see how they affect the results. In tutorial 2, you will see how to use the *caret* package to train a model over a grid of parameter values.

3.3. Example 2: Landslide susceptibility classification

The second example we will look at is landslide susceptibility classification in Italy. The data, called `landslides`, have been provided by the EU FP7 Project INFRARISK (http://www.infrarisk-fp7.eu/). It is a 100*100 raster, with each cell being 100m*100m. Each cell has a number of attributes associated with it, which are described in table 2. The variable we are trying to predict is a binary indicator of landslide susceptibility. Susceptible cells are those in which a landslide has occurred in the past based on SiFRAP (Sistema Informativo Frane in Piemonte- Landslide information system in Piedmont), a dataset containing 30439 landslides dating from the early 20th century to 2006. Visualise the data with *ggplot2* using the following code (figure 8):

```
library(ggplot2)
X <- as.matrix(landslides[,-c(1,16)])
y <- as.matrix(landslides[,"Landslide_susceptibility"])
y[which(y==0)] <- -1

options(stringsAsFactors = FALSE)
p <- ggplot(as.data.frame(X), aes(x=POINT_X,y=POINT_Y))
+geom_tile(aes(fill=as.factor(y)))
p
```

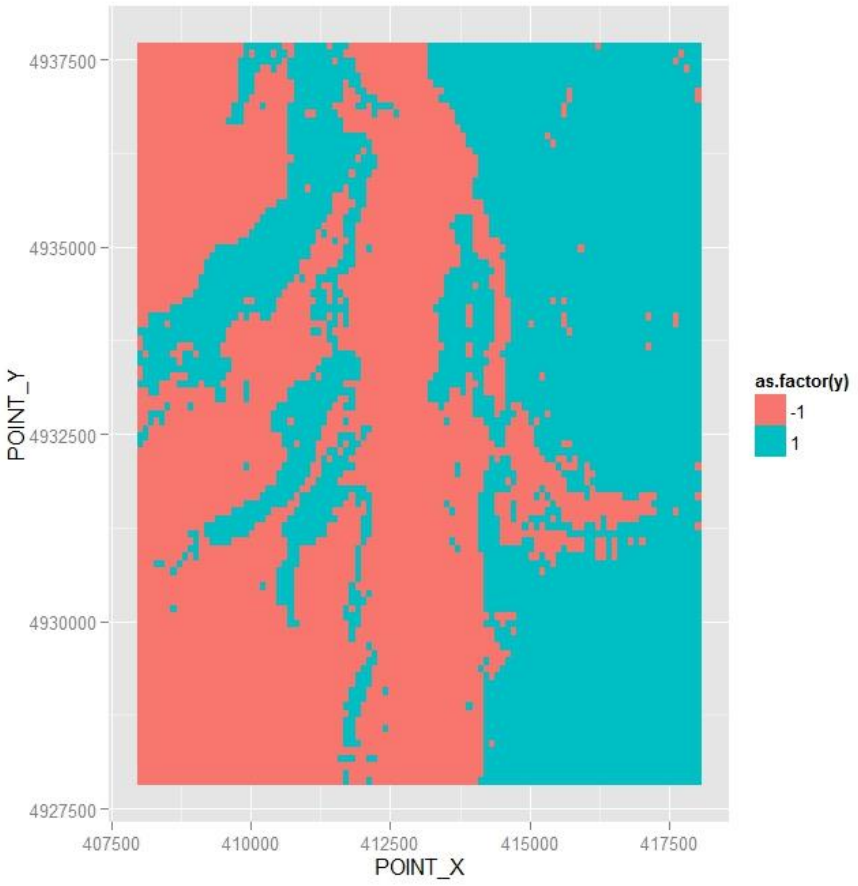

Figure 8 – Landslide susceptibility classes: -1 is unsusceptible, 1 is susceptible

To predict susceptibility, we will use a number of covariates, details of which are given in table 2. The following code will produce a multiplot (a function defined in the workspace and sourced from

here: http://www.cookbook-r.com/Graphs/Multiple_graphs_on_one_page_%28ggplot2%29/) of four of the covariates: `elevation`, `slope`, `parent_material` and `aspect` (shown in figure 9):

```
p1 <- ggplot(as.data.frame(X),
aes(x=POINT_X,y=POINT_Y))+geom_tile(aes(fill=elevation))+
theme(axis.text.x=element_text(angle=90, hjust=1))+
scale_fill_gradient(low="black", high="white")

p2 <- ggplot(as.data.frame(X),
aes(x=POINT_X,y=POINT_Y))+geom_tile(aes(fill=slope))+
theme(axis.text.x=element_text(angle=90, hjust=1))+
scale_fill_gradient(low="black", high="white")

p3 <- ggplot(as.data.frame(X),
aes(x=POINT_X,y=POINT_Y))+geom_tile(aes(fill=parent_material))+
theme(axis.text.x=element_text(angle=90, hjust=1))+
scale_fill_gradient(low="black", high="white")

p4 <- ggplot(as.data.frame(X),
aes(x=POINT_X,y=POINT_Y))+geom_tile(aes(fill=aspect))+
theme(axis.text.x=element_text(angle=90, hjust=1))+
scale_fill_gradient(low="black", high="white")

multiplot(p1,p2,p3,p4, cols=2)
```

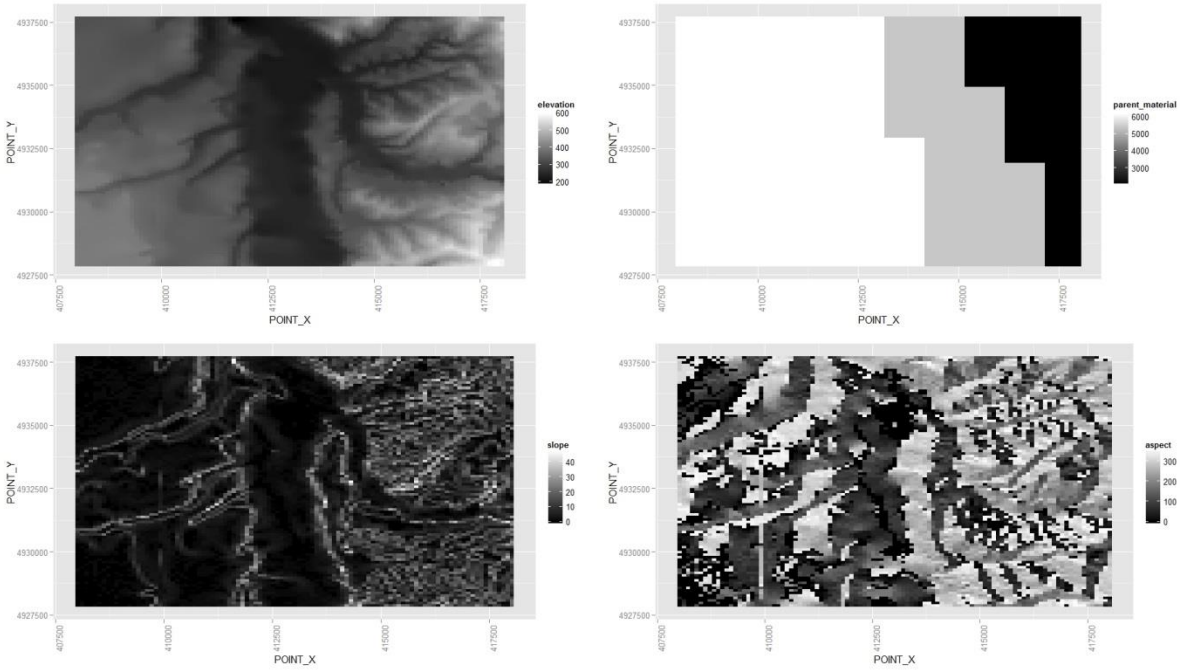

Figure 9 – A selection of covariates of landslide susceptibility: (a) elevation (b) parent material (c) slope (d) aspect

If you have time, try visualising some of the other covariates to assess their spatial distribution. As before we will set up training and testing datasets and build a classification model using the default parameter values:

```
n <- nrow(X)
trainInd <- sort(sample(1:n, n*.8))
XTrain <- X[trainInd,]
```

```
XTest <- X[-trainInd,]

yTrain <- y[trainInd]
yTest <- y[-trainInd]

lsModel <- ksvm(x=XTrain, y=yTrain, type="C-svc")
lsPred <- predict(lsModel, XTest)
```

Table 2 – Landslide covariates description

Variable	Description
FID	Unique identifier
POINT_X	X coordinate
POINT_Y	Y coordinate
parent_material	The dominant parent material on 1 km raster grid divided in 12 classes in the region
Soil_class	World Reference Base (WRB) soil classification on a 1 km grid divided in 15 classes in the region
landfor_classification	Pennock land form classification (Pennock et al., 1987)divided into 7 classes in the region
aspect	The compass direction of a slope. From -1 (flat) to 360°
curvature	Curvature can be thought of as the slope of a slope. Derived from the DEM on a 20m grid. The values range from -510 to 192
slope	Slope is the angle formed between any part of the surface of the earth and the horizontal. This is on a 20 m grid derived from the DEM ranging between 0°-87°
elevation	A 20m resolution raster showing elevation above sea level. Ranging from 61-4615 meters
average_annual_rainfall	Average annual rainfall on a 20 km grid ranging from 684-2640 mm y^{-1}
Hydro_geological_complex	
land_use_code	The CORINE land cover map 2006. A 1:100,000 scale land cover map divided into 16 land cover classes in the region, produced by interpretation of Landsat TM and SPOT HRV satellite imagery
topograpic_wetness_index	Topographic wetness index represents a hypothetical measure of the accumulation of water flow at any point within a river basin. This was derived from the DEM on a 100m grid. Values range from 3.9-30.4.
distance_from_roads	This is derived from the OpenStreetMap road network map of Italy using a GIS operation to calculate distance from a line. This produced a raster grid of 100 resolution
Landslide_susceptibility	Areas where landslides have occurred before are deemed to be susceptible, areas where landslide have not occurred (beyond a 200m buffer for the previous landslides) are deemed to be stable, hence non-suseptible.

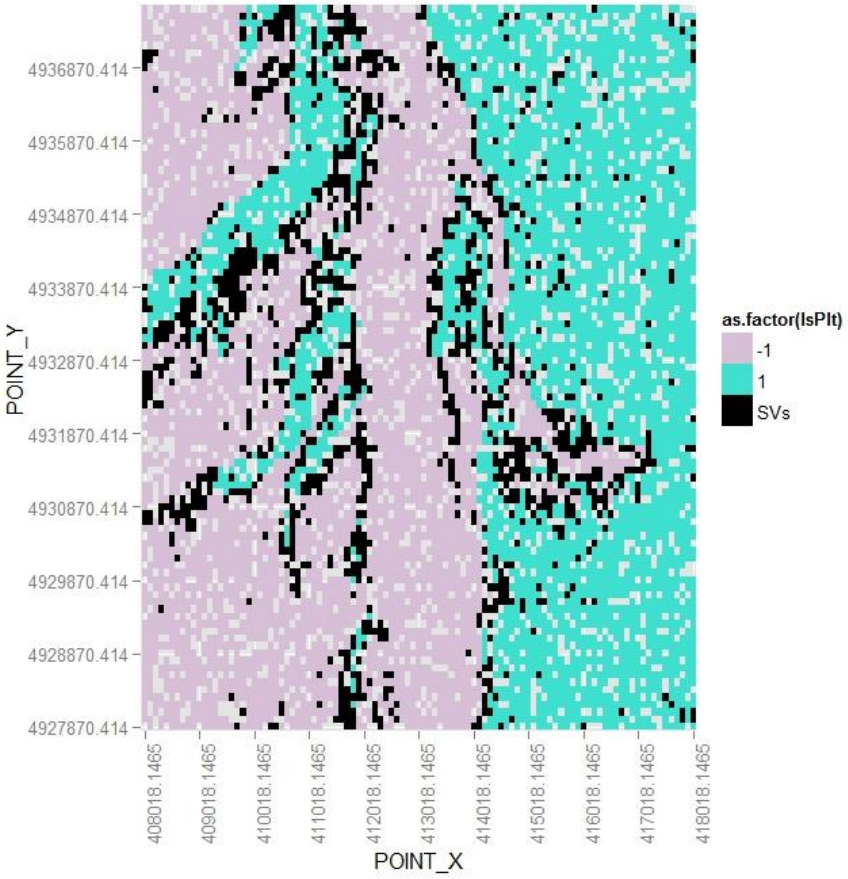

Figure 10 – Position of support vectors in landslide classification with SVM (grey cells are data in the testing set).

This time, we will plot the fitted model and support vectors using *ggplot2* rather than base *graphics* (figure 10):

```
lsPlt <- as.character(yTrain)
lsPlt[lsModel@SVindex] <- "SVs"

brksX <- seq(min(XTrain[,"POINT_X"]), max(XTrain[,"POINT_X"]), 1000)
brksY <- seq(min(XTrain[,"POINT_Y"]), max(XTrain[,"POINT_Y"]), 1000)
svPlt <- ggplot(as.data.frame(cbind(XTrain, lsPlt)),
aes(x=POINT_X,y=POINT_Y))+
geom_tile(aes(fill=as.factor(lsPlt)))+
scale_x_discrete(breaks=brksX)+
scale_y_discrete(breaks=brksY)+
theme(axis.text.x=element_text(angle=90, hjust=1))+
#scale_colour_brewer(palette="Greys")
scale_fill_manual(values=c("thistle", "turquoise", "black"))
svPlt
```

To build the susceptibility classification model we used 80% of the available data, which covers the whole region of interest. This means that the SVM algorithm has a lot of information to work with. Often when sampling a spatial process, our coverage of the region is much sparser and we want to interpolate points where we do not have data. To simulate this scenario we will attempt to train a classification model using a smaller subset of the data, just 20%:

```
trainInd20 <- sort(sample(1:n, n*.2))
XTrain20 <- X[trainInd20,]
```

```
XTest20 <- X[-trainInd20,]

yTrain20 <- y[trainInd20]
yTest20 <- y[-trainInd20]

lsModel20 <- ksvm(x=XTrain20, y=yTrain20, type="C-svc")
lsPred20 <- predict(lsModel20, XTest20)
lsErr20 <- yTest20-lsPred20
length(which((lsErr20)>0))/length(lsErr20)
```

How does the error from the model using 20% of the data compare with the one that uses 80%? Does using fewer data points affect the performance significantly? If you have time, try training one or more of the following:

- Fit a model with different sizes of training data. How low can the percentage be before the performance suffers significantly?
- Try using some different values of `C` and `sigma` to see how they affect the performance of the model and the number of support vectors.
- Try training the model with a polynomial kernel instead.

4. Next tutorial

In the next tutorial, you will learn how to use SVMs for regression, so called support vector regression (SVR). Using the same ideas as SVM, SVR enables a form of linear regression to be carried out in feature space, resulting in a nonlinear solution in the input space. The solution is sparse, depending only on the points that lie outside a tube defined by a parameter ε, which are the support vectors. The tutorial will start by revisiting the Gauss1 data from this tutorial in order to illustrate the way in which the method works. Then, you will use another dataset to build a model for time series forecasting. Instead of using *kernlab* directly, you will use the *caret* package to streamline the process of training your model over a grid of parameters and comparing it with a benchmark model.