Predictive Analytics

L. Torgo

ltorgo@dal.ca
Faculty of Computer Science / Institute for Big Data Analytics
Dalhousie University

May, 2021

ACADIA
UNIVERSITY

What is Prediction?

Definition

- Prediction (forecasting) is the ability to anticipate the future.
- Prediction is possible if we assume that there is some regularity in what we observe, i.e. if the observed events are not random.

Example

Medical Diagnosis: given an historical record containing the symptoms observed in several patients and the respective diagnosis, try to forecast the correct diagnosis for a new patient for which we know the symptoms.

Prediction Models

- Are obtained on the basis of the assumption that there is an unknown mechanism that maps the characteristics of the observations into conclusions/diagnoses. The goal of prediction models is to discover this mechanism.
  - Going back to the medical diagnosis what we want is to know how symptoms influence the diagnosis.
- Have access to a data set with "examples" of this mapping, e.g. this patient had symptoms x, y, z and the conclusion was that he had disease p
- Try to obtain, using the available data, an approximation of the unknown function that maps the observation descriptors into the conclusions, i.e. $Prediction = f(Descriptors)$

"Entities" involved in Predictive Modelling

- Descriptors of the observation:
  set of variables that describe the properties (features, attributes) of the cases in the data set
- Target variable:
  what we want to predict/conclude regards the observations
- The goal is to obtain an approximation of the function $Y = f(X_1, X_{,2}, \cdots, X_p)$, where Y is the target variable and $X_1, X_{,2}, \cdots, X_p$ the variables describing the characteristics of the cases.
- It is assumed that Y is a variable whose values depend on the values of the variables which describe the cases. We just do not know how!
- The goal of the modelling techniques is thus to obtain a good approximation of the unknown function $f()$

How are the Models Used?

Predictive models have two main uses:

1. Prediction
   use the obtained models to make predictions regards the target variable of new cases given their descriptors.
2. Comprehensibility
   use the models to better understand which are the factors that influence the conclusions.

Types of Prediction Problems

- Depending on the type of the target variable (Y) we may be facing two different types of prediction models:
  1. Classification Problems - the target variable Y is nominal
     e.g. medical diagnosis - given the symptoms of a patient try to predict the diagnosis
  2. Regression Problems - the target variable Y is numeric
     e.g. forecast the market value of a certain asset given its characteristics

Evaluation Metrics

Classification Error

Error Rate

- Given a set of test cases N_{test} we can obtain the predictions for these cases using some classification model.
- The *Error Rate* ($L_{0/1}$) measures the proportion of these predictions that are incorrect.
- In order to calculate the error rate we need to obtain the information on the true class values of the N_{test} cases.

Classification Error

Error Rate

- Given a test set for which we know the true class the error rate can be calculated as follows,

$$L_{0/1} = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} I(\hat{h}_\theta(\mathbf{x}_i), y_i)$$

where $I()$ is an indicator function such that $I(x, y) = 0$ if $x = y$ and 1 otherwise; and $\hat{h}_\theta(\mathbf{x}_i)$ is the prediction of the model being evaluated for the test case i that has as true class the value y_i.

Confusion Matrices

- A square $nc \times nc$ matrix, where nc is the number of class values of the problem
- The matrix contains the number of times each pair (ObservedClass,PredictedClass) occurred when testing a classification model on a set of cases

		Pred.		
		c_1	c_2	c_3
Obs.	c_1	n_{c_1,c_1}	n_{c_1,c_2}	n_{c_1,c_3}
	c_2	n_{c_2,c_1}	n_{c_2,c_2}	n_{c_2,c_3}
	c_3	n_{c_3,c_1}	n_{c_3,c_2}	n_{c_3,c_3}

- The error rate can be calculated from the information on this table.

An Example in R

```
trueVals <- c("c1","c1","c2","c1","c3","c1","c2","c3","c2","c3")
preds <- c("c1","c2","c1","c3","c3","c1","c1","c3","c1","c2")
confMatrix <- table(trueVals,preds)
confMatrix

##          preds
## trueVals c1 c2 c3
##       c1  2  1  1
##       c2  3  0  0
##       c3  0  1  2

errorRate <- 1-sum(diag(confMatrix))/sum(confMatrix)
errorRate

## [1] 0.6
```

Measuring Regression Error

Mean Squared Error

- Given a set of test cases N_{test} we can obtain the predictions for these cases using some regression model.
- The *Mean Squared Error* (*MSE*) measures the average squared deviation between the predictions and the true values.
- In order to calculate the value of *MSE* we need to have both the predicitons and the true values of the N_{test} cases.

Measuring Regression Error

Mean Squared Error (cont.)

- If we have such information the *MSE* can be calculated as follows,

$$MSE = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} (\hat{y}_i - y_i)^2$$

  where $\hat{y}_i$ is the prediction of the model under evaluation for the case i and y_i the respective true target variable value.
- Note that the *MSE* is measured in a unit that is squared of the original variable scale. Because of the this is sometimes common to use the *Root Mean Squared Error* (*RMSE*), defined as $RMSE = \sqrt{MSE}$

Measuring Regression Error

Mean Absolute Error

- The *Mean Absolute Error* (*MAE*) measures the average absolute deviation between the predictions and the true values.
- The value of the *MAE* can be calculated as follows,

$$MAE = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} |\hat{y}_i - y_i|$$

  where $\hat{y}_i$ is the prediction of the model under evaluation for the case i and y_i the respective true target variable value.
- Note that the *MAE* is measured in the same unit as the original variable scale.

Relative Error Metrics

- Relative error metrics are unit less which means that their scores can be compared across different domains.
- They are calculated by comparing the scores of the model under evaluation against the scores of some baseline model.
- The relative score is expected to be a value between 0 and 1, with values nearer (or even above) 1 representing performances as bad as the baseline model, which is usually chosen as something too naive.

Relative Error Metrics (cont.)

- The most common baseline model is the constant model consisting of predicting for all test cases the average target variable value calculated in the training data.
- The *Normalized Mean Squared Error* (*NMSE*) is given by,

$$NMSE = \frac{\sum_{i=1}^{N_{test}} (\hat{y}_i - y_i)^2}{\sum_{i=1}^{N_{test}} (\bar{y} - y_i)^2}$$

- The *Normalized Mean Absolute Error* (*NMAE*) is given by,

$$NMAE = \frac{\sum_{i=1}^{N_{test}} |\hat{y}_i - y_i|}{\sum_{i=1}^{N_{test}} |\bar{y} - y_i|}$$

Relative Error Metrics (cont.)

- The *Mean Average Percentage Error* (*MAPE*) is given by,

$$MAPE = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} \frac{|\hat{y}_i - y_i|}{y_i}$$

- The *Symmetric Mean Absolute Percentage Error* (*sMAPE*) is given by,

$$sMAPE = \frac{1}{n} \sum_{i=1}^{N_{test}} \frac{|\hat{y}_i - y_i|}{|\hat{y}_i| + |y_i|}$$

Relative Error Metrics (cont.)

- The *Correlation* between the predictions and the true values ($\rho_{\hat{y},y}$) is given by,

$$\rho_{\hat{y},y} = \frac{\sum_{i=1}^{N_{test}} (\hat{y}_i - \bar{\hat{y}})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{N_{test}} (\hat{y}_i - \bar{\hat{y}})^2 \sum_{i=1}^{N_{test}} (y_i - \bar{y})^2}}$$

An Example in R

```
trueVals <- c(10.2,-3,5.4,3,-43,21,
              32.4,10.4,-65,23)
preds <-  c(13.1,-6,0.4,-1.3,-30,1.6,
            3.9,16.2,-6,20.4)
mse <- mean((trueVals-preds)^2)
mse

## [1] 493.991

rmse <- sqrt(mse)
rmse

## [1] 22.22591

mae <- mean(abs(trueVals-preds))
mae

## [1] 14.35

nmse <- sum((trueVals-preds)^2) /
        sum((trueVals-mean(trueVals))^2)
nmse

## [1] 0.5916071
```

```
nmae <- sum(abs(trueVals-preds)) /
        sum(abs(trueVals-mean(trueVals)))
nmae

## [1] 0.65633

mape <- mean(abs(trueVals-preds)/trueVals)
mape

## [1] 0.290773

smape <- 1/length(preds) * sum(abs(preds - trueVals) /
                        (abs(preds)+abs(trueVals)))
smape

## [1] 0.5250418

corr <- cor(trueVals,preds)
corr

## [1] 0.6745381
```

Multiple Linear Regression

Multiple Linear Regression

- Multiple linear regression is probably the most used statistical method
- It is one of the many possible approaches to the multiple regression problem where given a training data set $\mathbf{D} = \{\langle \mathbf{x}_i, y_i \rangle\}_{i=1}^{n}$ we want to obtain an approximation of the unknown regression function $f()$ that maps the predictors values into a target continuous variable value.
- In matrix notation we have $\mathbf{D} = \mathbf{X}|\mathbf{Y}$, where $\mathbf{X}$ is a matrix $n \times p$, and $\mathbf{Y}$ is a matrix $n \times 1$.

Multiple Linear Regression (cont.)

- In the case of multiple linear regression the functional form that is assumed is the following:

$$Y = \beta_0 + \beta_1 \cdot X_1 + \cdots + \beta_p \cdot X_p$$

- The goal is to find the vector of parameters β that minimizes the sum of the squared errors $\sum_{i=1}^{n}(y_i - (\beta_0 + \beta_1 \cdot X_1 + \cdots + \beta_p \cdot X_p))^2$

Multiple Linear Regression

Pros and Cons

- Well-known and over-studied topic with many variants of this simple methodology (e.g. Drapper and Smith, 1981)
- Simple and effective approach when the “linearity” assumption is adequate to the data.
- Form of the model is intuitive - a set of additive effects of each variable towards the prediction
- Computationally very efficient
- Too strong assumptions on the shape of the unknown regression function

Drapper and Smith (1981): Applied Regression Analysis, 2nd edition. Wiley Series in Probability and Mathematical Statistics.

Obtaining Multiple Linear Regression Models in R

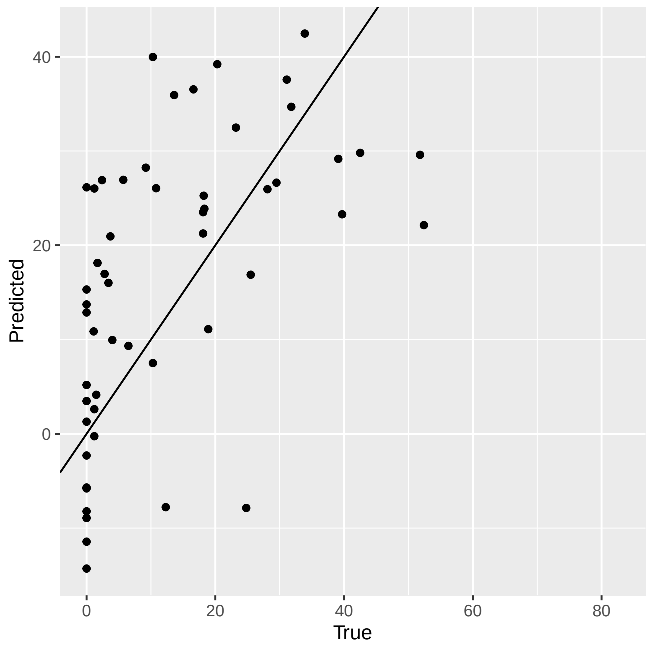
```
library(tidymodels)
data(algae, package="DMwR2")
alg <- as_tibble(algae) %>%     # Preparing the data
  select(1:12) %>% slice(-c(62,199))

lmSpec <-
  linear_reg() %>%     # the type of model
  set_engine("lm")     # the implementation to use

lm <- lmSpec %>% fit(a1 ~ ., data = alg)   # fit the model to the data

tidy(lm)    # showing the model

## # A tibble: 16 x 5
##    term      estimate std.error statistic
##    <chr>        <dbl>     <dbl>     <dbl>
##  1 (Interc~ 28.1      29.7          0.946
##  2 seasons~ -1.09      4.31        -0.252
##  3 seasons~ -0.917     4.08        -0.225
##  4 seasonw~  1.82      3.98         0.457
##  5 sizemed~  2.60      3.83         0.680
##  6 sizesma~  8.96      4.24         2.11
##  7 speedlow  1.70      4.93         0.345
##  8 speedme~ -2.24      3.41        -0.658
##  9 mxPH     -1.06      3.49        -0.305
## 10 mnO2      0.754     0.709        1.06
## 11 Cl       -0.0325    0.0331      -0.982
## 12 NO3      -1.60      0.551       -2.90
## 13 NH4       0.00178   0.000993     1.80
## 14 oPO4     -0.0151    0.0396      -0.380
## 15 PO4       0.0402    0.0308       1.30
```

```
## # ... with 1 more variable:
```

Using the Models for Prediction

```
dataSplit <- initial_split(alg, prop = 0.7)
algTr <- training(dataSplit)  # training set
algTs <- testing(dataSplit)   # test set

lmTr <-
  lmSpec %>% fit(a1 ~ ., data = algTr)

preds <- predict(lmTr, new_data = algTs)
algTs %>% bind_cols(preds) %>% metrics(a1,.pred)

## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard      14.6
## 2 rsq     standard       0.299
## 3 mae     standard      11.9
```

```
library(ggplot2)
ggplot(bind_cols(algTs,preds), aes(x=a1,y=.pred)) + geom_
```

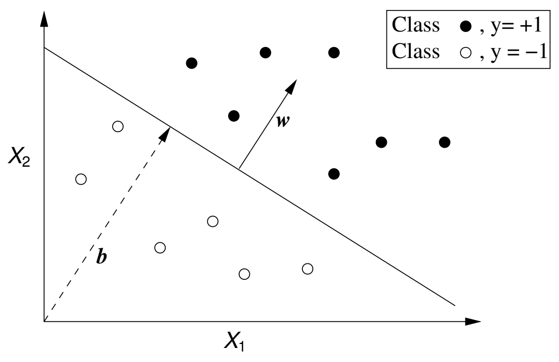


Support Vector Machines

A Bit of History...

- SVM's were introduced in 1992 at the COLT-92 conference
- They gave origin to a new class of algorithms named *kernel machines*
- Since then there has been a growing interest on these methods
- More information may be obtained at `www.kernel-machines.org`
- A good reference on SVMs:
  N. Cristianini and J. Shawe-Taylor: An introduction to Support Vector Machines. Cambridge University Press, 2000.
- SVMs have been applied with success in a wide range of areas like: bio-informatics, text mining, hand-written character recognition, etc.

Two Linearly Separable Classes

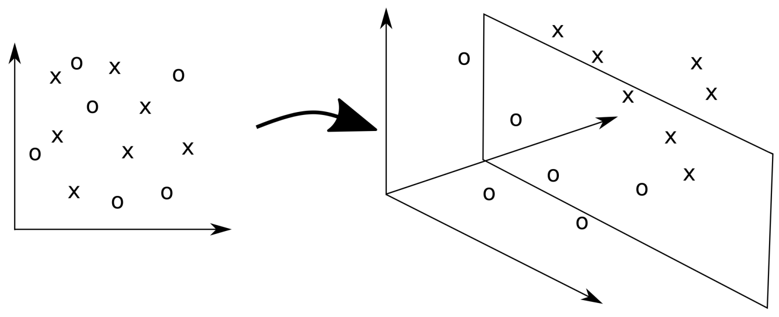


- Obtain a linear separation of the cases (binary classification problems)
- Very simple and effective for linearly separable problems
- Most real-world problems are not linearly separable!

The Basic Idea of SVMs

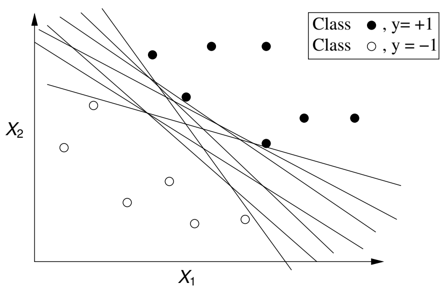
- Map the original data into a new space of variables with very high dimension.
- Use a linear approximation on this new input space.

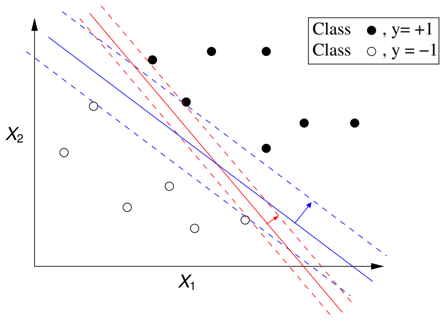
The Idea in a Figure



Map the original data into a new (higher dimension) coordinates system where the classes are linearly separable

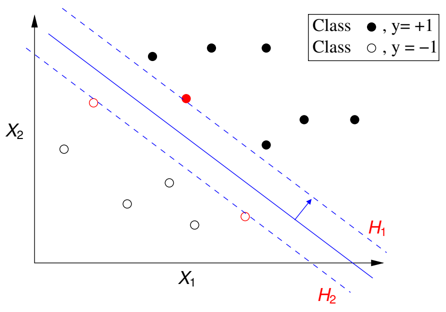
Maximum Margin Hyperplane





- There is an infinite number of hyperplanes separating the two classes!
- Which one should we choose?!
- We want the one that ensures a better classification accuracy on unseen data
- SVMs approach this problem by searching for the maximum margin hyperplane

The Support Vectors



- All cases that fall on the hyperplanes H_1 and H_2 are called the support vectors.
- Removing all other cases would not change the solution!

The Optimal Hyperplane

- SVMs use quadratic optimization algorithms to find the optimal hyperplane that maximizes the margin that separates the cases from the 2 classes
- Namely, these methods are used to find a solution to the following equation,

$$L_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j}^{n} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

Subject to :

$$\alpha_i \geq 0$$

$$\sum_i \alpha_i y_i = 0$$

- In the found solution, the α_i's > 0 correspond to the support vectors that represent the optimal solution

Recap

- Most real world problems are not linearly separable
- SVMs solve this by “moving” into a extended input space where classes are already linearly separable
- This means the maximum margin hyperplane needs to be found on this new very high dimension space

The Kernel trick

- The solution to the optimization equation involves dot products that are computationally heavy on high-dimensional spaces
- It was demonstrated that the result of these complex calculations is equivalent to the result of applying certain functions (the kernel functions) in the space of the original variables.

The Kernel Trick

Instead of calculating the dot products in a high dimensional space, take advantage of the proof that $K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{z})$ and simply replace the complex dot products by these simpler and efficient calculations

Summary of the SVMs Method

- As problems are usually non-linear on the original feature space, move into a high-dimension space where linear separability is possible
- Find the optimal separating hyperplane on this new space using quadratic optimization algorithms
- Avoid the heavy computational costs of the dot products using the kernel trick

How to handle more than 2 classes?

- Solve several binary classification tasks
- Essentially find the support vectors that separate each class from all others

The Algorithm

- Given a *m* classes task
- Obtain *m* SVM classifiers, one for each class
- Given a test case assign it to the class whose separating hyperplane is more distant from the test case

Obtaining an SVM in R

Training

```
library(tidymodels)
data(iris)

svmSpec <-
  svm_rbf() %>%                  # the type of model
  set_engine("kernlab") %>%     # the implementation to use
  set_mode("classification")    # type of task

s <- svmSpec %>% fit(Species ~ ., data = iris)   # fit the model to the data

svmSpec2 <-
  svm_rbf(cost=10, margin = 0.01) %>%
  set_engine("kernlab") %>%
  set_mode("classification")

s2 <- svmSpec2 %>% fit(Species ~ ., data = iris)
```

Obtaining an SVM in R (2)

Predicting

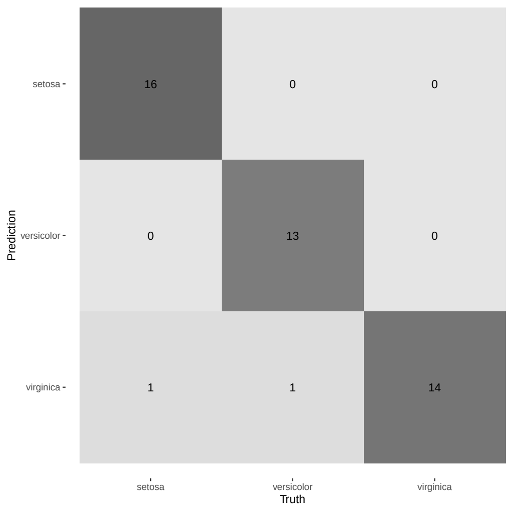
```
dataSplit <- initial_split(iris, prop = 0.7)
irTr <- training(dataSplit)   # training set
irTs <- testing(dataSplit)    # test set

svmTr <-
  svmSpec %>% fit(Species ~ ., data = irTr)

results <- irTs %>% select(Species) %>% bind_cols(predict(svmTr, new_data = irTs))
head(results)

##   Species .pred_class
## 1  setosa      setosa
## 2  setosa      setosa
## 3  setosa      setosa
## 4  setosa      setosa
## 5  setosa      setosa
## 6  setosa      setosa

results %>% metrics(Species,.pred_class)

## # A tibble: 2 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.956
## 2 kap      multiclass     0.933
```

Obtaining an SVM in R (2)

Predicting (cont.)

```
results %>% conf_mat(Species,.pred_class)

##             Truth
## Prediction   setosa versicolor
##   setosa         16          0
##   versicolor      0         13
##   virginica       1          1
##             Truth
## Prediction   virginica
##   setosa             0
##   versicolor         0
##   virginica         14
```

```
autoplot(results %>% conf_mat(Species,.pred_class),
         type="heatmap")
```



ε-SV Regression

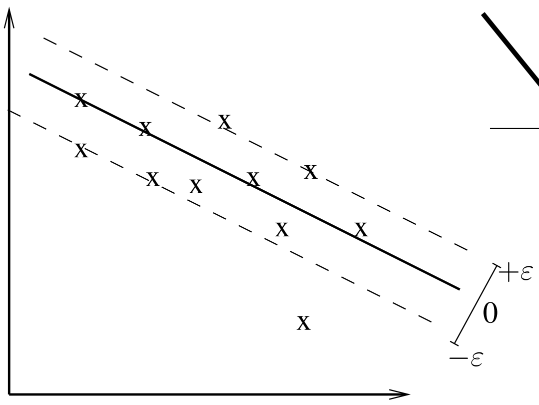
- Vapnik (1995) proposed the notion of ε support vector regression
- The goal in ε-SV Regression is to find a function $f(x)$ that has at most ε deviation from the given training cases
- In other words we do not care about errors smaller than ε

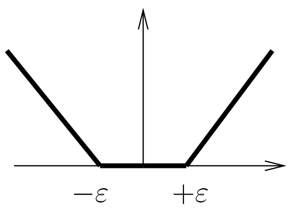
V. Vapnik (1995). The Nature of Statistical Learning Theory. Springer.

ε-SV Regression (cont.)

- ε-SV Regression uses the following error metric,

$$|\xi|_\varepsilon = \begin{cases} 0 & \text{if } |\xi| \le \varepsilon \\ |\xi| - \varepsilon & \text{otherwise} \end{cases}$$





ε-SV Regression (cont.)

- The theoretical development of this idea leads to the following optimization problem,

$$\text{Minimize}: \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*)$$

$$\text{Subject to}: \begin{cases} y_i - \mathbf{w}\cdot\mathbf{x} - b & \leq \varepsilon + \xi_i \\ \mathbf{w}\cdot\mathbf{x} + b - y_i & \leq \varepsilon + \xi_i^* \\ \xi_i, \xi_i^* & \geq 0 \end{cases}$$

  where C corresponds to the cost to pay for each violation of the error limit ε

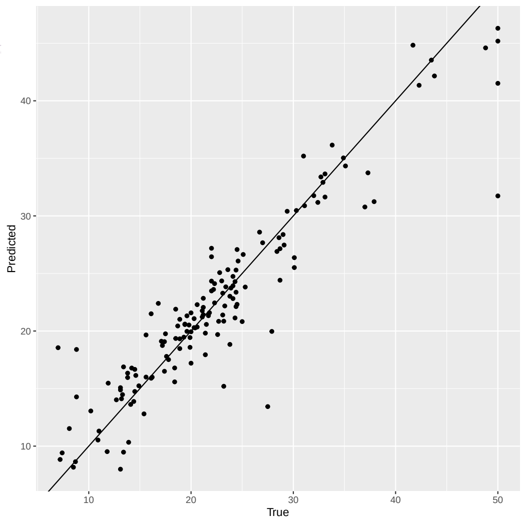
ε-SV Regression (cont.)

- As within classification we use the kernel trick to map a non-linear problem into a high dimensional space where we solve the same quadratic optimization problem as in the linear case
- In summary, by the use of the $|\xi|_\varepsilon$ loss function we reach a very similar optimization problem to find the support vectors of any non-linear regression problem.

SVMs for regression in R

```
data(Boston,package='MASS')
dataSplit <- initial_split(Boston, prop = 0.7)
bTr <- training(dataSplit)   # training set
bTs <- testing(dataSplit)    # test set

svmSpec <-
  svm_rbf() %>%                # the type of model
  set_engine("kernlab") %>%    # the implementation to
  set_mode("regression")   # type of task

sTr <-
  svmSpec %>% fit(medv ~ ., data = bTr)

preds <- predict(sTr, new_data = bTs)
bTs %>% bind_cols(preds) %>% metrics(medv,.pred)

## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       3.39
## 2 rsq     standard       0.862
## 3 mae     standard       2.20
```

```
library(ggplot2)
ggplot(bind_cols(bTs,preds), aes(x=medv,y=.pred)) +
  geom_point() + geom_abline(slope=1,intercept=0) +
  xlab("True") + ylab("Predicted")
```



Model Ensembles and Random Forests

Model Ensembles

What?

- Ensembles are collections of models that are used together to address a certain prediction problem

Why? (Diettrich, 2002)

- For complex problems it is hard to find a model that “explains” all observed data.
- Averaging over a set of models typically leads to significantly better results.

Dietterich, T. G. (2002). Ensemble Learning. In The Handbook of Brain Theory and Neural Networks, Second edition, (M.A. Arbib, Ed.), Cambridge, MA: The MIT Press, 2002. 405-408.

Random Forests (Breiman, 2001)

- One of the keys to sucessful ensembles is diversity among the models
- Random forests are formed by a set of decision tree models
- Diversity is achieved by obtaining each tree in different ways
  - There are differences in the training set
  - There are differences in the way the variables are used in the tree
- Random Forests consist of sets of tree-based models where each tree is obtained from a bootstrap sample of the original data and uses some form of random selection of variables during tree growth

Breiman, L. (2001): "Random Forests". Machine Learning 45 (1): 5—32.

Random Forests - the algorithm

- For each of the k models
  - Draw a random sample with replacement to obtain the training set
  - Grow a classification or regression tree
    - On each node of the tree choose the best split from a randomly selected subset m of the predictors
- The trees are fully grown, i.e. no pruning is carried out

Random Forests in R

The package `randomForest`

```
library(tidymodels)
data(Boston,package='MASS')
dataSplit <- initial_split(Boston, prop = 0.7)
bTr <- training(dataSplit)  # training set
bTs <- testing(dataSplit)   # test set

rfSpec <-
  rand_forest() %>%          # the type of model
  set_engine("ranger") %>%   # the implementation to use
  set_mode("regression")     # type of task

rfTr <-
  rfSpec %>% fit(medv ~ ., data = bTr)

preds <- predict(rfTr, new_data = bTs)
bTs %>% bind_cols(preds) %>% metrics(medv,.pred)

## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       3.44
## 2 rsq     standard       0.879
## 3 mae     standard       2.27
```

A classification example

```
data(iris)
dataSplit <- initial_split(iris, prop = 0.7)
irTr <- training(dataSplit)  # training set
irTs <- testing(dataSplit)   # test set

rfSpec <-
  rand_forest() %>%                # the type of model
  set_engine("randomForest") %>%   # the implementation to use
  set_mode("classification")     # type of task

rfTr <-
  rfSpec %>% fit(Species ~ ., data = irTr)

results <- irTs %>% select(Species) %>% bind_cols(predict(rfTr, new_data = irTs))
results %>% metrics(Species,.pred_class)

## # A tibble: 2 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.933
## 2 kap      multiclass     0.900
```

Hands on Linear Regression and Random Forests

the Servo data set

Load in the data set `Servo` from package **mlbench** and answer the following questions:

1. How would you obtain a random forest with 750 trees to forecast the value of *Class* (it is a numeric variable)
2. Repeat the previous exercise but now using a linear regression model.
3. Obtain the predictions of the two previous models for the data used to obtain them. Draw a scatterplot comparing these predictions
4. Split the data in train and test sets (80%-20%). Obtain the two previous models on the training data and get their predictions for the test set. Compare the predictions of the models.