Week 6

Predictions

- motivation:
- ~~key ideas~~ steps in predictive studies
- error measure

kaggle.com - prediction contests

Steps

1. Find the right data
2. Define your error rate
3. Split data into
   - training
   - testing
   - validation (optional)
4. On the training set pick features
5. On the training set pick prediction function
6. On the training set - cross-validate

if no validation:

apply ~~a~~ 1x to test set

if validation

apply to test set and refine

apply 1x to validation

True/False Positives (for binary classification)

Positive = identified Negative - rejected

	Positive	Negative
True	correctly identified	~~in~~correctly ~~identified~~ rejected
False	incorrectly identified	incorrectly rejected

e.g.

	Positive	Negative
True	Sick people correctly diagnosed	Healthy identified as healthy
False	Healthy people diagnosed as sick ~~healthy~~	sick people identified as healthy

Error rate:

Positive predictive value =

$$\frac{\sum TP}{\sum \text{test outcome positive}}$$ ← true value ← test value

~~Condition~~

Negative predictive value

$$\frac{\sum TN}{\sum \text{test outcome negative}}$$ true test

Sensitivity = $$\frac{\sum TP}{\sum \text{Condition positive}}$$ true value test value

(sick / test sick)

Specificity = $$\frac{\sum TN}{\sum \text{condition Negative}}$$ (healthy / test healthy)

False positive - Type 1 error

False negative - Type 2 error

Common error measures

- mean square error

  continuous data, sensitive to outliers

- median absolute deviation

  continuous data, often more robust

- sensitivity (recall)

  if you want few miss~~ing~~ed positives

- specificity

  if you want few negatives called positives

- accuracy

  weight false positives/negatives equally

Study design

all data

training data

quiz | test

actual data

for evaluating

applied in the ~~fo~~ very end

key issues:

- accuracy
- over fitting (you fit well to training sets but don't to other samples)
- interpretability
- computational speed

Resources:

- practical machine learning
- elements of statistical learning
- coursera machine learning
- machine learning for hackers

Cross-validation

how to estimate error rate for your predictive functions

key ideas:

- sub-sampling the training data
- avoiding overfitting
- making predictions generalizable

Goal of cross-validation - how well your predictions will work on different samples

- accuracy on training set is optimistic
- a better estimate comes from an independent set

but we cannot ~~test~~ use the test set when building the model - or it becomes the part of the training set

test set accuracy

basic approach

1. Use the training set

2. Split it into training/test sets

3. Build a model on the training set

4. Evaluate on the test set

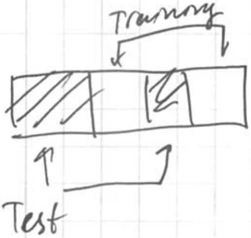
5. Repeat and average the estimate errors

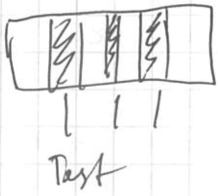
Used for:

- picking variables to include in a model
- picking the type of prediction function to use
- picking the parameters in the prediction function
- comparing different predictors

Ways:

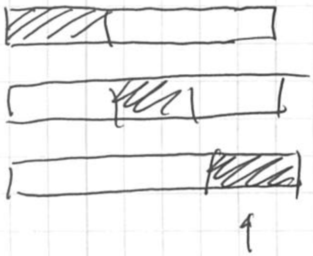
- random subsampling





← do over and over again

K-fold



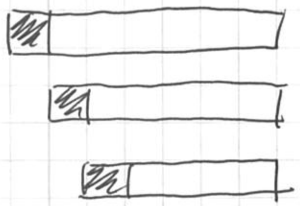
3-fold cross validation

1/3rd

testing

training

leave one out



at each step, leaving out ~~testing~~ training sample

quite stable, less biased

- testing

Notes

- training sets and testing sets must come from the same population
- sampling should be designed to mimic real patterns

(sample chunks of times - not just random sampling for time series)

Predicting with regressions

- lm or glm -

predict new values with coefficients

prons:

easy to implement
easy to interpret

cons:

poor perfomance in non-linear settings

predicting:

coef(lm1)[1] + coef(lm2)[2] * 80

what you want to predict

equivalent

predict function

Calculating training set/test set errors

#RMSE

sqrt(sum((lm1$fitted - train$eruptions)^2))

test

~~war~~ var we're predicting

sqrt (sum ((

predict(lm1, newdata = test) -
test$eruptions) ^2

))

(how but not why)

↓

error's slightly larger

prediction intervals

95%-prediction

good for normaly distributed data

Choosing a cut-off. (re-substitution)

a value at which you get the smallest amount of errors

Comparing models

cost= function (win, pred=0) mean(abs
$\underbrace{\text{win - pred)}}_{\text{when the error is high enough}}$ > 0.5))

library(boot)

```
cv1 = cv.glm(ravens data, glm1, cost, K=3)
```

error est. function

regression model

number of folds

```
cv1$delta
```

the less the better

for checking what regression model is better

Predicting with Trees

Better at capturing non-linearity

key ideas:

- iteratively split variables into groups
- split where maximally predictive
- evaluate "homogeneity" within each branch (how similar the objects are within the groups) and maximize it in each group
- fitting multiple trees often works better

Pros:

- easy to implement
- easy to interpret
- better perfomance in non-linear settings

Cons:

- without pruning / cross-validating can lead to overfitting
- harder to estimate uncertainty
- results may be variable

example – decision tree

Basic algorithm:

1. Start will all variables in one group
2. Find the variable / split that best separates the outcomes (in homogenious)
3. Divide the data into two groups ("leaves") on that split ("node")
4. Within each split, find the best variable / split that separates the outcomes
5. Continue until the groups are small enough or sufficiently „pure"

~~predict~~

Example

data(iris) – need to classify it

library(tree) outcome

tree1 = tree(Species ~ Sepal.Width + Petal.Width)

summary(tree1)

says the number of terminal nodes: 5

(measure of impurity)

Residual mean deviance: 0.204 = 29.6 / 145

Misclassification error rate: 0.0333 = 5/150

(how often you misclassify)

plot(tree1)

text(tree1)

partition.tree(tree1)

instead of lm

names have to match

pred1 = predict(tree1, newdata) – returns probabilities

pred1 = predict(tree1, newdata, type = "class")

returns not probabilities, but a class

partition.tree(tree1, "Species", add = T)

Pruning

if we built a tree and want to cross-validate it

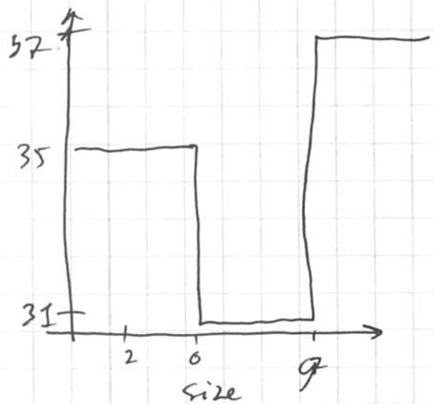
cv.tree - for this

```
plot(cv.tree(treeCars, FUN=prune.tree, method=
                "misclass")
```

number of misclassification errors

```
cv.tree(treeCars)
```

default method is deviance



as the model increases in size

- start with 35 misclassifications
- between 6-9: fewer missclassifications
- then we have misclassification again

what happens to different sizes of the model prune.tree (tree1, best=4)

give me the best tree that has exactly
4 terminal leaves / nodes

And it gives a smaller subset of the tree -

And this smaller set will get a smaller error
rate

(prune - обрезать,
подрезать [дерево])

Resubstitution Error

table (orig, predict(pruneTree, type="class"))

↓

Shows a table

works better than not pruned trees.