Data Analysis and Knowledge Discovery

Model validation

Jukka Heikkonen

University of Turku
Department of Information Technology

Jukka.Heikkonen@utu.fi

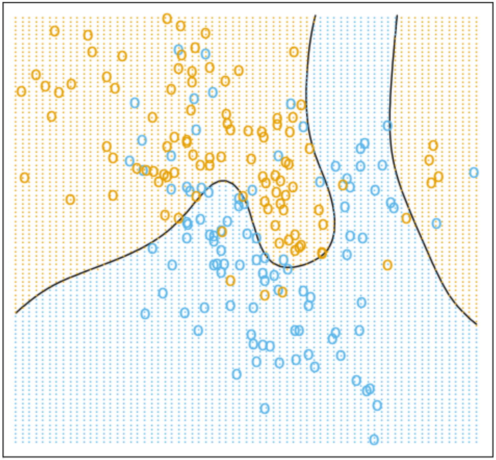
Where are we now

- Using a machine learning method (such as k-nearest neighbours) we can learn a classifier or regressor
- some classification performance measures
    - misclassification rate
    - misclassification rate with costs
    - area under ROC curve (AUC)
- some regression performance measures
    - mean squared error
    - mean absolute error
    - correlation measures also applicable (see early lectures)
- One way or another, all these measures compute how well predicted outputs match true outputs. But what data should we use to compute these?

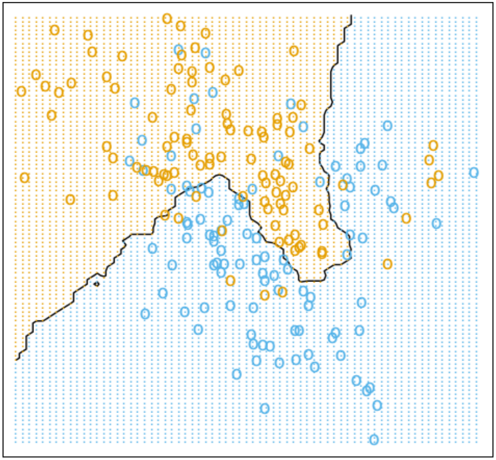
Where are we now

- Evaluation: given a model, estimate how well it would predict on new data (arriving from similar source as the training data)
- True error rate: predictors expected error rate on the whole population
- Remember: fit to training data does not guarantee generalization to new data
- a complex model may overfit
  - low error on training set, but captures poorly (or not at all) a real predictive pattern
  - when tested on new data, overfitted model predicts inaccurately (or even randomly)
- analogy to human learning: training data is $2 + 2 = 4$, $3 + 2 = 5$ and $4 + 11 = 15$
  - overfitting: the model predicts these correctly, but has no idea what $1 + 1$ equals, since it was not in training set
  - learning: a good model captures the underlying pattern, and thus knows that $1 + 1 = 2$

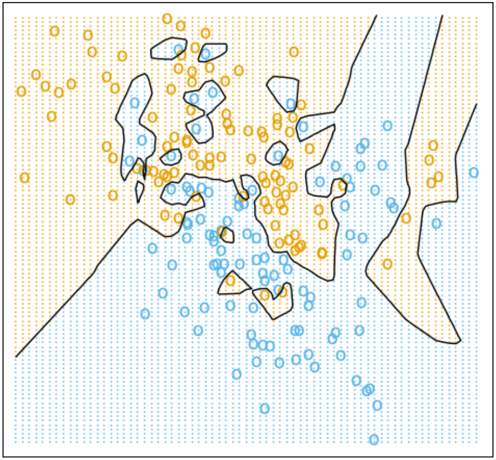
True model



Reasonable model (15-nearest neighbour)



Overfitting (1-nearest neighbour)



How to choose the best possible k ?

Overfitting

- 1-nearest neighbour classifier/regressor has always 0 training error
- model memorizes all training data, and thus knows the correct output to predict for these instances
- this guarantees nothing about generalization to new data
- a general phenomenom in machine learning, also linear models, neural networks etc. may overfit
- thus models should be tested on independent test data

Validation

- Validation means measuring the predictor's behavior on data points other than those in the training set
- We need this in order to reliably select the hyperparameters of our model (such as number of neighbours), or in order to choose between models produced by different algorithms
- Also needed for evaluating how well the final model can be expected to generalize to completely new data

Training and test data

To avoid overfitting and to have a more realistic estimation for the error that the model will make for new data, the data set is often split into training data and test data.

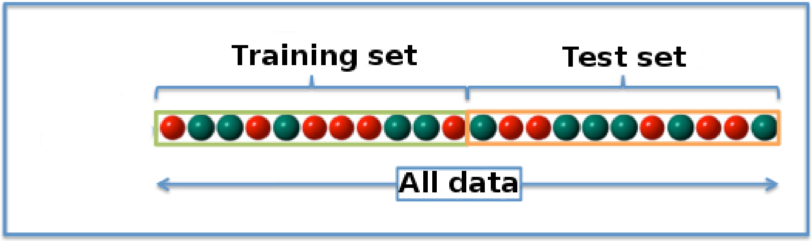
- The model is computed ("trained") only with the training data.
- The test data set is used to evaluate the model (for instance, to calculate the misclassification rate).
- Sometimes known as the holdout method

Typically, 2/3 of the data are used for training and 1/3 for testing.

Instances are assigned randomly to the training and the test data.

For classification problems, stratification is often applied, guaranteeing that the training and test data have the distribution of classes as the original data.

The holdout method



- In practice, once the model performance has been evaluated, combine all data and use it to train the final model in order to construct as accurate predictor as the data allows.
- Pessimistic bias, the model evaluated, that was trained only on the training set, can be less accurate than the final model trained on all data

The holdout method

- Using a separate test set gives an unbiased estimate of the true expected error (or AUC, or...) on new test data
- Can still be unreliable on too small test sets due to variance, we might get very "lucky" or "unlucky" in which instances happen to fall in the test set
    - you may compute confidence intervals using standard statistical approaches, not covered in this course
- two conflicting goals
    - we want as much data as possible for training in order to be able to discover the possible pattern as well as possible
    - we want as much data as possible for testing in order to get as reliable estimate of true error rate as possible
- when working with Big Data not a problem, we have as much training and test data as we want

Limitations of the holdout method

- What if we have only a data set of 100 instances
- Using e.g. 70 for training and 30 for testing, we get a very unreliable estimate of test performance (too small test set)
- Using e.g. 30 for training and 70 for testing does not help much, now too little training data
- especially bad on imbalanced data, where we could have e.g. 90 instances from majority and only 10 from minority class
- would be nice to use all data for both training and testing, but we can't use training error directly due to overfitting
- Cross-validation to the rescue!

Leave-one-out cross-validation

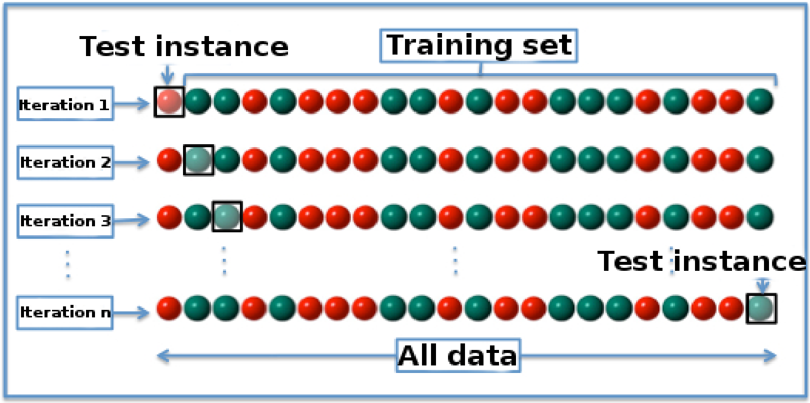
Basic idea

- we have a data set of n instances
- a model cannot be tested on its own training data due to overfitting, so need to divide into training and test set
- question we ask for each instance $i \in \{1, ..., n\}$
  - "what would the model have predicted for the i:th instance, if it had not seen it during training?"
- we can answer this question by removing the i:th instance, training the model on the $n - 1$ remaining instances, and making a prediction for the i:th one (a test set of size 1)
- this is known as leave-one-out cross-validation

Leave-one-out cross-validation

```
1 for $i = 1$ to $n$ do
2         train model on all but the $i$th instance
3         predict output for the $i$th instance with the model
4 end for
5 compute error between true and predicted outputs
```

Leave-one-out cross-validation



Analyzing leave-one-out cross-validation

- validating the models trained on $n-1$ training instances gives us a good (almost unbiased) estimate of how the final model trained on all data will behave
- one each iteration, the model cannot overfit to the test instance since it is not part of training set
- leave-one-out allows us to use the whole data set for both training and testing simultaneously
- parameter selection: pick the model with lowest leave-one-out error (or highest AUC or correlation or...)
- final evaluation can also be done with leave-one-out

Implementing leave-one-out

- for nearest neighbor extremely simple to implement
  - for each instance in training set, compute prediction from k nearest neigbors, excluding the instance itself
- generally, for other types of learning methods you need a loop where the machine learning method is trained n times
- can be computationally expensive
  - 5000 instances, 1 minute training time $\rightarrow$ leave-one-out cross-validation takes 3.5 days
- Advanced topic: some machine learning methods allow computational shortcuts for computing leave-one-out much faster (k-nn, ridge regression, Naive Bayes...)

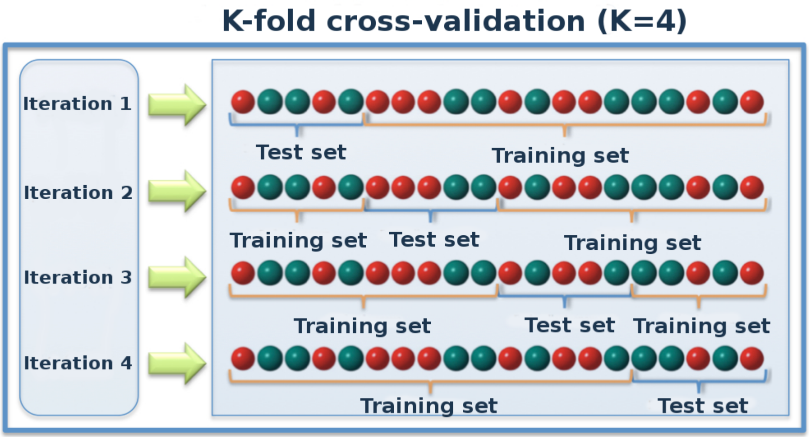
K-fold cross-validation

- for larger datasets leave-one-out can be impractical due to high computational costs
- K-fold cross-validation: same basic idea, but instead of leaving a single instance out at a time, leave several
- data is randomly divided into K (approximately) equally sized non-overlapping parts called folds
- on each round of cross-validation, use one fold for testing and remaining $K-1$ for training
- usual choices for K: 10 or 5
- leave-one-out as extreme case, where $K=n$

K-fold cross-validation

```
1 split the data randomly into $K$ equally sized folds
2 **for** $i = 1$ to $K$ **do**
3         train model on data from all but the $i$th fold
4         predict outputs for the instances belonging to the $i$th fold with the model
5 **end for**
6 compute error between true and predicted outputs
```

K-fold cross-validation



Analyzing K-fold cross-validation

- much faster than leave-one-out; 5000 instances, 1 minute training time $\rightarrow$ 10-fold CV takes 10 minutes
- small pessimistic bias, the model evaluated can be slightly less accurate than the final model trained on all data
- some variance depending on the random fold split
- rule of thumb: you can use leave-one-out on small data sets (e.g. 100 instances), 10-fold CV on medium sized (e.g. couple of thousands instances) and a single test set on large data sets

Some variants of K-fold cross-validation

- stratified K-fold cross-validation
  - especially imbalanced classification problems
  - stratification: split the data into folds so, that each fold roughly has the same fraction of members from each class as the full data set
  - guarantees that on all the iterations, the class distributions of the training and test set resemble that of the full data set
  - otherwise with very bad luck, it might happen that all instances from minority class end up in one fold
- M-times repeated K-fold cross-validation
  - some variance in the K-fold estimate due to the randomness in how the folds are split
  - can reduce the variance and thus obtain more reliable error estimates by repeating the whole procedure with different fold divisions M times (e.g. M=100) and taking the average
  - 5000 instances, 1 minute training time $\rightarrow$ 100 times repeated 10-fold CV takes almost 17 hours (1000 minutes)

Combining model selection and final evaluation

- using cross-validation (or a single test set) we can do model selection
    - choose hyperparameter value / feature set / learning algorithm that gives lowest cross-validation error (or conversely, highest accuracy/AUC/correlation...)
- using cross-validation (or a single test set) we can evaluate the expected error rate of the final model
- what if we want to do both at the same time?

Combining model selection and final evaluation

- The error rate estimate of the final model biased (smaller than the true error rate) since out of several choices we pick the one having smallest estimated error
- example: 6 models, true expected error rates are [0.40, 0.20, 0.10, 0.10, 0.10]
- leave-one-out cross-validation gives us estimates: [0.42, 0.18, 0.13, 0.10, 0.07]
- we pick the last one, leading to a good choice of model, but the leave-one-out-estimate now biased (0.07 smaller than the true error rate 0.10)
- can be problematic especially on small datasets and when testing a large number of different models
- need additional independent test data, that was not used for model selection

Combining model selection and final evaluation

- first idea: instead of splitting data to training and test sets, split into three
- e.g. 50%, 25%, 25% split
- training set used for training the models to be evaluated
- we test all models against the validation set, pick one with lowest error
- final model is evaluated against the test set, which has not been used in any way until this time point
- good protocol if you have lots of data, can we combine this same idea with cross-validation?

Nested cross-validation

- nested, aka external cross-validation
- cross-validation within cross-validation
- inner loop for model selection, outer for evaluation
- on each round of the outer loop, data split to training and test set
    - compute a cross-validation estimate for each tested parameter on training set, choose best parameter
    - train on training set and predict on test set
- repeat for each fold

Nested cross-validation

- can be computationally very expensive (combinatorial explosion)
- leave-one-out within leave-one-out: need to train model $n * (n-1)*(\#\text{tested params})$ times
- 5000 instances, 1 minute training time $\rightarrow$ nested leave-one-out cross-validation takes about 48 years
- nested-leave-one-out feasible only on very small data sets, more often nested K-fold
- nesting especially important when you test a really large number of hyperparameters

Software for cross-validation

- you should program cross-validation once by yourself to get the hang of it
- all major data analysis environments offer also this functionality
- see e.g. `http://scikit-learn.org/stable/modules/cross_validation.html`

Alternative criteria for model selection

- there are alternative approaches for model selection compared to using cross-validation
- information criteria, that balance fit to training data with model complexity
- e.g. Akaike's and the Bayesian information criterion (see book)
- general underlying principles, but the way the complexity can be evaluated is highly dependent on our assumptions about the type of models considered
- cross-validation more than competitive with these approaches, and generally applicable to any type of model used