Norwegian University of Science and Technology
Department for Computer Science

# Methods in Artificial Intelligence TDT4171 Spring 2021

**Jørgen Rosager**

**Exercise 4**

1 **a)** The continuous variables are *Age*, *Name*, *Ticket*, *Cabin*, *Fare* as these are pretty much unique for almost all (except for *Cabin* where more people can live in same cabin, but this does not warrant enough discrete cases to make it discrete). Both *SibSp* and *Parch* can be treated as both as for most persons this number should be somewhere between 0-10. However, there are edge cases and this is in reality a continuous variable. This means that when we train it with these as discrete attributes there is a chance that the DT meets a unhandled case (for example $SibSp = 11$) when used on testing data. In my DT model I have handled this by making an educating guess if *Parch* or *SibSp* are chosen as discrete attributes and there is no known case of this from the training data. This proved to be smart as the highest accuracy was gotten when using *SibSp* as a discrete attribute (see results below).

When trained on all discrete attributes these were the results:

This model is extremely complicated and got an accuracy of 86.8%.
The attribute *Embarked* shouldn't really affect the survival of the passenger so I decided to drop this. The results:

The graph got a bit simpler and the accuracy increased to 88.0%, this makes sense because the attribute *Embarked* was overfitting the problem.

In fact the best results are gotten by also removing the *Parch* attribute:

The graph is now easily readable the accuracy improved to 88.5%.

**b)** Support for continuous variables were added by changing the *gain* function to loop over the unique values of the continuous attribute and then find the best split point (taken by averaging between two points).

*Age* and *Cabin* was removed since they had some missing values. *Ticket* was removed as the ticket number itself should not have any influence on survival, and it is also not an easy value to split on as it has a combination of string and int which would need more processing.

The results by using the attributes $Fare$, $Parch$ and $SibSp$ as continuous attributes with the discrete values $Sex$ and $Pclass$:

The accuracy was also 88.5% which is the same as the best results using discrete attributes only. There is an argument for whether *Fare* actually should influence survival, but in this case the results would have been the same (88.5% accuracy on test set).

**c)** The best performance in both **a)** and **b)** was equal at 88.5%. Although we could expect better results when also using continuous values this was not the case, however a significant gain is that the DT is much simpler and although it is more heavy to train as atleast my algorithm goes through all distinct continuous values and checks the split gain. In this case the expected performance did not increase so it does not warrant the usage of continuous values.

Another interesting aspect would have been the *Age* attribute because of the women and children first code of conduct when saving passengers on the Titanic. Since there was a lot of missing values in the training data set we did not use this attribute, but it could have increased performance.

There are several methods we could have used to improve this simple DT:

1. **Bootstrapping**: With trainingsets which are this small (small in the machine learning world) there can be a lot of variance when training on different data. For example if we were to split the trainingset in two halves and then train a DT on each one we could get two trees with very different results (high variance). In order to combat this we could use Bootstrapping. The idea is to minizime the variance by using sub-samples of our trainingset with replacement and train several DTs on these sub-sample trainingsets. The key is that we don't use just one of the trees when predicting, but we calculate the *average* over all the DTs, which will reduce the overall variance and could give better accuracy.
2. **$\chi^2$-pruning**: Unless we remove some of the irrelevant attributes by hand there is a general problem of overfitting with our DT. For example when using the attribute *Embarked* which should have no bearing on survival. To combat this we could use a technique called $\chi^2$-pruning. This pruning removes irrelevant nodes, the key here is to determining the relevance of a node. This can be done by finding nodes that have close to 0 in information gain, however underlying noise can influence this, so we will also use a significance test of for example 5% deviation from 0. We can then replace these irrelevant nodes with leaf nodes and severly simplify the tree which combats overfitting and requires less domain knowledge by me who writes the DT.
3. **Gain ratios on multivalued attributes**: The information gain test we are using in this DT can be misleading when using it on multivalued attributes such as *Pclass*, *Parch* etc. In the extreme cases for example when *SibSp* has very few cases on each variable, this would result in a high information gain, but it would not necessarily give us the best tree. To combat this we could use gain ratio which takes this into account. This could possibly lead to a better tree because the splits would be chosen on a more equal basis among different attributes with different number of possible values.

2 The columns missing data are the *Age* and *Cabin* columns. *Age* is just an integer, but

*Cabin* can be a combination of several string and integers, for example "C32 C31", which makes this attribute harder to replace.

Both of these are regarded as continuous variables, which means in order to replace them we should use regression rather than classification (it would be possible to use classification, but it does not make as much sense).

These are the proposed solutions:

1. **k-Nearest-Neighbors**: This method relies on finding the nearest neighbors of the rows with missing values by averaging over them. The hard thing here is to find a good distance metric. This is how I see the implementation:
   (a) Calculate the distance metric, for discrete attributes and the continuous attributes *Name*, *Cabin* I would use the Hamming distance. For the continuous attributes *Fare*, *Ticket* and *Age* I would use the Euclidean or Cosine distance between them.
   (b) Find the $k$ nearest neighbors (design parameter), average over these to choose the continuous output on *Age* and *Cabin*.

   **Advantages**:
   - Non-parametric, does not rely on parameters like for example mean or variance
   - Simple to implement, we only need to worry about the distance metric which could be hard, but managable.

   **Weaknesses**:
   - Very expensive as it has to go through the whole dataset in order to find the nearest neighbors, however we have a smaller trainingset and not too many attributes so my guess is that it wouldn't be too expensive. High dimensions (number of attributes) and large datasets are the big drawback of this method.
   - Can be sensitive to noise in the dataset, we may need to manually remove outliers.
   - Requires high domain knowledge to find good distance metrics.

   This method assumes that the distance metrics are good enough, which is hard to determine before testing with it.
2. **Mean Imputation**: A single imputation method which works by imputating a variable with the mean of that variable. This is how I see the implementation:
   (a) Calculate the mean of the variable, for *Age* this is straight forward, but *Cabin* requires a bit more thought. A simple method is to just take the mean of the character (A, B, C etc.) and the mean of the integer part and then combine these, this will however not take into account the possibility of living in multiple cabins. This can be solved by for example by taking the mean of number of cabins and then choose the second cabin nearby of the first.
   (b) Assign the missing values with this imputated mean.

   **Advantages**:
   - Very simple to implement, requires little calculation.
   - Does not change the mean of the variable.

- Requires no domain knowledge to implement and use.

**Weaknesses**:

- This very simplistic way of viewing the data can attenuate any underlying correlations with other variables as it just chooses based on mean.

Assumes that the mean is representative and assumes that the variable has no correlation with the other attributes, which in this case I would say is wrong as for example *Cabin* correlates with the *Cabin* of people with same name (families are more likely to live in the same or nearby cabins). I do not believe this method should be chosen for filling out these missing values.

3. **Multiple Imputation**: This is a method which averages the outcome across multiple imputated datasets to deal with the issue of increased noise when imputating. This is how I see the implementation:
   (a) Draw $m$ imputated values from a distribution (create normal distribution for variable) until there are $m$ completed datasets. Here one could also use MCMC (Markov Chain Monte Carlo) to draw th $m$ imputated variables.
   (b) Analyze the $m$ datasets, returning $m$ analysis.
   (c) Pool the $m$ analysis into one result. This is done by calculating parameters such as mean, variance and confidence interval. The values are then imputated from this distribution.

   **Advantages**:

   - Easy to implement (although harder than the previous methods)
   - The problem with single imputation (such as mean imputation) is the negligence of uncertainty. This leads to overly precise results and errors in the imputated values. Multiple imputation combats this problem by doing the imputation several times and combining the results, this leads to no biases if the imputation model is correct.

   **Weaknesses**:

   - Can lead to errors if the imputation model is incorrect
   - Slightly more advanced (and therefore harder to implement) than the previous methods
   - More expensive calculations (drawing, analyzing and pooling) than single imputation methods

   This method is very dependent on the imputation model chosen, which can also be its downfall. However it should still be more accurate than the single imputation methods like mean imputation.