1 Hypothesentest

1.1 Test Manual

1. Check samples
   - If 1 sample: σ_0 known or unknown
   - If 2 sample: dependent or independent
2. State H_0 and H_1
3. Select and calculate the test statistics
4. Select α (given)
5. Find the critical value in the table
6. Result

1.1.1 Step 1

dependent: A dependent variable is the variable being tested and measured in a scientific experiment.
independent: An independent variable is the variable that is changed or controlled in a scientific experiment to test the effects on the dependent variable.

1.1.2 Step 2

Hypothesis	H_0	H_1
Two-sided	$\mu_x = \mu_0$	$\mu_x \neq \mu_0$
One-sided	$\mu_x \leq \mu_0$	$\mu_x > \mu_0$
One-sided	$\mu_x \geq \mu_0$	$\mu_x < \mu_0$

1.1.3 Step 3

1 Sample:
σ_x known $\rightarrow$ Gauss/z-test $z_0 = \frac{\overline{x}-\mu_0}{\sigma_x} \quad$ N(0,1)
σ_x unknown $\rightarrow$ t-test $t_0 = \frac{\overline{x}-\mu_0}{s_x}\sqrt{n} \quad t_{n-1}$ with $s_2^x = \frac{1}{n-1}\sum(x_i - \overline{x})^2$

2 Sample:
independent $\rightarrow Welch - Test t_0 = \frac{\overline{x}-\overline{w}-\mu_0}{s_{\overline{x}-\overline{w}}} \; t_{df}$
dependent $\rightarrow Paired t - test$

1.1.4 Step 5

t^c critical value in the table
Gauss/z-Test $\rightarrow$ use normal distribution
t-test, Welch-Test and paired t-Test $\rightarrow$ t-distribution

H_1	$t^c range$	$t^c range$
$\mu_x \neq \mu_0$	can be any	$\lvert t^c_{1-\frac{\alpha}{2};df}\rvert = \lvert t^c_{\frac{\alpha}{2};df}\rvert$
$\mu_x > \mu_0$	must be positive	$\lvert t^c_{1-\alpha;df}\rvert$
$\mu_x < \mu_0$	must be negative	$\lvert t^c_{\alpha;df}\rvert$

1.1.5 Step 6

Reject H_0:

H_1	$t^c range$	$t^c range$
$\mu_x \neq \mu_0$	$p < a$	$\lvert t_0\rvert > \lvert t^c_{1-\frac{\alpha}{2};df}\rvert$
$\mu_x > \mu_0$	$p < a$	$t_0 > t^c_{1-\alpha;df}$
$\mu_x < \mu_0$	$p < a$	$t_0 < t^c_{\alpha;df}$

σ - Standard Deviation for variable
s - Standard Deviation for sample

1.2 Welch-Test

μ_0 - Value of Null-Hypothesis
$\overline{x}$ - Mean of first set
$\overline{w}$ - Mean of second set
t_0 - P-Value of Welch-Test
t^c - P-Value from from T-Table

1.3 T-Test

H1	Rejection Region
$\overline{x} \neq \mu_0$	$\lvert t_0\rvert \geq t_{1-\alpha/2,n-1}$
$\overline{x} > \mu_0$	$t_0 > t_{1-\alpha,n-1}$
$\overline{x} < \mu_0$	$t_0 < t_{1-\alpha,n-1}$

1.4 Gauss/z-Test

H1	Rejection Region
$\mu \neq \mu_0$	$\lvert z_0 \rvert \geq z_{1-\alpha/2}$
$\mu > \mu_0$	$z_0 > z_{1-\alpha}$
$\mu < \mu_0$	$z_0 < z_{1-\alpha}$

1.5 Confidence Intervals

σ_x known: $[\overline{x} - z^c_{1-\frac{\alpha}{2}} * \frac{\sigma_x}{\sqrt{n}}, \overline{x} + z^c_{1-\frac{\alpha}{2}} * \frac{\sigma_x}{\sqrt{n}}]$
σ_x unknown, use S_x as estimate instead: $[\overline{x} - t^c_{1-\frac{\alpha}{2};n-1} * \frac{s_x}{\sqrt{n}}, \overline{x} + t^c_{1-\frac{\alpha}{2};n-1} * \frac{s_x}{\sqrt{n}}]$

2 Linear Regression

Fit trough linear function: $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 * x_i$
Residual e is the difference between observed and predicted value: $\hat{e}_i = y_i - \hat{y}_i$
Residual Sum Squares: Sum of all squared errors
$\hat{\beta}_0 = \overline{y} - \hat{\beta}_1 * \overline{x}$
$\hat{\beta}_1 = \frac{Cov(x,y)}{Var(x)}$

Interpretation:
$\hat{\beta}_0$: Output of linear regression model when all predictor variables are set to 0. Also called the intercept on y.
$\hat{\beta}_1$: Change in y for each unit increase in x.

Hypothesis test:
$t_0 = \frac{\hat{\beta}_1}{SE(\hat{\beta}_1)}\ t_n - 2$
$SE(\hat{\beta}_1) = \sqrt{\frac{RSS}{\sum(x_i - \overline{x})^2} * \frac{1}{n-2}}$

Evaluation of Model:
RSS
Mean Squared Error (MSE) $= \frac{RSS}{n}$
Root Mean Squared Error (REMSE) $= \sqrt{MSE}$

R Code:

```
model=lm(demand~t_a)
pm=predict(model)
```

3 Naïve Bayes, 0 Rule, 1 Rule

3.1 1 Rule

1. Build table with pos/neg for each Attribute with every class
2. Find Error Rate for each class of each attribute.
3. Calculate total error rate (Lowest error rate wins)

Error Rate: $\frac{\sum Lowest\ values\ of\ class}{\sum All\ classes}$

Windy	Playing	Not Playing	Error Rate
True	**3**	3	$\frac{3}{6}$
False	**6**	2	$\frac{2}{8}$

Total Error Rate: $\frac{3+2}{6+8}$

3.2 Naïve Bayes

1. Count each class for target attribute. Calculate prior.
2. If a **used** class is $= 0$, increment all classes by one.
3. Calculate likelihood of prior given a certain attribute.
4. Calculate the normalized likelihood of prior given an attribute
5. Highest normalized likelihood of prior given an attribute wins

Prior: $\Pr(h_l)$ $\frac{Amount\ of\ class}{\sum class}$
Likelihood of prior given an attribute: $\Pr(e_i|h_l)$: $\prod_{i=1}^{n} Pr(e_i|h_l) \cdot Pr(h_l)$
Normalized $\Pr(e_i|h_l)$: $\frac{Pr(e_i|h_l)}{\sum_{i=1}^{l} Pr(e_i|h_l)}$

4 Decision Trees

Classification:

- Internal *node* is a test on an attribute
- *Branch* represents an outcome of the test
- *Leaf* represents a class
- New *Instance* is classified by following a matching path to a leaf node

Optimal Tree

- For n attributes there are $2^{2^n} possibletreesFindingoptimaltreeisNP-complete$ Solution: **Greedy Algorithm** for tree construction:
    - Top down approach: Start with root
    - Every split is assessed with a measure
    - Best split is chosen
    - Repeat until all leaf nodes are pure or all attributes have been used

4.1 Information and Entropy

Entropy measures information content in bits → **Uncertainty of nodes**
$entropy(p_1, ..., p_n) = -\sum_{c=1}^{n} p_c * log_2(p_c)$

Information necessary to classify for $C = \sum c_i$
$info([c_1, ..., c_n]) = entropy(\frac{c_1}{C}, ..., c_n C)$

Average Information
Information weighted with the amount of outcomes on one leaf

Information Gain: Quality is spolit equal to the gained information gain
$gain(attribute) = gain(beforesplitbyattribute) - info(aftersplitbyattribute)$

Problem with information gain:

- Biased against attributes with a lot of edges like IDs
- Results in overfitting

→ Solution: Measurement that takes number and size of leafes into account
Intrinsic Information: (s is size of a leaf)
$intrinsicinfo([s_1, ..., s_n]) = info([s_1, ..., s_n])$

Gain Ratio:
$gainRatio(attribute) = \frac{gain(attribute)}{intrinsicInfo(attribute)}$

Log Calculation:
$log_2 x = \frac{logx}{log2}$

Example (Figure 1):
Salary:
Leaf 1:
$info([2, 4]) = entropy(2/6, 4/6) = -2/6 * log_2 2/6 - 4/6 * log_2 4/6 = 0,918bit$
Leaf 2:
$info([3, 1]) = entropy(3/4, 1/4) = -3/4 * log_2 3/4 - 0,25 * log_2 0,25 = 0,811bit$
Average Information
$info([2, 4], [3, 1]) = \frac{6}{10} * 0,918 + \frac{4}{10} * 0,811 = 0,875bit$

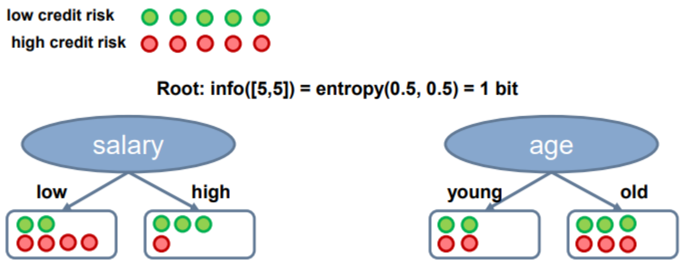

Figure 1: Decision Tree example

Information Gain:
$gain(salary) = info([5,5]) - info([2,4],[3,1]) = 1bit - 0.875bit = 0.125bit$
Gain Ratio:
$intrinsicInfo(salary) = info([6,4]) = 0,9709$
$gainRatio(salary) = 0.125/0,9709 = 0,128$

4.2 Construct a tree:

1. Calculate the information gain / gain ratio for the remaining attributes
2. Choose attribute with **highest** information gain
3. Remove attribute from list of attributes
4. Repeat if attributes are left

4.3 Pruning

- Shortening, simplifying, optimizing
- avoid overfitting

Types of pruning:

- Prepruning (during construction of the tree)
  - Abort construction before the tree becomes to complex
  - Hard to decide because of the number of possible combinations
- Postpruning
  - Construct tree and prune afterwards
  - "Waste of computing time"

4.3.1 Subtree replacement

Replace a subtree with a leaf node.
→ Decreases accuracy on the training set
→ May increase accuracy on the test set

Criterion for replacement:

- error rate of a node and error rate of a leaf is estimated
- Replace if:
  - $e_{Node} < e_{Leaf}$

Estimating the error rate:

- Estimate error based on the training set → Bad Choice!
- Withhold part of the training set and use it as test set
- **Method of C4.5**
  - Pessimistic estimation of error rate e
  - Based on observed error rate f=E/N → E = error and N = instances
  - Confidence level: c (e.g. 25%) is called confidence limit
  - Formula → Not important! Table is given

Example (Figure 2):
Leaf 1:
$N = 2 + 4 = 6, E = 2, f = 1/3$
Look in table at row 6 (N) and column 2 (E) → $e = 0,474$
Leaf 2:
$N = 1 + 1 = 2, E = 1, f = 1/3$
Look in table at row 2 (N) and column 1 (E) → $e = 0,719$
Leaf 3:
$N = 2 + 4 = 6, E = 2, f = 1/3$
Look in table at row 6 (N) and column 2 (E) → $e = 0,474$
Calculate e for all leafs together:
$e = \frac{6}{14} * 0,474 + \frac{2}{14} * 0,719 + \frac{6}{14} * 0,474 = 0.51$
$\frac{6}{14}$ → Sum of outcomes of one leaf (leaf 1 - 4+2=6) divided by the sum of outcomes of the node (health plan contribution - 6+2+6=14)
Calculate e of the node:
Sum all positive/negative entries: [2,4] + [1,1] + [2,4] = [5,9]
$N = 5 + 9 = 14, E = 5, f = 5/14$
Look in table at row 14 (N) and column 5 (E) → $e = 0,449$
Replace if necessary:
$e_{Node} < e_{Leaf}$ → Replace Node with one leaf! See Figure 3

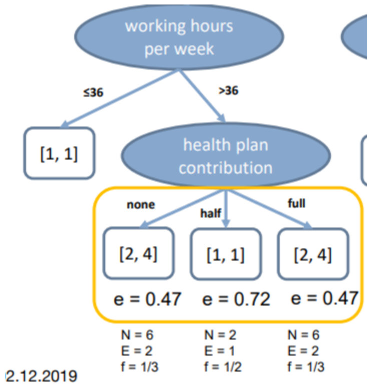

Figure 2: DT before pruning

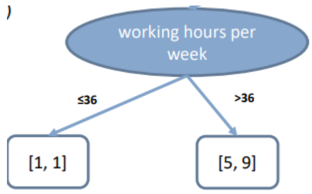

Figure 3: DT after pruning

5 Data preparation

Important R code:

- *mutate()* → adds new variables and preserves existing ones. New variables with the same name overwrite existing ones.

```
CPS1988 <- CPS1988 %>% mutate(wage = wage*2)
```

- *select()* → Is used to select certain columns from a DF. Can be used to rename:

```
df <- df %>%select(newName=column1)
```

- *factor()* → Encode vector as a vector. Gives a number to nominal values like a enumeration.

```
#Gives the numbers 2,3,4 in salutation the names
   Company, Mr and Mrs. and saves column as fct.
salutation = factor(salutation, labels = c("Company",
    "Mr.", "Mrs."))
```

- *toupper()* → Makes all characters to uppercase.
- *mutate_at* → Used to apply multiple functions on selected variables.

```
df <- df %>% mutate_at(vars(order_date, delivery_date
  , date_of_birth), as_date)
```

- *mutate_all* → same as mutate_at but for all variables.
- *as_date()* → transfers data to date format

- *is.na()* → checks if a value is NA.
- *na.omit* → removes all entries which contain NA values.

```
df <- df %>% na.omit()
```

- *if_else()* → takes condition, yes, no

```
if_else(is.na(price), mean(price,na.rm = T),price)
```

- *case_when()* → Like switch case in Java

```
delivery_time_discrete = case_when(
is.na(delivery_time) ~ "NA",
delivery_time<=5 ~ "5d",
delivery_time>5 ~ ">5d"
)
```

6 Data Evaluation

6.1 Holdout

Holding out certain values for evaluation, not training:

- Reduce number of training instances
- Composition influence results

Solution:

- More reliable results using stratification
- Some instances may never be used for training/testing

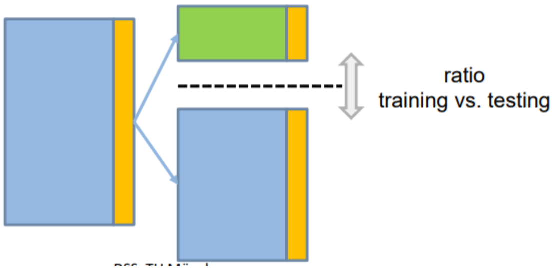

Figure 4: Holdout Example

6.2 k-fold Cross Validation

- Partition the data set into k complementary subsets (make k groups)
- Train with k-1 subsets and test on one subset
- Every subset is used k-1 times for training and 1 time for testing

Example:
Dataset: [+, +, +, +, -, +, +, -, -, -, +, +, +, -, -]
3-fold Cross Validation:

- Divide the sample into 3 different partitions:
  P1:(1,2,3,4,5)
  P2:(6,7,8,9,10)
  P3:(11,12,13,14,15)
- Create k Folds which each use k-1 partitions for training and 1 for testing
  Fold1: Train with P2 and P3 and test with P1 $\rightarrow$ classes[4,1]
  Fold2: Train with P3 and P1 and test with P2 $\rightarrow$ classes[2,3]
  Fold3: Train with P1 and P2 and test with P3 $\rightarrow$ classes[3,2]

Stratified: The splitting of data into folds may be governed by criteria such as ensuring that each fold has the same proportion of observations with a given categorical value, such as the class outcome value. This is called stratified cross-validation.

Example: Find partitions so that Folds result in the same class values: P1 = 1,2,3,5,8, P2 = 4,6,7,9,10, P3 = 11,12,13,14,15
Fold 1: Train: P2 & P3, Test: P1, classes: [3,2]
Fold 2: Train: P1 & P3, Test: P2, classes: [3,2]
Fold 3: Train: P1 & P2, Test: P3, classes: [3,2]

6.2.1 Leave One Out Validation

k-fold validation with k=N $->$ N = Number of training instances

- Deterministic but requires a lot of time for computing
- extreme class distribution of test data

		Actual class	
		Cat	Dog
Predicted class	Cat	**5**	2
	Dog	3	**3**

Figure 5: Example of an Confusion Matrix

7 Confusion Matrix

Used to test an algorithm. Is divided in (see Figure 5):

- True Class
- Predicted Class
- True Positive Rate: $tpr = \frac{TP}{TP+FN}$
  'How many positive instances have been predicted positive?'
- False Positive Rate: $fpr = \frac{FP}{TN+FP}$
  'How many negative instances have been predicted positive?'
- True Negative Rate: $tnr = \frac{TN}{TN+FP}$
  'How many negative instances have been predicted negative?'
- Accuracy $accuracy = \frac{TP+TN}{TP+FP+TN+FN}$
  'How many instances have been predicted correctly?

Example:

True Class	Pedicted Class
0	0
0	1
1	1
1	0
0	0
1	0
0	0
1	1
0	1
1	0

$$tpr = \frac{2}{2+3} = 0,4$$
$$fpr = \frac{2}{3+2} = 0,4$$
$$tnr = \frac{3}{3+2} = 0,6$$
$$accuracy = \frac{2+3}{2+2+3+3} = 0,5$$

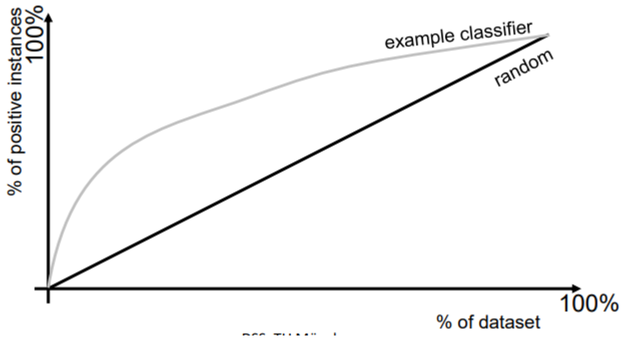

Figure 6: Gain Curve

8 t-Test

Paired t-Test to test if the difference between two classifiers is significant.
$H_0 : d = 0$
$H_1 : d \neq 0$ **(depends on the question!)**

Formulas:
$\overline{d} = \frac{1}{k} \sum_i d_i$
$s_d = \sqrt{\frac{1}{k-1} \sum_i (d_i - \overline{d})^2}$
$t = \frac{\overline{d}}{\frac{s_d}{\sqrt{k}} \; t_{k-1}}$

9 Curves

9.1 Gain Curve

Used to show the difference between different cut-offs. **Instances are sorted descending by the probability!**
x-axis: percentage of the data set (0-100% meaning 10% of the instances in the data set)
y-axis: percentage of number of **positive** instances. Check the percentage of the positive instances **within the cut-off** of the X percent of the x-axis.

9.2 Lift Curve

Displays the factor between the classifier and random value
x-axis: Percentage of the data set
y-axis: $\frac{Gainatx}{x}$

9.3 ROC Curve

Displays ratio of False Positive and True Positive

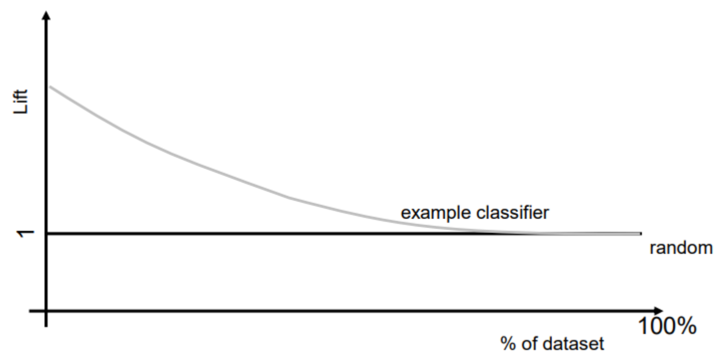

Figure 7: Lift Curve

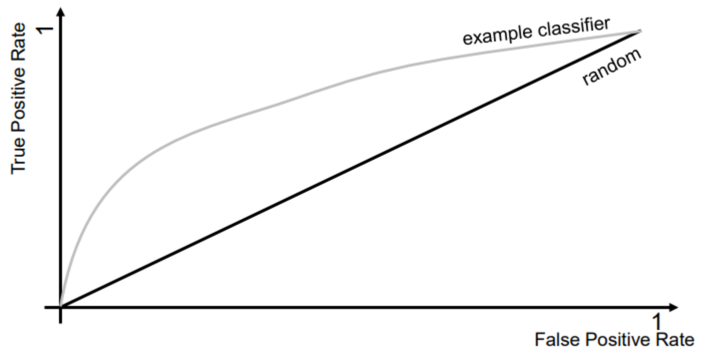

Figure 8: ROC Curve

10 Clustering

Definition:
Given: A p-dimensional data set with n instances.
Want: Partition data set into a number of clusters.

- Items in the same cluster are identical: Intra-cluster similarity is maximized
- Items from different cluster are different: Inter-cluster similarity is minimized

Difference between Classification and Clustering:

Classification:

- Supervised learning
- Target is known
- Training data
- Naive Bayes
- Decision Tree
- Ensemble Methods

Clustering:

- Unsupervised learning
- Target is unknown
- No labels meaning no true classes
- k-means
- Minimum spanning tree
- Expectation maximization

10.1 k-means

Divide instances into k clusters $\text{C}_1, ..., C_k$

- Randomly choose k centers or pick k instances as initial centers
- Repeat until no changes:
  - Assign instances to the closes cluster
    $d(p,c) = \sqrt{(x(p) - x(c))^2 + (y(p) - y(c))^2}$
  - Recalculate the center of the clusters
    $x'(c_i) = \frac{\sum_{p \in C_i} x(p)}{|c_i|}$
    $y'(c_i) = \frac{\sum_{p \in C_i} y(p)}{|c_i|}$

10.2 Expectation Maximation (EM) - Fuzzy Clustering

Currently: Each item of the dataset is assigned to one cluster but sometimes a fuzzy or more flexible cluster can be more realistic!
Method:

- Start with guessing for cluster centers and define k
  - calculate the probability that instance p belongs to cluster c
    $f(x, \mu_A, \sigma_A) = \frac{1}{\sigma_A \cdot \sqrt{2\pi}} \cdot e^{-\frac{(x-\mu_A)^2}{2 \cdot \sigma_A^2}}$
    $Pr[X] = f(x, \mu_A, \sigma_A) \cdot p_A + f(x, \mu_B, \sigma_B) \cdot p_B$
    $Pr[A|x] =$
  - optimize distribution parameters based on the likelihoods
    $\mu_A = \frac{w_1 + .. + w_n}{w_1 + .. + w_n}$