JEM221 Data Science with R I
Week #7-#10

Choosing, evaluating, and validating models
&
Memorization methods

Ladislav Krištoufek

Outline

Outline

Outline

Classification problems

- Classification problems usually reduce to assigning (known) labels to an object.
- Good example of the *supervised learning* – we need a dataset (training set) that has already been classified.
- Building such dataset is quite often a burdensome but essential part of the problem.

Classification problems – approaches

- Naive Bayes
  - Especially useful for problems with many input variables, categorical input variables with a very large number of possible values, and text classification.
- Decision trees
  - Useful when input variables interact with the output in *if-then* manner, or when inputs have many *AND* relationships, or when they are correlated. In addition, these are usually quite easy to understand for nontechnical users.
- Logistic regression
  - Appropriate when you want to estimate class probability (which is a scoring problem) in addition to class assignment. Useful for interpretational purposes as well.
- Support vector machines
  - Useful when there are many input variables, or when input variables interact with the outcome or with each other in complicated ways.

Scoring problems

- For economists/econometricians, these are regression problems.
- Returning probabilities or marginal effects.
- Classical examples in the machine learning: linear and logistic regression.

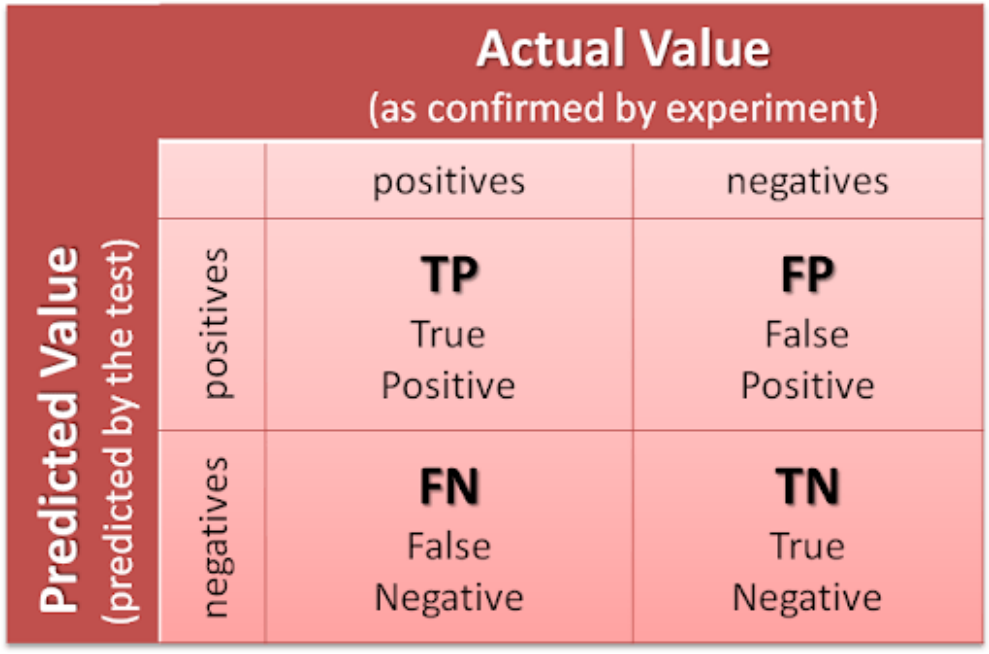
Working without known targets

- Sometimes, there is not a specific outcome that you want to predict.
- Instead, you are looking for patterns and relationships in the data.
- These types of problems lead to the *unsupervised learning* methods, e.g. (covered in this course):
  - k-means clustering
  - finding association rules
  - nearest neighbor methods

Outline

Evaluating classification models

Confusion matrix: a table that summarizes the classifier's prediction against the true data categories.



Evaluating classification models

- *Accuracy*: the number of items categorized correctly divided by the total number of items, i.e. *(TP+TN)/(TP+FP+TN+FN)*.
- *Precision*: fraction of items the classifier flags as being in the class that actually are in the class, i.e. *TP/(TP+FP)*.
- *Recall*: what fraction of the things that are in the class are detected by the classifier, i.e. *TP/(TP+FN)*
- *F1*: combines the precision and recall measures, specifically *F1=2*precision*recall/(precision+recall)*. The idea is that a classifier that improves precision or recall by sacrificing a lot of the complementary measure will have a lower F1.
- *Sensitivity*: a.k.a. the true positive rate a.k.a. recall.
- *Specificity*: a.k.a the true negative rate, i.e. *TN/(TN+FP)*.

Evaluating classification models

		True condition				
	Total population	Condition positive	Condition negative	Prevalence = $\frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$	
Predicted condition	Predicted condition positive	**True positive**	**False positive**, Type I error	Positive predictive value (PPV), Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$	
	Predicted condition negative	**False negative**, Type II error	**True negative**	False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$	
		True positive rate (TPR), Recall, Sensitivity, probability of detection, Power = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) = $\frac{\text{LR+}}{\text{LR−}}$	F_1 score = $2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$
		False negative rate (FNR), Miss rate = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) = $\frac{\text{FNR}}{\text{TNR}}$		

Evaluating scoring models

- *RMSE*: root mean squared error
- R^2: coefficient of determination
- *correlation*: Pearson, Spearman, Kendall
- *MAE, RAE*: mean absolute error and relative absolute error
- You know these from statistics and econometrics.

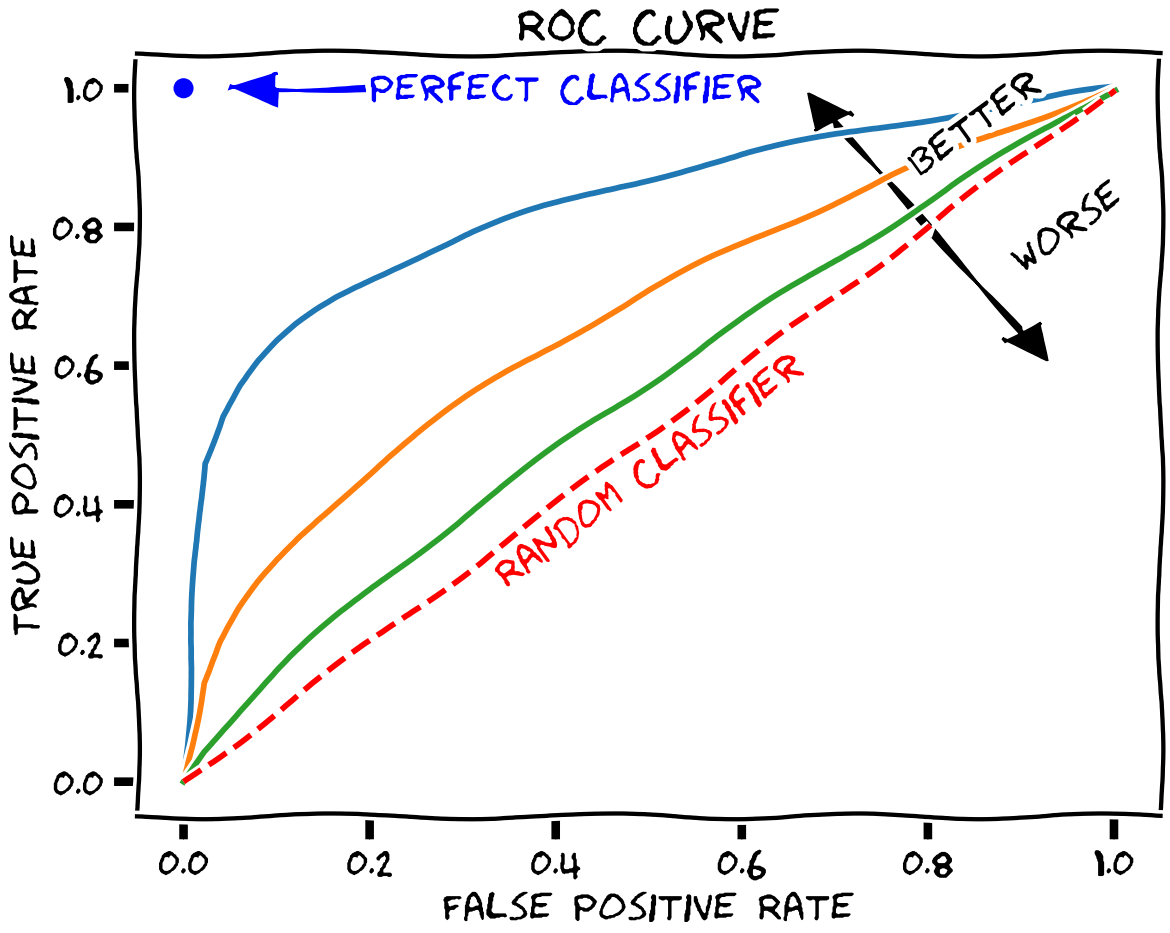
Evaluating probability models

- *ROC/AUC*: receiver operating characteristic curve (/area under curve)
- *LL*: log-likelihood
- *Pseudo-R^2*: McFadden's R^2
- *dev*: deviance
- *IC*: information criteria

Receiver Operating Curve (ROC)

- For each threshold, we plot the false positive rate – $p(\hat{y}_j = 1 | y_j \neq 1)$ – against the true positive rate – $p(\hat{y}_j = 1 | y_j = 1)$.
- ROC is thus threshold independent as it contains all of them.
- As a measure of quality, the Area Under the *receiver operating* Curve (AUC) is used. This is simply an integral of the ROC.
- The maximum value is 1 and for a flip-coin model, we have $AUC = 0.5$.
- Available in the *ROCR* package under the *performance()* function.

Receiver Operating Curve (ROC)



Log likelihood (LL)

- Logarithm of the product of the probability the model assigned to each alternative (here V is the number of possible outcomes):

$$\log L = \sum_{i=1}^{V} (y_i \log(\hat{\pi}(y_i = 1|x_i)) + (1 - y_i) \log(1 - \hat{\pi}(y_i = 1|x_i)))$$

- Logarithm of the likelihood is usually used as it is more convenient to work with than the likelihood itself (products vs. sums).
- (Log) Likelihood is maximized.
- Using the *ifelse()* function.

Goodness-of-fit

- In probabilistic models, the idea of coefficient of determination R^2, i.e. comparing our model with a constant model, can be translated using two likelihoods:
  - $\log L_1$: log-likelihood function of our model
  - $\log L_0$: log-likelihood function of a base model with a constant probability
- The most popular goodness-of-fit measure is:

$$\text{McFadden} - R^2 = 1 - \log L_1 / \log L_0$$

- Is this necessarily between 0 and 1? Can we do worse than the average probability model?

Deviance

- Deviance of the model is defined as $-2 * LL$ and it is analogous to the residual sum of squares (SSR).
- $-2 * LL$ is also referred to as the *model deviance* (how much variation is left unexplained by the model).
- The benchmark *null deviance* is defined as the deviance of the constant model, i.e. the same probability for each observation (how much variation there is to explain).
- Using these, we can define $pseudo - R^2$ as

$$pseudo - R^2 = 1 - \frac{\text{model deviance}}{\text{null deviance}}.$$

Information criteria

- Information criteria extend deviance $-2 * LL$ by penalizing for model complexity.
- For k being the number of parameters and N number of observations, specific information criteria are defined as:
  - Akaike IC (AIC): $-2 * LL + 2 * k$
  - Bayesian/Schwarz IC (BIC, SBC, SBIC): $-2 * LL + k * \ln(N)$
  - Hannan-Quinn IC (HQC): $-2 * LL + 2 * k * \ln(\ln(N))$
- AIC is the best founded on theoretical basis (as an estimator of Kullback-Leibler divergence and being asymptotically efficient).

Outline

Overfitting and sample splitting

- Overfitting is a common pitfall of data science as we usually work with large datasets (and large number of independent variables).
- The constructed models are usually needed for predictions not only fitting.
- For these purposes, we split the dataset into two (three) parts:
  - training – estimate the model
  - *calibration – pseudo out-of-sample performance; if the difference between the training and calibration samples is too high, we are at risk of overfitting*
  - testing – actual out of sample performance

Cross-validation

- How much is the model performance dependent on separation between the training set and the calibration set?
- Repeat the procedure many times to see how the model fares, i.e. how stable the performance of the model is.

Outline

Memorization methods

- Methods that generate answers by returning a majority category (in the case of classification) or average value (in the case of scoring).
- Vary from single variable models, to decision trees, nearest neighbor and Naive Bayes methods.
- These can be seen as prequels to (but sometimes valid building blocks towards) the regression analysis.

Outline

Dataset

- We use the KDD Cup 2009 dataset.
- KDD is the Conference on Knowledge Discovery and Data Mining.
- Every year, the conference hosts the KDD Cup.
- Our dataset contains 250 independent variables about 50,000 credit card accounts.
- Three variables of interest: *churn* (account cancelation), *appetency* (tendency to use new products and services), and *upselling* (willingness to respond favorably to marketing pitches)
- Independent variables as well as credit card account owners are undisclosed.
- Aim: Find an independent variable (for single-variable models) which best predicts a selected dependent variable.

Using categorical features

- Simply inspect the levels of the dependent variable with respect to a specific independent variable.
  - Usually done in a table, using the *table()* command.
- Practically, this is a representation of probability distribution functions (categorical values are discrete).
- More formally:
  - Let's have a dependent variable Y and we are interested in the case $y = 1$.
  - We have a finite number of levels/values of an independent variable $X - x_i$, $i = 1, \ldots, k$.
  - For each level i, we find conditional sample mean $\hat{\pi}_i = \frac{1}{N} \sum_{j=1}^{N} \mathbb{I}_{y_j=1|X_j=x_i}$, where N is the sample size and $\mathbb{I}_\bullet$ is the indicator function equal to 1 if the condition in $\bullet$ is met, and 0 otherwise.
  - If the occurrence of NAs in the dataset is high, it usually pays off to include it as a separate level 0. Then it holds that $\sum_{i=0}^{k} \hat{\pi}_i = 1$.
  - As there is a chance that a new level $k + 1$ emerges in the testing dataset, we also find the unconditional mean $\hat{\pi} = \frac{1}{N} \sum_{j=1}^{N} \mathbb{I}_{y_j=1}$.

Predictions with categorical features

- For specific levels x_i (possibly including x_0 of NAs), conditional means $\hat{\pi}_i$ serve as predictions.
- For new levels x_i with $i > k$, unconditional mean $\hat{\pi}$ is used as a prediction.
- As we are given probabilities as predictions (and not actual ones or zeros, or other), some threshold needs to be set.

What should our function(s) do?

- For a specified character variable, detect its possible values.
- Find conditional counts and probabilities of these values linked to our dependent variable of interest.
- Find unconditional probability of the dependent variable outcome for the case of new independent variable values in the calibration/testing sample.
- These probabilities are then used as predictions.

Using numeric features

- Parallel to the previous approach, numeric data can be transformed into bins.
- These bins then form the conditioning levels.
- Training and calibration data are usually much closer than in the categorical features category (as the overfitting is controlled by binning).

What should our function(s) do?

- For a specified numeric variable, split the values into meaningful bins (ranges).
- These bins (ranges) can be used as “character” values, i.e. we input these into the function for character variables.

Outline

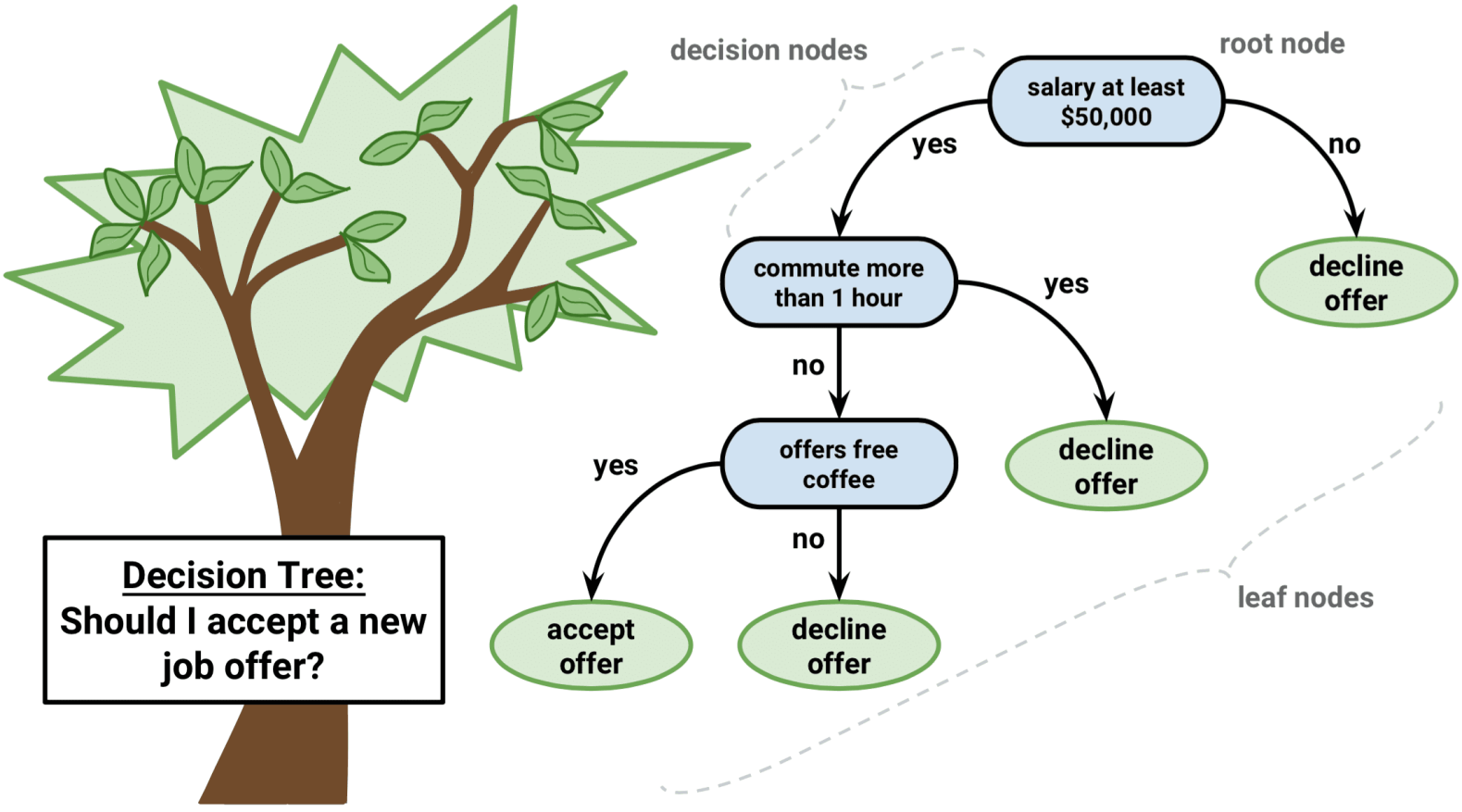
Variable (pre-)selection

- Adding variables in general improves the model performance (in the training set, i.e. in-sample).
- Adding variables also increases the risk of overfitting.
- Therefore, we usually don't want to use all available variables but only the "worthy" ones.
- To do so, we need to choose a selection criterion and a threshold.
  - We already know likelihood, goodness-of-fit measures, and various information criteria.
  - In large datasets, it is usually enough to keep the variables which perform at least as well as the coin flip.

Decision trees

- Decision trees can be seen as procedures to split the training data into pieces and use a simple memorized constant (probability) on each piece.
- The tree is then a set of branches which represent conditions to be met.
- In the *rpart* library, there is the *rpart()* function that constructs the decision tree for a given sample.
- Various parameters can be specified via the *rpart.control()* function.

Decision tree example



Nearest neighbor methods

- Predictions based on the *k-nearest neighbor (KNN)* method are simply averages of the outcomes of the k nearest neighbors.
- The nearness is usually based on the basic Euclidean distance:

$$d(p, q) = \sqrt{\sum_{i=1}^{N} (q_i - p_i)^2}$$

- The number of neighbors must be chosen with care, i.e. in the case of the churn rate with average probability of around 0.07, it does not make much sense to have only 4 neighbors.
- The number of neighbors is always arbitrary but standardly, it is suggested to use approximately $dh/\bar{\pi}$, where dh is the desired hits number in the neighborhood and $\bar{\pi}$ is the average probability of a hit (success).
- There is the *knn()* command in the *class* library which performs the KNN for us (well, almost).

Naive Bayes (NB)

- NB memorizes how each training variable is related to outcome, and then makes predictions by multiplying together the effects of each variable.
- NB reverses the standard causal logic and asks “If y is true, what is the probability of x having level x_i?” Such question can be answered from the data.

Naive Bayes – procedure

- Let's call a specific variable X_1 take a specific value x_1 as a piece of evidence ev_1.
- From this, we can get a conditional probability $\pi(ev_1|y = 1)$. We can also easily get unconditional probability $\pi(y = 1)$.
- Using Bayes' law, we get:

$$\pi(y = 1|ev_1) = \frac{\pi(y = 1) * \pi(ev_1|y = 1)}{\pi(ev_1)}$$

$$\pi(y = 0|ev_1) = \frac{\pi(y = 0) * \pi(ev_1|y = 0)}{\pi(ev_1)}$$

- For a single evidence, there is no advantage of taking this discourse as the left-hand side probability can be estimated from the data easily. However, this cannot be done for multiple evidences, i.e. multiple variables.

Naive Bayes – procedure

- Assuming that all the evidence (having E pieces of evidence) is conditionally independent of each other, we get:

$$\pi(y = 1|ev_1 \& \ldots \& ev_E) = \frac{\pi(y = 1) * \pi(ev_1|y = 1) * \ldots * \pi(ev_E|y = 1)}{\pi(ev_1 \& \ldots \& ev_E)}$$

$$\pi(y = 0|ev_1 \& \ldots \& ev_E) = \frac{\pi(y = 0) * \pi(ev_1|y = 0) * \ldots * \pi(ev_E|y = 0)}{\pi(ev_1 \& \ldots \& ev_E)}$$

- The numerator terms can be calculated from the data, the denominator ones cannot. However, we know that

$$\pi(y = 1|ev_1 \& \ldots \& ev_E) + \pi(y = 0|ev_1 \& \ldots \& ev_E) = 1$$

so that the calculated probabilities can be rescaled to sum to one, i.e. the denominators are not needed here.

Naive Bayes – procedure

- The problem is usually transformed into logarithms as working with the sums is more efficient than with products, i.e. we work with a sum of logarithms of probabilities, the left-hand side of the equation is referred to as the score:

$$score(y = 1|ev_1 \& \ldots \& ev_E) = \log(\pi(y = 1)) + \sum_{i=1}^{E} \log(\pi(ev_i|y = 1))$$

$$score(y = 0|ev_1 \& \ldots \& ev_E) = \log(\pi(y = 0)) + \sum_{i=1}^{E} \log(\pi(ev_i|y = 0))$$

- Apparently, zero probabilities will cause problems so that a smoothing operator (a tiny constant) is added to each (unconditional and conditional) probability.
- From the definition of conditional probability $P(A|B) = \frac{P(A \bigcap B)}{P(B)}$, we can find $\sum_{i=1}^{E} \log(\pi(ev_i|y = 1))$ as $\sum_{i=1}^{E} \log \frac{\pi(y=1 \& e_i)}{\pi(y=1)}$, and in the same way for the failure probability.

Next (final) block

- Linear and logistic regression
- Local polynomial regression
- LASSO (least absolute shrinkage and selection operator) algorithm
- GAMs (generalized additive models)