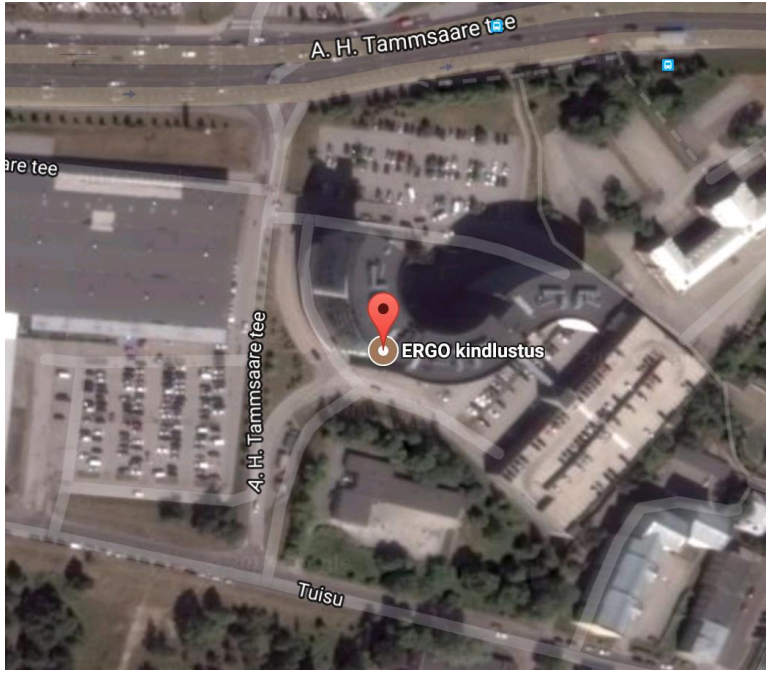
Machine Learning

(a perspective influenced by actuarial and tech worlds)

Vlad Belous

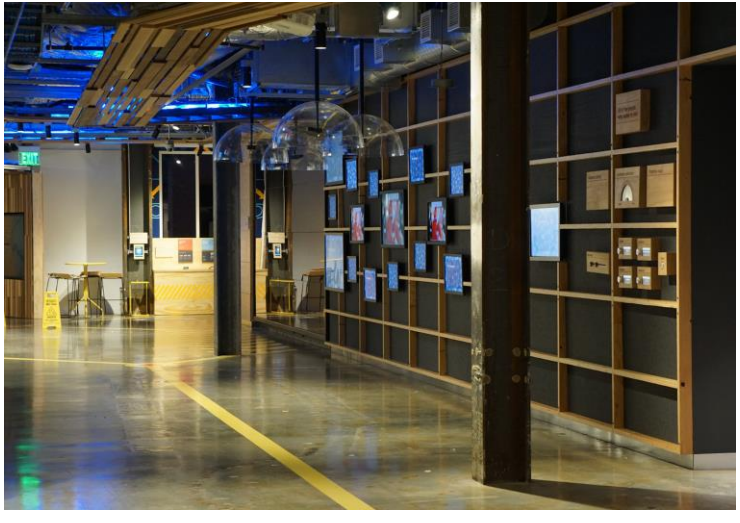
Baltic Actuarial Summer Days, 2017

Background

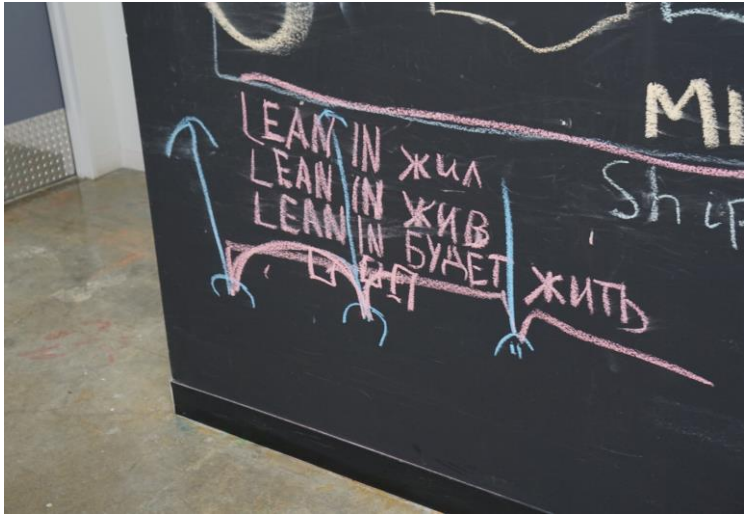

A. H. Tammsaare tee
ERGO kindlustus
A. H. Tammsaare tee
Tuisu

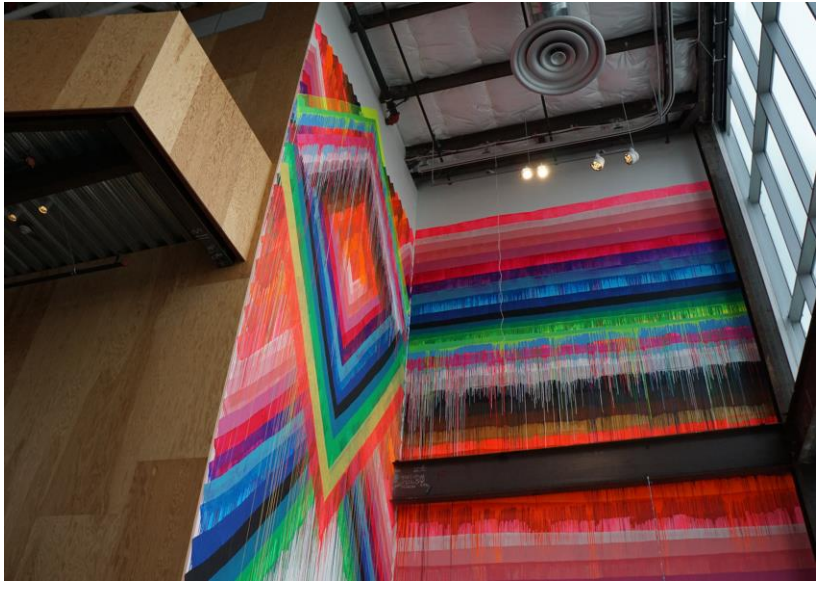

Ravenswood Slough
Facebook HQ
Bayfront Expy
Facebook Building 20
Willow Rd
University Ave
84

Pictures



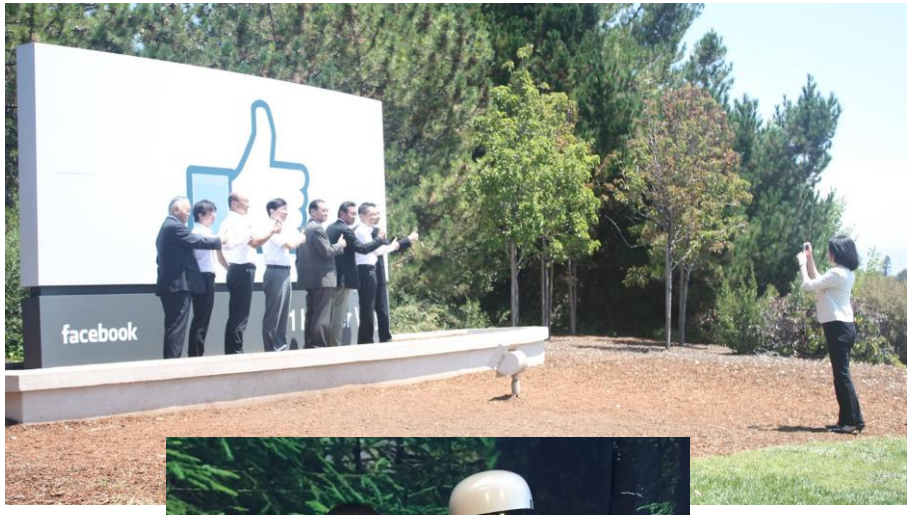
Pictures


LEAN IN
LEAN IN
LEAN IN
жил
жив
будет
жить




facebook
FEEDBACK IS A GIFT
BIG BETS
BRAIN

Pictures



Introduction

From Wikipedia:

> “Machine learning [...] gives computers the ability to learn without being explicitly programmed.”

Introduction

From Wikipedia:

> “Machine learning [...] gives computers the ability to learn without being explicitly programmed.”

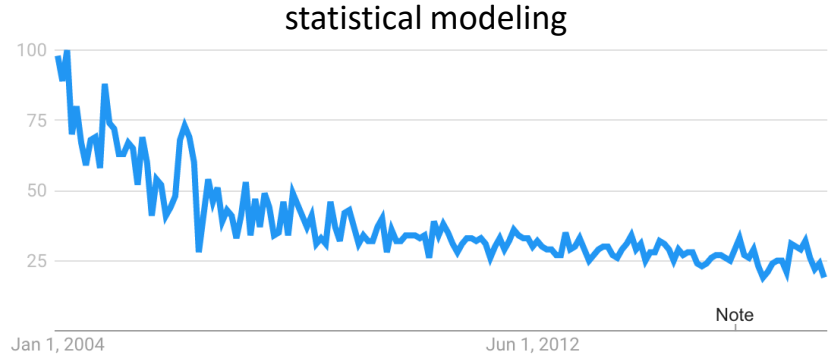
> “As of 2016, machine learning is a buzzword, and [… is] at its peak of inflated expectations.”

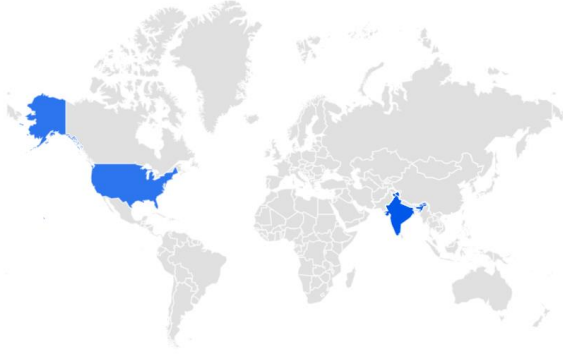
Inflation of terms

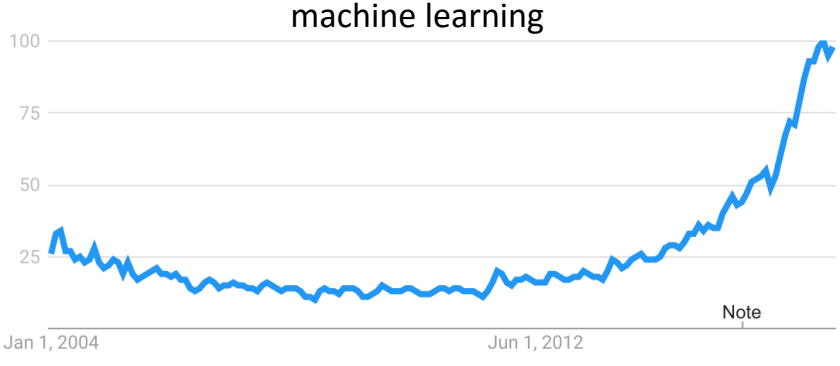
Statistical Modeling → Machine Learning → Artificial Intelligence

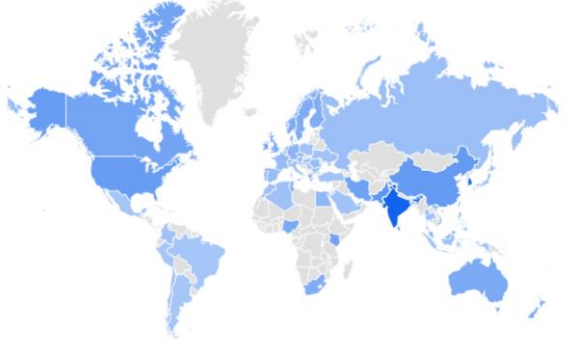
Artificial Intelligence → Artificial General Intelligence

Google trends



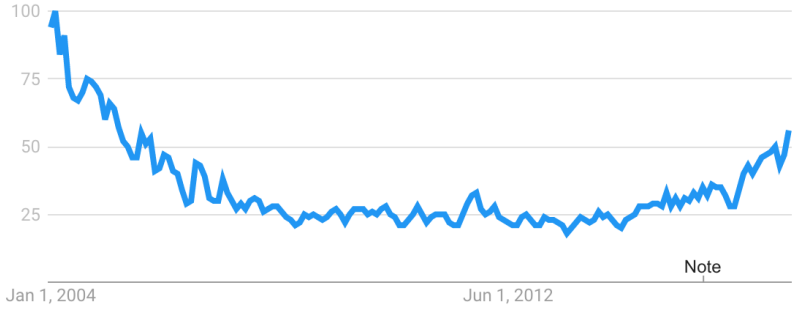


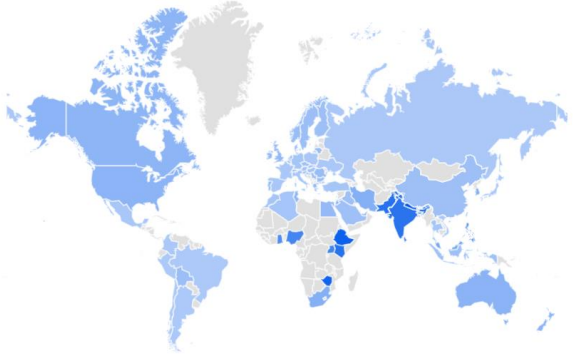




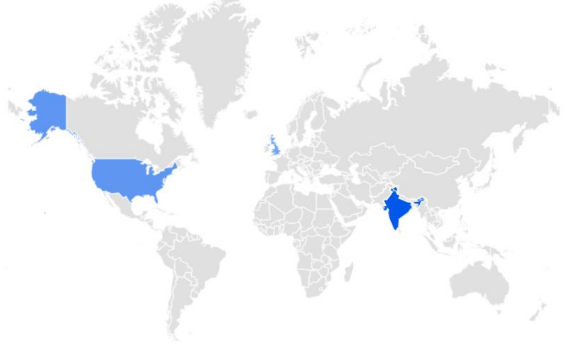
Google trends

artificial intelligence





artificial general intelligence



(Supervised) Machine Learning

label (usually a scalar)

Given a dataset of *labeled* samples $D = \{ (\mathbf{x}^{(1)}, y^{(1)}), \ldots (\mathbf{x}^{(n)}, y^{(n)}) \}, \ldots$

≈ we know the correct answer;
this is called *ground-truth*

each $\mathbf{x}^{(i)}$ is a vector – called a *feature vector*

(Supervised) Machine Learning

label (usually a scalar)

Given a dataset of *labeled* samples $D = \{ (\mathbf{x}^{(1)}, y^{(1)}), \ldots (\mathbf{x}^{(n)}, y^{(n)}) \}, \ldots$

≈ we know the correct answer;
this is called *ground-truth*

each $\mathbf{x}^{(i)}$ is a vector – called a *feature vector*

… find a function F, such that $F(\mathbf{x}) = \hat{\hat{y}}$ is "close" to the correct label y, …

the *model*

model *output* for $\mathbf{x}$
(≈prediction)

according to some pre-defined error measure,
often called *loss function*

(Supervised) Machine Learning

label (usually a scalar)

Given a dataset of *labeled* samples $D = \{ (\mathbf{x}^{(1)}, y^{(1)}), \ldots (\mathbf{x}^{(n)}, y^{(n)}) \}, \ldots$

≈ we know the correct answer; this is called *ground-truth*

each $\mathbf{x}^{(i)}$ is a vector – called a *feature vector*

… find a function F, such that $F(\mathbf{x}) = \hat{y}$ is "close" to the correct label y, …

the *model*

model *output* for $\mathbf{x}$ (≈prediction)

according to some pre-defined error measure, often called *loss function*

… even on previously unseen data.

this is the critical part – our model must *generalize* in order to be useful!

(Supervised) Machine Learning

Examples:

$\hat{\hat{y}} = F(\mathbf{x}) = \boldsymbol{\beta}^T\mathbf{x} = \beta_0 + \beta_1 x_1 + \ldots \beta_k x_k$

for convenience, we set $x_0 = 1$

$\text{Loss}^{(i)} = [y^{(i)} - \hat{\hat{y}}^{(i)}]^2$

we then sum over all *i* (i.e. all samples)

This is the most basic linear model.
Least squares regression.

(Supervised) Machine Learning

Examples:

$\hat{\hat{y}} = F(\mathbf{x}) = \boldsymbol{\beta}^T\mathbf{x} = \beta_0 + \beta_1 x_1 + \ldots \beta_k x_k$

for convenience, we set $x_0 = 1$

$\text{Loss}^{(i)} = [y^{(i)} - \hat{\hat{y}}^{(i)}]^2$

we then sum over all *i* (i.e. all samples)

This is the most basic linear model.
Least squares regression.

$\hat{\hat{y}} = F(\mathbf{x}) = 1 / (1 + \exp[-\boldsymbol{\beta}^T\mathbf{x}])$

$$\text{Loss}^{(i)} = \begin{cases} -\log(\hat{\hat{y}}^{(i)}) & \text{if } y^{(i)}=1 \\ -\log(1 - \hat{\hat{y}}^{(i)}) & \text{if } y^{(i)}=0 \end{cases}$$

$\hat{\hat{y}}$ is meant to be interpreted as Pr[y=1]

The well known logistic regression.

In both cases the process of learning means minimizing $\sum_i \text{Loss}^{(i)}$ with respect to $\boldsymbol{\beta}$.
(note that $\text{Loss}^{(i)}$ depends on $\hat{\hat{y}}$, and $\hat{\hat{y}}$ depends on $\boldsymbol{\beta}$)

(Supervised) Machine Learning

Some common ML techniques for supervised learning:

- Linear models and generalized linear models.
  - e.g. logistic regression is a type of GLM
- Decision trees – often combined with boosting or bagging.
  - e.g. gradient-boosted decision trees.
  - random forests
- Artificial neural networks (NNs).
- Support vector machines (SVMs).
  (seems to be popular in literature, but I've never seen used in practice…)

ML tasks in insurance

• Estimating probability of future claim on a MTPL (motor liability) policy, given parameters of policy, vehicle, policyholder, etc – this is machine learning! ☺

Model type:	logistic regression, Poisson regression
Feature ideas:	driver_age, car_age, horse_power, region, time_since_last_accident, is_bmw_and_young_driver, etc…

coming up with these is called *feature engineering*

ML tasks in insurance

- Estimating probability of future claim on a MTPL (motor liability) policy, given parameters of policy, vehicle, policyholder, etc – this is machine learning! ☺

  Model type: logistic regression, Poisson regression
  Feature ideas: driver_age, car_age, horse_power, region, time_since_last_accident, is_bmw_and_young_driver, etc…

- Retention modeling: estimating probability that client will renew policy.

  Model type: logistic regression
  Feature ideas: <audience actively making suggestions>

ML tasks in insurance

- Estimating probability of future claim on a MTPL (motor liability) policy, given parameters of policy, vehicle, policyholder, etc – this is machine learning! ☺

  Model type: logistic regression, Poisson regression

  Feature ideas: driver_age, car_age, horse_power, region, time_since_last_accident, is_bmw_and_young_driver, etc…

- Retention modeling: estimating probability that client will renew policy.

  Model type: logistic regression

  Feature ideas: price_change, how_long_our_client, age, gender, car_price, previous_accepted_price_change, has_other_policies_with_us, policy_price_at_competitors, acquire_channel, etc…

ML tasks in insurance

- Estimating probability of future claim on a MTPL (motor liability) policy, given parameters of policy, vehicle, policyholder, etc – this is machine learning! ☺

  Model type: logistic regression, Poisson regression
  Feature ideas: driver_age, car_age, horse_power, region, time_since_last_accident, is_bmw_and_young_driver, etc…

- Retention modeling: estimating probability that client will renew policy.

  Model type: logistic regression
  Feature ideas: price_change, how_long_our_client, age, gender, car_price, previous_accepted_price_change, has_other_policies_with_us, policy_price_at_competitors, acquire_channel, etc…

- Other ideas for ML application?

ML tasks in Facebook

Assuming you've used or seen Facebook, where do you think Facebook might apply machine learning?

<audience actively making suggestions>

ML tasks in Facebook

• Relevancy: probability that a user will ‘like’ a story he is shown. More generally, probability that a user will engage with story (like, comment, share, photo view, etc).

Model type: logistic regression, other binary classification type models

Feature ideas: <audience actively making suggestions>

ML tasks in Facebook

• Relevancy: probability that a user will ‘like’ a story he is shown. More generally, probability that a user will engage with story (like, comment, share, photo view, etc).

Model type:	logistic regression, other binary classification type models
Feature ideas:	user_interests, story_keywords, has_photo, photo_has_face, story_likes_total, story_likes_among_friends, story_likes_among_same_age, story_likes_among_same_gender, has_word_congratulations, has_word_congratulations_in_comment, etc..

thousands of features

ML tasks in Facebook

• Relevancy: probability that a user will 'like' a story he is shown. More generally, probability that a user will engage with story (like, comment, share, photo view, etc).

Model type:	logistic regression, other binary classification type models
Feature ideas:	user_interests, story_keywords, has_photo, photo_has_face, story_likes_total, story_likes_among_friends, story_likes_among_same_age, story_likes_among_same_gender, has_word_congratulations, has_word_congratulations_in_comment, etc..

thousands of features

- Probability user will click on an ad.
- How likely image contains offensive content (e.g. nudity) and needs to be reviewed?
- Is account legitimate or fake? Is account secured or hacked?
- Does photo contain a face? (using ML to generate features for the next ML model)
- Likelihood you know a certain person ("People you may know" suggestion)

High-level differences in approach to ML between insurance vs. tech companies

* based on a personal subjective opinion

High-level differences in approach to ML between insurance vs. tech companies

* based on a personal subjective opinion

- Amount of training data.

In case of insurance, the typical case is to have one sample per policy, and you can't have too many of those.

Any web-site visit, any online activity can be generating a sample (B per day).
Photos uploaded (MM per day).
Texts posted (MM per day).

High-level differences in approach to ML between insurance vs. tech companies

* based on a personal subjective opinion

- Amount of training data.
- Interpretability of models.

Insurance seems to favor interpretable models.
Possible reasons:
- client often sees inputs and outputs
- there are market expectations
- need to explain to regulators?

Accuracy is priority over interpretability. The user doesn't "see" inputs/outputs anyway. The model can be a black box.

High-level differences in approach to ML between insurance vs. tech companies

* based on a personal subjective opinion

- Amount of training data.
- Interpretability of models.
- Selection bias problem – the current model has influence on the choice of samples for the next training or evaluation dataset. <How?>

High-level differences in approach to ML between insurance vs. tech companies

* based on a personal subjective opinion

- Amount of training data.
- Interpretability of models.
- Selection bias problem – the current model has influence on the choice of samples for the next training or evaluation dataset.

A concern, but doesn't seem very significant (at least short-term).

Can be a major issue, even short-term. Leads to false conclusion about which model is better. <How?>
Necessitates running true A/B tests.

High-level differences in approach to ML between insurance vs. tech companies

* based on a personal subjective opinion

- Amount of training data.
- Interpretability of models.
- Selection bias problem – the current model has influence on the choice of samples for the next training or evaluation dataset.
- Ability to experiment.

Quite limited? Difficult to justify pricing inconsistencies observed by clients.
Also constrained by contracts?

Widespread. End-user is exposed to the model rather indirectly.
No strong expectations about how things are supposed to work.

Problem with LMs and GLMs

$$\boldsymbol{\beta}^T\mathbf{x} = \beta_0 + \beta_1 x_1 + \ldots \beta_k x_k$$

Input → output relationship isn't necessarily well described as a straight line.

- this model structure is a rather significant constraint, which reduces the "expressiveness" of the model and the type of relationships one can learn. Generally this is a good thing! But not always.

Solution: create *derived* features!

$$x_1 \rightarrow x_1, x_1^2 \rightarrow x'_1, x'_2$$

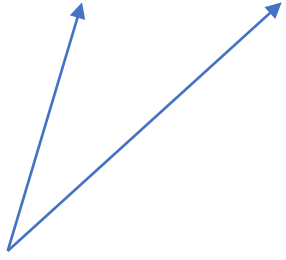
transform rename

We can now use x'_1, x'_2 as "normal" features in a linear model.

Note that we are still within linear model class!

Problem with LMs and GLMs

$$\boldsymbol{\beta}^T\mathbf{x} = \beta_0 + \beta_1 x_1 + \ldots \beta_k x_k$$



The effect of x_1 might change significantly as value of x_2 changes. We want to capture this relationship, but current model structure cannot.

- this model structure is a rather significant constraint, which reduces the "expressiveness" of the model and the type of relationships one can learn.

Solution: create *bigrams*! (often called *interaction terms*)

For simple binary features:

$$x_1 , x_2 \rightarrow x_1, x_2, x_1x_2$$

need to train three β parameters here

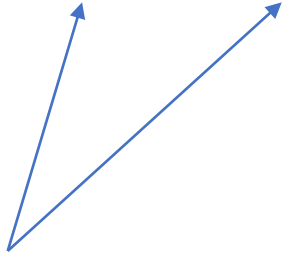
For example:

$$\beta_0 + \beta_1 I_{young} + \beta_2 I_{BMW} + \beta_3 I_{young\_with\_BMW}$$

Still within linear model class!

Problem with LMs and GLMs

$$\boldsymbol{\beta}^T\mathbf{x} = \beta_0 + \beta_1 x_1 + \ldots \beta_k x_k$$



The effect of x_1 might change significantly as value of x_2 changes. We want to capture this relationship, but current model structure cannot.

- this model structure is a rather significant constraint, which reduces the "expressiveness" of the model and the type of relationships one can learn.

Solution: create *bigrams*! (often called *interaction terms*)

For higher cardinality categorical features, you get quite a bit more combinations:

For example:

$$\beta_0 + \beta_1 I_{age=[0\text{-}18],make=BMW} + \beta_2 I_{age=[19\text{-}25],make=BMW} + \ldots$$
$$+ \beta_8 I_{age=[0\text{-}18],make=Audi} + \beta_9 I_{age=[19\text{-}25],make=Audi} + \ldots$$
$$+ \beta_{15} I_{age=[0\text{-}18],make=MB} + \beta_{16} I_{age=[19\text{-}25],make=MB} + \ldots$$

Problem with LMs and GLMs

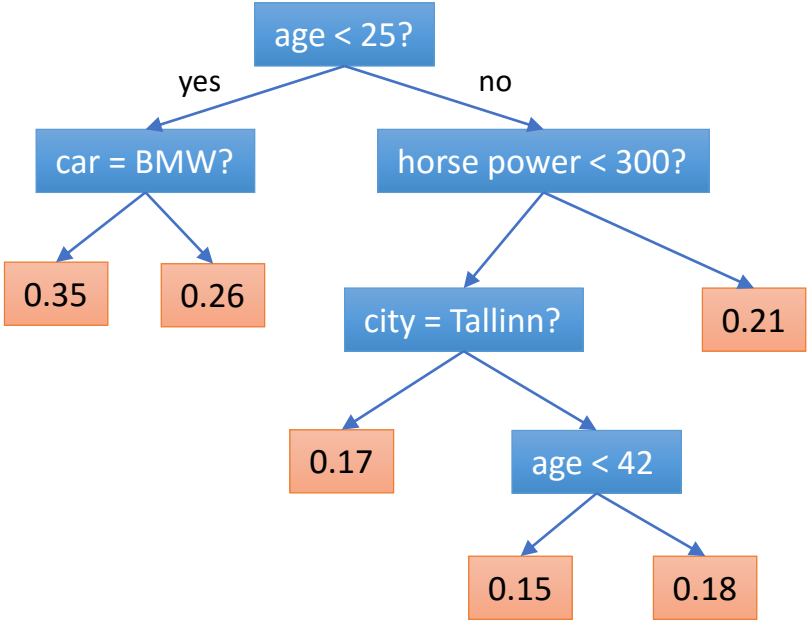
Have we solved the above mentioned problems using derived features and bigrams?

Sort of, but there are significant downsides:

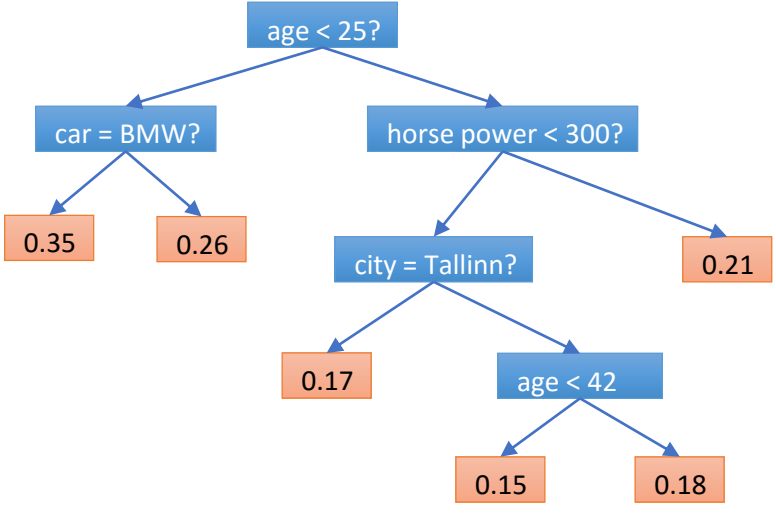
- Number of weights to train grows very quickly if we add derived features and put them all into bigrams! Becomes infeasible quickly, if original number of features or their cardinalities was already large.
- So we should only add *some* bigrams – but this involves extra step of feature selection, possibly manual feature engineering work.
- And still, we haven't considered tri-grams …

Decision trees

Instead of having a model expressed as linear combination of features, we can try a structure that “asks” a series of binary questions about a sample, and gives output in the end.



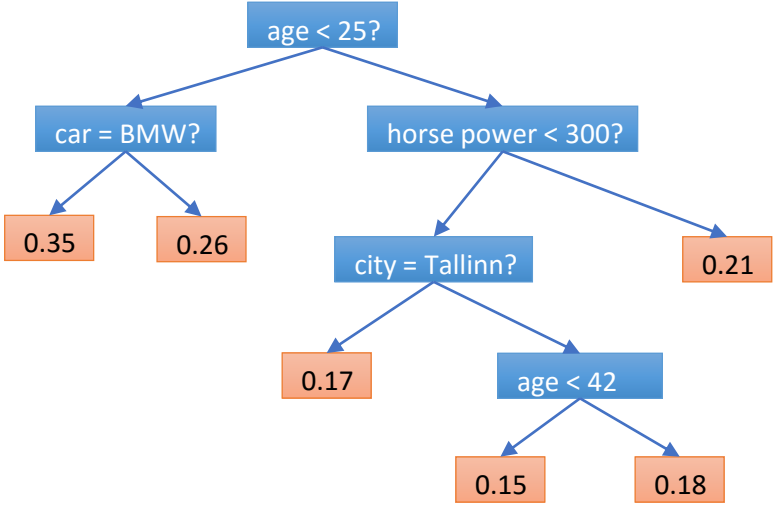
Decision trees



Good:

- Naturally expresses bigrams, trigrams, 4-grams, etc.
- No need normalize (scale) features.
- In principle, no need for manually derived features (the tree can express nonlinearities).

Decision trees



Good:

- Naturally expresses bigrams, trigrams, 4-grams, etc.
- No need normalize (scale) features.
- In principle, no need for manually derived features (the tree can express nonlinearities).

Bad:

- Optimal trees are difficult to construct. Typically use some greedy method. (optimal trees also don't necessarily generalize better)
- Sensitive to noise in the training data.
- Building one big decision tree, as a general method, doesn't produce good results. ?!

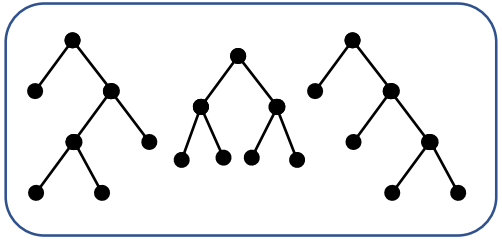
Gradient-boosted decision trees

Decision trees can be a building block in a really powerful algorithm – GBDTs.

Instead of training a single large tree, we train a sequence of smaller trees, where at each iteration, the next tree is given the task of correcting errors of the current forest (set of trees).

The more general concept behind this is *boosting* – a method of turning weak predictors into a strong ones.

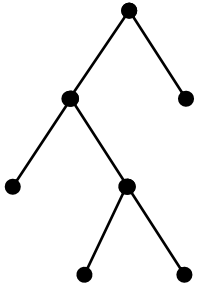
Gradient-boosted decision trees



model from iteration i-1

$$g_j = - dL^{(i-1)}_j / d\hat{\hat{y}}^{(i-1)}_j$$

for each sample j in training dataset, compute the negative "gradient" of the loss wrt prediction output, evaluated at current prediction output.

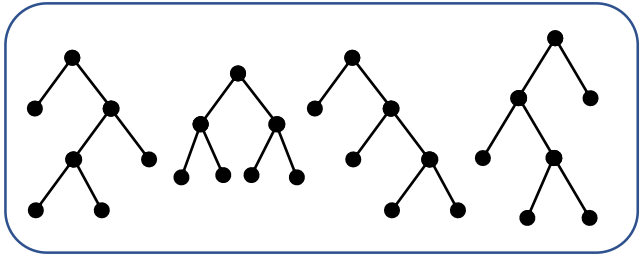


build a **DT**, optimizing for $\mathbf{g}_j$, instead of $\mathbf{y}_j$!

learning rate

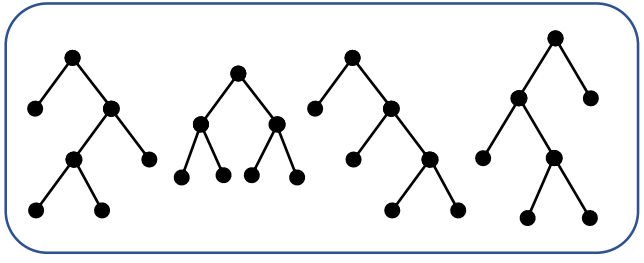
$$\hat{\hat{y}}^{(i)}(\mathbf{x}) = \hat{\hat{y}}^{(i-1)}(\mathbf{x}) + \alpha\, DT(\mathbf{x})$$

new DT contributes to the model additively*.



model from iteration i

Gradient-boosted decision trees



model from iteration i

Number of trees usually around 100 to 1000.

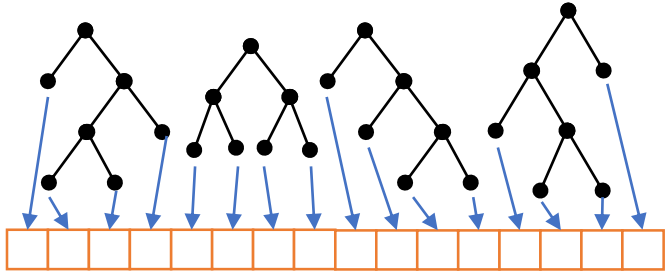
Initialization often done by simply taking the average of all labels, i.e. model at iteration 0 is simply the label average.

Learning rate around 0.05 to 0.20 seems to work fine (might need experimenting).

This description as is works for least squares loss, but changing it, e.g. to logistic loss, is easy – the principle stays the same.

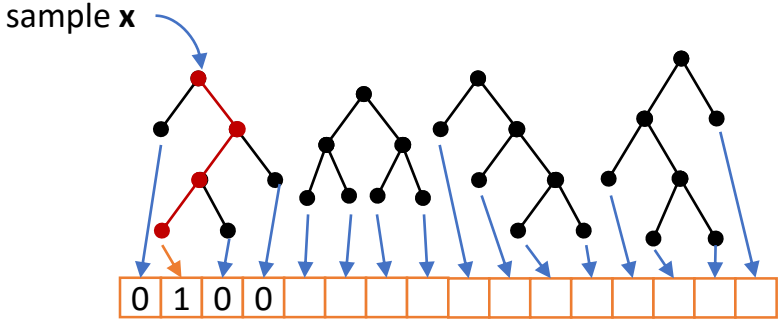
Excellent general purpose predictor/classifier (except, probably, for image recognition)

GBDT + LR



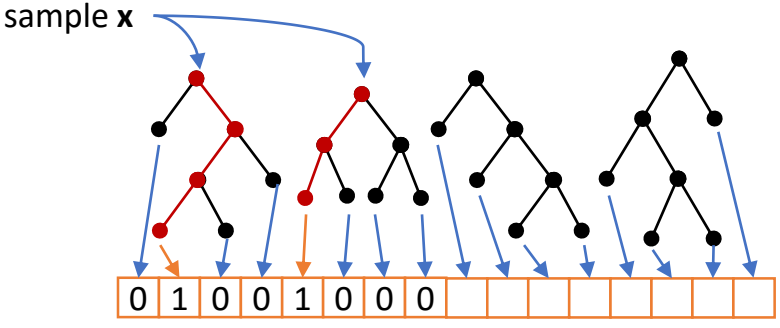
Idea: forget the leaf values found by the tree algorithm, and treat the forest as a *feature transformer*

GBDT + LR



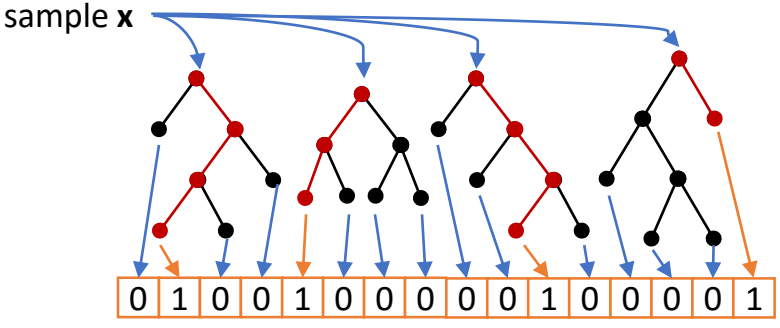
Idea: forget the leaf values found by the tree algorithm, and treat the forest as a *feature transformer*

GBDT + LR



Idea: forget the leaf values found by the tree algorithm, and treat the forest as a *feature transformer*

GBDT + LR

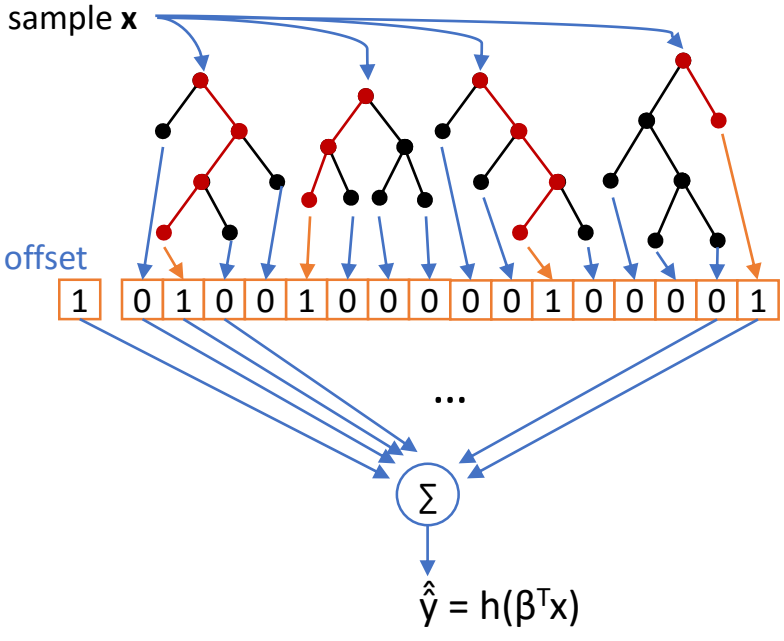


Idea: forget the leaf values found by the tree algorithm, and treat the forest as a *feature transformer*

The output is a binary feature vector of length num_leaves_per_tree * num_trees

What's next? (hint in title)

GBDT + LR

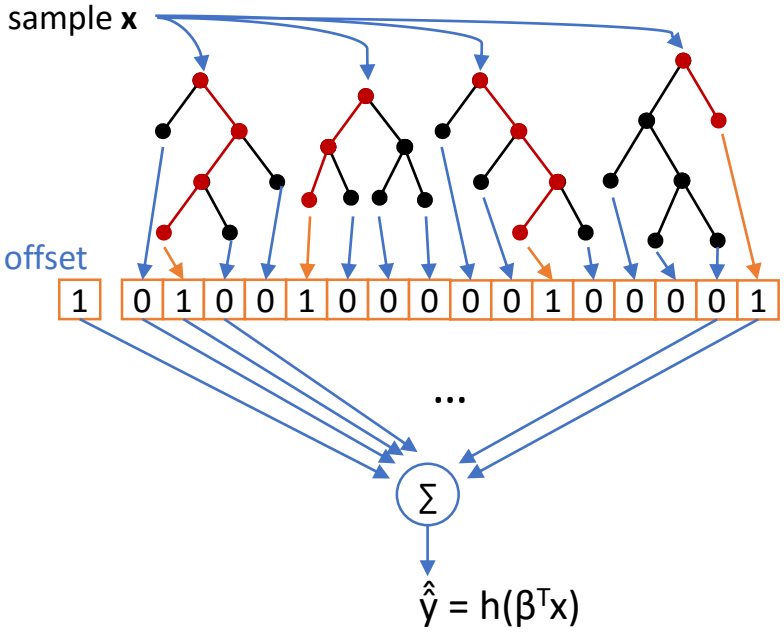


Idea: forget the leaf values found by the tree algorithm, and treat the forest as a *feature transformer*

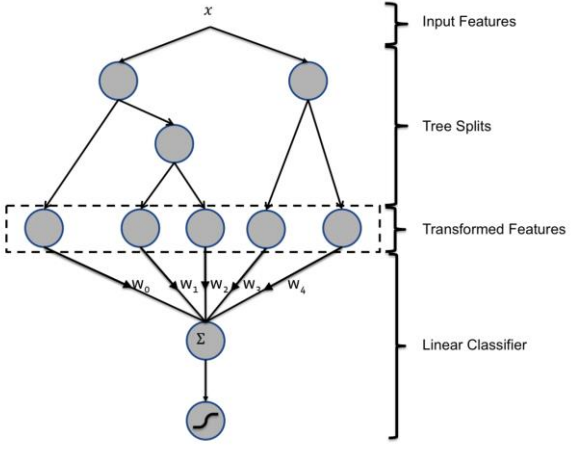
The output is a binary feature vector of length num_leaves_per_tree * num_trees

For example, you could apply a logistic regression model on the transformed features

GBDT + LR

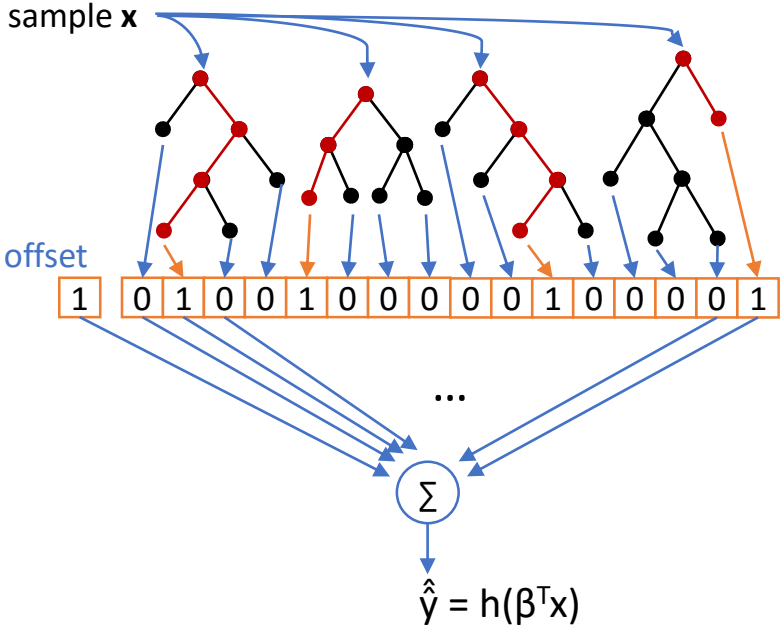


Xinran He, Junfen Pan, Ou Jin, et. al.
"Practical Lessons from Predicting Clicks on Ads at Facebook"



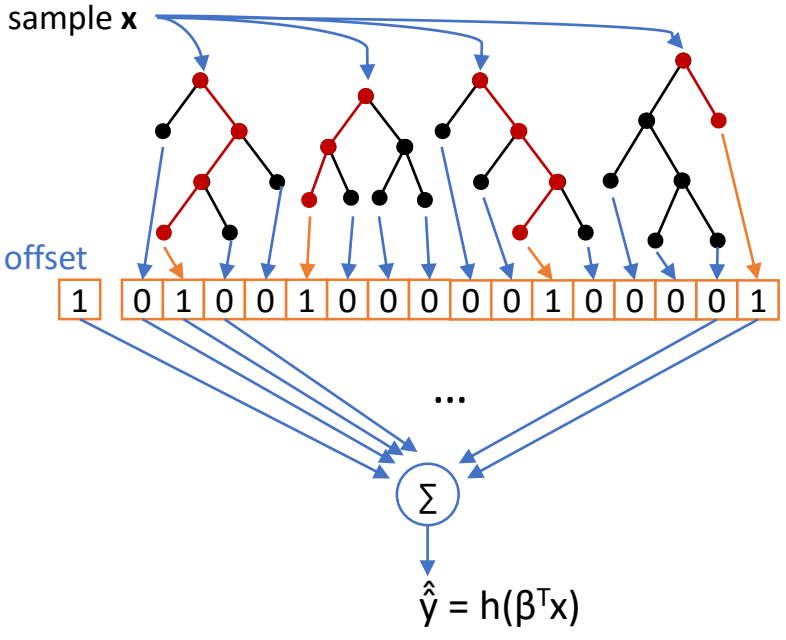
Model Structure	NE (relative to Trees only)
LR + Trees	96.58%
LR only	99.43%
Trees only	100% (reference)

GBDT + LR



Training of these two stages can be done on different datasets. Why is that useful?

GBDT + LR

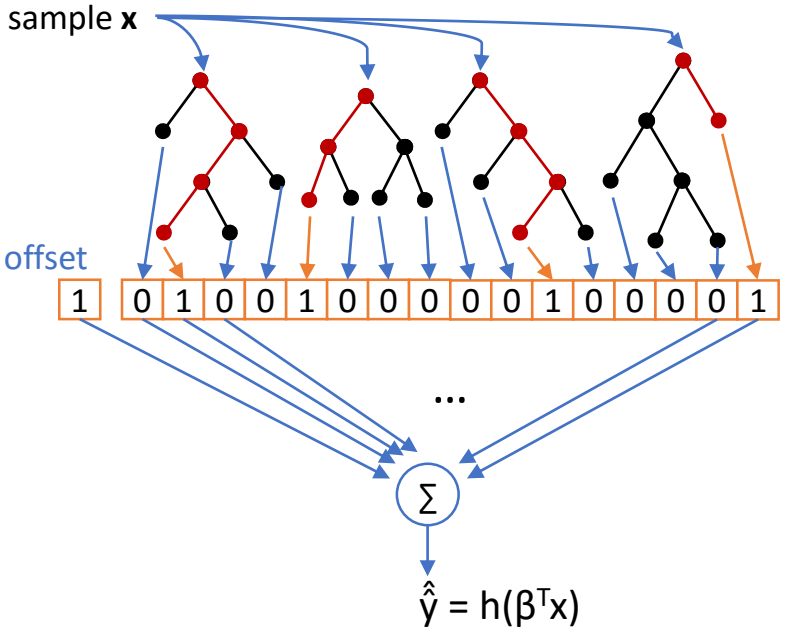


Training of these two stages can be done on different datasets. Why is that useful?

- Somewhat expensive to train,
- Limited maximum training dataset,
- Difficult to do online / real-time.

⇒Let's train the feature transformer stage only daily or weekly.

GBDT + LR



Training of these two stages can be done on different datasets. Why is that useful?

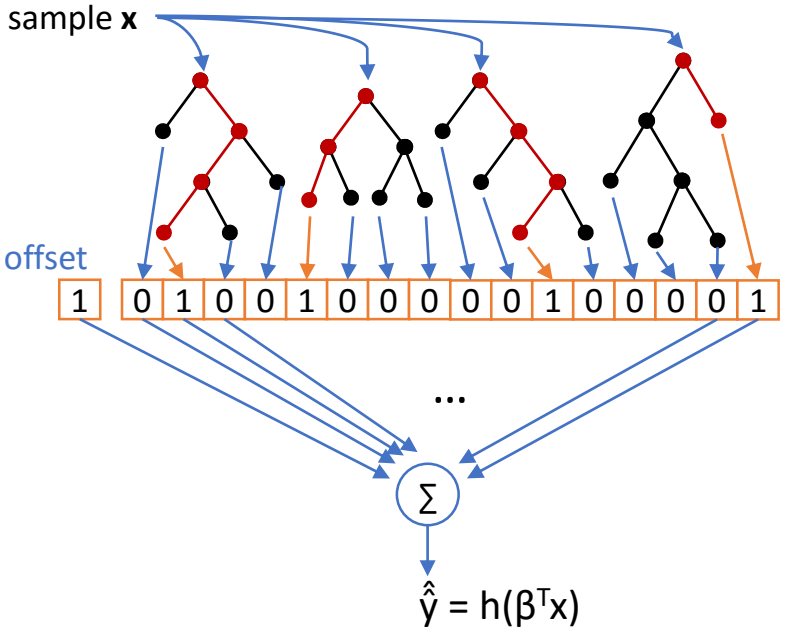
- Somewhat expensive to train,
- Limited maximum training dataset,
- Difficult to do online / real-time.

⇒Let's train the feature transformer stage only daily or weekly.

- Cheap to train.
- Can consume unlimited data.,
- Simple online/real-time training.

⇒Let's train this final stage continuously on all the data as it comes.

GBDT + LR



Training of these two stages can be done on different datasets. Why is that useful?

- Somewhat expensive to train,
- Limited maximum training dataset,
- Difficult to do online / real-time.

⇒Let's train the feature transformer stage only daily or weekly.

- Cheap to train.
- Can consume unlimited data.,
- Simple online/real-time training.

⇒Let's train this final stage continuously on all the data as it comes.

TL;DR: Get the power of GBDTs, combined with data freshness of LR.

(until recent successes with deep NNs, this was the best default choice)

Thank you!

Ask me anything.