inzva Applied AI Program
Week 2

Anomaly Detection & Recommender Systems

Onur Boyar

02.01.2021

Today’s Topics

- Anomaly Detection
- Isolation Forest
- Local Outlier Factor
- Autoencoders
- XGBoost
- CatBoost
- Hyperparameter Optimization
- TabNet Discussion (Optional)
- Recommender Systems
- FunkSVD
- Mult-DAE

Anomaly Detection

Anomaly detection is the process of identifying unexpected or rare events in the data.

Anomaly Detection

- Identifying unexpected events in data sets.
- Banks, servers, factories
- Medical Industry
- E-mail services, any retail company
- Behavioral anomalies

Anomaly Detection

Why it is critical?

- Potential risks are too costly.

- Being unable to detect a cyber attack

- Being unable to detect the malign cells

- Being unable to detect fraud transactions etc..

Each comes with huge costs.

Anomaly Detection

Why it is hard?

- Rare event detection is a hard task in complex systems.

- In dynamic environments like cyber environment, actors are changing their way of acting in the system to be not detected.

- Misclassification cost might be too high.

Anomaly Detection

Approaches

- Rule Based Methods
- Statistical Methods
- Traditional Machine Learning Methods
- Deep Learning Methods

Anomaly Detection

Main challenge

- The main challenge in a prediction based anomaly detection system is learning from an imbalanced data.

- Unsupervised techniques are promising approaches to overcome that challenge.

Anomaly Detection

In this week, we use Isolation Forest, Local Outlier Factor, Autoencoders, XGBoost, and CatBoost algorithms to detect cyber attacks.

KDDCUP ‘99 Dataset

Dataset we are going to use to build our models is KDDCUP’99 dataset. This dataset includes information about network connections and whether a connection is a part of an intrusion attempt or a normal connection.

By intrusion attempt we mean a cyber attack like Probing, Denial of Service Attack etc.

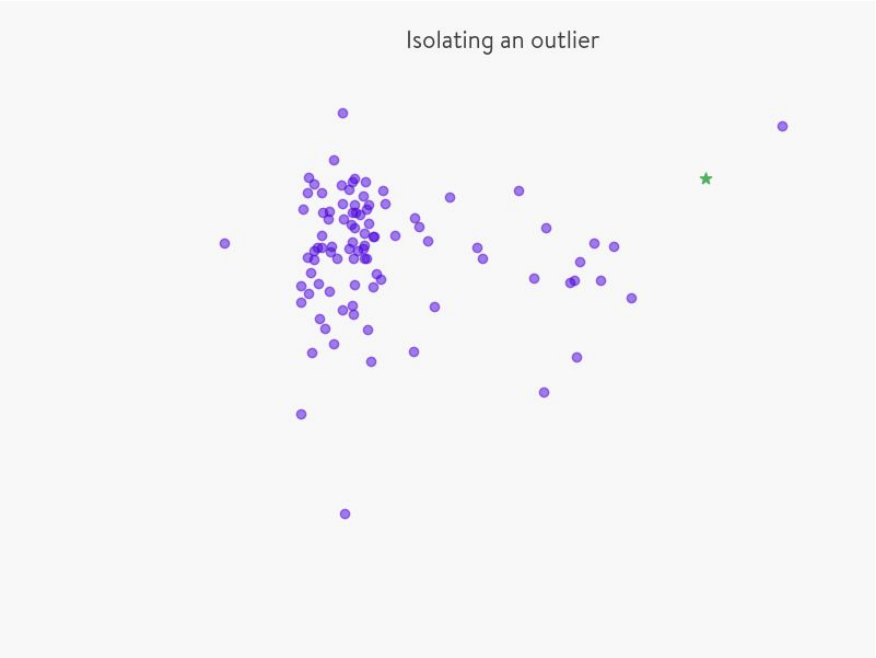
In our algorithms, we will try to detect Probing attacks. It is an attack to investigate targeted network to explore the technologies & hardwares being used in that network.

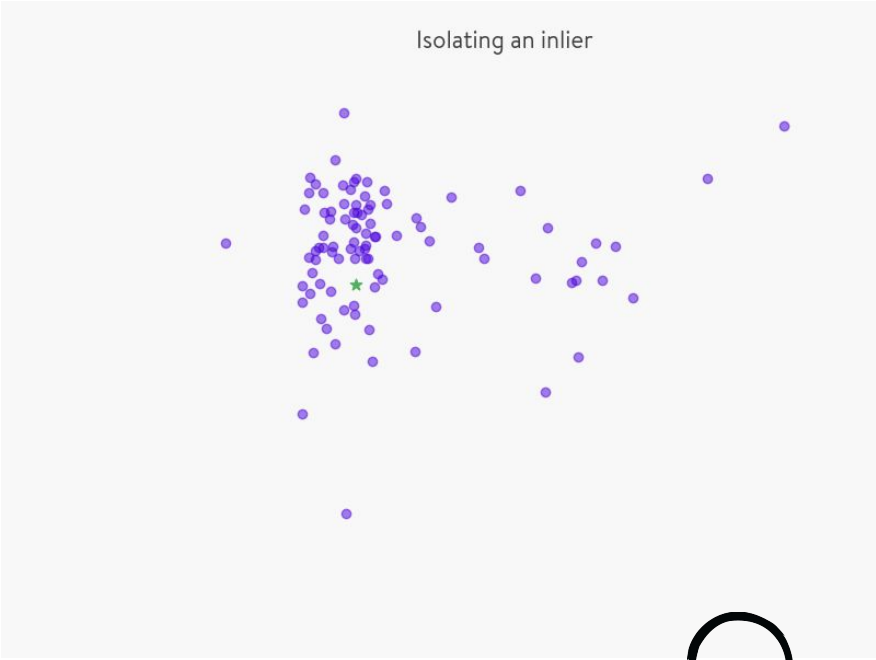
Isolation Forest

Isolation Forest algorithm works based on an intuition that anomalous observations are easier to ‘isolate’.

Normal observations tend to stay together in an euclidean space where anomalous observations tend to far apart from them.

Isolation Forest





Hyperparameters

- **n_estimators, default=100**: The number of base estimators in the ensemble.
- **max_samples**: The number of samples to draw from X to train each base estimator
- **contamination**: The amount of contamination of the data set, i.e. the proportion of outliers in the data set. Used when fitting to define the threshold on the scores of the samples.
- **max_features**: The number of features to draw from X to train each base estimator.

Isolation Forest

It generates a ‘Anomaly Score’ and returns this Anomaly Score for each observation and we set a threshold based on ‘Anomaly Score’.

This score affected mostly by number of isolating lines to isolate a point.

You can refer to this link for further information.

Anomaly Detection

We can switch to **our notebook** to apply Isolation Forest algorithm.

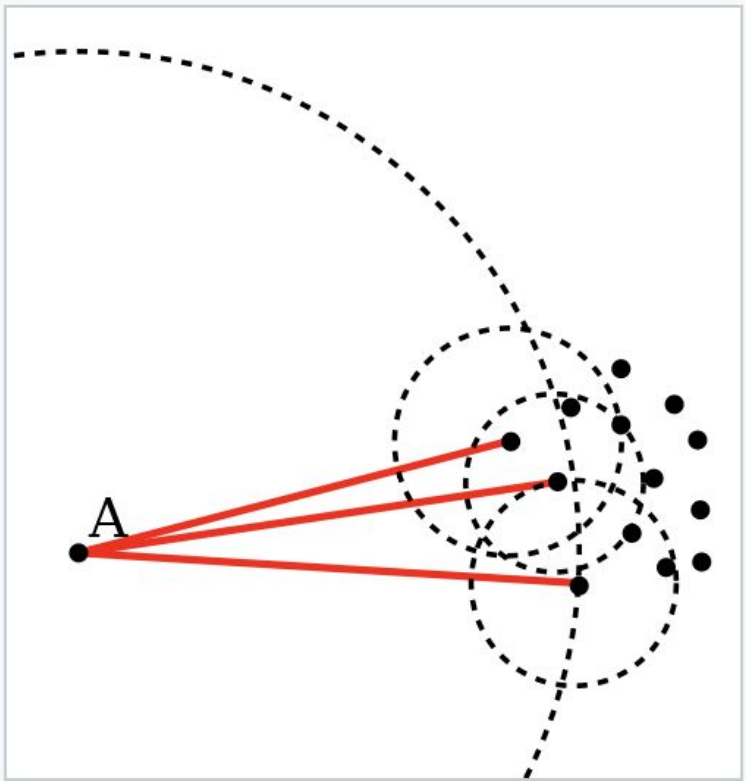
Local Outlier Factor

The local outlier factor is based on a concept of a local density, where locality is given by *k* nearest neighbors, whose distance is used to estimate the density.

By comparing the local density of an object to the local densities of its neighbors, one can identify regions of similar density, and points that have a substantially lower density than their neighbors. These are considered to be outliers.

https://en.wikipedia.org/wiki/Local_outlier_factor

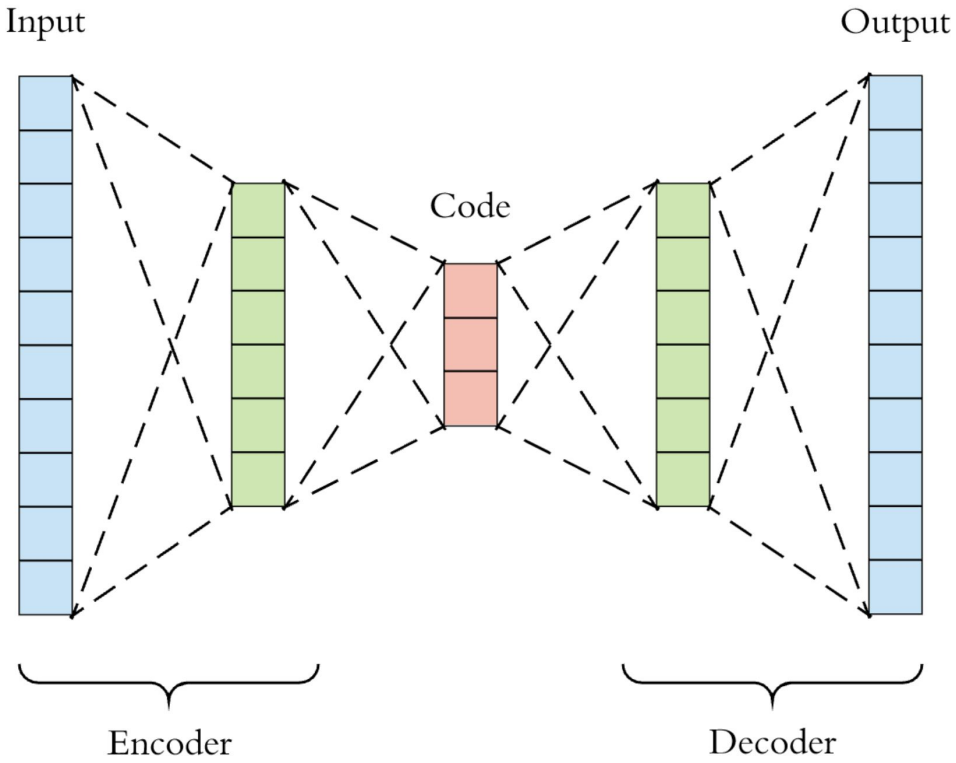
Local Outlier Factor



Local Outlier Factor

- Unsupervised technique
- Takes more time as dimensionality increase
- Refer to this paper for details.

Autoencoders



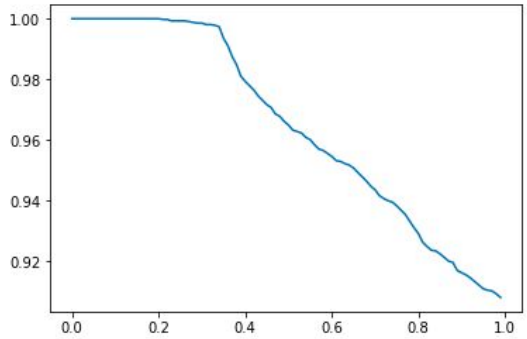
Source: https://towardsdatascience.com/generating-images-with-autoencoders-77fd3a8dd368

Autoencoder Model

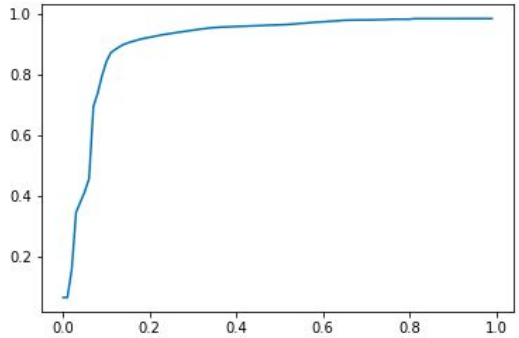
- Autoencoder model for detecting cyber attacks.
- Encoding dimension: 14
- Bottleneck dimension:7
- Mean squared error as a loss function

Autoencoder Model

TP Rate vs Reconstruction Error



Accuracy Rate vs Reconstruction Error



Probing attack detection model

Anomaly Detection

We can switch to **notebook to apply autoencoder model.**

XGBoosting

Extreme Gradient Boosting Algorithm.

Gradient Boosting is an ensemble learning algorithm. We create set of ‘weak’ classifiers and ensemble them. ‘Weak’ classifiers do not use all set of our features. They sample from our set of features and create a classifier.

We separate XGBoosting into two parts. First is “Boosting” part, second is “XGBoosting” part.

What is “Boosting”?

- For a given dataset with n examples and m features, the explanatory variables $\boldsymbol{x}_i$ are defined :

$$\boldsymbol{x}_i = (x_{i1}, x_{i2}, \cdots, x_{im})$$

- Define the $i-th$ objective variable $y_i; \mathrm{i} = 1,2, \cdots, n$

What is “Boosting”?

- Define the output of $t - th$ decision tree: $\hat{y}_i^{(t)}$

- The error $\epsilon_i^{(1)}$ between first decision tree’s output and objective variable y_i is following:

$$\epsilon_i^{(1)} = \hat{y}_i^{(1)} - y_i$$

What is “Boosting”?

- The second decision tree predict $\epsilon_i^{(1)}$
- Define the second decision tree’s output $\hat{\epsilon}_i^{(2)}$
- The predicted value $\hat{y}_i^{(2)}$ is following:

$$\hat{y}_i^{(2)} = \hat{y}_i^{(1)} + \hat{\epsilon}_i^{(2)}$$

What is “Boosting”?

- The error $\epsilon_i^{(2)}$ between predicted value using two decision tree and objective value is following:

$$\epsilon_i^{(2)} = y_i - \hat{y}_i^{(2)}$$
$$= y_i - \left(\hat{y}_i^{(1)} + \epsilon_i^{(2)}\right)$$

What is “Boosting”?

- Define the third decision tree’s predicted value $\hat{\epsilon}_i^{(2)}$
- The predicted value $\hat{y}_i^{(3)}$ using three decision trees is:

$$\hat{y}_i^{(3)} = \hat{y}_i^{(2)} + \hat{\epsilon}_i^{(3)}$$
$$= \hat{y}_i^{(1)} + \hat{\epsilon}_i^{(2)} + \hat{\epsilon}_i^{(3)}$$

What makes “XGBoosting” Extreme?

XGBoosting is nothing but an unique implementation of Gradient Boosting algorithm.

It is extreme because it is quite good at avoiding overfitting. It has number of hyperparameters to regularize the learning process.

Overview of XGBoost Algorithm

Until specified number of trees/error threshold **do**:

- Start with an initial prediction. (0.5 probability for classification)

- Find residuals for each target variable

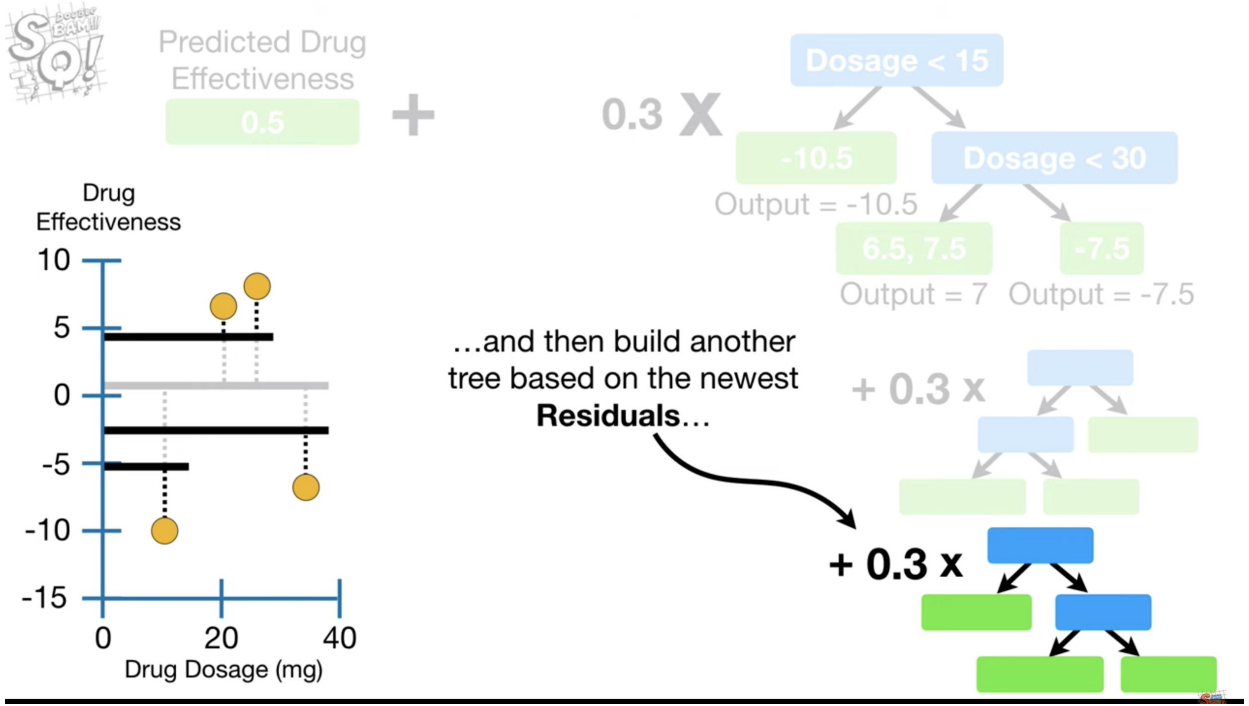
- Fit a XGBoost tree using these residuals by choosing specified number of features from our feature set.

- Find "Gain" value of the fitted tree. It is a measure for how well we fit to the residuals. Gain value can be regularized (lambda).

- If gain value is less than a threshold, gamma, we can prune the tree on that level.

- Make new predictions using regularized/pruned tree. Predicted values are added to the initial prediction and they are multiplied by learning rate.

XGBoost



Ref: [StatQuest, XGBoosting]

Some of the hyperparameters

Eta: Learning rate.
Gamma: Minimum loss reduction required to make a further partition.
Max_depth: Maximum number of features used in each tree.
Min_child_weight: Minimum number of points in the node.
Subsample: Sampling ratio of the training instances.
Lambda: L2 regularization for trees.
Scale_pos_weight: To control the balance of positive/negative points. Especially useful when the data is unbalanced.

XGBoosting

As it can be understood, XGBoost model has a lot of hyperparameters. For full list of parameters of XGBoost implementation on R/Python please refer to

https://xgboost.readthedocs.io/en/latest/parameter.html

Catboost Algorithm

- Developed by Yandex
- Has advanced way of dealing with the categorical variables
- Boosting trees grow symmetrically as opposed to XGBoost and LightGBM.

Catboost Algorithm

- XGBoost and CatBoost differs in some different aspects.
- XGBoost creates deeper and more complex decision trees while CatBoost creates simpler decision trees
- https://catboost.ai/docs/concepts/python-reference_parameters-list.html

Training of CatBoost

For each tree

1. Dataset is splitted
2. Features to be used in the tree are selected
3. Encodings for categorical (and text) features are performed
4. Optimal splittings are found and tree grows

Encoding of Categorical Features

3. All categorical feature values are transformed to numerical using the following formula:

$$avg\_target = \frac{countInClass + prior}{totalCount + 1}$$

- $countInClass$ is how many times the label value was equal to "1" for objects with the current categorical feature value.
- $prior$ is the preliminary value for the numerator. It is determined by the starting parameters.
- $totalCount$ is the total number of objects (up to the current one) that have a categorical feature value matching the current one.

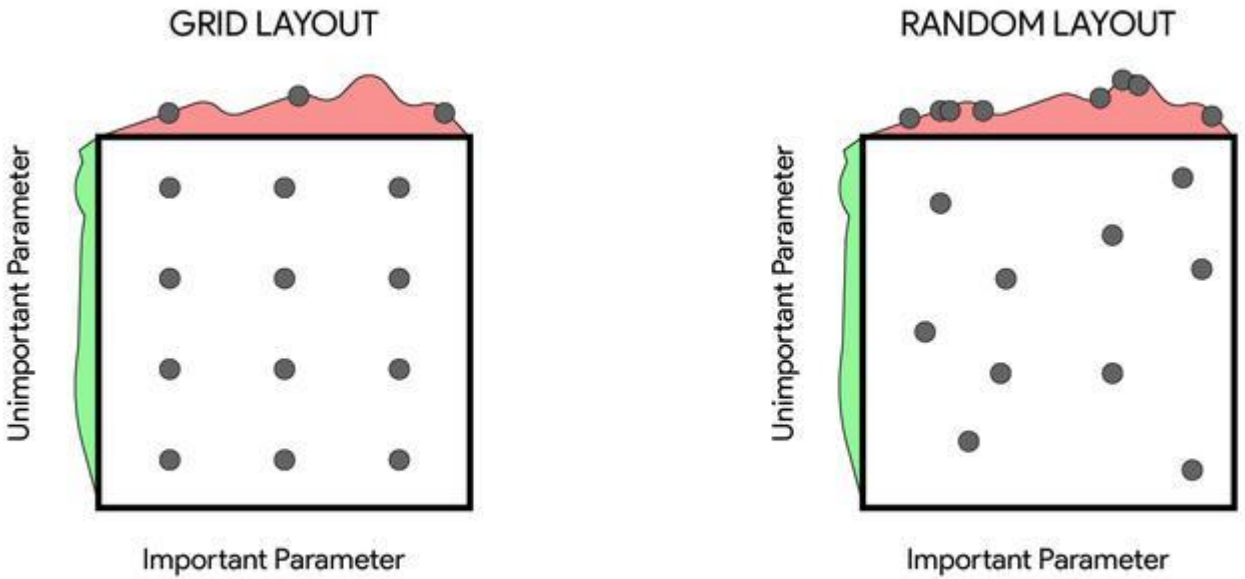
Source: https://catboost.ai/docs/concepts/algorithm-main-stages_cat-to-numeric.html#algorithm-main-stages_cat-to-numeric

Hyperparameter Optimization

- In order to get most out of the Gradient Boosting trees, we should apply hyperparameter optimization techniques.

- We can use RandomSearch, GridSearch, and Bayesian Optimization.

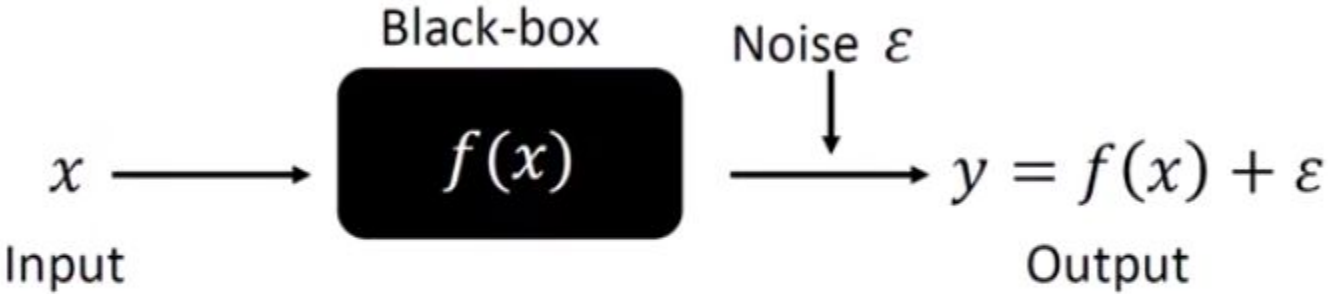
RandomSearch & GridSearch



Bayesian Optimization

- We have a function that is expensive to evaluate
- It lacks special structure like concavity. We say f is ‘black-box’
- When we evaluate f, we obtain f(x) and it is not differentiable
- We try to find a global optimum, not local

Black-box function



- Black-box function is a function in which the relationship between input and output is not expressed as an explicit formula.
- Typically, observing the output corresponding to each input is assumed to be expensive (Ex: Hyperparameter tuning of DNN).

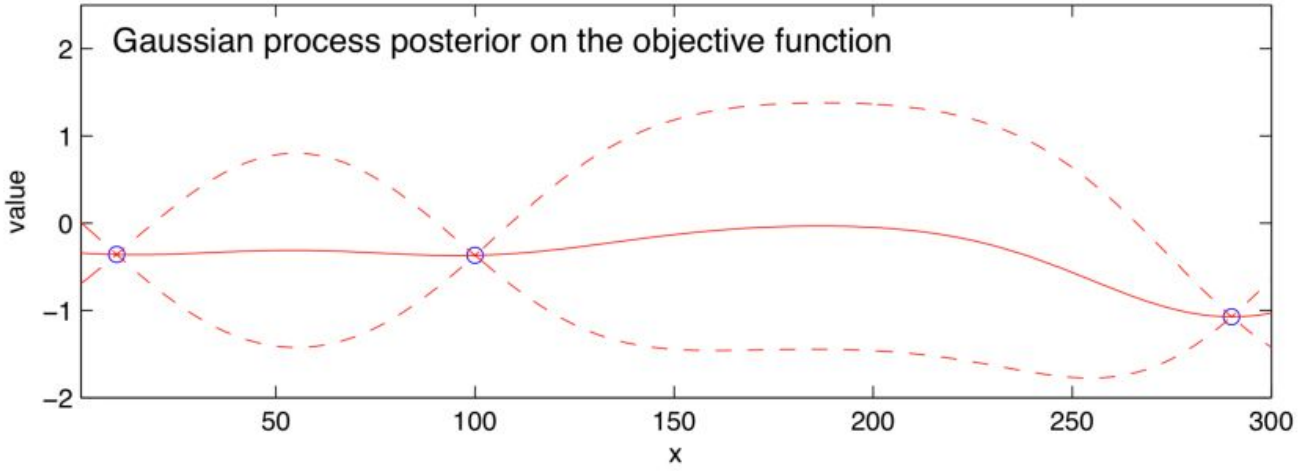
Source: https://www.youtube.com/watch?v=pPTF2S59KCE&t=2917s&ab_channel=AIPRIKEN

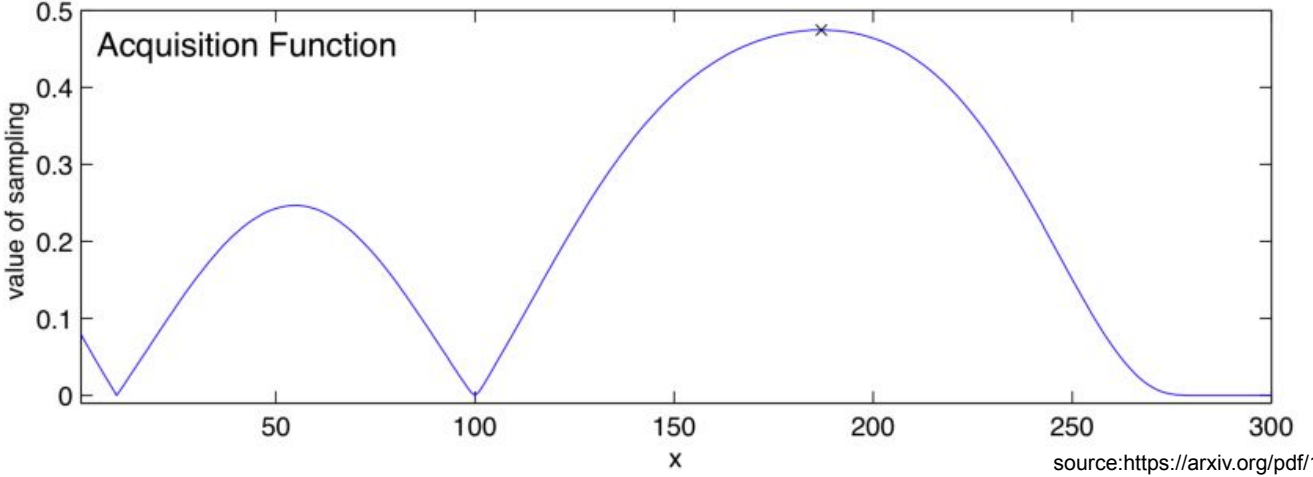
Bayesian Optimization

- Initially explores hyperparameter space randomly
- Has a guiding mechanism that tells the BayesOpt where to pick next set of values next
- Includes uncertainty
- For a tutorial on Bayesian Optimization, please refer to this paper.

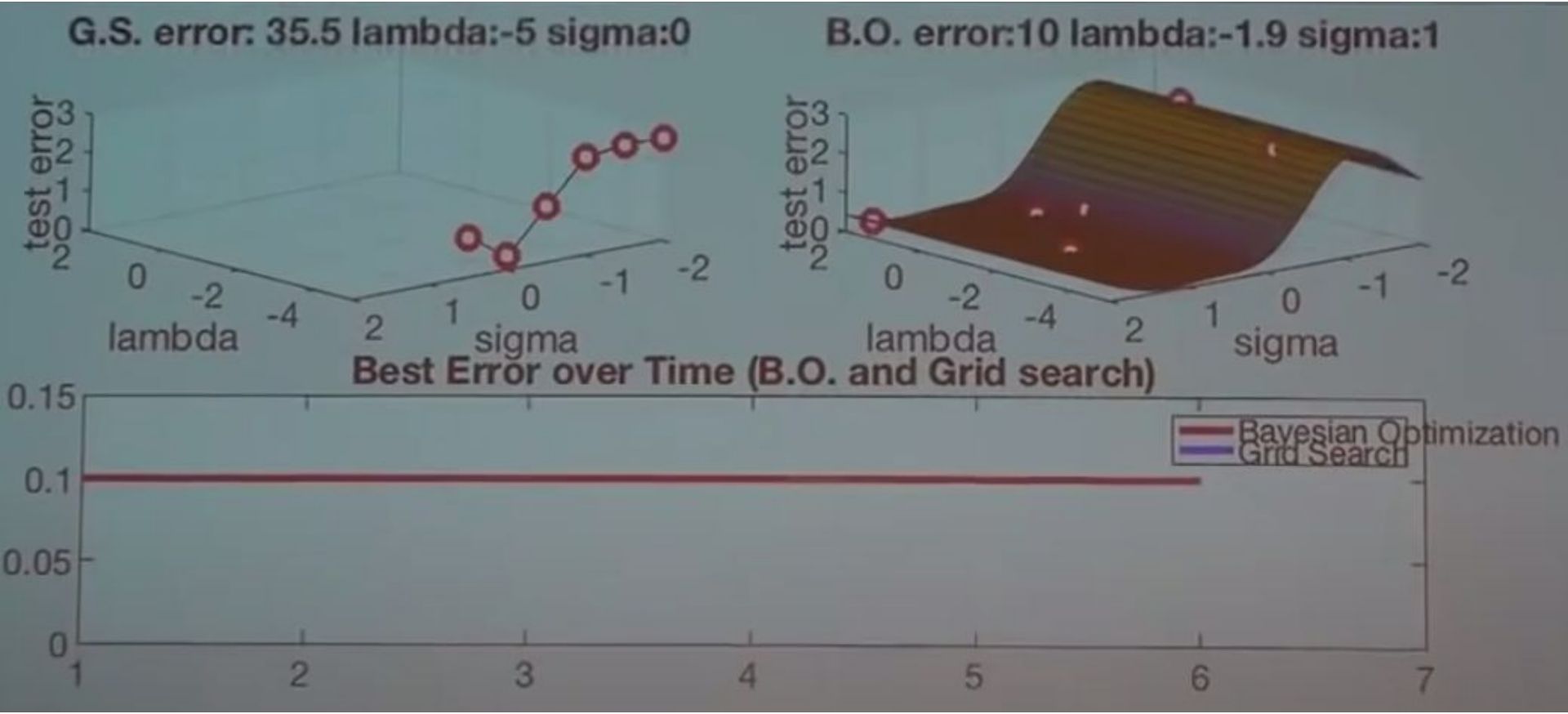
Other Links

- https://sheffieldml.github.io/GPy/
- https://sheffieldml.github.io/GPyOpt/
- https://github.com/fmfn/BayesianOptimization
- https://towardsdatascience.com/an-introductory-example-of-bayesian-optimization-in-python-with-hyperopt-aae40fff4ff0
- https://papers.nips.cc/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper.pdf



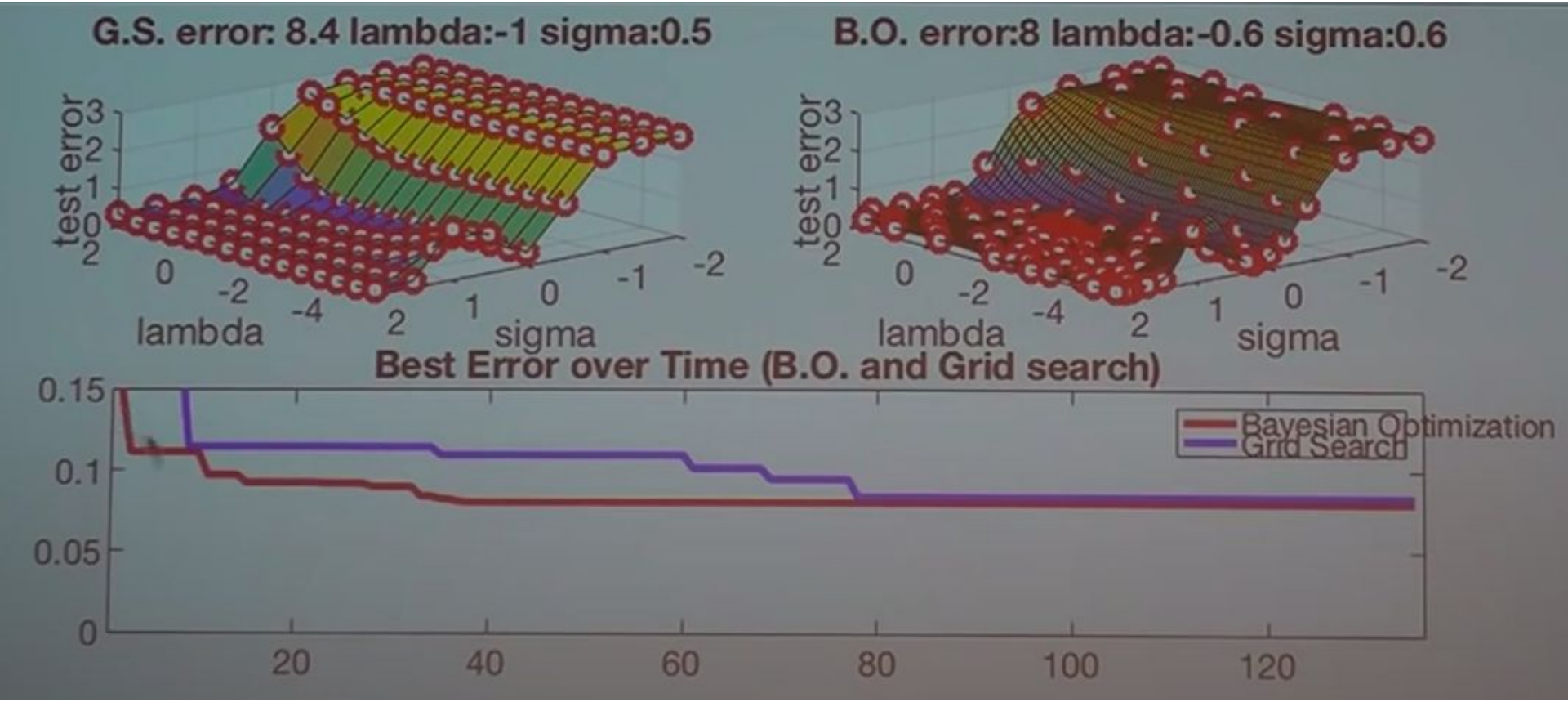


source:https://arxiv.org/pdf/1807.02811.pdf



Source: https://www.youtube.com/watch?v=BzHJ57QCdVo&t=1086s&ab_channel=KilianWeinberger


G.S. error: 11.4 lambda:-4 sigma:1.5
B.O. error:8.1 lambda:-0.9 sigma:0.6
test error
lambda
sigma
Best Error over Time (B.O. and Grid search)
Bayesian Optimization
Grid Search
0.15
0.1
0.05
0
5
10
15
20
25
30


G.S. error: 8.4 lambda:-1 sigma:0.5
B.O. error:8 lambda:-0.6 sigma:0.6
test error
lambda
sigma
Best Error over Time (B.O. and Grid search)
Bayesian Optimization
Grid Search
0.15
0.1
0.05
0
20
40
60
80
100
120

TabNet

TabNet: Attentive Interpretable Tabular Learning

Sercan Ö. Arık, Tomas Pfister
Google Cloud AI
Sunnyvale, CA
soarik@google.com, tpfister@google.com

Abstract

We propose a novel high-performance and interpretable canonical deep tabular data learning architecture, TabNet. TabNet uses sequential attention to choose which features to reason from at each decision step, enabling interpretability and more efficient learning as the learning capacity is used for the most salient features. We demonstrate that TabNet outperforms other variants on a wide range of non-performance-saturated tabular datasets and yields interpretable feature attributions plus insights into its global behavior. Finally, we demonstrate self-supervised learning for tabular data, significantly improving performance when unlabeled data is abundant.

ments particularly for large datasets (Hestness et al. 2017). In addition, unlike tree learning, DNNs enable gradient descent-based end-to-end learning for tabular data which can have a multitude of benefits: (i) efficiently encoding multiple data types like images along with tabular data; (ii) alleviating the need for feature engineering, which is currently a key aspect in tree-based tabular data learning methods; (iii) learning from streaming data and perhaps most importantly (iv) end-to-end models allow representation learning which enables many valuable application scenarios including data-efficient domain adaptation (Goodfellow, Bengio, and Courville 2016), generative modeling (Radford, Metz, and Chintala 2015) and semi-supervised learning (Dai et al. 2017).

Attention Transformer Blocks

- Instance-wise feature selection for better performances
- Built-in explainability derived from masks
- Efficient Learning Capacity

Feature Transformer Blocks

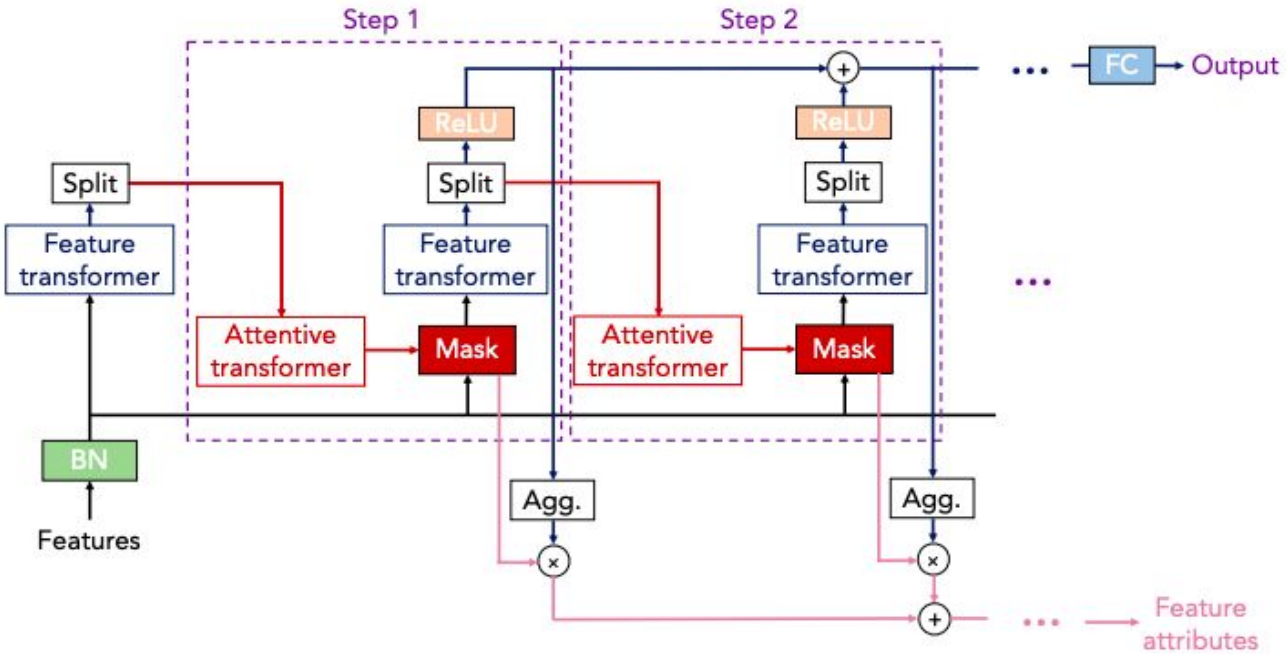
- Basic MLP block with Gated Linear Unit Activation
- Shared layers accross different steps

Sequential Steps

- Mimics ensembling
- Large Model Capacity

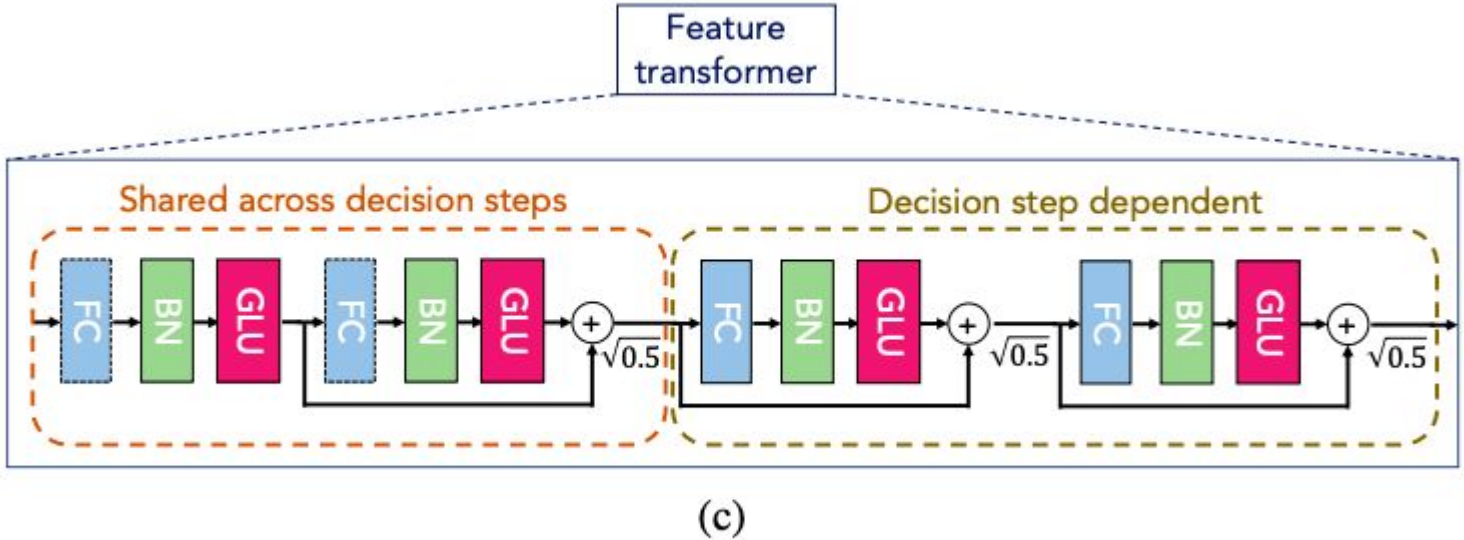
Source: https://www.youtube.com/watch?v=ysBaZO8YmX8&t=881s&ab_channel=AbhishekThakur

TabNet



(a) TabNet encoder architecture

TabNet


Feature transformer
Shared across decision steps
Decision step dependent
FC
BN
GLU
FC
BN
GLU
+
√0.5
FC
BN
GLU
+
√0.5
FC
BN
GLU
+
√0.5

(c)

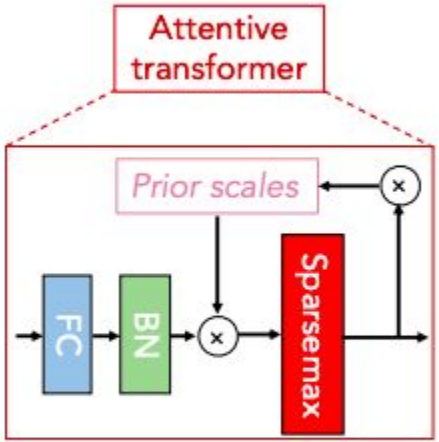
GLU

- GLU(x) = sigmoid(x).x

  GLU control what information will be passed to the following layer.

  In a language modeling task, the gating mechanism allows selection of words or features that are important for predicting the next word.

TabNet



(d)

Recommender Systems

Recommender Systems are one of the most common applications of ML-Based systems on production.

70% of the movies/series watched on Netflix comes from Recommended Items. *(Mass Communication: Living in a Media World, Ralph E. Hanson)*

E-commerce companies, YouTube, Dating Apps etc. are relied heavily on Recommender Systems.

Recommender Systems

Implicit / Explicit feedback.

Evaluation of Recommender Systems:

- Normalized Discounted Cumulative Gain

  How relevant are the top N recommendations? Number of recommendations might change for every customer.

- Recall @ k

  Same intuition with NDCG. Top k recommendations and their actual labels are compared.

- Using test set to evaluate set of recommendations. They have different calculations.

Recommender Systems

In this week, we will use different algorithms to build recommender systems.

- Matrix Factorization model based on FunkSVD
- Deep Learning Collaborative Filtering Model using Denoising Autoencoders.

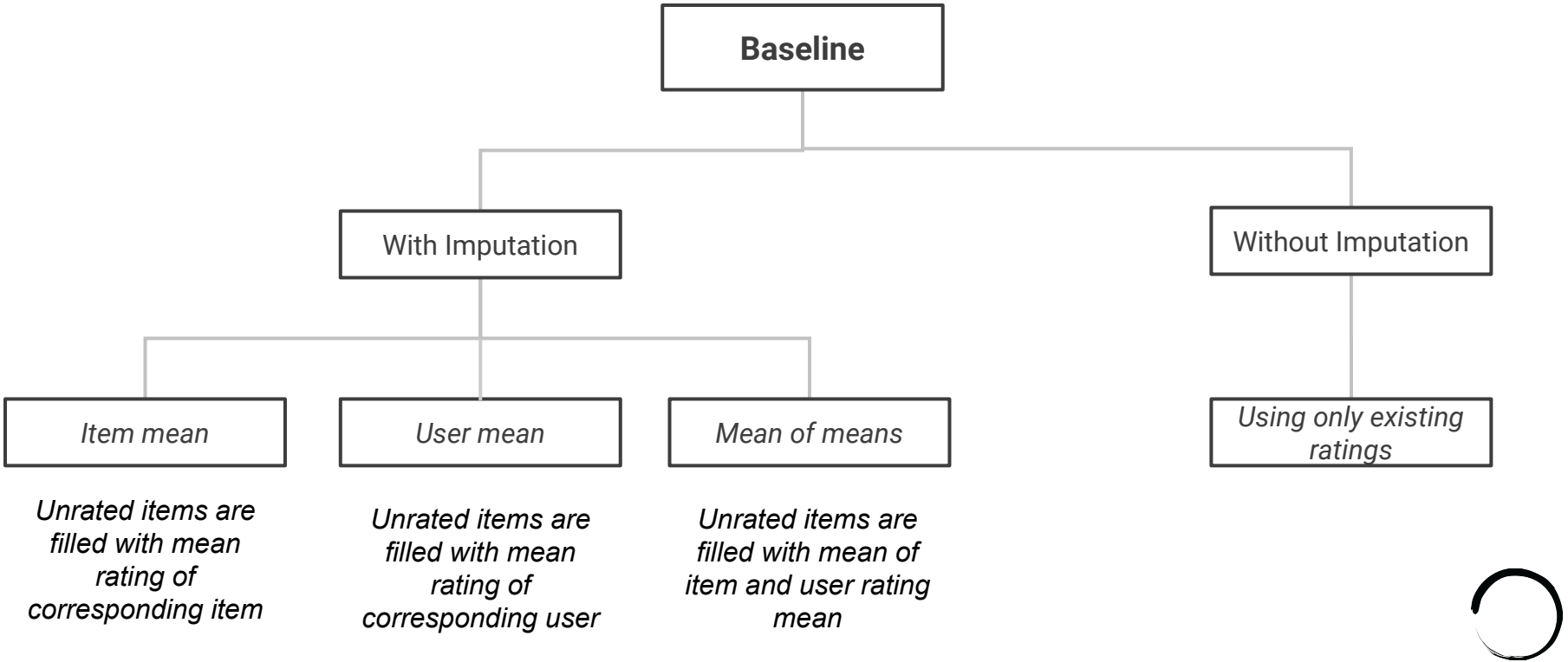
Recommender Systems

We will first build the foundation by discussing about Nearest Neighbors. Nearest Neighbors with/without imputing missing cells of the sparse User-Rating matrix can be considered as baseline approach of Recommender System algorithms.

Nearest Neighbour

- Nearest Neighbour sorts the users with respect to their distance to new user, and recommends items from nearest neighbours.
- Different similarity measures are used, such as L2 norm, cosine similarity etc.
- To calculate distance, only the intersected items between two vectors are used.

Baseline Approach



FunkSVD

Initialize matrices randomly (p,q)
for f = 1…k:
  until feature has converged:
for *rui* in Ratings:
  predict *rui*
  calculate error
  update *puf, qif*

$$\epsilon_{ui} = r_{ui} - r_{ui,estimated}$$
$$= r_{ui} - b_{ui} - \sum_f p_{uf} q_{if}$$

Update Rules

$$\epsilon_{ui} = r_{ui} - \tilde{r}_{ui}$$
$$q_{if} \mathrel{+}= -\lambda \frac{d}{dq_{if}} e_{ui}^2 = \lambda(\epsilon_{ui} p_{uf})$$
$$p_{uf} \mathrel{+}= -\lambda \frac{d}{dp_{uf}} e_{ui}^2 = \lambda(\epsilon_{ui} q_{if})$$

Regularization

Add regularization, to discourage p_{uf}/q_{if} from being large:

$$\epsilon_{ui} = r_{ui} - \tilde{r}_{ui}$$
$$q_{if} \mathrel{+}= \lambda(\epsilon_{ui} p_{uf} - \gamma q_{if})$$
$$p_{uf} \mathrel{+}= \lambda(\epsilon_{ui} q_{if} - \gamma p_{uf})$$

Update Step

$$\epsilon_{ui} = r_{ui} - s(u,i)$$
$$\Delta q_{if} = \lambda(\epsilon_{ui} p_{uf} - \gamma q_{if})$$
$$\Delta p_{uf} = \lambda(\epsilon_{ui} q_{if} - \gamma p_{uf})$$

where,

- λ is the learning rate
- γ is the regularization

Autoencoder Based Approach

Liang et al., Variational Autoencoders for Collaborative Filtering, 2018.

In the work above a different recommender systems using Variational Autoencoders and Denoising Autoencoders are proposed.

We will learn about their proposed method, Mult-DAE.

Ref: https://arxiv.org/pdf/1802.05814.pdf

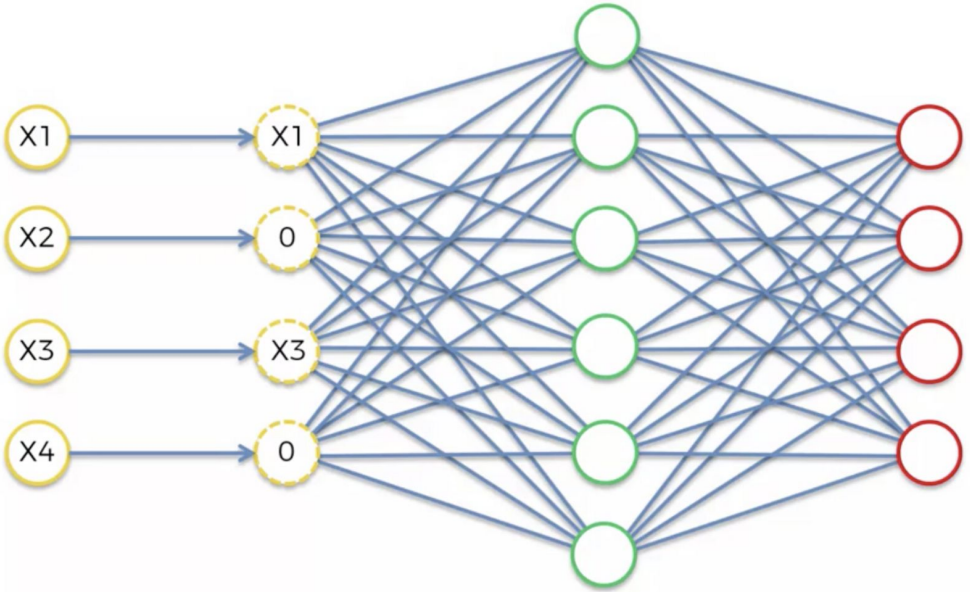
What is ‘Denoising Autoencoder’?

We already saw that an Autoencoder model aims to reconstruct the input while learning low dimensional representation of the data. In order to prevent AE to totally copy input to output (input = output with MSE = 0) we can have some options.

- Regularization, dropout, early stopping
- Using Denoising Autoencoder Model

Denoising Autoencoders

We purposefully corrupt the input data by turning some of the values to zero.



Denoising Autoencoders

Crucial part is that when calculating the loss, we use original input values. In that way, we eliminate the risk of learning the identity function and we are able to extract features / learn weights.

Homework

- Compare XGBoost and CatBoost algorithms using their default parameters.

- Compare XGBoost and CatBoost algorithms after applying hyperparameter optimization.

- Select the better model and apply two other hyperparameter optimization technique and see their results.

- Which one takes so much time? Which one is more efficient?