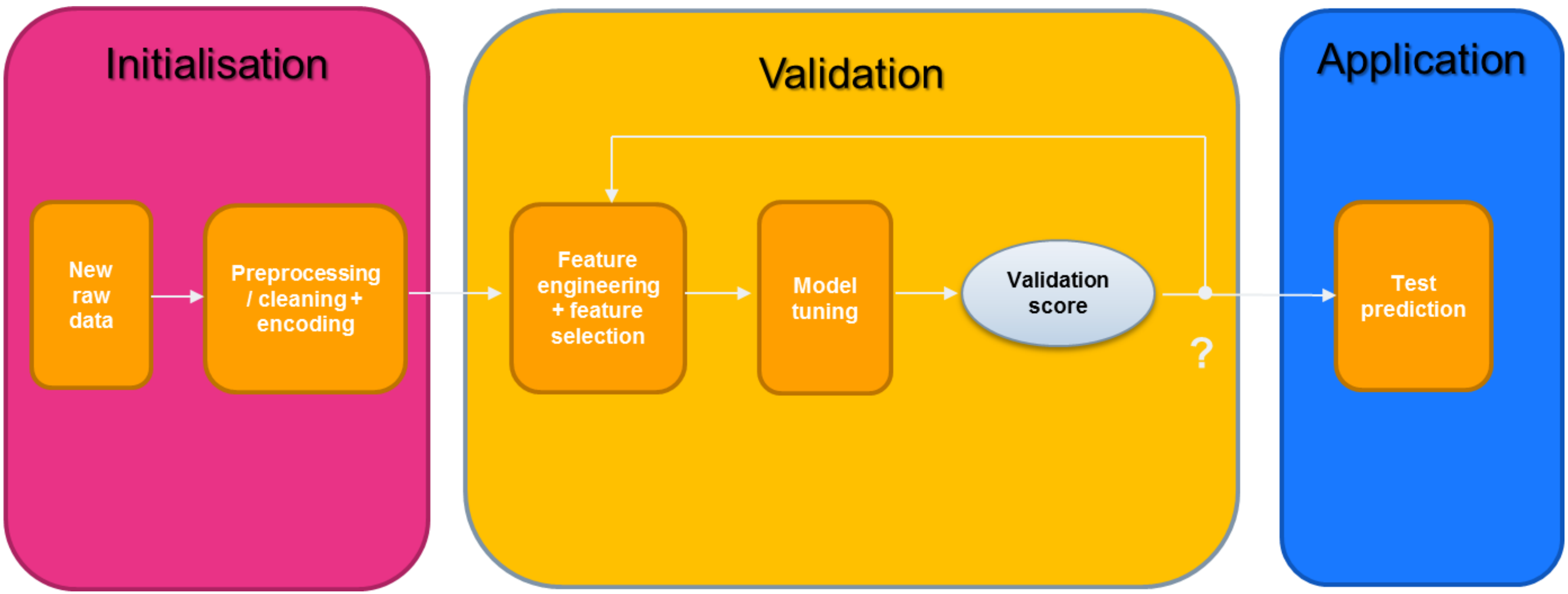
2 - MLBox

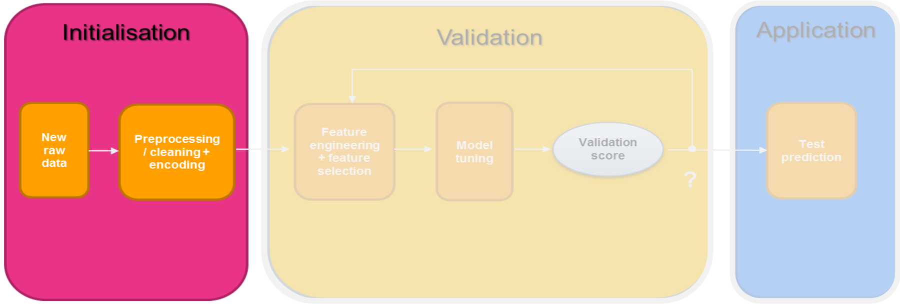
2 - MLBox

a – Features

MLBox : a fully automated pipeline



MLBox: features

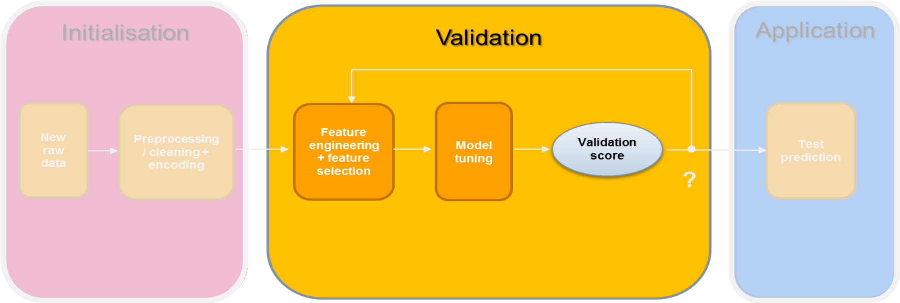


Initialisation:

From a **raw dataset** to a **cleaned dataset** with numerical features.

- **Files reading:**
  - Reading of several files (csv, xls, json and hdf5)
  - **Task detection** (binary/multiclass classification or regression)
  - **Creation of the training data set and the test data set**
- **Preprocessing/cleaning:**
  - Dropping duplicates and constant features
  - **Dropping drifting features → see « drift »**
- **Encoding:**
  - Converting features to a **unique format** (float if possible or str)
  - Converting **lists**
  - Converting **dates** into timestamp
  - Target encoding for classification task only
  - Categorical features encoding (several strategies available !)
  - Missing values encoding (several strategies available !)

MLBox: features



Validation:

Several models are **tested** and **cross-validated** and the best one is fitted.

- **On features:**

  - **Feature engineering** : neural network features engineering → [see « entity embedding »]
  - **Feature selection :** filter methods, wrapper methods and L1 regularization

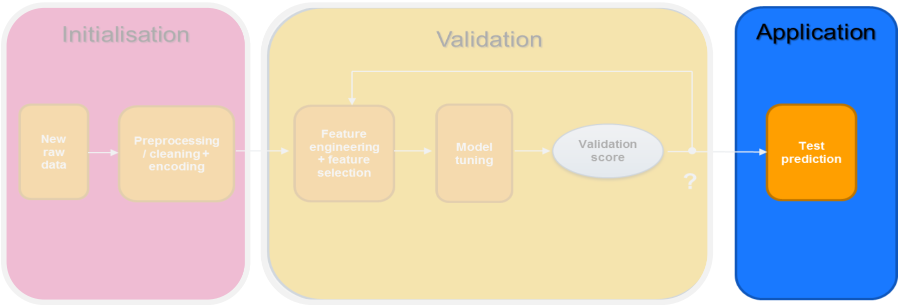
- **On estimator:**

  - TestIng of a wide range of accurate estimators: Linear model, Random Forest, XGBoost, LightGBM…
  - Model blending: **stacking**, boosting, bagging
  - **Hyper-parameters tuning** (using TPE algorithm)

- **On validation :**

  - **Choice of several metrics**: accuracy, log-loss, AUC, f1-score, MSE, MAE, … **or customized**
  - **Validation parameters:** number of folds, random state, …

MLBox: features



Application:

We **fit** the **whole** pipeline and **predict** the target on the test set.

- **<u>Prediction:</u>**
  - ➢ **Target prediction** (class probabilities for classification)
  - ➢ **Dumping fitted models and final predictions** (.csv file)
  - ➢ CPU **time display**

- **<u>Models interpretation</u>:**
  - ➢ **Features importance**
  - ➢ **Leak detection**

2 - MLBox

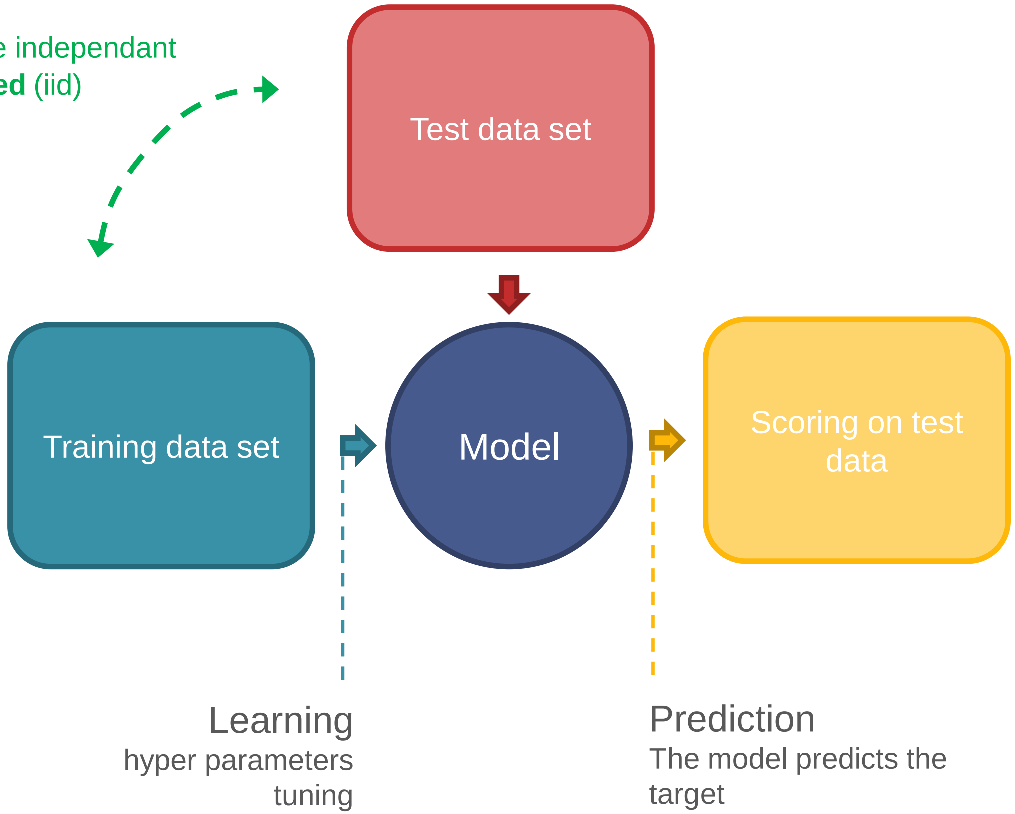
b – focus #1 on MLBox

« Drift »: a brand new algorithm !

Focus 1 on MLBox: « drift detection »

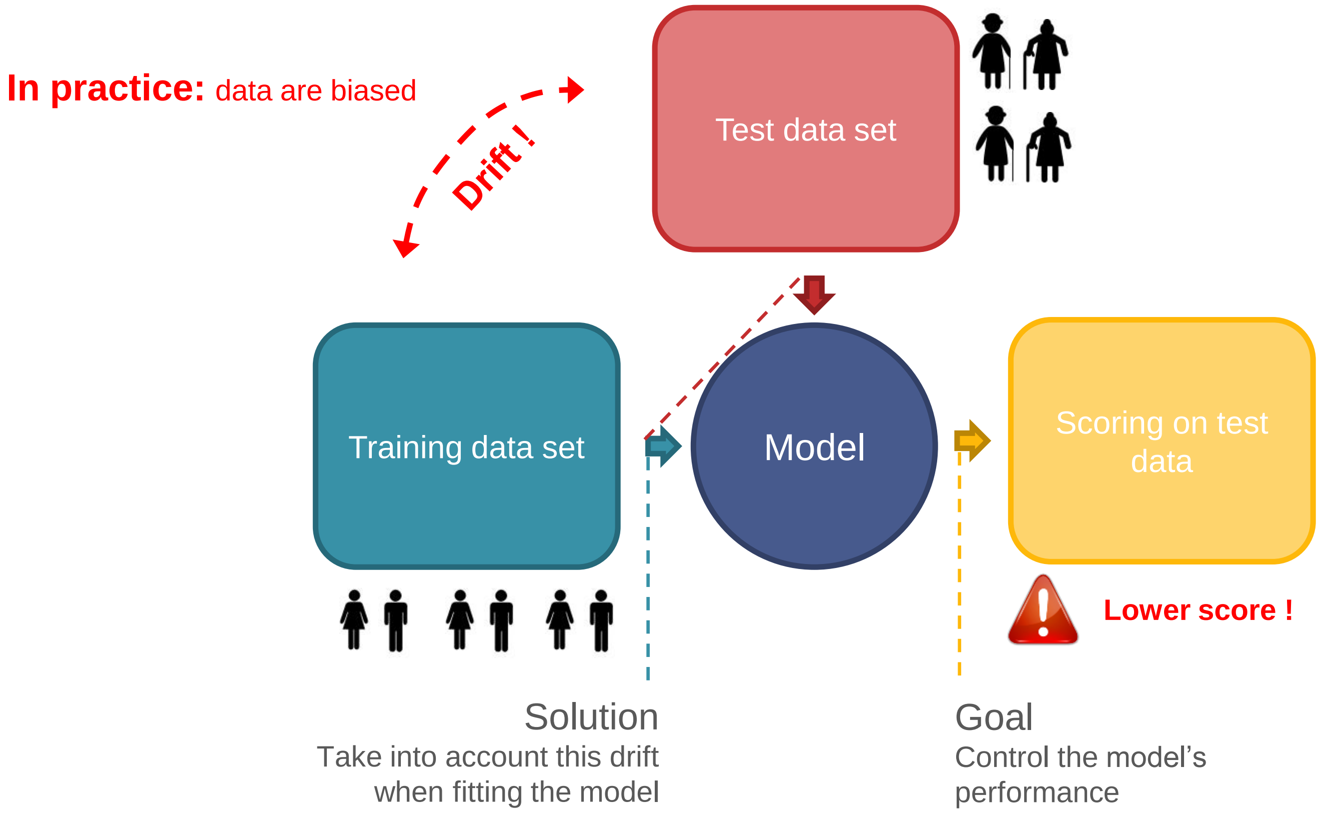
➢ <u>**The issue**</u>

Hypothesis : data are independant and **identically distributed** (iid)



Focus 1 on MLBox: « drift detection »

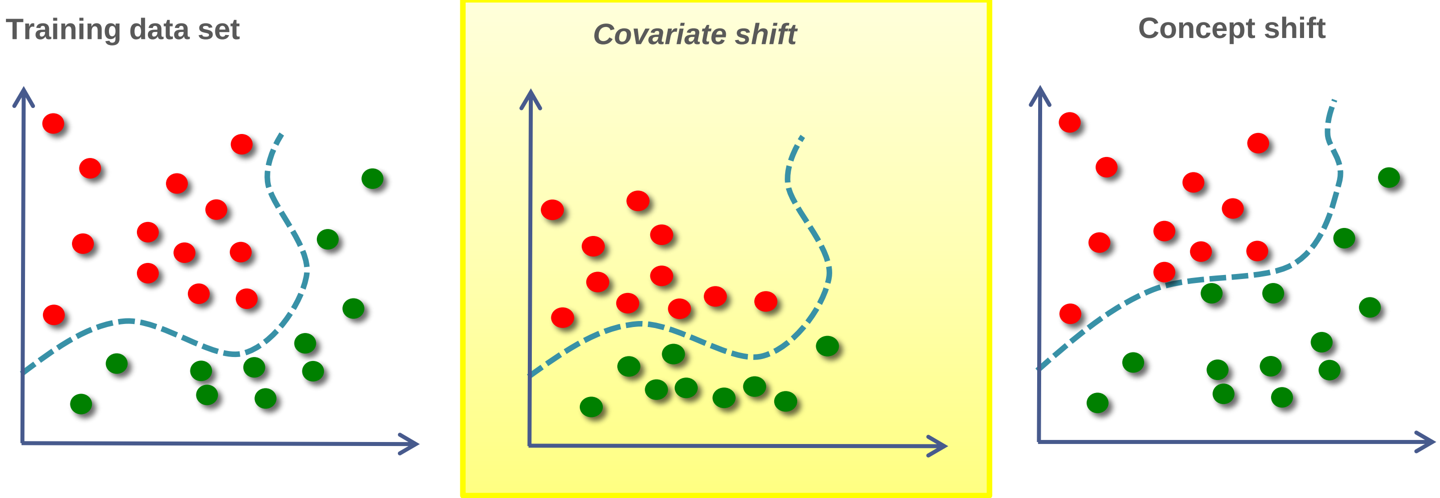
➢ **The issue**



Focus 1 on MLBox: « drift detection »

➢ **Definition**

$$P(x,y) = P(x) \,.\, P(y|x)$$



We deal with this case

Focus 1 on MLBox: « drift detection »

➢ **The solution**

1 – Drift estimation for each feature - *i.e. covariate shift*

2 – Feature selection in order to:

- maintain the model’s performance
- lower the drift

Focus 1 on MLBox: « drift detection »

➢ **The algorithm**

1

Univariate drift estimation

- **Labelling data** whether it belongs to the training or the test set
- **Fitting** a classifier to separate both classes
- **Validation:** roc AUC score gives us a drift measure

2

(Greedy) Recursive Drift Elimination

For each feature in decreasing drift order:

- We drop the feature
- We cross-validate the new model
- If delta score is > p :
  - We keep the feature
  - If not we drop it permanently

NB :

- Greedy algorithm: we can set a limited number of features to try or a drift threshold under which features are kept
- We can set different p

Focus 1 on MLBox: « drift detection »

➢ Best case scenario

- **100%** drifting features (we drop it)
- **0%** drifting features (we keep it)
- **Drifting** features that are **not important** (we drop it)

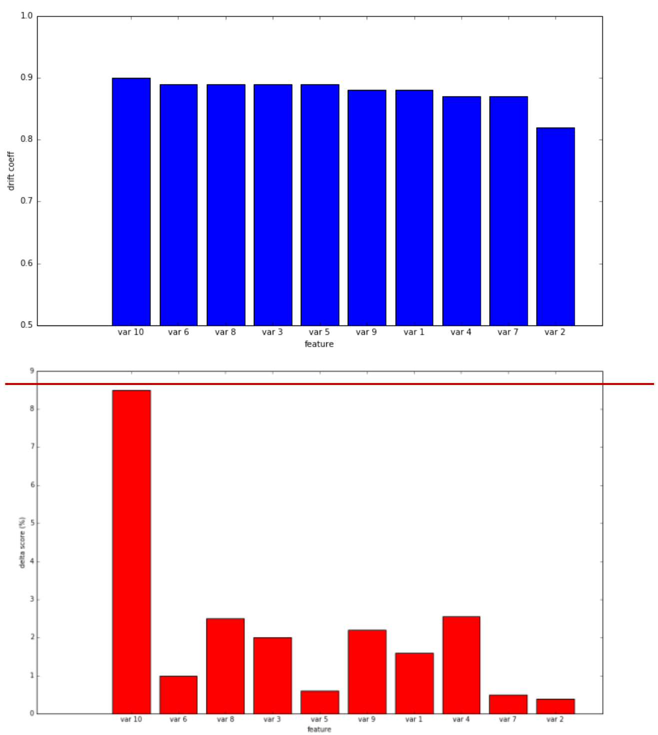
➢ Worst case scenario

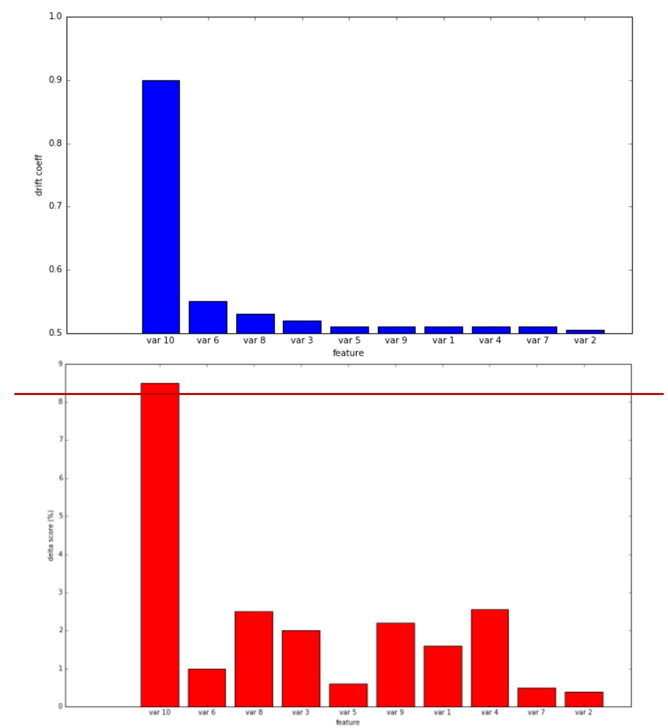
- **Important drifting** features (2 sub-cases) :

Drift measure

Maximum threshold for delta score

Delta score (%)





2 - MLBox

c – Focus #2 on MLBox

« Entity embeddings »

Focus 2 on MLBox: « entity embeddings »

➢ **How to encode a categorical feature?**

Methods	Explanation	Advantages	Drawbacks
label encoding	For each categorical feature, we encode each value using the lexicographic order (A -> 1, B -> 2, …)	**Quick** + **scalable**	**Naive encoding** + **non understandable**
dummification	We « binarise » all categorical features (ex : var1_A, var1_B, …)	**Quick** + **less naive encoding** + **understandable**	**Not scalable**
Random projection	Random label encoding in k dimension	**Quick** + **scalable** + **less naive encoding**	**Non understandable**
discretization	We gather values	**scalable** + **understandable**	**Loss of information**
entity embedding	How to do ???	**Accurate** + **scalable** + **understandable** + **quick**	**none ?** ☺

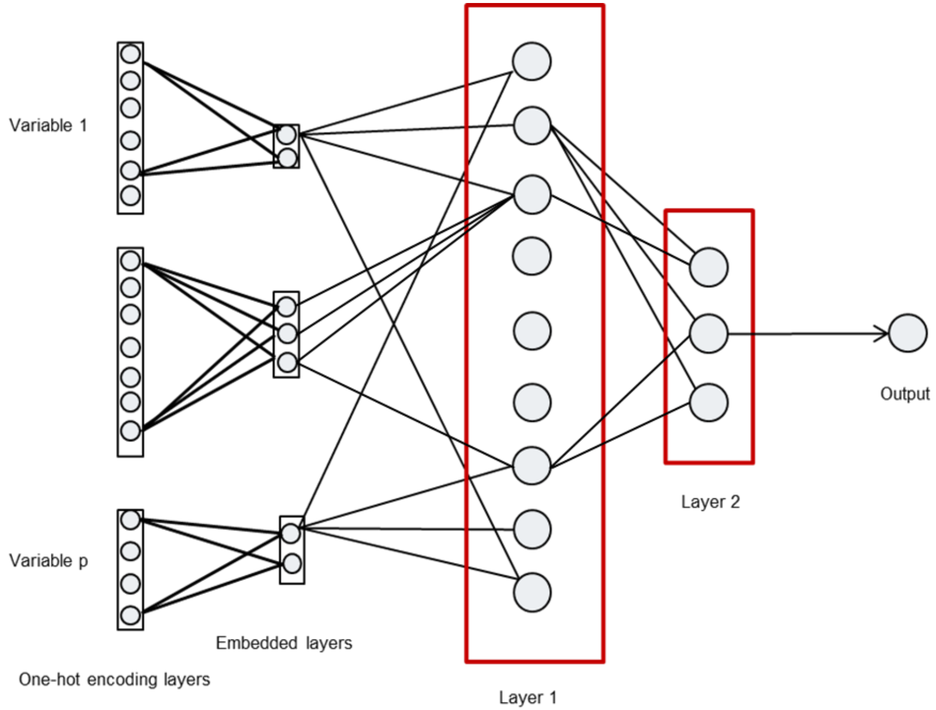
Focus 2 on MLBox: « entity embeddings »

➢ The idea of entity embedding

We would like to learn the **best vectorial representation** in an **euclidian space**:

- **Label encoding/random projection : distance between 2 values = random**
- **Dummification : distance between 2 values = $\sqrt{2}$**
- **Entity embedding : distance between 2 « similar » values low and vice versa**

➢ Solution : use a neural network to learn this representation



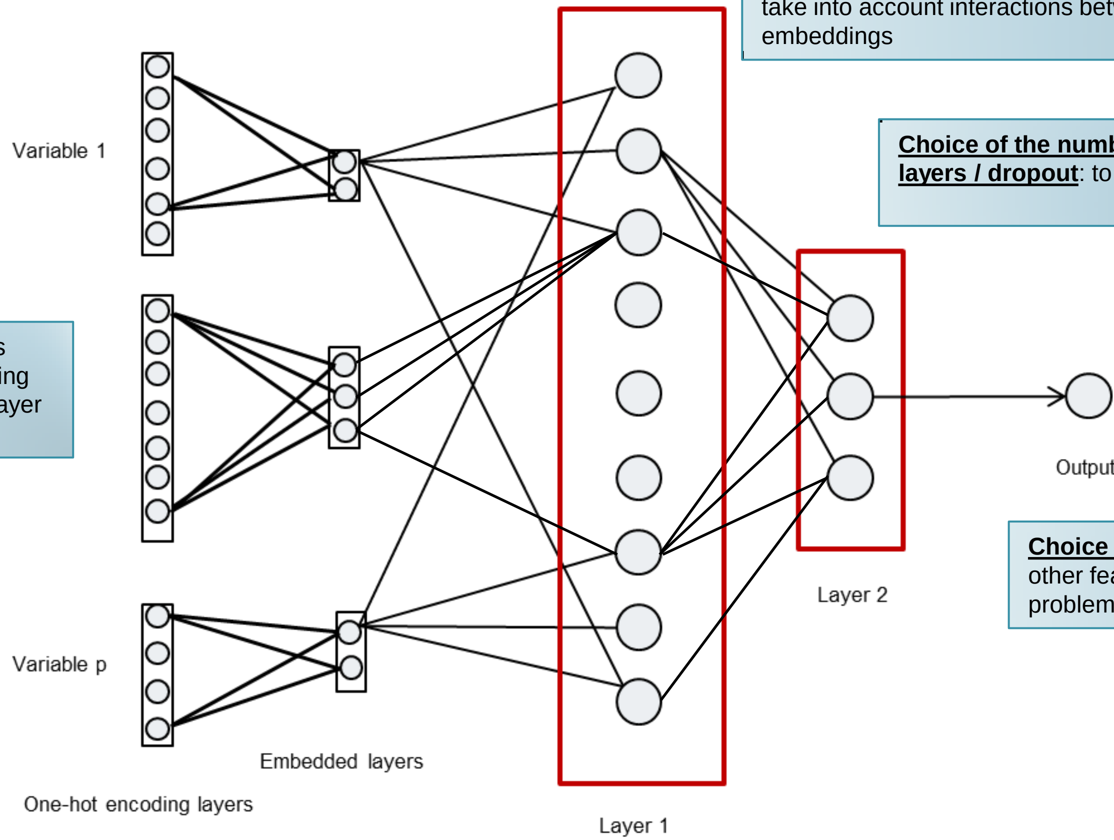
Focus 2 on MLBox: « entity embeddings »

➢ **<u>The principle of entity embedding</u>**

<u>Choice of the number of units in the layer 1</u> : take into account interactions between embeddings

<u>Choice of the number of layers / dropout</u>: to be tested !

<u>Encoding</u> : we get the weighs between the « one-hot-encoding layer » and the « embedded layer »



<u>Choice of the ouput</u>: the target or other features for an unsupervised problem

<u>Choice of the number of units for the « embedded layer » :</u>

- Constant (2 or 3)
- Proportionnal to the number of values (threshold = 5)
- Using a discretization method (Edge ML)

Focus 2 on MLBox: « entity embeddings »

➢ **Test on a real data set**

- DataScience.net challenge: https://www.datascience.net/fr/challenge/26/details
- We would like to predict the price of an **automobile insurance policy**
- Training data set: 300.000 rows and 43 features like: brand, age, postcode, energy consumption, …

Top 10:

	id	annee_naissance	annee_permis	marque	puis_fiscale	anc_veh	codepostal	energie_veh	kmage_annuel	crm	...	var14	var15	var16	var17	var18	var19	var20	var21	var22	prime_tot_ttc
0	1.0	1986.0	2006.0	RENAULT	4.0	1.0	1034	gpl	2924.0	68.0	...	N	1.0	1	1.0	27.0	0.0	0.0	0.0	1.0	254.75
1	2.0	1986.0	2006.0	RENAULT	8.0	2.0	1034	gpl	11580.0	50.0	...	N	2.0	1	1.0	28.0	0.0	0.0	0.0	1.0	259.89
2	3.0	1982.0	2001.0	RENAULT	7.0	2.0	1034	gpl	7149.0	95.0	...	N	3.0	1	1.0	29.0	0.0	0.0	0.0	1.0	431.65
3	4.0	1987.0	2006.0	DACIA	5.0	2.0	1034	gpl	6526.0	100.0	...	N	3.0	1	1.0	29.0	0.0	0.0	0.0	1.0	577.99
4	5.0	1994.0	2013.0	CITROEN	4.0	2.0	1034	gpl	2872.0	50.0	...	N	2.0	1	4.0	26.0	0.0	1.0	1.0	4.0	222.67
5	6.0	1985.0	2006.0	HONDA	5.0	2.0	1034	hybride essence	7191.0	100.0	...	N	0.0	1	4.0	27.0	0.0	1.0	1.0	4.0	595.23
6	7.0	1974.0	1994.0	CHRYSLER	11.0	2.0	1034	gpl	12387.0	100.0	...	N	2.0	1	4.0	27.0	0.0	1.0	1.0	4.0	540.89
7	8.0	1989.0	2008.0	DAEWOO	9.0	1.0	1034	gpl	7227.0	100.0	...	N	2.0	1	4.0	27.0	0.0	1.0	1.0	4.0	728.57
8	9.0	1987.0	2006.0	RENAULT	8.0	2.0	1034	gpl	11580.0	50.0	...	N	2.0	1	1.0	33.0	0.0	0.0	1.0	2.0	259.89
9	10.0	1990.0	2009.0	RENAULT	7.0	4.0	1034	gpl	6496.0	50.0	...	N	1.0	1	1.0	34.0	0.0	0.0	1.0	2.0	207.58

155 brands

6 energy consumptions

target

Focus 2 on MLBox: « entity embeddings »

➢ <u>Test on a real data set</u>: accuracy + quickness

```
In [7]: opt = Optimiser(scoring, n_folds, verbose=verbose)
        opt.evaluate(None, df)

No parameters set. Defaut configuration is tested

########################################################## testing hyper-parameters... ##########################################################

>>> NA ENCODER : {'numerical_strategy': 'mean', 'categorical_strategy': '<NULL>'}

>>> CA ENCODER : {'strategy': 'label_encoding'}

>>> ESTIMATOR : {'reg_alpha': 0, 'colsample_bytree': 0.9, 'silent': True, 'colsample_bylevel': 0.6, 'scale_pos_weight': 1, 'learning_rate': 0.05, 'missing': None, 'max_delta_step': 0, 'nthread': -1, 'base_score': 0.5, 'strategy': 'XGBoost', 'n_estimators': 500, 'subsample': 0.9, 'reg_lambda': 1, 'seed': 0, 'min_child_weight': 1, 'objective': 'reg:linear', 'max_depth': 7, 'gamma': 0}

MEAN SCORE :  mean_absolute_error  =  -29.9181289037
CPU time: 52.185188055 seconds
```

```
In [8]: opt.evaluate({"ce__strategy":"entity_embedding"}, df)

########################################################## testing hyper-parameters... ##########################################################

>>> NA ENCODER : {'numerical_strategy': 'mean', 'categorical_strategy': '<NULL>'}

>>> CA ENCODER : {'strategy': 'entity_embedding'}

>>> ESTIMATOR : {'reg_alpha': 0, 'colsample_bytree': 0.9, 'silent': True, 'colsample_bylevel': 0.6, 'scale_pos_weight': 1, 'learning_rate': 0.05, 'missing': None, 'max_delta_step': 0, 'nthread': -1, 'base_score': 0.5, 'strategy': 'XGBoost', 'n_estimators': 500, 'subsample': 0.9, 'reg_lambda': 1, 'seed': 0, 'min_child_weight': 1, 'objective': 'reg:linear', 'max_depth': 7, 'gamma': 0}

MEAN SCORE :  mean_absolute_error  =  -29.7093714336
CPU time: 122.252818823 seconds
```

Focus 2 on MLBox: « entity embeddings »

➢ **<u>Test on a real data set</u>: understanding + scalability**

```
In [9]: cols = ["marque","energie_veh"]

        ce = Categorical_encoder("entity_embedding",verbose=False)

        %time df_train_emb = ce.fit_transform(df['train'][cols],df['target'])
        df_train_emb.head()

CPU times: user 2min, sys: 2.9 s, total: 2min 2s
Wall time: 33.2 s
```

Out[9]:

	marque_emb1	marque_emb2	marque_emb3	marque_emb4	marque_emb5	energie_veh_emb1	energie_veh_emb2
0	0.799269	-0.767513	-0.802887	0.801199	0.812181	-0.885237	0.816619
1	0.799269	-0.767513	-0.802887	0.801199	0.812181	-0.885237	0.816619
2	0.799269	-0.767513	-0.802887	0.801199	0.812181	-0.885237	0.816619
3	2.610133	-2.577490	-2.639355	2.580082	2.603229	-0.885237	0.816619
4	0.843187	-0.868857	-0.869318	0.846686	0.846408	-0.885237	0.816619

Threshold = 5 units for the embedded layer

Focus 2 on MLBox: « entity embeddings »