Using target to generate features

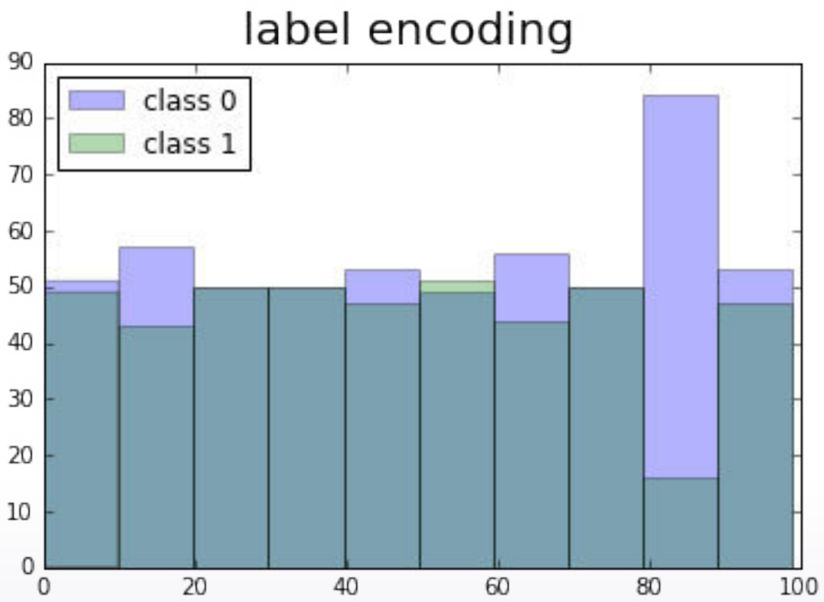
Simple example

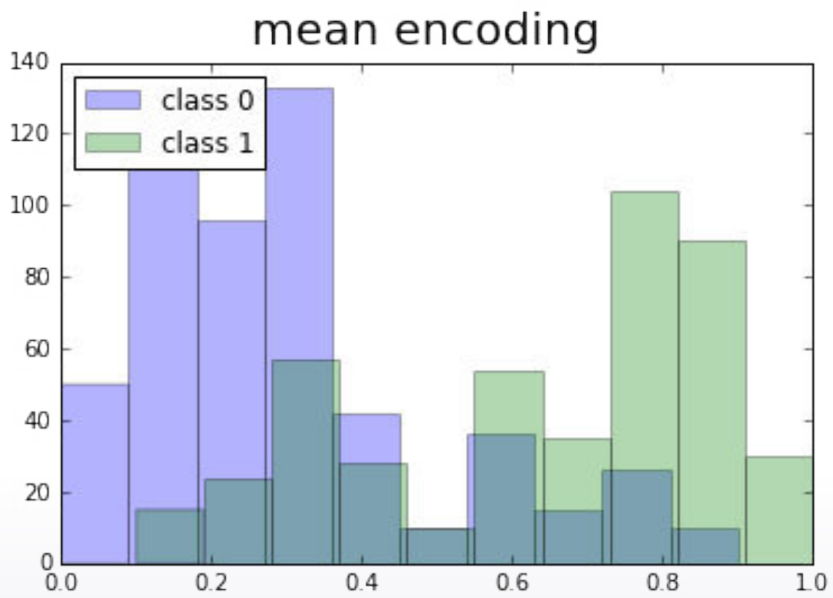
- Categorical feature
  - some city
- Binary classification

	feature	feature_label	feature_mean	target
0	Moscow	1	0.4	0
1	Moscow	1	0.4	1
2	Moscow	1	0.4	1
3	Moscow	1	0.4	0
4	Moscow	1	0.4	0
5	Tver	2	0.8	1
6	Tver	2	0.8	1
7	Tver	2	0.8	1
8	Tver	2	0.8	0
9	Klin	0	0.0	0
10	Klin	0	0.0	0
11	Tver	2	0.8	1

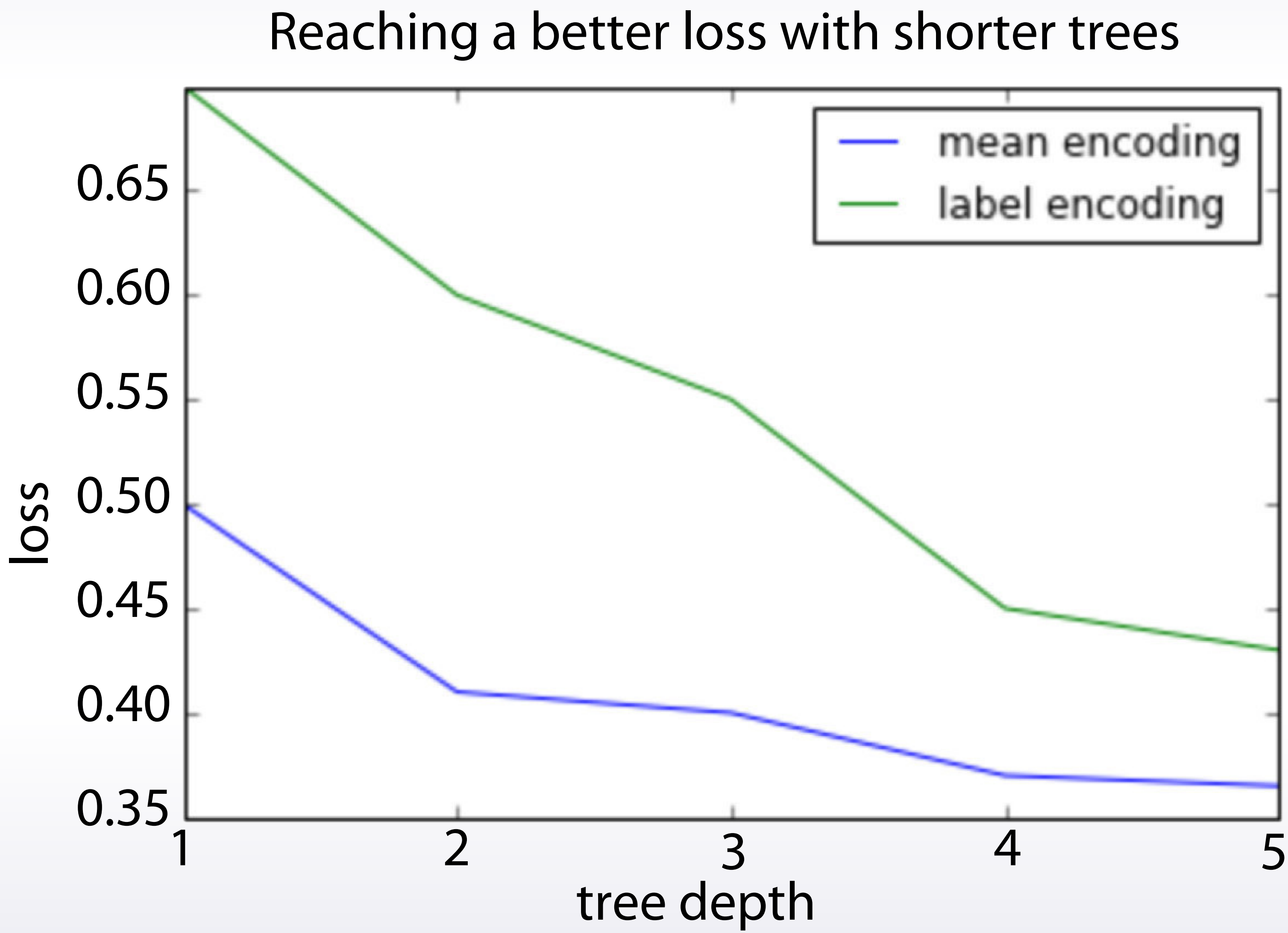
Why does it work?

1. Label encoding gives random order. No correlation with target
2. Mean encoding helps to separate zeros from ones





Why does it work?

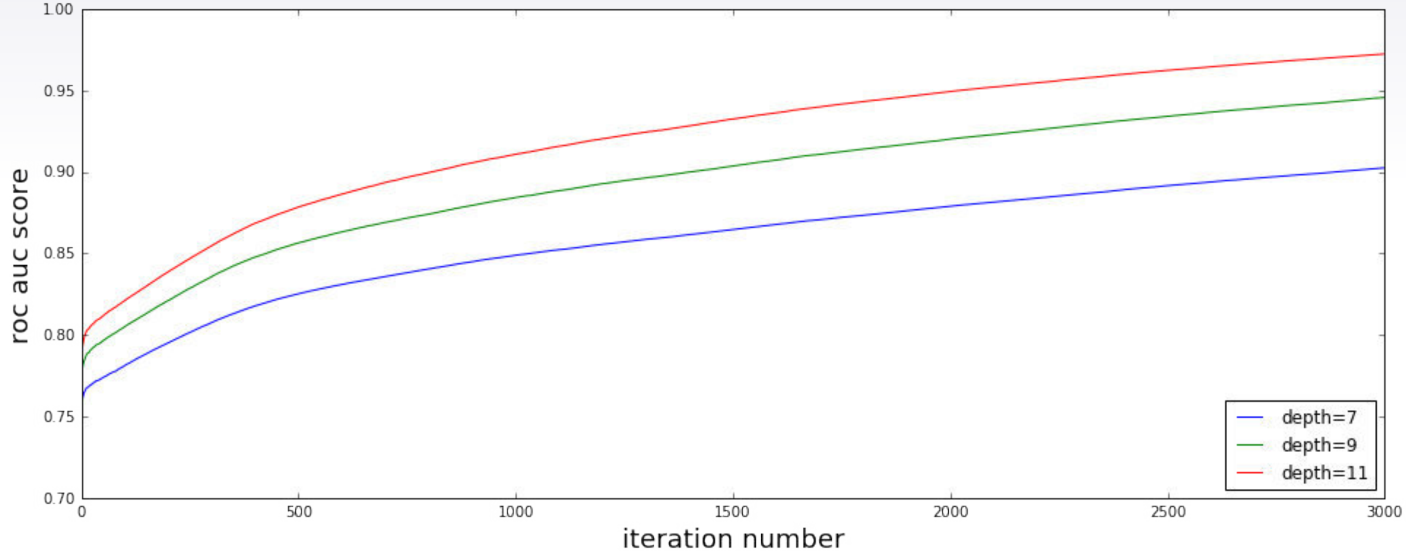


What will you learn?

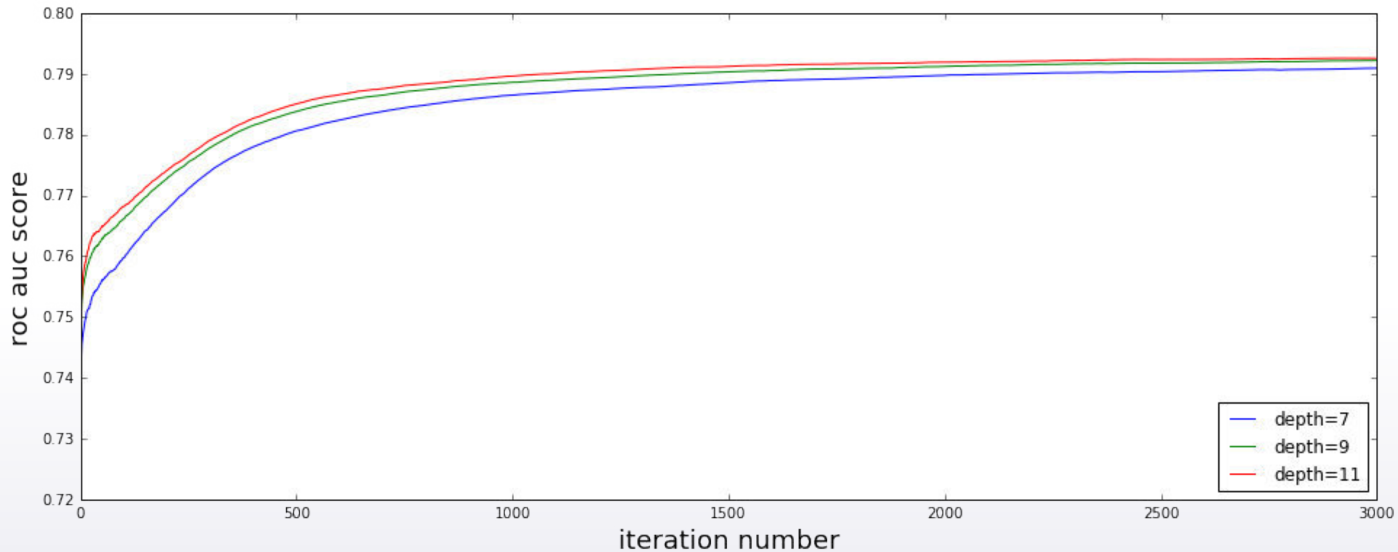
- ✓ Construct encodings
- ✓ Correctly validate them
- ✓ Extend them

Indicators of usefulness

Train



Validation



Ways to use target variable

Goods - number of ones in a group,

Bads - number of zeros

- $Likelihood = \frac{Goods}{Goods+Bads} = mean(target)$
- $Weight\ of\ Evidence = \ln\left(\frac{Goods}{Bads}\right) * 100$
- $Count = Goods = sum(target)$
- $Diff = Goods - Bads$

Springleaf example

In [4]:

```
means = X_tr.groupby(col).target.mean()
train_new[col+'_mean_target'] = train_new[col].map(means)
val_new[col+'_mean_target'] = val_new[col].map(means)

means
```

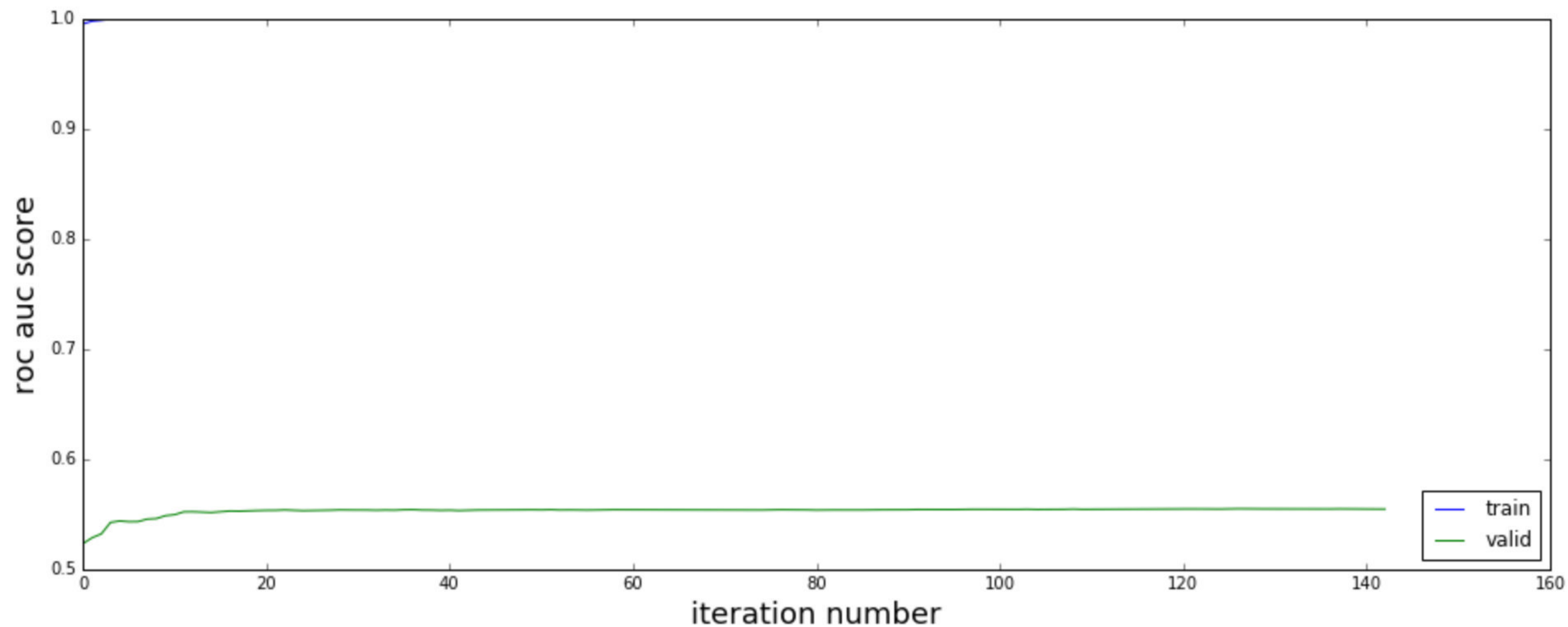
Out[4]:

```
VAR_1277
0.0       0.358965
1.0       0.219249
2.0       0.193671
3.0       0.191143
4.0       0.191080
5.0       0.185694
```

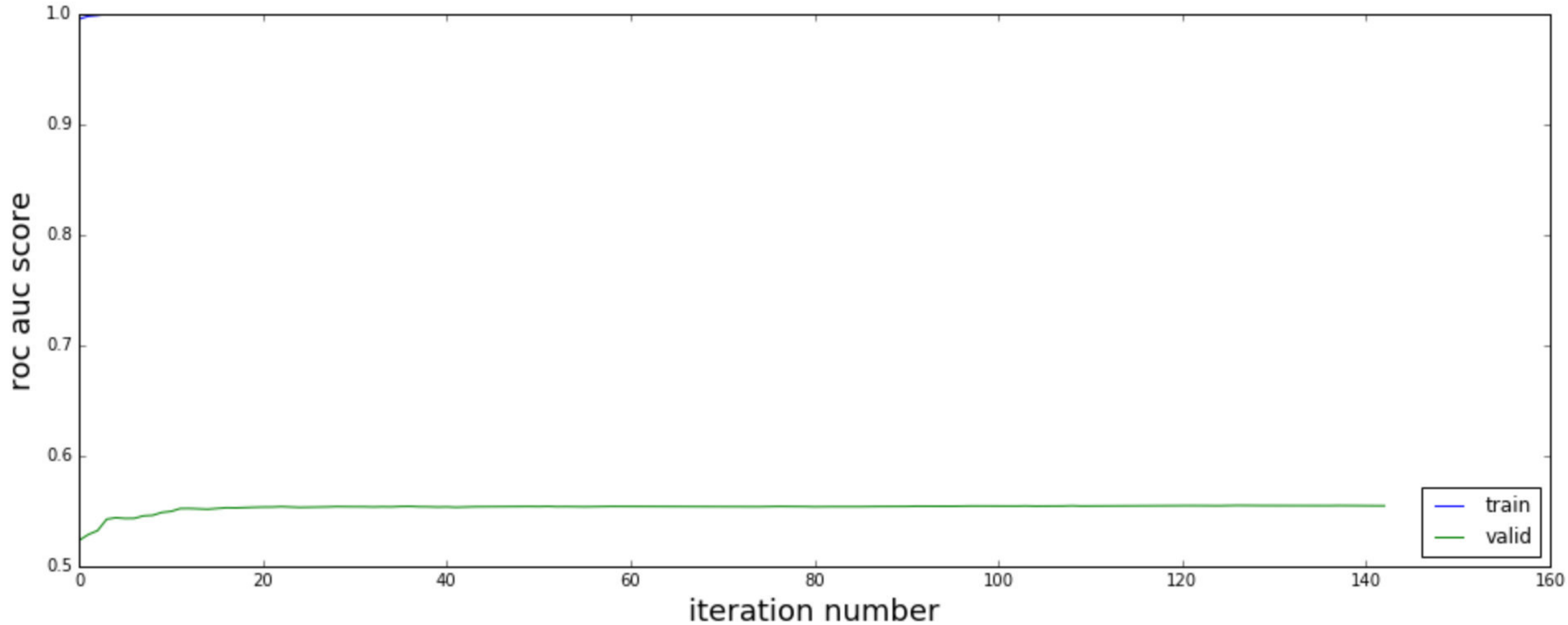
Springleaf example

```
dtrain = xgb.DMatrix(train_new, label=y_tr)
dvalid = xgb.DMatrix(val_new, label=y_val)

evallist = [(dtrain, 'train'),(dvalid, 'eval')]
evals_result3 = {}
model = xgb.train( xgb_par, dtrain,3000,evals=evallist,
verbose_eval=30,evals_result=evals_result3,early_stopping_rounds=50)
```



Overfit



Train

	feature	feature_label	feature_mean	target
8	Tver	2	0	0
9	Klin	0	0	0

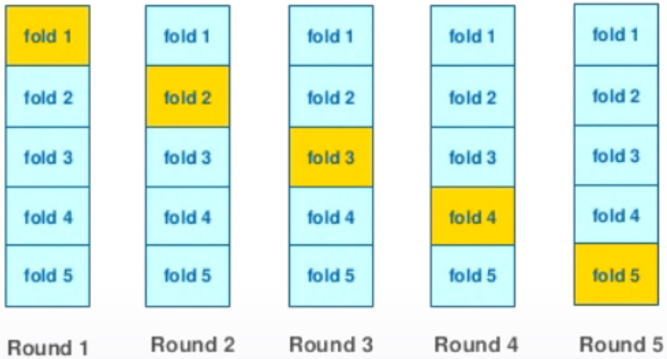
Validation

	feature	feature_label	feature_mean	target
10	Klin	0	1	1
11	Tver	2	1	1

Regularization

1. CV loop inside training data;
2. Smoothing;
3. Adding random noise;
4. Sorting and calculating expanding mean.

1.

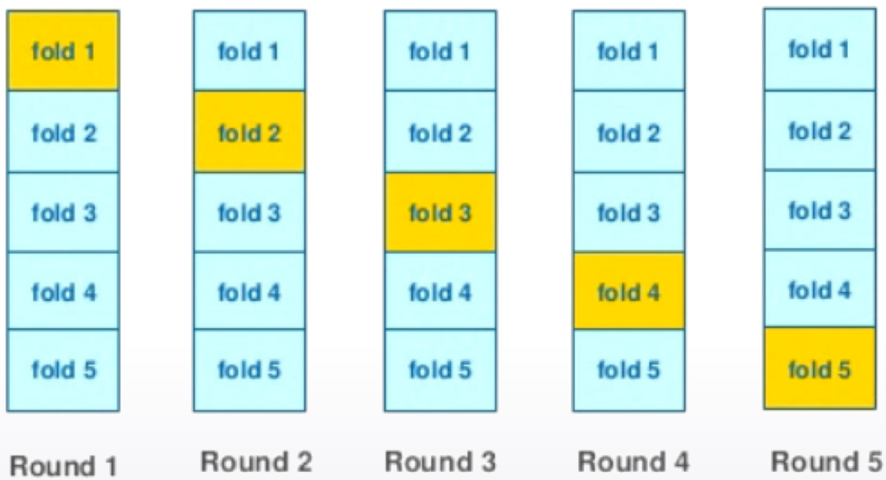


2. $$\frac{mean(target) * nrows + globalmean * alpha}{nrows + alpha}$$

Regularization. CV loop

- Robust and intuitive
- Usually decent results with 4-5 folds across different datasets
- Need to be careful with extreme situations like LOO

KFold scheme



Regularization. CV loop

```
y_tr = df_tr['target'].values #target variable
skf = StratifiedKFold(y_tr,5, shuffle=True,random_state=123)

for tr_ind, val_ind in skf:
    X_tr, X_val = df_tr.iloc[tr_ind], df_tr.iloc[val_ind]
    for col in cols: #iterate though the columns we want to encode
        means = X_val[col].map(X_tr.groupby(col).target.mean())
        X_val[col+'_mean_target'] = means
    train_new.iloc[val_ind] = X_val

prior = df_tr['target'].mean() #global mean
train_new.fillna(prior,inplace=True) #fill NANs with global mean
```

Regularization. CV loop

- Perfect feature for LOO scheme
- Target variable leakage is still present even for KFold scheme

Leave-one-out

	feature	feature_mean	target
0	Moscow	0.50	0
1	Moscow	0.25	1
2	Moscow	0.25	1
3	Moscow	0.50	0
4	Moscow	0.50	0

Regularization.Smoothing

- Alpha controls the amount of regularization
- Only works together with some other regularization method

$$\frac{mean(target) * nrows + globalmean * alpha}{nrows + alpha}$$

Regularization. Noise

- Noise degrades the quality of encoding
- How much noise should we add?
- Usually used together with LOO

Regularization. Expanding mean

- Least amount of leakage
- No hyper parameters
- Irregular encoding quality
- Built - in in CatBoost

```
cumsum = df_tr.groupby(col)['target'].cumsum() - df_tr['target']
cumcnt = df_tr.groupby(col).cumcount()
train_new[col+'_mean_target'] = cumsum/cumcnt
```

Regularization. Conclusion

- There are a lot ways to regularize mean encodings
- Unending battle with target variable leakage
- CV loop or Expanding mean for practical tasks

Generalizations and extensions

- Using target variable in different tasks. Regression, multiclass
- Domains with many-to-many relations
- Timeseries
- Encoding interactions and numerical features

Regression and multiclass

- More statistics for regression tasks. Percentiles, std, distribution bins.
- Introducing new information for one vs all classifiers in multi class tasks

Many-to-many relations

- Cross product of entities
- Statistics from vectors

User_id	APPS	Target
10	APP1; APP2; APP3	0
11	APP4; APP1	1
12	APP2	1
100	APP3; APP9	0

LONG REPRESENTATION

User_id	APP_id	Target
10	APP1	0
10	APP2	0
10	APP3	0
11	APP4	1
11	APP1	1

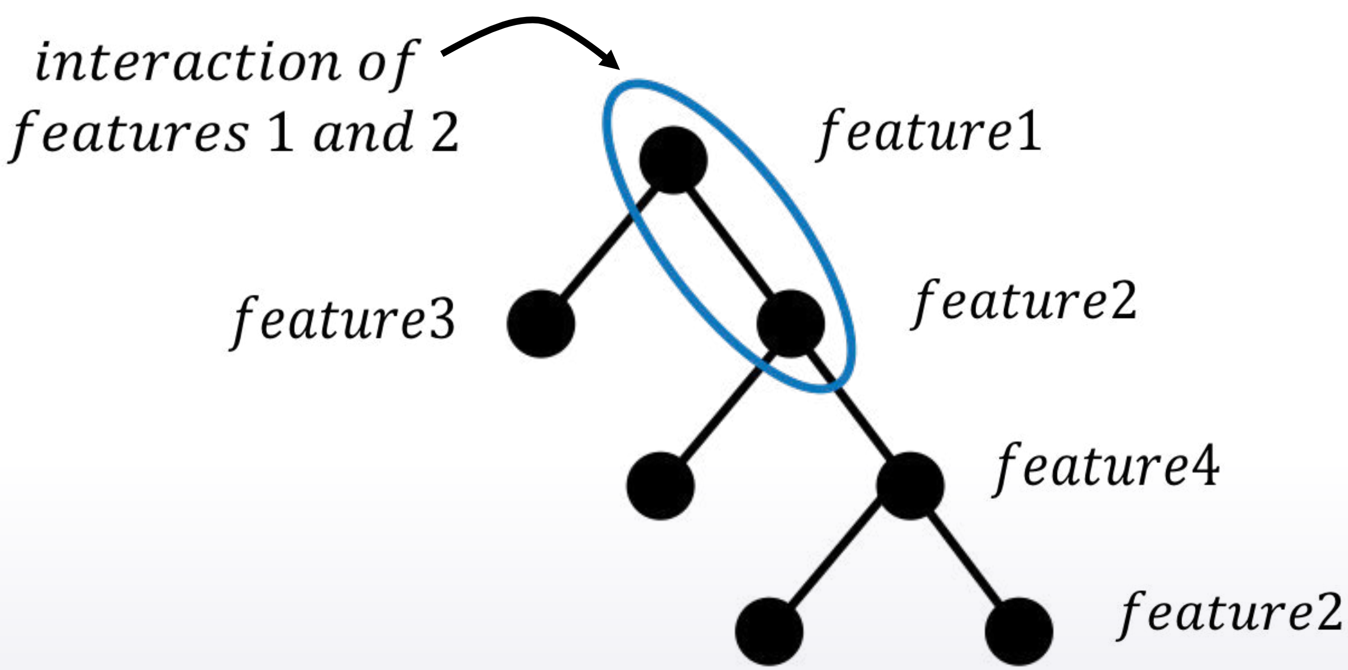
Time series

- Time structure allows us to make a lot of complicated features.
- Rolling statistics of target variable

Day	User	Spend	Amount	Prev_user	Prev_spend_avg
1	101	FOOD	2.0	0.0	0.0
1	101	GAS	4.0	0.0	0.0
1	102	FOOD	3.0	0.0	0.0
2	101	GAS	4.0	6.0	4.0
2	101	TV	8.0	6.0	0.0
2	102	FOOD	2.0	3.0	2.5

Interactions and numerical features

- Analyzing fitted model
- Binning numeric and selecting interactions



Amazon.com

Amazon.com - Employee Access Challenge Competition

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
cat_boost1.csv	a few seconds ago	0 seconds	0 seconds	0.91581

Complete

Jump to your position on the leaderboard

Your most recent submission

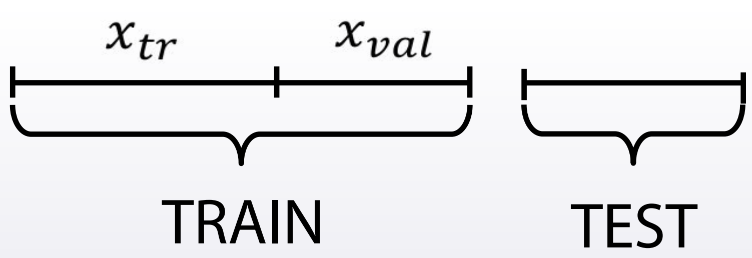
Name	Submitted	Wait time	Execution time	Score
lgb1.csv	just now	0 seconds	0 seconds	0.87209

Complete

Jump to your position on the leaderboard

Correct validation reminder

- Local experiments:
  - Estimate encodings on X_tr
  - Map them to X_tr and X_val
  - Regularize on X_tr
  - Validate model on X_tr/ X_val split
- Submission:
  - Estimate encodings on whole Train data
  - Map them to Train and Test
  - Regularize on Train
  - Fit on Train



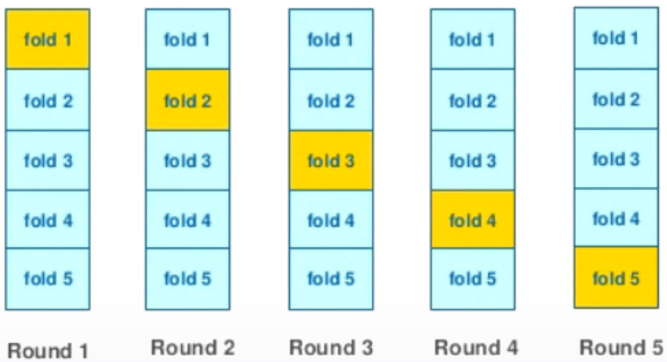
End

- Main advantages:
  - Compact transformation of categorical variables
  - Powerful basis for feature engineering
- Disadvantages:
  - Need careful validation, there a lot of ways to overfit
  - Significant improvements only on specific datasets

Regularization

1. CV loop inside training data;
2. Smoothing;
3. Adding random noise;
4. Sorting and calculating expanding mean.

1.

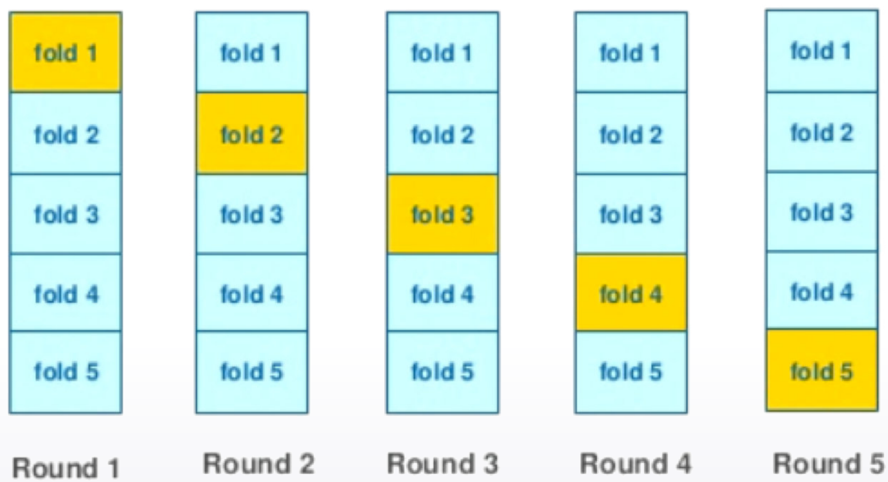


2. $$\frac{mean(target)*nrows+globalmean*alpha}{nrows+alpha}$$

Regularization. CV loop

- Robust and intuitive
- Usually decent results with 4-5 folds across different datasets
- Need to be careful with extreme situations like LOO

KFold scheme



Regularization. CV loop

```
y_tr = df_tr['target'].values #target variable
skf = StratifiedKFold(y_tr,5, shuffle=True,random_state=123)

for tr_ind, val_ind in skf:
    X_tr, X_val = df_tr.iloc[tr_ind], df_tr.iloc[val_ind]
    for col in cols: #iterate though the columns we want to encode
        means = X_val[col].map(X_tr.groupby(col).target.mean())
        X_val[col+'_mean_target'] = means
    train_new.iloc[val_ind] = X_val

prior = df_tr['target'].mean() #global mean
train_new.fillna(prior,inplace=True) #fill NANs with global mean
```

Regularization. CV loop

- Perfect feature for LOO scheme
- Target variable leakage is still present even for KFold scheme

Leave-one-out

	feature	feature_mean	target
0	Moscow	0.50	0
1	Moscow	0.25	1
2	Moscow	0.25	1
3	Moscow	0.50	0
4	Moscow	0.50	0

Regularization.Smoothing

- Alpha controls the amount of regularization
- Only works together with some other regularization method

$$\frac{mean(target) * nrows + globalmean * alpha}{nrows + alpha}$$

Regularization. Noise

- Noise degrades the quality of encoding
- How much noise should we add?
- Usually used together with LOO

Regularization. Expanding mean

- Least amount of leakage
- No hyper parameters
- Irregular encoding quality
- Built - in in CatBoost

```
cumsum = df_tr.groupby(col)['target'].cumsum() - df_tr['target']
cumcnt = df_tr.groupby(col).cumcount()
train_new[col+'_mean_target'] = cumsum/cumcnt
```

Regularization. Conclusion

- There are a lot ways to regularize mean encodings
- Unending battle with target variable leakage
- CV loop or Expanding mean for practical tasks