Topics:

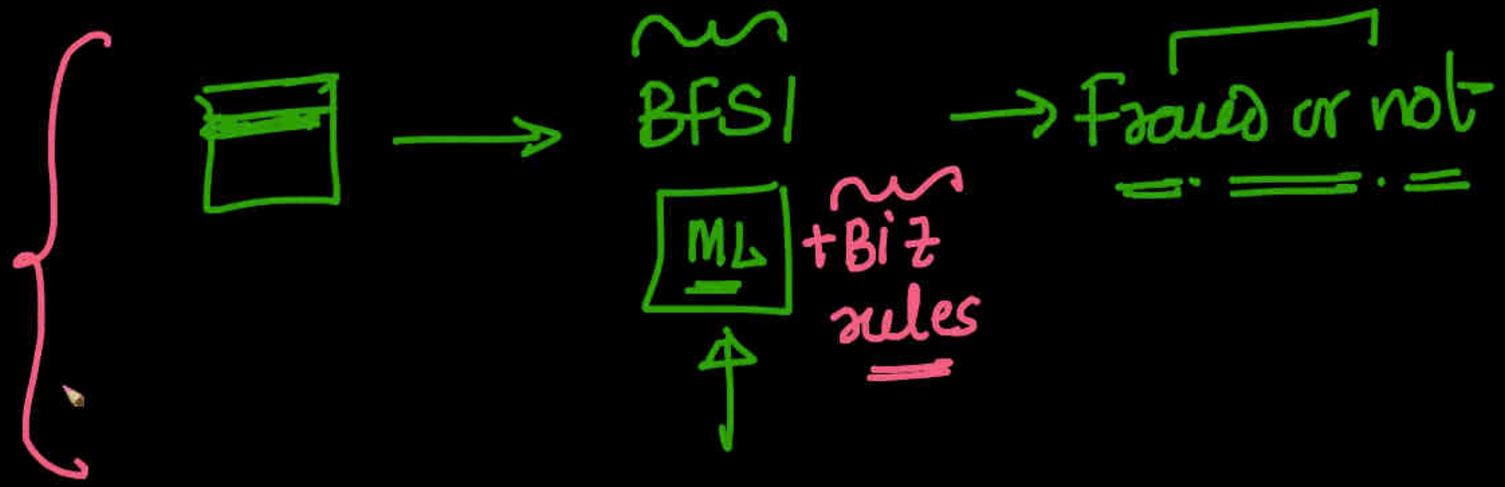
- Imbalanced data (code + Case-study) — Fraud-detection
- Decision Trees (intuition & Math)
- ✓
  - ROC & AU-ROC
  - GBDT → Xgboost

Next

- Algo
- Various scenarios

✓
- RF & GBDT

Credit card



Q

$y_i = 1$ → Fraud

$y_i = 0$ → not fraud

$x_i$: features corresponding to a Transaction

binary classfn.

imbalanced or balanced

Q

Business metric to measure the model perf- on:

✓ Recall ← what %age of fraudulent Xcns did our model flag as fraudulent

Problem: $y_i = 1 \ \forall i$

recall = 100%

✓ Precision : high

f1-score

OHE
Target encoding

# Category Encoders

A set of scikit-learn-style transformers for encoding categorical variables into numeric with different techniques. While ordinal, one-hot, and hashing encoders have similar equivalents in the existing scikit-learn version, the transformers in this library all share a few useful properties:

- First-class support for pandas dataframes as an input (and optionally as output)
- Can explicitly configure which columns in the data are encoded by name or index, or infer non-numeric columns regardless of input type
- Can drop any columns with very low variance based on training set optionally
- Portability: train a transformer on data, pickle it, reuse it later and get the same thing out.
- Full compatibility with sklearn pipelines, input an array-like dataset like any other transformer

## Usage

install as:

```
pip install category_encoders
```

or

```
conda install -c conda-forge category_encoders
```

To use:

```
[ ]  import pandas as pd
     import numpy as np
     from numpy import argmax

     from datetime import date, time, timedelta
     import pendulum # for time formatting

     import matplotlib.pyplot as plt
     import seaborn as sns

     import category_encoders as ce # for categorical encoding
     from sklearn.preprocessing import LabelEncoder, StandardScaler

     from sklearn.model_selection import train_test_split, cross_val_score
     from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, precision_re
```

```
[ ]  !wget "https://drive.google.com/uc?export=download&id=1HC7MfMyg4KphO51bc21JKIxeubmcl8C_" -O orders.csv
```

```
--2022-05-04 07:24:45--  https://drive.google.com/uc?export=download&id=1HC7MfMyg4KphO51bc21JKIxeubmcl8C_
Resolving drive.google.com (drive.google.com)... 172.253.122.138, 172.253.122.102, 172.253.122.100, ...
```

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:17

imbalanced Data.ipynb - Colab | plot (-x*log(x)-(1-x)*log(1-x)) | Category Encoders — Categor

colab.research.google.com/drive/1aJN0epwvGFkdUmxRBpBkxH6gykOnUagm#scrollTo=dp3Jn34iJCBE

+ Code + Text

RAM Disk

```
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packa
Installing collected packages: category-encoders
Successfully installed category-encoders-2.4.0
```

[2,4,6,8,1,5,12]

```
[ ] import pandas as pd
    import numpy as np
    from numpy import argmax

    from datetime import date, time, timedelta
    import pendulum # for time formatting

    import matplotlib.pyplot as plt
    import seaborn as sns

    import category_encoders as ce # for categorical encoding
    from sklearn.preprocessing import LabelEncoder, StandardScaler

    from sklearn.model_selection import train_test_split, cross_val_score
    from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, precision_re
```

```
[ ] !wget "https://drive.google.com/uc?export=download&id=1HC7MfMyg4KphO51bc21JKIxeubmcl8C_" -O orders.csv
```

```
--2022-05-04 07:24:45--  https://drive.google.com/uc?export=download&id=1HC7MfMyg4KphO51bc21JKIxeubmcl8C
```

```
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packa
Installing collected packages: category-encoders
Successfully installed category-encoders-2.4.0
```

```
[ ] import pandas as pd
    import numpy as np
    from numpy import argmax

    from datetime import date, time, timedelta
    import pendulum # for time formatting

    import matplotlib.pyplot as plt
    import seaborn as sns

    import category_encoders as ce # for categorical encoding
    from sklearn.preprocessing import LabelEncoder, StandardScaler

    from sklearn.model_selection import train_test_split, cross_val_score
    from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, precision_re
```

```
[ ] !wget "https://drive.google.com/uc?export=download&id=1HC7MfMyg4KphO51bc21JKIxeubmcl8C_" -O orders.csv
```

```
--2022-05-04 07:24:45--  https://drive.google.com/uc?export=download&id=1HC7MfMyg4KphO51bc21JKIxeubmcl8C
```

```
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packa
Installing collected packages: category-encoders
Successfully installed category-encoders-2.4.0
```

```
[ ] import pandas as pd
    import numpy as np
    from numpy import argmax

    from datetime import date, time, timedelta
    import pendulum # for time formatting

    import matplotlib.pyplot as plt
    import seaborn as sns

    import category_encoders as ce # for categorical encoding
    from sklearn.preprocessing import LabelEncoder, StandardScaler

    from sklearn.model_selection import train_test_split, cross_val_score
    from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, precision_re
```

```
[ ] !wget "https://drive.google.com/uc?export=download&id=1HC7MfMyg4KphO51bc21JKIxeubmcl8C_" -O orders.csv
```

```
--2022-05-04 07:24:45--  https://drive.google.com/uc?export=download&id=1HC7MfMyg4KphO51bc21JKIxeubmcl8C
```

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:18

imbalanced Data.ipynb - Colab | plot (-x*log(x)-(1-x)*log(1-x)) | Category Encoders — Categor

colab.research.google.com/drive/1aJN0epwvGFkdUmxRBpBkxH6gykOnUagm#scrollTo=dp3Jn34iJCBE

Update

+ Code + Text

RAM
Disk

```
[ ] import pandas as pd
    import numpy as np
    from numpy import argmax

    from datetime import date, time, timedelta
    import pendulum # for time formatting

    import matplotlib.pyplot as plt
    import seaborn as sns

    import category_encoders as ce # for categorical encoding
    from sklearn.preprocessing import LabelEncoder, StandardScaler

    from sklearn.model_selection import train_test_split, cross_val_score
    from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, precision_re
```

```
[ ] !wget "https://drive.google.com/uc?export=download&id=1HC7MfMyg4KphO51bc21JKIxeubmcl8C_" -O orders.csv
```

```
--2022-05-04 07:24:45--  https://drive.google.com/uc?export=download&id=1HC7MfMyg4KphO51bc21JKIxeubmcl8C_
Resolving drive.google.com (drive.google.com)... 172.253.122.138, 172.253.122.102, 172.253.122.100, ...
Connecting to drive.google.com (drive.google.com)|172.253.122.138|:443... connected.
HTTP request sent, awaiting response... 303 See Other
Location: https://doc-0k-14-docs.googleusercontent.com/docs/securesc/ha0ro937gcuc717deffksulhg5h7mbp1/8bsmgm1ggd4mt1
```

10 / 10

[11]

```
       'amount', 'device', 'payment_id', 'customer_ip', 'customer_id',
       'payment_method', 'payment_method_provider', 'payment_method_bin',
       'payment_method_type', 'payment_method_product',
       'payment_method_card_category', 'payment_method_issuer_bank',
       'payment_method_issuer_country', 'is_fraudulent', 'time_diff',
       'created_at_hod_sin', 'created_at_hod_cos', 'created_dom_sin',
       'created_dom_cos', 'created_dow_sin', 'created_dow_cos',
       'created_wom_sin', 'created_wom_cos', 'experience_dom_sin',
       'experience_dom_cos', 'experience_dow_sin', 'experience_dow_cos',
       'experience_wom_sin', 'experience_wom_cos'],
      dtype='object')
```

```
df["is_fraudulent"].value_counts()
```

1 0

1:75

```
False    75817
True      1012
Name: is_fraudulent, dtype: int64
```

```
df.nunique() # unique values per feature
```

```
order_id                    76829
city                            2
category_name                  96
product_id                    353
product_name                  340
```

[11]

```
 'amount', 'device', 'payment_id', 'customer_ip', 'customer_id',
 'payment_method', 'payment_method_provider', 'payment_method_bin',
 'payment_method_type', 'payment_method_product',
 'payment_method_card_category', 'payment_method_issuer_bank',
 'payment_method_issuer_country', 'is_fraudulent', 'time_diff',
 'created_at_hod_sin', 'created_at_hod_cos', 'created_dom_sin',
 'created_dom_cos', 'created_dow_sin', 'created_dow_cos',
 'created_wom_sin', 'created_wom_cos', 'experience_dom_sin',
 'experience_dom_cos', 'experience_dow_sin', 'experience_dow_cos',
 'experience_wom_sin', 'experience_wom_cos'],
dtype='object')
```

```
df["is_fraudulent"].value_counts()
```

```
False    75817
True      1012
Name: is_fraudulent, dtype: int64
```

```
df.nunique() # unique values per feature
```

```
order_id                    76829
city                            2
category_name                  96
product_id                    353
product_name                  340
```

[11]

```
 'amount', 'device', 'payment_id', 'customer_ip', 'customer_id',
 'payment_method', 'payment_method_provider', 'payment_method_bin',
 'payment_method_type', 'payment_method_product',
 'payment_method_card_category', 'payment_method_issuer_bank',
 'payment_method_issuer_country', 'is_fraudulent', 'time_diff',
 'created_at_hod_sin', 'created_at_hod_cos', 'created_dom_sin',
 'created_dom_cos', 'created_dow_sin', 'created_dow_cos',
 'created_wom_sin', 'created_wom_cos', 'experience_dom_sin',
 'experience_dom_cos', 'experience_dow_sin', 'experience_dow_cos',
 'experience_wom_sin', 'experience_wom_cos'],
dtype='object')
```

```
df["is_fraudulent"].value_counts()
```

```
False    75817
True      1012
Name: is_fraudulent, dtype: int64
```

[1:1000] ✓

```
df.nunique() # unique values per feature
```

```
order_id                76829
city                        2
category_name              96
product_id                353
product_name              340
```

[11]

```
 'amount', 'device', 'payment_id', 'customer_ip', 'customer_id',
 'payment_method', 'payment_method_provider', 'payment_method_bin',
 'payment_method_type', 'payment_method_product',
 'payment_method_card_category', 'payment_method_issuer_bank',
 'payment_method_issuer_country', 'is_fraudulent', 'time_diff',
 'created_at_hod_sin', 'created_at_hod_cos', 'created_dom_sin',
 'created_dom_cos', 'created_dow_sin', 'created_dow_cos',
 'created_wom_sin', 'created_wom_cos', 'experience_dom_sin',
 'experience_dom_cos', 'experience_dow_sin', 'experience_dow_cos',
 'experience_wom_sin', 'experience_wom_cos'],
dtype='object')
```

```
df["is_fraudulent"].value_counts()
```

```
False    75817
True      1012
Name: is_fraudulent, dtype: int64
```

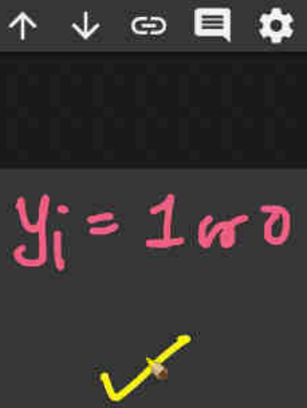
```
df.nunique() # unique values per feature
```

```
order_id                  76829
city                          2
category_name                96
product_id                  353
product_name                340
```

```
[ ] df["is_fraudulent"].value_counts()

False    75817
True      1012
Name: is_fraudulent, dtype: int64
```

```
df.nunique() # unique values per feature
```

```
order_id                      76829
city                              2
category_name                    96
product_id                      353
product_name                    340
amount                         8627
device                            5
payment_id                    76829
customer_ip                   44208
customer_id                   51109
payment_method                    1
payment_method_provider           5
payment_method_bin             6991
payment_method_type               8
payment_method_product          144
payment_method_card_category      2
```

```
[ ] df["is_fraudulent"].value_counts()
```

```
False    75817
True      1012
Name: is_fraudulent, dtype: int64
```

```
df.nunique() # unique values per feature
```

```
[redacted]                    76829
city                              2
category_name                    96
product_id                      353
product_name                    340
amount                         8627
device                            5
payment_id                    76829
customer_ip                   44208
customer_id                   51109
payment_method                    1
payment_method_provider           5
payment_method_bin             6991
payment_method_type               8
payment_method_product          144
payment_method_card_category      2
payment_method_issuer_bank     2046
```

```
[ ]  df["is_fraudulent"].value_counts()
```

```
False    75817
True      1012
Name: is_fraudulent, dtype: int64
```

```
df.nunique() # unique values per feature
```

```
order_id                      76829
city                              2
category_name                    96
product_id                      353
product_name                    340
amount                         8627
device                            5
payment_id                    76829
customer_ip                   44208
customer_id                   51109
payment_method                    1
payment_method_provider           5
payment_method_bin             6991
payment_method_type               8
payment_method_product          144
payment_method_card_category      2
payment_method_issuer_bank     2046
```

```
[ ] df["is_fraudulent"].value_counts()
```

```
False    75817
True      1012
Name: is_fraudulent, dtype: int64
```

```
df.nunique() # unique values per feature
```

```
order_id                      76829
city                              2
category_name                    96
product_id                      353
product_name                    340
amount                         8627
device                            5
payment_id                    76829
customer_ip                   44208
customer_id                   51109
payment_method                    1
payment_method_provider           5
payment_method_bin             6991
payment_method_type               8
payment_method_product          144
payment_method_card_category      2
payment_method_issuer_bank     2046
```

```
True      1012
Name: is_fraudulent, dtype: int64
```

```
df.nunique() # unique values per feature
```

```
order_id                        76829
city                                2
category_name                      96
product_id                        353
product_name                      340
amount                           8627
device                              5
payment_id                      76829
customer_ip                     44208
customer_id                     51109
payment_method                      1
payment_method_provider             5
payment_method_bin               6991
payment_method_type                 8
payment_method_product            144
payment_method_card_category        2
payment_method_issuer_bank       2046
payment_method_issuer_country     151
is_fraudulent                       2
time_diff                         117
```

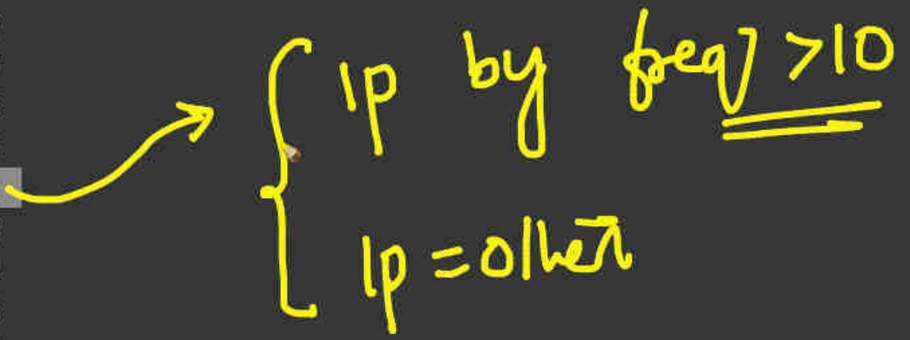
| | |
|---|---|
| order_id | 76829 |
| city | 2 |
| category_name | 96 |
| product_id | 353 |
| product_name | 340 |
| amount | 8627 |
| device | 5 |
| payment_id | 76829 |
| customer_ip | 44208 |
| customer_id | 51109 |
| payment_method | 1 |
| payment_method_provider | 5 |
| payment_method_bin | 6991 |
| payment_method_type | 8 |
| payment_method_product | 144 |
| payment_method_card_category | 2 |
| payment_method_issuer_bank | 2046 |
| payment_method_issuer_country | 151 |
| is_fraudulent | 2 |
| time_diff | 117 |
| created_at_hod_sin | 24 |
| created_at_hod_cos | 13 |
| created_dom_sin | 31 |
| created_dom_cos | 18 |
| created_dow_sin | 6 |
| created_dow_cos | 4 |
| created_wom_sin | 5 |
| created_wom_cos | 4 |

encrypted_IP

[redacted] →

IP1 → 20 → 5

| | |
|---|---|
| order_id | 76829 |
| city | 2 |
| category_name | 96 |
| product_id | 353 |
| product_name | 340 |
| amount | 8627 |
| device | 5 |
| payment_id | 76829 |
| customer_ip | 44208 |
| customer_id | 51109 |
| payment_method | 1 |
| payment_method_provider | 5 |
| payment_method_bin | 6991 |
| payment_method_type | 8 |
| payment_method_product | 144 |
| payment_method_card_category | 2 |
| payment_method_issuer_bank | 2046 |
| payment_method_issuer_country | 151 |
| is_fraudulent | 2 |
| time_diff | 117 |
| created_at_hod_sin | 24 |
| created_at_hod_cos | 13 |
| created_dom_sin | 31 |
| created_dom_cos | 18 |
| created_dow_sin | 6 |
| created_dow_cos | 4 |
| created_wom_sin | 5 |
| created_wom_cos | 4 |



| | |
|---|---|
| order_id | 76829 |
| city | 2 |
| category_name | 96 |
| product_id | 353 |
| product_name | 340 |
| amount | 8627 |
| device | 5 |
| payment_id | 76829 |
| customer_ip | 44208 |
| customer_id | 51109 |
| payment_method | 1 |
| payment_method_provider | 5 |
| payment_method_bin | 6991 |
| payment_method_type | 8 |
| payment_method_product | 144 |
| payment_method_card_category | 2 |
| payment_method_issuer_bank | 2046 |
| payment_method_issuer_country | 151 |
| is_fraudulent | 2 |
| time_diff | 117 |
| created_at_hod_sin | 24 |
| created_at_hod_cos | 13 |
| created_dom_sin | 31 |
| created_dom_cos | 18 |
| created_dow_sin | 6 |
| created_dow_cos | 4 |
| created_wom_sin | 5 |
| created_wom_cos | 4 |

| | |
|---|---|
| order_id | 76829 |
| city | 2 |
| category_name | 96 |
| product_id | 353 |
| product_name | 340 |
| amount | 8627 |
| device | 5 |
| payment_id | 76829 |
| customer_ip | 44208 |
| customer_id | 51109 |
| payment_method | 1 |
| payment_method_provider | 5 |
| payment_method_bin | 6991 |
| payment_method_type | 8 |
| payment_method_product | 144 |
| payment_method_card_category | 2 |
| payment_method_issuer_bank | 2046 |
| payment_method_issuer_country | 151 |
| is_fraudulent | 2 |
| time_diff | 117 |
| created_at_hod_sin | 24 |
| created_at_hod_cos | 13 |
| created_dom_sin | 31 |
| created_dom_cos | 18 |
| created_dow_sin | 6 |
| created_dow_cos | 4 |
| created_wom_sin | 5 |
| created_wom_cos | 4 |


Chrome File Edit View History Bookmarks Profiles Tab Window Help

imbalanced Data.ipynb - Colab | plot (-x*log(x)-(1-x)*log(1-x)) | Category Encoders — Categor | News - Latest News, Breaking

colab.research.google.com/drive/1aJN0epwvGFkdUmxRBpBkxH6gykOnUagm#scrollTo=R79B_o5dLPOR

+ Code + Text

RAM
Disk

[14]

0s

[15] df.shape

```
(76829, 30)
```

0s

```
# drop duplicates
if df.shape[0] == df.drop_duplicates().shape[0] :
    print('No duplicates Found')
else:
    duplicates = df.shape[0] - df.drop_duplicates().shape[0]
    print('{} duplicates found'.format(duplicates))
```

```
6821 duplicates found
```

```
df = df.drop_duplicates()
```

```
#NAN values
df.isna().sum()
```

24 / 24

```
[ ] #NAN values
    df.isna().sum()
```

```
city                          0
category_name                83
product_id                    0
amount                        0
device                        0
customer_ip                   0
customer_id                   0
payment_method_provider       0
payment_method_bin           46
payment_method_type          61
payment_method_product     1886
payment_method_card_category 2413
payment_method_issuer_bank   2838
payment_method_issuer_country  66
is_fraudulent                 0
time_diff                     0
created_at_hod_sin            0
created_at_hod_cos            0
created_dom_sin               0
created_dom_cos               0
created_dow_sin               0
created_dow_cos               0
created_wom_sin               0
```

```
created_dow_sin                0
created_dow_cos                0
created_wom_sin                0
created_wom_cos                0
experience_dom_sin             0
experience_dom_cos             0
experience_dow_sin             0
experience_dow_cos             0
experience_wom_sin             0
experience_wom_cos             0
dtype: int64
```

```
[ ]  df = df.dropna(axis = 0, how= 'any',
                    subset = ['payment_method_issuer_bank','payment_method_product', 'payment_method_card_category'])
```

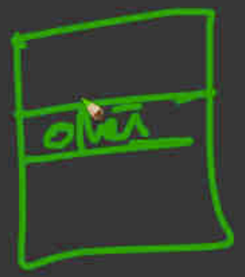
```
[ ]  df.isna().sum()
```

```
city                           0
category_name                 73
product_id                     0
amount                         0
device                         0
customer_ip                    0
customer_id                    0
payment_method_provider        0
```

```
[18]  subset = ['payment_method_issuer_bank','payment_method_product', 'payment_method_card_category'])
```

```
df.isna().sum()
```

```
city                          0
category_name                75
product_id                    0
amount                        0
device                        0
customer_ip                   0
customer_id                   0
payment_method_provider       0
payment_method_bin            0
payment_method_type           0
payment_method_product        0
payment_method_card_category  0
payment_method_issuer_bank    0
payment_method_issuer_country 11
is_fraudulent                 0
time_diff                     0
created_at_hod_sin            0
created_at_hod_cos            0
created_dom_sin               0
created_dom_cos               0
created_dow_sin               0
created_dow_cos               0
```



Chrome File Edit View History Bookmarks Profiles Tab Window Help

imbalanced Data.ipynb - Colab | plot (-x*log(x)-(1-x)*log(1-x)) | Category Encoders — Categor | News - Latest News, Breaking

colab.research.google.com/drive/1aJN0epwvGFkdUmxRBpBkxH6gykOnUagm#scrollTo=WSqiBdfvMx2_

+ Code + Text

RAM
Disk

[22]

0s

```
df.isna().sum()
```

```
city                          0
category_name                 0
product_id                    0
amount                        0
device                        0
customer_ip                   0
customer_id                   0
payment_method_provider       0
payment_method_bin            0
payment_method_type           0
payment_method_product        0
payment_method_card_category  0
payment_method_issuer_bank    0
payment_method_issuer_country 0
is_fraudulent                 0
time_diff                     0
created_at_hod_sin            0
created_at_hod_cos            0
created_dom_sin               0
created_dom_cos               0
created_dow_sin               0
created_dow_cos               0
created_wom_sin               0
```

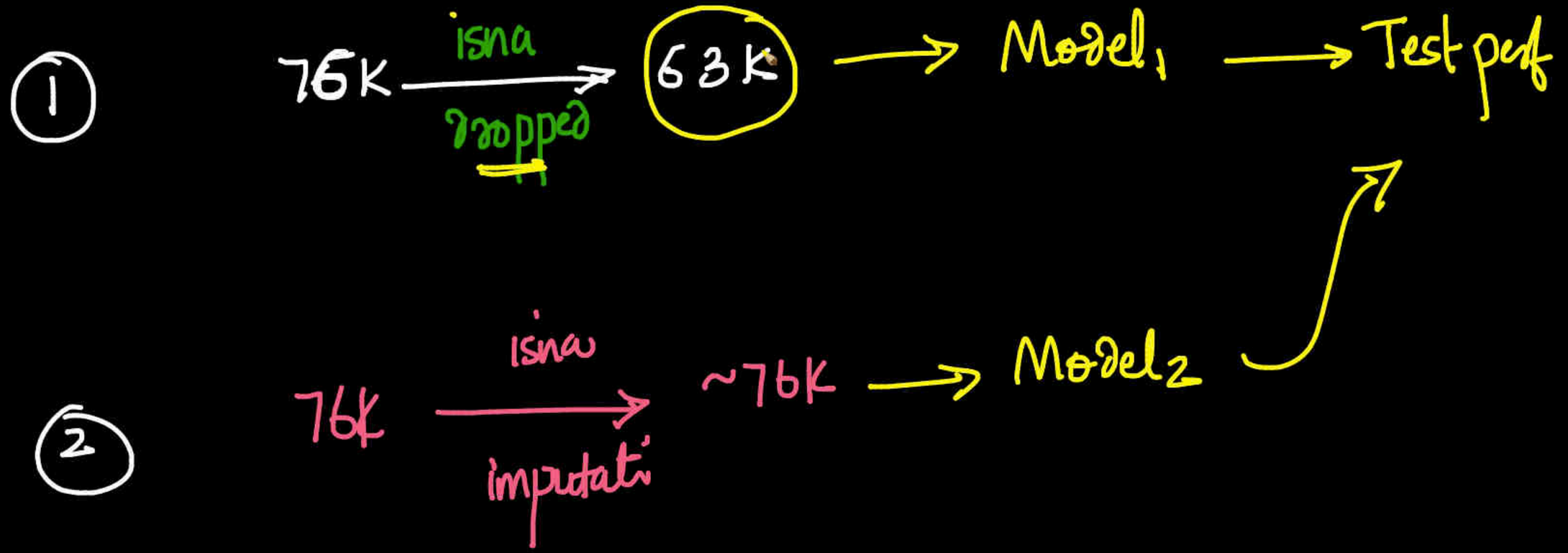
drop

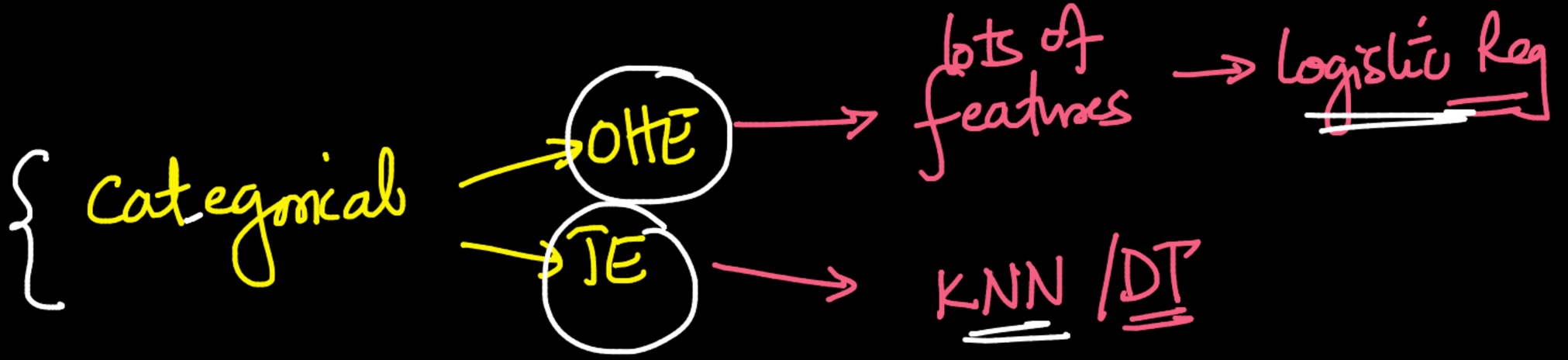
isna

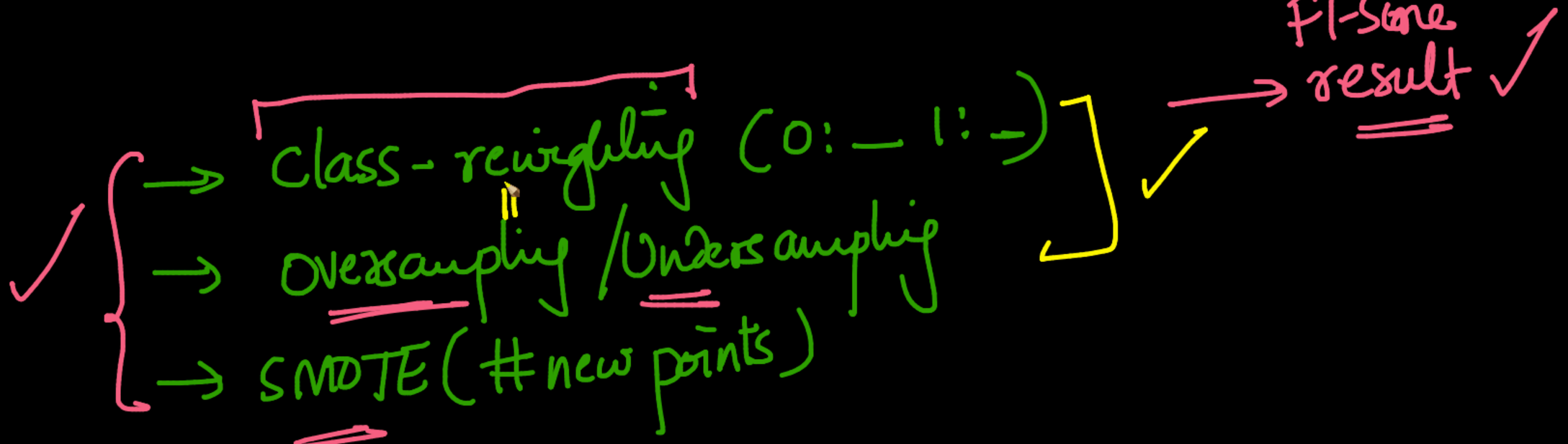
imputation

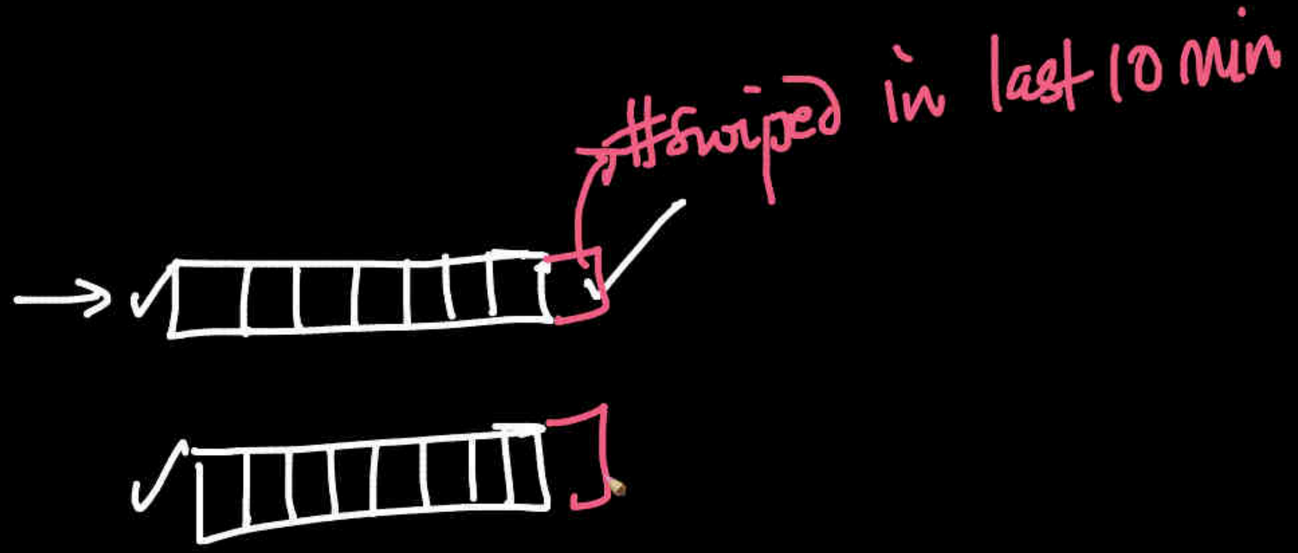
75K → 63K

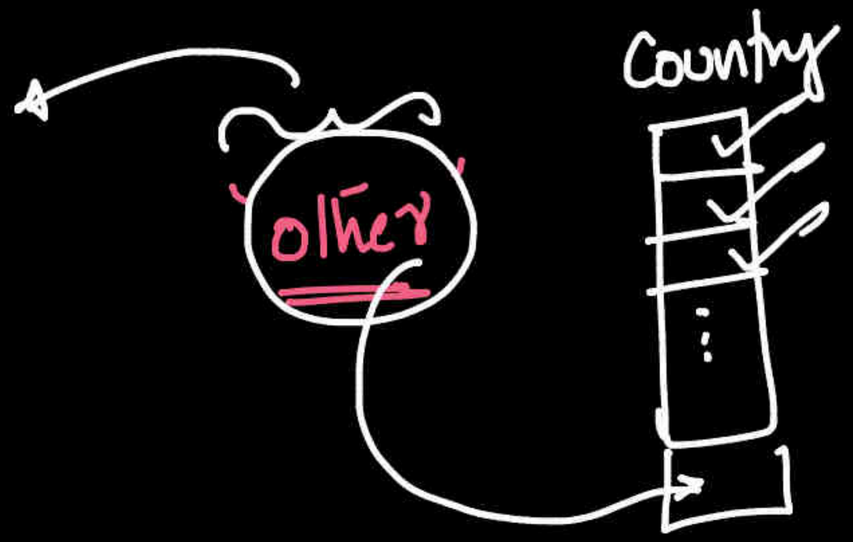
28 / 28




Categorical
OHE
TE
lots of features
Logistic Reg
KNN / DT


Class-weighting (0:_ 1:_)
Oversampling / Undersampling
SMOTE (# new points)
F1-Score result



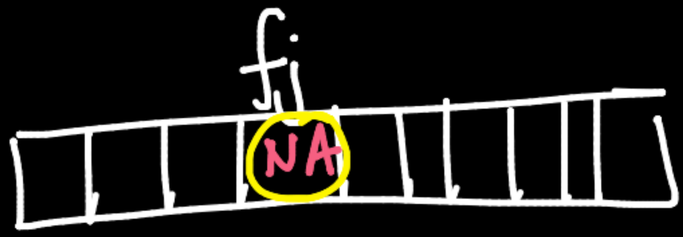



mode - imputation

Q

Test-time

$x_q$:




$y_q$:

X isna ✓ same, imputation

- fj: numeric → mean/median
- fj: Categorical → Mode

Split the data

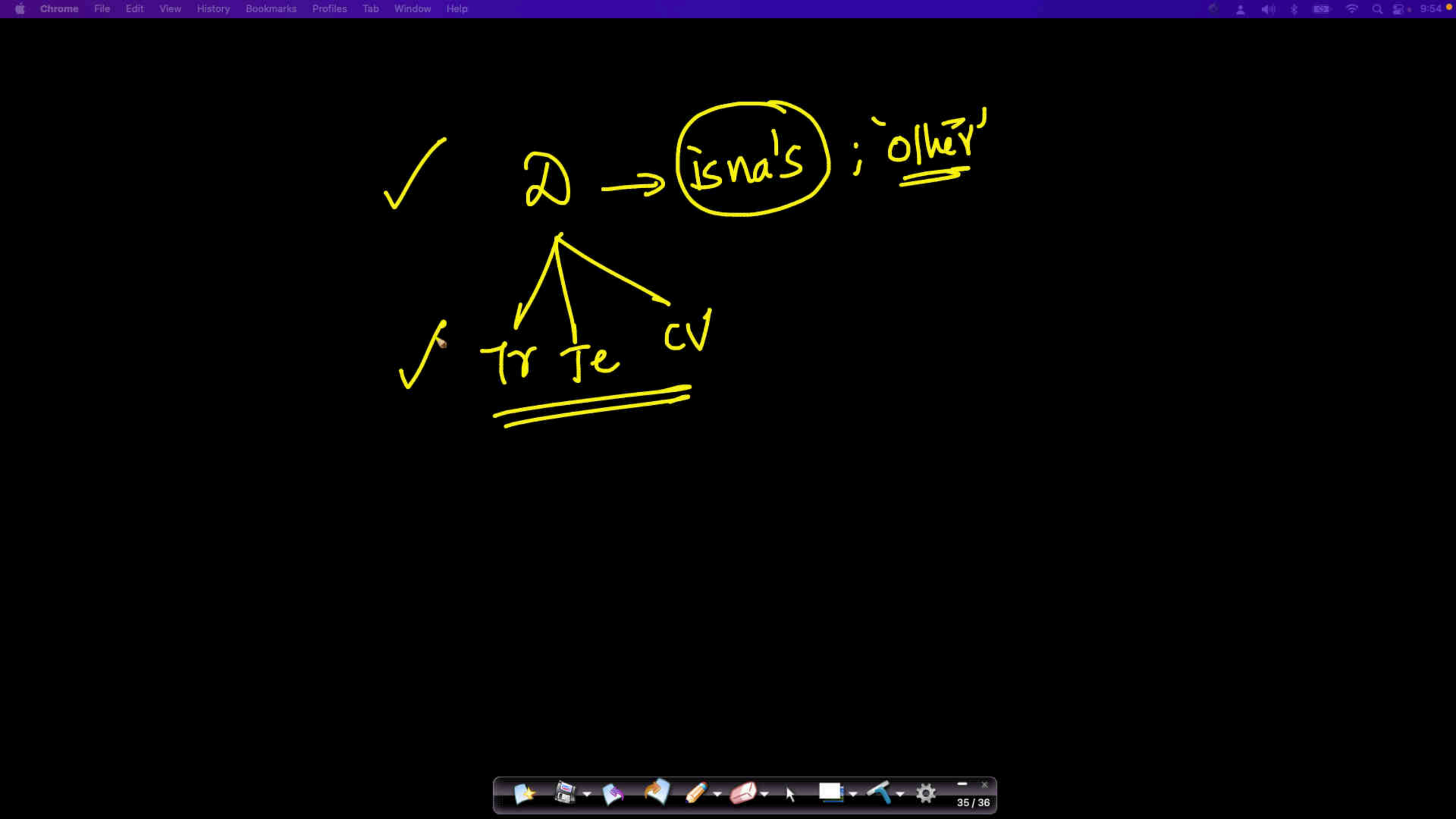
↓

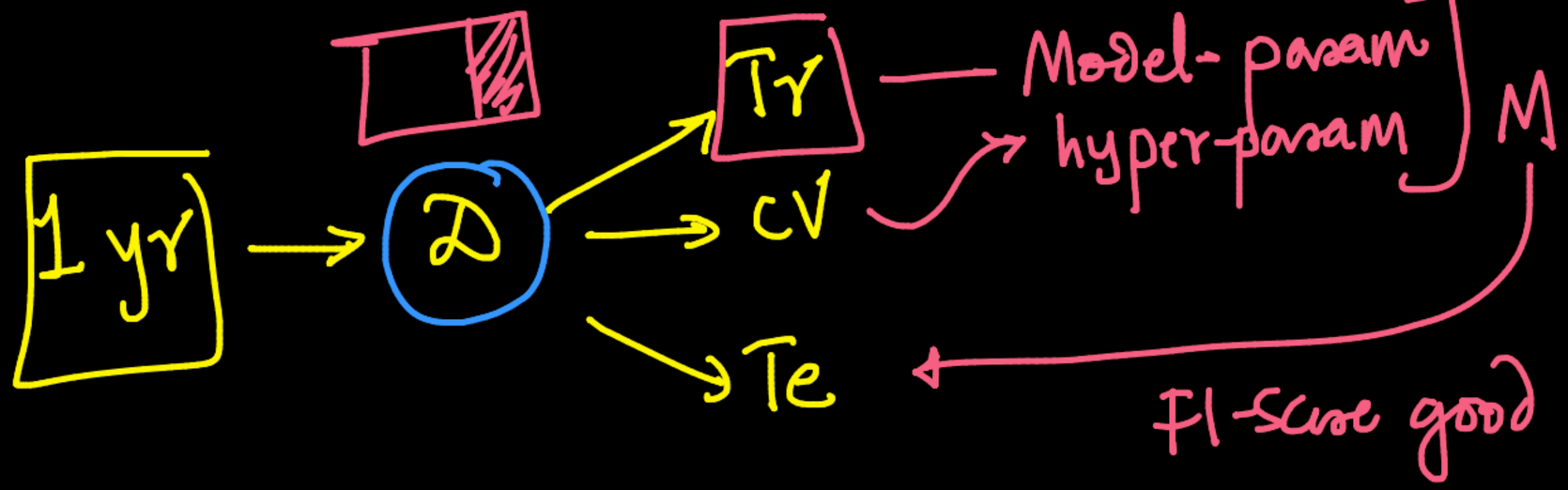
X_train : imputation

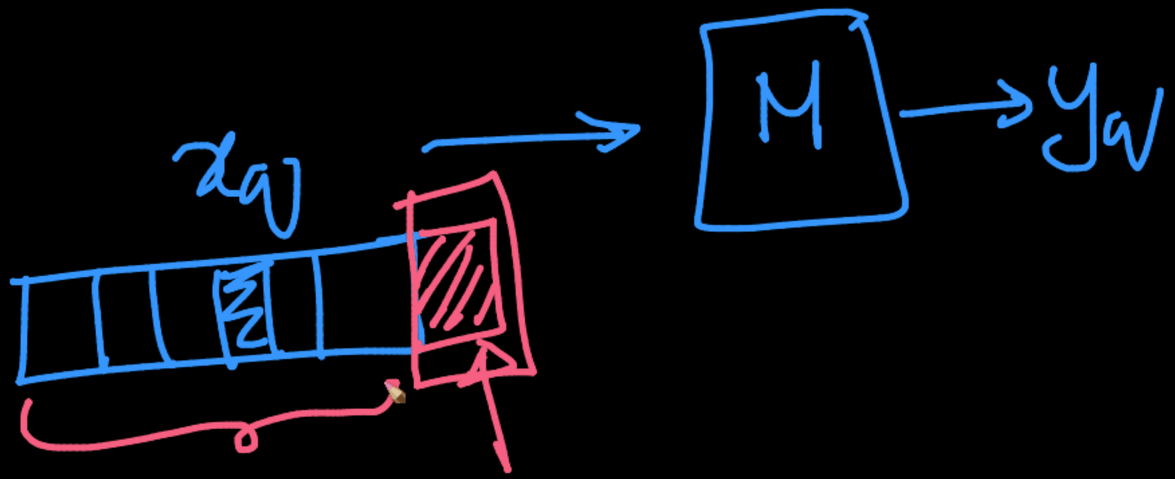
↓

X_val

X_test


D
isna's
; other
Tr
Te
CV

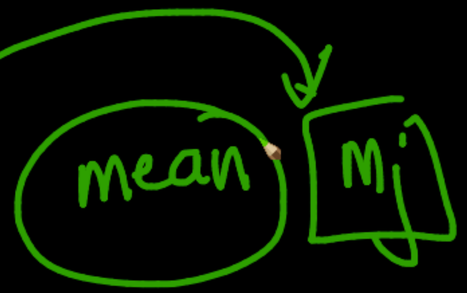

1 yr
D
Tr
CV
Te
Model- param
hyper-param
M
F1-score good


xq
M
yq

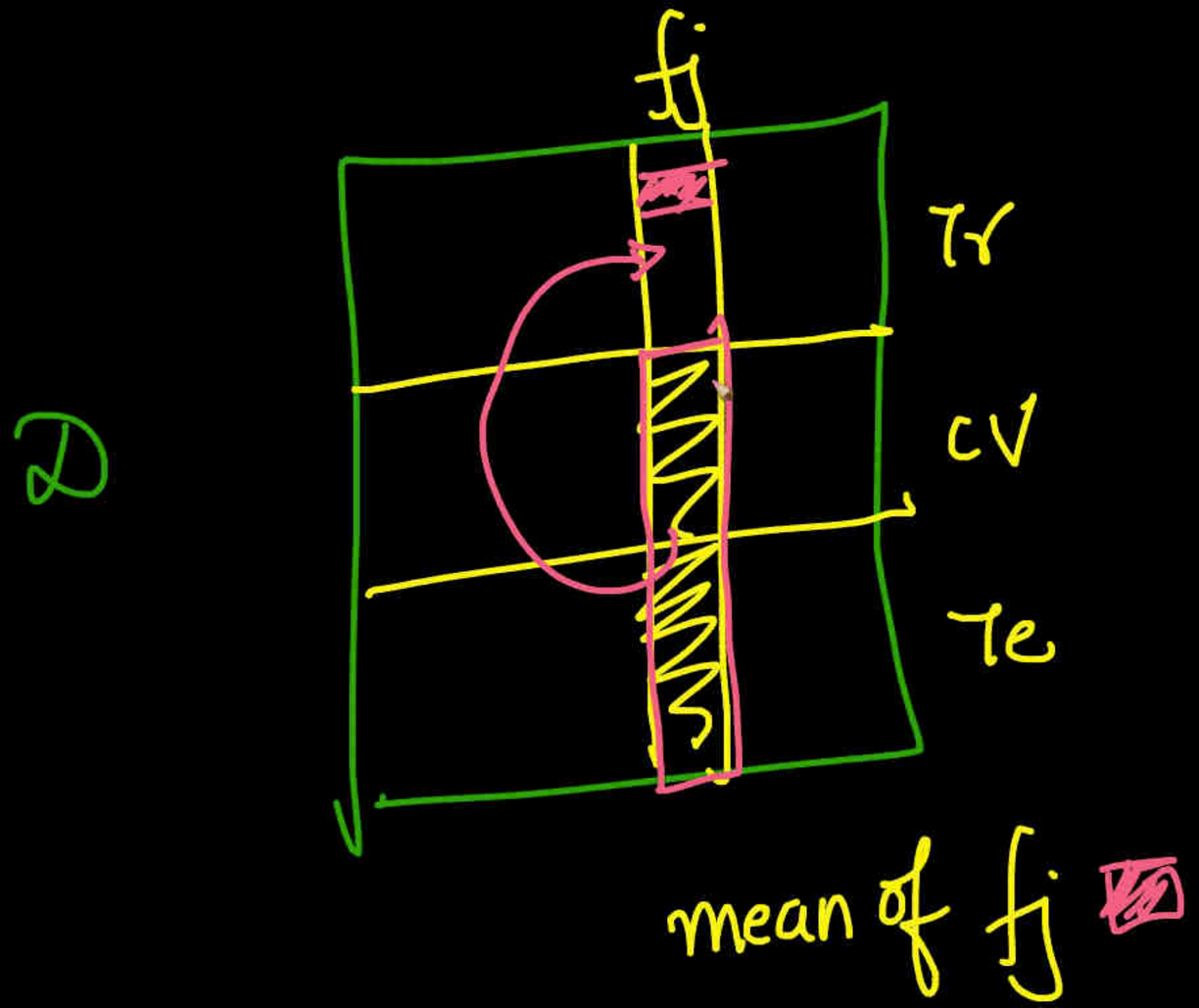
1. Donot NANs in production
   → isna's drop

2. NANs in production
   → imputation (mean/mode/median)

# Imputation:

1. $D$ — $D_{Tr}$, $D_{Te}$, $D_{CV}$
2. Impute using $D_{Tr}$ 

3. CV: 3 $m_j$'s
   Test


fj
Tr
CV
Te
D
mean of fj


```
[ ] set(X.columns) - set(X.select_dtypes(['number']).columns)
```

```
{'category_name',
 'city',
 'customer_id',
 'customer_ip',
 'device',
 'payment_method_card_category',
 'payment_method_issuer_bank',
 'payment_method_issuer_country',
 'payment_method_product',
 'payment_method_provider',
 'payment_method_type'}
```

```
[ ] # Target encoding using mean
    #
    ce_target = ce.TargetEncoder(cols = ['category_name','city','customer_id','customer_ip','device','payment_method_ca
    X_train = ce_target.fit_transform(X_train, y_train)
```

```
[ ] X_val = ce_target.transform(X_val)
    X_test = ce_target.transform(X_test)
```

```
    'customer_id',
    'customer_ip',
    'device',
    'payment_method_card_category',
    'payment_method_issuer_bank',
    'payment_method_issuer_country',
    'payment_method_product',
    'payment_method_provider',
    'payment_method_type'}
```

```
# Target encoding using mean
#
ce_target = ce.TargetEncoder(cols = ['category_name','city','customer_id','customer_ip','device','payment_method_ca
X_train = ce_target.fit_transform(X_train, y_train)
```

```
X_val = ce_target.transform(X_val)
X_test = ce_target.transform(X_test)
```

```
# Hyper-pram tuning
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.metrics import f1_score

train_scores = []
```

```
[ ]   'customer_id',
      'customer_ip',
      'device',
      'payment_method_card_category',
      'payment_method_issuer_bank',
      'payment_method_issuer_country',
      'payment_method_product',
      'payment_method_provider',
      'payment_method_type'}
```

non-numeric

```
[ ]  # Target encoding using mean
     #
     ce_target = ce.TargetEncoder(cols = ['category_name','city','customer_id','customer_ip','device','payment_method_ca
     X_train = ce_target.fit_transform(X_train, y_train)
```

```
[ ]  X_val = ce_target.transform(X_val)
     X_test = ce_target.transform(X_test)
```

```
[ ]  # Hyper-pram tuning
     from sklearn.linear_model import LogisticRegression
     from sklearn.pipeline import make_pipeline
     from sklearn.metrics import f1_score

     train_scores = []
```

```
    'customer_id',
    'customer_ip',
    'device',
    'payment_method_card_category',
    'payment_method_issuer_bank',
    'payment_method_issuer_country',
    'payment_method_product',
    'payment_method_provider',
    'payment_method_type'}
```

```
# Target encoding using mean
#
ce_target = ce.TargetEncoder(cols = ['category_name','city','customer_id','customer_ip','device','payment_method_ca
X_train = ce_target.fit_transform(X_train, y_train)
```

```
X_val = ce_target.transform(X_val)
X_test = ce_target.transform(X_test)
```

```
# Hyper-pram tuning
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.metrics import f1_score

train_scores = []
```

$P(y_i=1 \mid \text{city}=\text{singapore}) \leftarrow \frac{1000}{20{,}000}$

city: Singapore: 20,000 (let) 1000

Dubai: 30,000 1200

$\frac{1200}{30{,}000} = P(y=1 \mid \text{city}=\text{Dubai})$

$f_j$ : Categorical- features

$$c_1 = \frac{300}{1000} = 0.3$$

$$P(y=1 \mid f_j = c_2) = c_2 = \frac{4+\alpha}{10+\beta} = 0.4$$

$\vdots$

$c_{10}$


$\frac{1}{75} \sim P(y=1)$

Smoothing Strategies

$$C_1: \quad \frac{300 + \alpha}{1000 + \beta} \xrightarrow{\alpha \to 1,\ \beta \to 75} = \frac{301}{1075} \approx \frac{300}{1000}$$

$$C_2: \quad \frac{4 + \alpha}{10 + \beta} \xrightarrow{\alpha \to 1,\ \beta \to 75} = \frac{5}{85} \neq \frac{4}{10}$$

**return_df: bool**

boolean for whether to return a pandas DataFrame from transform (otherwise it will be a numpy array).

**handle_missing: str**

options are 'error', 'return_nan' and 'value', defaults to 'value', which returns the target mean.

**handle_unknown: str**

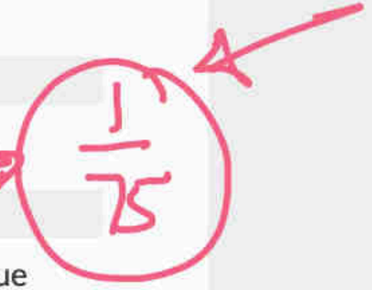
options are 'error', 'return_nan' and 'value', defaults to 'value', which returns the target mean.

**min_samples_leaf: int**

minimum samples to take category average into account.

**smoothing: float**

smoothing effect to balance categorical average vs prior. Higher value means stronger regularization. The value must be strictly bigger than 0.




## References

1

A Preprocessing Scheme for High-Cardinality Categorical Attributes in Classification and Prediction Problems, from

https://d

**return_df: bool**

boolean for whether to return a pandas DataFrame from transform (otherwise it will be a numpy array).

**handle_missing: str**

options are 'error', 'return_nan' and 'value', defaults to 'value', which returns the target mean.

**handle_unknown: str**

options are 'error', 'return_nan' and 'value', defaults to 'value', which returns the target mean.

**min_samples_leaf: int**

minimum samples to take category average into account.

**smoothing: float**

smoothing effect to balance categorical average vs prior. Higher value means stronger regularization. The value must be strictly bigger than 0.

## References

1

A Preprocessing Scheme for High-Cardinality Categorical Attributes in Classification and Prediction Problems, from

https://d

```
'device',
'payment_method_card_category',
'payment_method_issuer_bank',
'payment_method_issuer_country',
'payment_method_product',
'payment_method_provider',
'payment_method_type'}
```

```
# Target encoding using mean
#
ce_target = ce.TargetEncoder(cols = ['category_name','city','customer_id','customer_ip','device','payment_method_ca
X_train = ce_target.fit_transform(X_train, y_train)
```

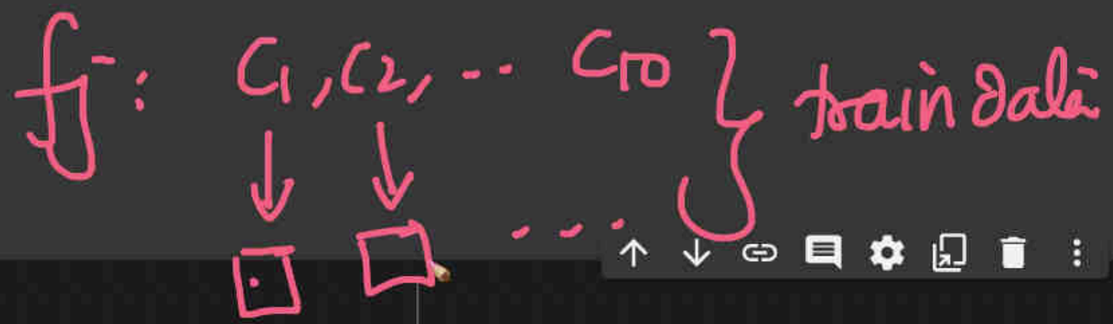
```
X_val = ce_target.transform(X_val)
X_test = ce_target.transform(X_test)
```

```
# Hyper-pram tuning
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.metrics import f1_score

train_scores = []
val_scores = []
scaler = StandardScaler()
```

+ Code + Text

RAM
Disk

```
[27]  'device',
      'payment_method_card_category',
      'payment_method_issuer_bank',
      'payment_method_issuer_country',
      'payment_method_product',
      'payment_method_provider',
      'payment_method_type'}
```




```
# Target encoding using mean
#
ce_target = ce.TargetEncoder(cols = ['category_name','city','customer_id','customer_ip','device','payment_method_ca
X_train = ce_target.fit_transform(X_train, y_train)
```

```
[ ]  X_val = ce_target.transform(X_val)
     X_test = ce_target.transform(X_test)
```

```
[ ]  # Hyper-pram tuning
     from sklearn.linear_model import LogisticRegression
     from sklearn.pipeline import make_pipeline
     from sklearn.metrics import f1_score

     train_scores = []
     val_scores = []
     scaler = StandardScaler()
```

```
# Hyper-pram tuning
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.metrics import f1_score

train_scores = []
val_scores = []
scaler = StandardScaler()
l=0.01
h= 1000.0
d=50.0

for la in np.arange(l,h,d):
  scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/la))
  scaled_lr.fit(X_train, y_train)
  train_y_pred = scaled_lr.predict(X_train)
  val_y_pred = scaled_lr.predict(X_val)
  train_score = f1_score(y_train, train_y_pred)
  val_score = f1_score(y_val, val_y_pred)
  train_scores.append(train_score)
  val_scores.append(val_score)
```

```
plt.figure()
```

Chrome File Edit View History Bookmarks Profiles Tab Window Help 10:14

imbalanced Data.ipynb - Colab | plot (-x*log(x)-(1-x)*log(1-x)) | Category Encoders — Categor | Target Encoder — Category En

colab.research.google.com/drive/1aJN0epwvGFkdUmxRBpBkxH6gykOnUagm#scrollTo=xGwDDiXTNq8W

Update

+ Code + Text

RAM Disk

```
[ ]  # Hyper-pram tuning
     from sklearn.linear_model import LogisticRegression
     from sklearn.pipeline import make_pipeline
     from sklearn.metrics import f1_score

     train_scores = []
     val_scores = []
     scaler = StandardScaler()
     l=0.01
     h= 1000.0
     d=50.0

     for la in np.arange(l,h,d):
       scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/la))
       scaled_lr.fit(X_train, y_train)
       train_y_pred = scaled_lr.predict(X_train)
       val_y_pred = scaled_lr.predict(X_val)
       train_score = f1_score(y_train, train_y_pred)
       val_score = f1_score(y_val, val_y_pred)
       train_scores.append(train_score)
       val_scores.append(val_score)
```

No class reweighting

```
[ ]  plt.figure()
```

51 / 52

```
[ ]  # Hyper-pram tuning
     from sklearn.linear_model import LogisticRegression
     from sklearn.pipeline import make_pipeline
     from sklearn.metrics import f1_score

     train_scores = []
     val_scores = []
     scaler = StandardScaler()
     l=0.01
     h= 1000.0
     d=50.0

     for la in np.arange(l,h,d):
       scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/la))
       scaled_lr.fit(X_train, y_train)
       train_y_pred = scaled_lr.predict(X_train)
       val_y_pred = scaled_lr.predict(X_val)
       train_score = f1_score(y_train, train_y_pred)
       val_score = f1_score(y_val, val_y_pred)
       train_scores.append(train_score)
       val_scores.append(val_score)
```

```
[ ]  plt.figure()
```

```
# Hyper-pram tuning
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.metrics import f1_score

train_scores = []
val_scores = []
scaler = StandardScaler()
l=0.01
h= 1000.0
d=50.0

for la in np.arange(l,h,d):
  scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/la))
  scaled_lr.fit(X_train, y_train)
  train_y_pred = scaled_lr.predict(X_train)
  val_y_pred = scaled_lr.predict(X_val)
  train_score = f1_score(y_train, train_y_pred)
  val_score = f1_score(y_val, val_y_pred)
  train_scores.append(train_score)
  val_scores.append(val_score)
```

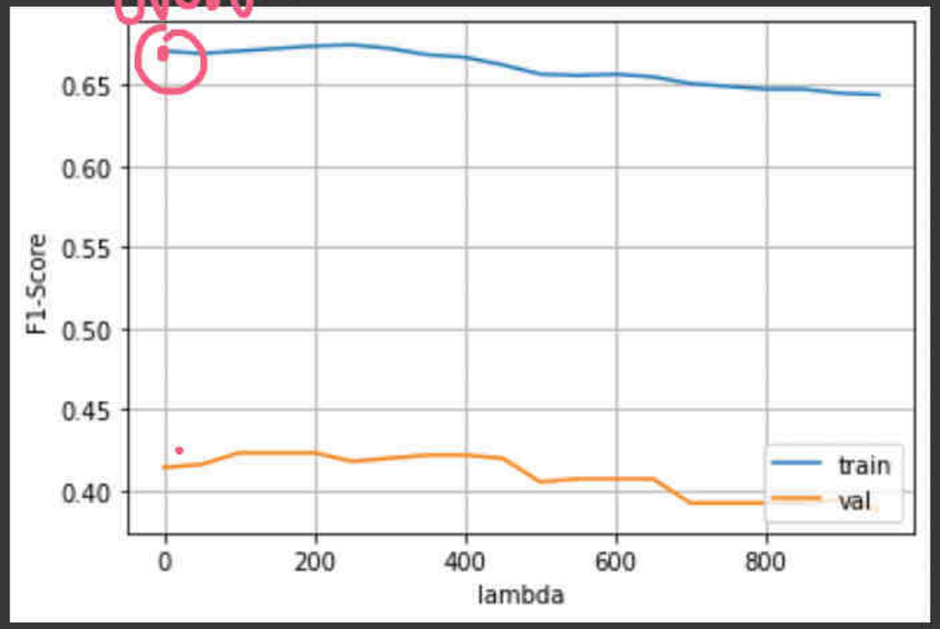
```
plt.figure()
```

```
# Hyper-pram tuning
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.metrics import f1_score

train_scores = []
val_scores = []
scaler = StandardScaler()
l=0.01
h= 1000.0
d=50.0

for la in np.arange(l,h,d):
  scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/la))
  scaled_lr.fit(X_train, y_train)
  train_y_pred = scaled_lr.predict(X_train)
  val_y_pred = scaled_lr.predict(X_val)
  train_score = f1_score(y_train, train_y_pred)
  val_score = f1_score(y_val, val_y_pred)
  train_scores.append(train_score)
  val_scores.append(val_score)
```

```
plt.figure()
```

```
plt.plot(list(np.arange(l,h,d)), val_scores, label="val")
plt.legend(loc='lower right')
plt.xlabel("lambda")
plt.ylabel("F1-Score")
plt.grid()
plt.show()
```

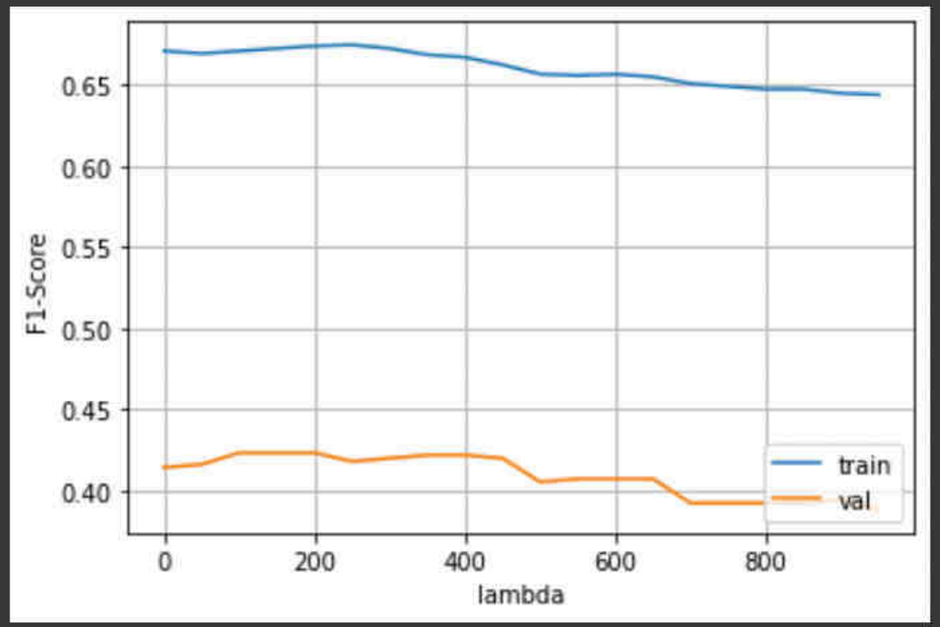
overfit




0.425 ← no class rebalancing

```
# minority class needs more re-weighting

# Hyper-pram tuning
```

```
plt.plot(list(np.arange(l,h,d)), val_scores, label="val")
plt.legend(loc='lower right')
plt.xlabel("lambda")
plt.ylabel("F1-Score")
plt.grid()
plt.show()
```

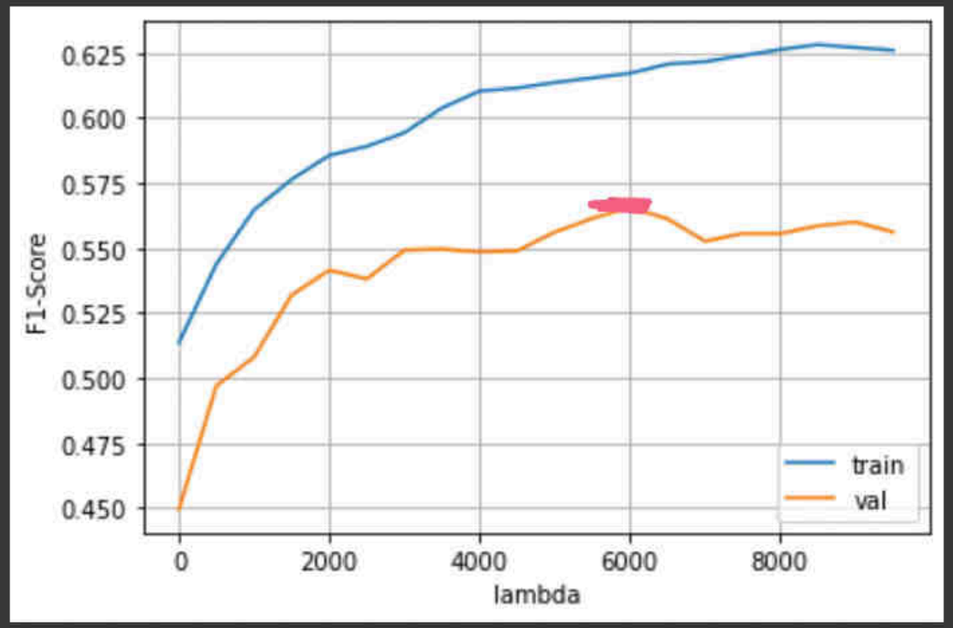



```
# minority class needs more re-weighting

# Hyper-pram tuning
```

```
[ ]  train_scores = []
     val_scores = []
     scaler = StandardScaler()
     l=0.01
     h= 1000.0
     d=50.0

     for la in np.arange(l,h,d):
       scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/la, class_weight={ 0:0.1, 1:0.9 }))
       scaled_lr.fit(X_train, y_train)
       train_y_pred = scaled_lr.predict(X_train)
       val_y_pred = scaled_lr.predict(X_val)
       train_score = f1_score(y_train, train_y_pred)
       val_score = f1_score(y_val, val_y_pred)
       train_scores.append(train_score)
       val_scores.append(val_score)

     #plotting
     plt.figure()
     plt.plot(list(np.arange(l,h,d)), train_scores, label="train")
     plt.plot(list(np.arange(l,h,d)), val_scores, label="val")
     plt.legend(loc='lower right')
     plt.xlabel("lambda")
     plt.ylabel("F1-Score")
     plt.grid()
```

1/9

```
plt.xlabel("lambda")
plt.ylabel("F1-Score")
plt.grid()
plt.show()
```




F1-score CV

1:1 → 0.425

1:9 → 0.55

hyper-param

1:75 → 0.565

1:200 → ?

```
best_idx = np.argmax(val_scores)
print(val_scores[best_idx])
```

```
0.5656565656565656
```


```
best_idx = np.argmax(val_scores)
print(val_scores[best_idx])
```

```
0.5656565656565656
```

0.565 : Val-data F1

```
# Model with lambda_best
best_idx = np.argmax(val_scores)
l_best = l+d*best_idx
scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/l_best, class_weight={ 0:0.01, 1:0.75 }))
scaled_lr.fit(X_train, y_train)

y_pred_test = scaled_lr.predict(X_test)
test_score = f1_score(y_test, y_pred_test)

print(test_score)
```

```
0.5721925133689839
```

f1-score test

```
# Alternatives metrics for imbalanced data:
# Brier-Score: 1/n*SUM[Y_i - Y_i_hat]^2 = AVG Squared Error
```

```
0.5656565656565656
```

```
[ ] # Model with lambda_best
    best_idx = np.argmax(val_scores)
    l_best = l+d*best_idx
    scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/l_best, class_weight={ 0:0.01, 1:0.75 }))
    scaled_lr.fit(X_train, y_train)

    y_pred_test = scaled_lr.predict(X_test)
    test_score = f1_score(y_test, y_pred_test)

    print(test_score)
```

```
0.5721925133689839
```

```
[30] # minority class needs more weighting

     # Hyper-pram tuning
     from sklearn.linear_model import LogisticRegression
     from sklearn.pipeline import make_pipeline
     from sklearn.metrics import f1_score

     train_scores = []
```

Chrome File Edit View History Bookmarks Profiles Tab Window Help 10:24

imbalanced Data.ipynb - Colab | plot (-x*log(x)-(1-x)*log(1-x)) | Category Encoders — Categor | Target Encoder — Category En

colab.research.google.com/drive/1aJN0epwvGFkdUmxRBpBkxH6gykOnUagm#scrollTo=JM5J906__gLj

+ Code + Text

RAM Disk

```
[33] # Model with lambda_best
     best_idx = np.argmax(val_scores)
     l_best = l+d*best_idx
     scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/l_best, class_weight={ 0:0.01, 1:0.75 }))
     scaled_lr.fit(X_train, y_train)

     y_pred_test = scaled_lr.predict(X_test)
     test_score = f1_score(y_test, y_pred_test)

     print(test_score)
```

```
0.6007751937984496
```

```
confusion_matrix(y_test, y_pred_test)
```

```
array([[13549,   175],
       [   31,   155]])
```

0

1

```
[30] # minority class needs more weighting

     # Hyper-pram tuning
     from sklearn.linear_model import LogisticRegression
     from sklearn.pipeline import make_pipeline
```

61 / 61

Chrome File Edit View History Bookmarks Profiles Tab Window Help 10:24

imbalanced Data.ipynb - Cola | plot (-x*log(x)-(1-x)*log(1-x)) | Category Encoders — Categor | Target Encoder — Category En

colab.research.google.com/drive/1aJN0epwvGFkdUmxRBpBkxH6gykOnUagm#scrollTo=JM5J906__gLj

Update

+ Code + Text

RAM Disk

```
[33] # Model with lambda_best
     best_idx = np.argmax(val_scores)
     l_best = l+d*best_idx
     scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/l_best, class_weight={ 0:0.01, 1:0.75 }))
     scaled_lr.fit(X_train, y_train)

     y_pred_test = scaled_lr.predict(X_test)
     test_score = f1_score(y_test, y_pred_test)

     print(test_score)
```

```
0.6007751937984496
```

155 / 186

```
confusion_matrix(y_test, y_pred_test)
```

```
array([[13549,   175],
       [   31,   155]])
```

```
[30] # minority class needs more weighting

     # Hyper-pram tuning
     from sklearn.linear_model import LogisticRegression
     from sklearn.pipeline import make_pipeline
```

62 / 62

Chrome File Edit View History Bookmarks Profiles Tab Window Help 10:25

imbalanced Data.ipynb - Cola | plot (-x*log(x)-(1-x)*log(1-x)) | Category Encoders — Categor | Target Encoder — Category En

colab.research.google.com/drive/1aJN0epwvGFkdUmxRBpBkxH6gykOnUagm#scrollTo=JM5J906__gLj

Update

+ Code + Text

RAM Disk

```
[33] # Model with lambda_best
     best_idx = np.argmax(val_scores)
     l_best = l+d*best_idx
     scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/l_best, class_weight={ 0:0.01, 1:0.75 }))
     scaled_lr.fit(X_train, y_train)

     y_pred_test = scaled_lr.predict(X_test)
     test_score = f1_score(y_test, y_pred_test)

     print(test_score)
```

```
0.6007751937984496
```

```
confusion_matrix(y_test, y_pred_test)
```

```
array([[13549,   175],
       [   31,   155]])
```

```
[30] # minority class needs more weighting

     # Hyper-pram tuning
     from sklearn.linear_model import LogisticRegression
     from sklearn.pipeline import make_pipeline
```

63 / 63

+ Code + Text

RAM Disk

[33]
```
# Model with lambda_best
best_idx = np.argmax(val_scores)
l_best = l+d*best_idx
scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/l_best, class_weight={ 0:0.01, 1:0.75 }))
scaled_lr.fit(X_train, y_train)

y_pred_test = scaled_lr.predict(X_test)
test_score = f1_score(y_test, y_pred_test)

print(test_score)
```

```
0.6007751937984496
```

```
confusion_matrix(y_test, y_pred_test)
```

```
array([[13549,   175],
       [   31,   155]])
```

[30]
```
# minority class needs more weighting

# Hyper-pram tuning
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
```

Chrome File Edit View History Bookmarks Profiles Tab Window Help 10:26

imbalanced Data.ipynb - Cola | plot (-x*log(x)-(1-x)*log(1-x)) | Category Encoders — Categor | Target Encoder — Category En

colab.research.google.com/drive/1aJN0epwvGFkdUmxRBpBkxH6gykOnUagm#scrollTo=JM5J906__gLj

+ Code + Text

RAM Disk

```
[33] # Model with lambda_best
     best_idx = np.argmax(val_scores)
     l_best = l+d*best_idx
     scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/l_best, class_weight={ 0:0.01, 1:0.75 }))
     scaled_lr.fit(X_train, y_train)

     y_pred_test = scaled_lr.predict(X_test)
     test_score = f1_score(y_test, y_pred_test)

     print(test_score)
```

```
0.6007751937984496
```

```
confusion_matrix(y_test, y_pred_test)
```

```
array([[13549,   175],
       [   31,   155]])
```

```
[30] # minority class needs more weighting

     # Hyper-pram tuning
     from sklearn.linear_model import LogisticRegression
     from sklearn.pipeline import make_pipeline
```

| actual \ predicted | 0 | 1 |
|---|---|---|
| 0 | 13549 | 175 |
| 1 | 31 | 155 |

Recall = 155/186 ~83%

Pr: 155/330 ~48%

65 / 65

Chrome File Edit View History Bookmarks Profiles Tab Window Help 10:30

imbalanced Data.ipynb - Colab | plot (-x*log(x)-(1-x)*log(1-x)) | Category Encoders — Categor | Target Encoder — Category En

colab.research.google.com/drive/1aJN0epwvGFkdUmxRBpBkxH6gykOnUagm#scrollTo=-BsklhG4-zo4

Update

+ Code + Text

RAM Disk

```
# Alternatives metrics for imbalanced data:
# Brier-Score: 1/n*SUM[Y_i - Y_i_hat]^2 = AVG Squared Error
# G-mean: SQRT(Sensitivity * Specificity)=SQRT(TPR*TNR) ----Seldom used
```

- F1-Score ✓
- AUC (ROC) — later

$$\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i^1\right)^2$$

$y_i$ → 0 or 1

$\hat{y}_i^1$ → Prob

→ avg squared error with probabilities

$$\sqrt{TPR * TNR}$$

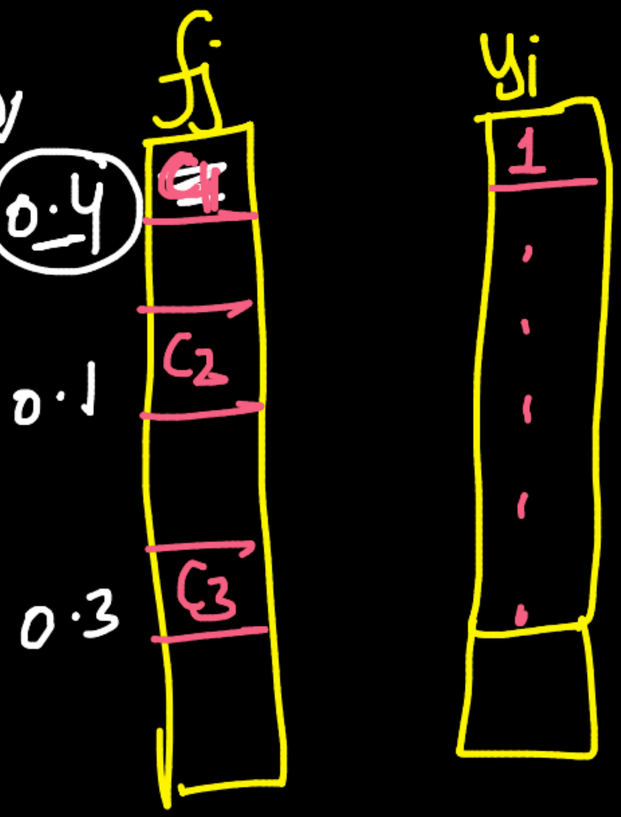
66 / 66

```
[ ]  # Alternatives metrics for imbalanced data:
     # Brier-Score: 1/n*SUM[Y_i - Y_i_hat]^2 = AVG Squared Error
     # G-mean: SQRT(Sensitivity * Specificity)=SQRT(TPR*TNR) ----Seldom used
```

```
[ ]
```

## Correlation:

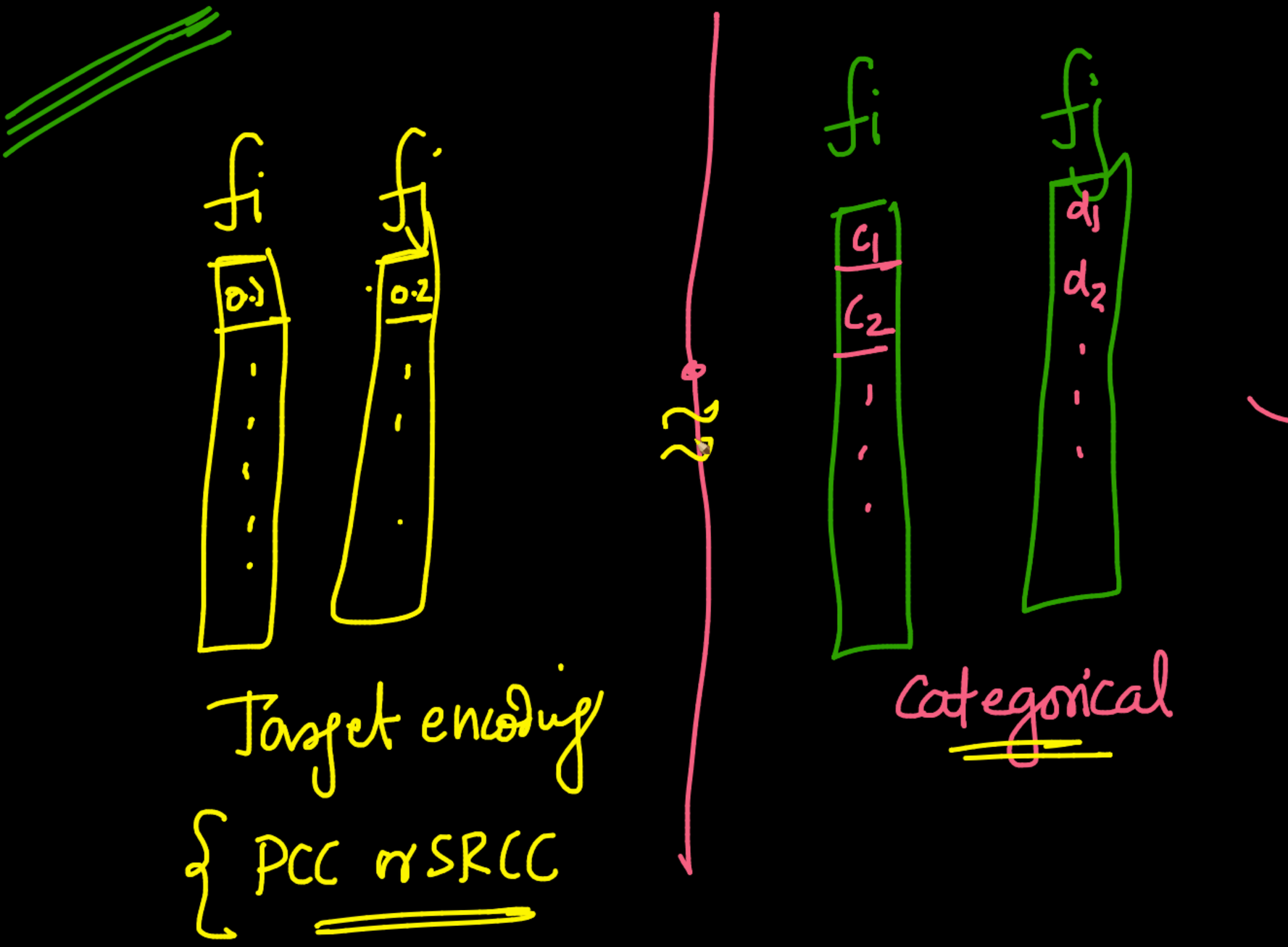
$P(y_i=1 \mid f_j=c_1)$ → 0.4




## Target encoding

$c_1 \xrightarrow{\text{highly corr}} y_i=1$

$P(y_i=1 \mid c_1) \rightarrow y_i=1$


fi
fj
o.1
o.2
Target encoding
{ PCC or SRCC
fi
fj
c1
c2
d1
d2
Categorical
χ²-test

✓ { Z-Test
or T-Test

✓ { ANOVA

fi
C1
C1
C2
C2
cat

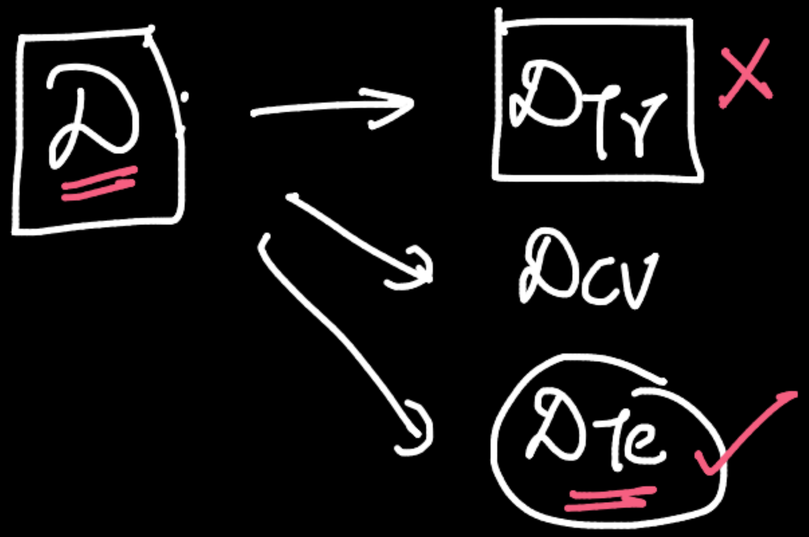
fj
0.1
0.2
numeric

fj | fi = C1
fj | fi = C2

C1 C2

✓ KL-Divergence

best $D_{train}$ ✓




time-series : TBSplitting


$$p(y_i = 1 \mid f_j = c_1) = \frac{40 + \alpha}{100 + \beta}$$

train

test

- F1-Score ✓ (↳ Pr, recall)
- G-mean: $\sqrt{sens * spec}$

Community portal
Recent changes
Upload file

Tools
What links here
Related changes
Special pages
Permanent link
Page information
Cite this page
Wikidata item

Print/export
Download as PDF
Printable version

Languages
العربية
Deutsch
Español
Français
한국어
Italiano
日本語
Português
中文
5 more
Edit links

identical sets of "classes" in both dimensions (each combination of dimension and class is a variable in the contingency table).

Contents [hide]



## Example [ edit ]

Given a sample of 12 individuals, 8 that have been diagnosed with cancer and 4 that are cancer-free, where individuals with cancer belong to class 1 (positive) and non-cancer individuals belong to class 0 (negative), we can display that data as follows:

| Individual Number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Actual Classification | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

Assume that we have a classifier that distinguishes between individuals with and without cancer in some way, we can take the 12 individuals and run them through the classifier. The classifier then makes 9 accurate predictions and misses 3: 2 individuals with cancer wrongly predicted as being cancer-free (sample 1 and 2), and 1 person without cancer that is wrongly predicted to have cancer (sample 9).

| Individual Number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Actual Classification | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| Predicted Classification | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |

Notice, that if we compare the actual classification set to the predicted classification set

**true negative (TN)**
A test result that correctly indicates the absence of a condition or characteristic

**false positive (FP)**
A test result which wrongly indicates that a particular condition or attribute is present

**false negative (FN)**
A test result which wrongly indicates that a particular condition or attribute is absent

---

**sensitivity, recall, hit rate, or true positive rate (TPR)**

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} = 1 - \mathrm{FNR}$$

**specificity, selectivity or true negative rate (TNR)**

$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{N}} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}} = 1 - \mathrm{FPR}$$

**precision or positive predictive value (PPV)**

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} = 1 - \mathrm{FDR}$$

**negative predictive value (NPV)**

$$\mathrm{NPV} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FN}} = 1 - \mathrm{FOR}$$

**miss rate or false negative rate (FNR)**

$$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{P}} = \frac{\mathrm{FN}}{\mathrm{FN} + \mathrm{TP}} = 1 - \mathrm{TPR}$$

**fall-out or false positive rate (FPR)**

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{N}} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} = 1 - \mathrm{TNR}$$

**false discovery rate (FDR)**

$$\frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TP}} = 1 - \mathrm{PPV}$$

The template for any binary confusion matrix uses the four kinds of results discussed above (true positives, false negatives, false positives, and true negatives) along with the positive and negative classifications. The four outcomes can be formulated in a 2×2 *confusion matrix*, as follows:

| | | Predicted condition | |
|---|---|---|---|
| | Total population = P + N | Positive (PP) | Negative (PN) |
| Actual condition | Positive (P) | True positive (TP) | False negative (FN) |
| | Negative (N) | False positive (FP) | True negative (TN) |

Sources: [11][12][13][14][15][16][17][18]

view · talk · edit

The color convention of the three data tables above were picked to match this confusion matrix, in order to easily differentiate the data.

Now, we can simply total up each type of result, substitute into the template, and create a confusion matrix that will concisely summarize the results of testing the classifier:

| | | Predicted condition | |
|---|---|---|---|
| | Total 8 + 4 = 12 | Cancer | Non-cancer |
| al condition | Cancer | 6 | 2 |

$$\frac{P + N}{TP + TN + FP + FN}$$

**balanced accuracy (BA)**

$$\mathrm{BA} = \frac{TPR + TNR}{2}$$

**F1 score**

is the harmonic mean of precision and sensitivity:

$$\mathrm{F_1} = 2 \times \frac{\mathrm{PPV} \times \mathrm{TPR}}{\mathrm{PPV} + \mathrm{TPR}} = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}$$

**phi coefficient (ϕ or $r_\phi$) or Matthews correlation coefficient (MCC)**

$$\mathrm{MCC} = \frac{\mathrm{TP} \times \mathrm{TN} - \mathrm{FP} \times \mathrm{FN}}{\sqrt{(\mathrm{TP} + \mathrm{FP})(\mathrm{TP} + \mathrm{FN})(\mathrm{TN} + \mathrm{FP})(\mathrm{TN} + \mathrm{FN})}}$$

**Fowlkes–Mallows index (FM)**

$$\mathrm{FM} = \sqrt{\frac{TP}{TP + FP} \times \frac{TP}{TP + FN}} = \sqrt{PPV \times TPR}$$

**informedness or bookmaker informedness (BM)**

$$\mathrm{BM} = \mathrm{TPR} + \mathrm{TNR} - 1$$

**markedness (MK) or deltaP (Δp)**

$$\mathrm{MK} = \mathrm{PPV} + \mathrm{NPV} - 1$$

**Diagnostic odds ratio (DOR)**

$$\mathrm{DOR} = \frac{\mathrm{LR+}}{\mathrm{LR-}}$$

*Sources:* Fawcett (2006),[1] Piryonesi and El-Diraby (2020),[2] Powers (2011),[3] Ting (2011),[4] CAWCR,[5] D. Chicco & G. Jurman (2020, 2021),[6][7] Tharwat (2018).[8]

Recent changes
Upload file

Tools
What links here
Related changes
Special pages
Permanent link
Page information
Cite this page
Wikidata item

Print/export
Download as PDF
Printable version

Languages
العربية
Deutsch
Español
Français
한국어
Italiano
日本語
Português
中文
5 more
Edit links

variable in the contingency table).

Contents [hide]



G-mean

## Example [ edit ]

Given a sample of 12 individuals, 8 that have been diagnosed with cancer and 4 that are cancer-free, where individuals with cancer belong to class 1 (positive) and non-cancer individuals belong to class 0 (negative), we can display that data as follows:

| Individual Number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Actual Classification | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

Assume that we have a classifier that distinguishes between individuals with and without cancer in some way, we can take the 12 individuals and run them through the classifier. The classifier then makes 9 accurate predictions and misses 3: 2 individuals with cancer wrongly predicted as being cancer-free (sample 1 and 2), and 1 person without cancer that is wrongly predicted to have cancer (sample 9).

| Individual Number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Actual Classification | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| Predicted Classification | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |

Notice, that if we compare the actual

there are 4 different outcomes that could result in any particular column. One, if the actual

A test result that correctly indicates the absence of a condition or characteristic

**false positive (FP)**

A test result which wrongly indicates that a particular condition or attribute is present

**false negative (FN)**

A test result which wrongly indicates that a particular condition or attribute is absent

---

**sensitivity, recall, hit rate, or true positive rate (TPR)**

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} = 1 - \mathrm{FNR}$$

**specificity, selectivity or true negative rate (TNR)**

$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{N}} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}} = 1 - \mathrm{FPR}$$

**precision or positive predictive value (PPV)**

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} = 1 - \mathrm{FDR}$$

**negative predictive value (NPV)**

$$\mathrm{NPV} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FN}} = 1 - \mathrm{FOR}$$

**miss rate or false negative rate (FNR)**

$$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{P}} = \frac{\mathrm{FN}}{\mathrm{FN} + \mathrm{TP}} = 1 - \mathrm{TPR}$$

**fall-out or false positive rate (FPR)**

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{N}} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} = 1 - \mathrm{TNR}$$

**false discovery rate (FDR)**

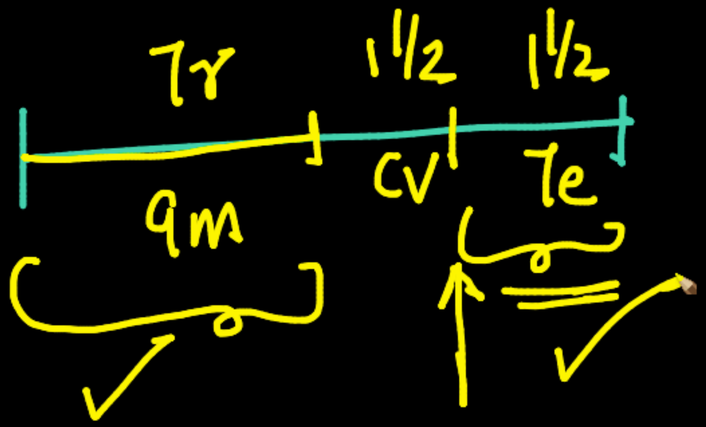
$$\mathrm{FDR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TP}} = 1 - \mathrm{PPV}$$

**false omission rate (FOR)**

76 / 76

Domain → F1, Gmean

random splitting → $D_{Tr}$, $D_{CV}$, $D_{Te}$




Time based Splitting ← things changes over time significantly

Search the docs ...

Under-sampling methods
Over-sampling methods
RandomOverSampler
SMOTE
SMOTENC
SMOTEN
ADASYN
BorderlineSMOTE
KMeansSMOTE
SVMSMOTE
Combination of over- and under-sampling methods
Ensemble methods
Batch generator for Keras
Batch generator for TensorFlow
Miscellaneous

# SMOTE

*class* imblearn.over_sampling.SMOTE(*, *sampling_strategy='auto', random_state=None, k_neighbors=5, n_jobs=None*) [source]

Class to perform over-sampling using SMOTE.

This object is an implementation of SMOTE - Synthetic Minority Over-sampling Technique as presented in [1].

Read more in the User Guide.

**Parameters:** **sampling_strategy** : *float, str, dict or callable, default='auto'*

Sampling information to resample the data set.

- When `float`, it corresponds to the desired ratio of the number of samples in the minority class over the number of samples in the majority class after resampling. Therefore, the ratio is expressed as $\alpha_{os} = N_{rm}/N_M$ where $N_{rm}$ is the number of samples in the minority class after resampling and $N_M$ is the number of samples in the majority class.

On this page

Examples using imblearn.over_sampling.SMO

Edit this page

Search the docs ...

Under-sampling methods
Over-sampling methods
RandomOverSampler
SMOTE
SMOTENC
SMOTEN
ADASYN
BorderlineSMOTE
KMeansSMOTE
SVMSMOTE
Combination of over- and under-sampling methods
Ensemble methods
Batch generator for Keras
Batch generator for TensorFlow
Miscellaneous

# SMOTE

***class*** **imblearn.over_sampling.SMOTE**(*, ***sampling_strategy='auto'***, ***random_state=None***, ***k_neighbors=5***, ***n_jobs=None***) [source]

Class to perform over-sampling using SMOTE.

This object is an implementation of SMOTE - Synthetic Minority Over-sampling Technique as presented in [1].

Read more in the User Guide.

**Parameters:** **sampling_strategy** : ***float, str, dict or callable, default='auto'***

Sampling information to resample the data set.

- When `float`, it corresponds to the desired ratio of the number of samples in the minority class over the number of samples in the majority class after resampling. Therefore, the ratio is expressed as $\alpha_{os} = N_{rm}/N_M$ where $N_{rm}$ is the number of samples in the minority class after resampling and $N_M$ is the number of samples in the majority class.

On this page
Examples using imblearn.over_sampling.SMO
Edit this page

Search the docs ...

Under-sampling methods
Over-sampling methods
RandomOverSampler
SMOTE
SMOTENC
SMOTEN
ADASYN
BorderlineSMOTE
KMeansSMOTE
SVMSMOTE
Combination of over- and under-sampling methods
Ensemble methods
Batch generator for Keras
Batch generator for TensorFlow
Miscellaneous

> **Warning**
>
> `float` is only available for **binary** classification. An error is raised for multi-class classification.

- When `str`, specify the class targeted by the resampling. The number of samples in the different classes will be equalized. Possible choices are:

  `'minority'`: resample only the minority class;
  `'not minority'`: resample all classes but the minority class;
  `'not majority'`: resample all classes but the majority class;
  `'all'`: resample all classes;
  `'auto'`: equivalent to `'not majority'`.

- When `dict`, the keys correspond to the targeted classes. The values correspond to the desired number of samples for each targeted class.
- When callable, function taking `y` and returns a `dict`. The keys correspond to the targeted classes. The values correspond to the desired number of samples for each class.

**random_state** : *int, RandomState instance, default=None*

Control the randomization of the algorithm.

- If int, `random_state` is the seed used by the random number generator;

On this page
Examples using `imblearn.over_sampling.SMO`
Edit this page

sampling technique," Journal of artificial intelligence research, 321-357, 2002.

Search the docs ...

Under-sampling methods
Over-sampling methods
RandomOverSampler
SMOTE
SMOTENC
SMOTEN
ADASYN
BorderlineSMOTE
KMeansSMOTE
SVMSMOTE
Combination of over- and under-sampling methods
Ensemble methods

## Examples

```
>>> from collections import Counter
>>> from sklearn.datasets import make_classification
>>> from imblearn.over_sampling import SMOTE
>>> X, y = make_classification(n_classes=2, class_sep=2,
... weights=[0.1, 0.9], n_informative=3, n_redundant=1, flip_y=0,
... n_features=20, n_clusters_per_class=1, n_samples=1000, random_state=10)
>>> print('Original dataset shape %s' % Counter(y))
Original dataset shape Counter({1: 900, 0: 100})
>>> sm = SMOTE(random_state=42)
>>> X_res, y_res = sm.fit_resample(X, y)
>>> print('Resampled dataset shape %s' % Counter(y_res))
Resampled dataset shape Counter({0: 900, 1: 900})
```

## Methods

| | |
|---|---|
| fit(X, y) | Check inputs and statistics of the sampler. |

sampling technique," Journal of artificial intelligence research, 321-357, 2002.

Search the docs ...

Under-sampling methods
Over-sampling methods
RandomOverSampler
SMOTE
SMOTENC
SMOTEN
ADASYN
BorderlineSMOTE
KMeansSMOTE
SVMSMOTE
Combination of over- and under-sampling methods
Ensemble methods

## Examples

```
>>> from collections import Counter
>>> from sklearn.datasets import make_classification
>>> from imblearn.over_sampling import SMOTE
>>> X, y = make_classification(n_classes=2, class_sep=2,
... weights=[0.1, 0.9], n_informative=3, n_redundant=1, flip_y=0,
... n_features=20, n_clusters_per_class=1, n_samples=1000, random_state=10)
>>> print('Original dataset shape %s' % Counter(y))
Original dataset shape Counter({1: 900, 0: 100})
>>> sm = SMOTE(random_state=42)
>>> X_res, y_res = sm.fit_resample(X, y)
>>> print('Resampled dataset shape %s' % Counter(y_res))
Resampled dataset shape Counter({0: 900, 1: 900})
```

1:900
0:100

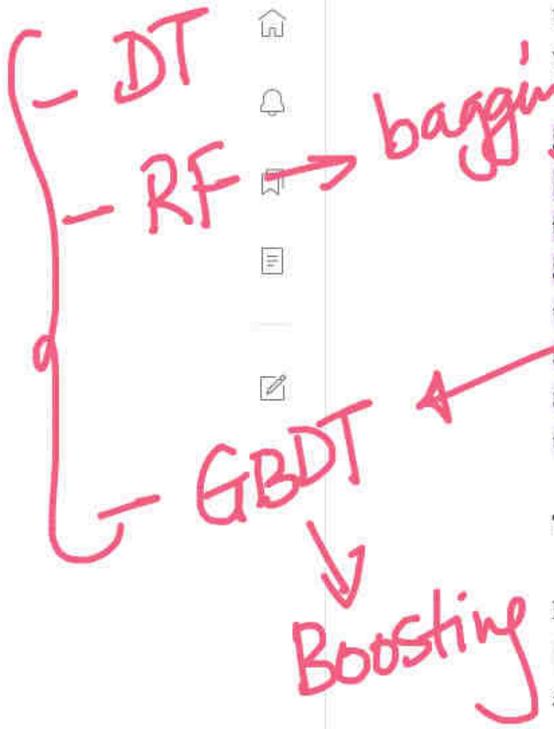
## Methods

| | |
|---|---|
| fit(X, y) | Check inputs and statistics of the sampler. |

learner, the weights are recalculated such that a higher weight is assigned to samples that the current weak learner misclassified. This weight determines the probability that the sample will appear in the training of the next weak learner. For this reason, boosting algorithms like AdaBoost are particularly useful for class imbalance problems because higher weight is given to the minority class at each successive iteration as data from this class is often misclassified.

Data sampling methods combined with boosting can be an effective way to deal with class imbalance problems. To demonstrate, let's use Lending Club loan data (available here) to try to predict whether someone will default on a loan using two interesting sampling approaches that have been suggested and combined with AdaBoost: SMOTEBoost and RUSBoost.

SMOTEBoost is an oversampling method based on the SMOTE algorithm (Synthetic Minority Oversampling Technique). SMOTE uses k-nearest neighbors to create synthetic examples of the minority class. SMOTEBoost then injects the SMOTE method at each boosting iteration. The advantage of this approach is that while standard boosting gives equal weights to all misclassified data, SMOTE gives more examples of the minority class at each boosting step. Similarly, RUSBoost achieves the same goal by performing random undersampling (RUS) at each boosting iteration instead of SMOTE.




### Trying it out: data preparation

First, I import the data and get rid of all entries that do not correspond to `loan_data = "Fully Paid"` or `"Charged Off"`. I set the `"Fully Paid"` target to `0` and `"Charged Off"` to `1`.

```
import pandas as pd
import numpy as np
```


Logistic-regr

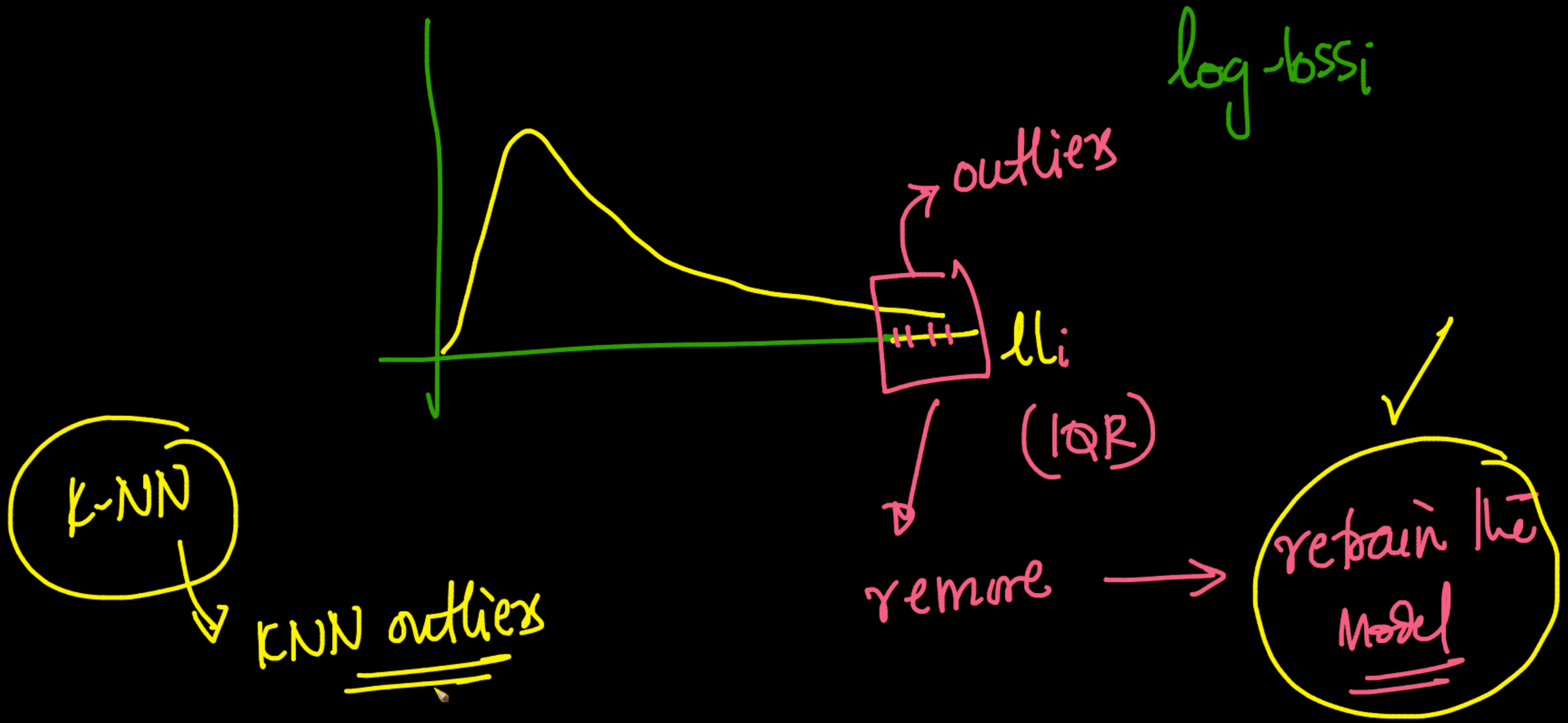
✓ IQR-based → pcr-column

hint: outlier impact Logistic Regr

$M_1$: model → extreme outliers

log-loss$_i$

(or) $\sigma(w^T x_i + b) \; \forall i$


log-loss
outliers
lli
(IQR)
K-NN
KNN outliers
remove
retrain the Model

Task

↓ outliers w.r.t. logistic regr. model

$k=5$

KNN for regression


→ 5 NNs → $y_i$'s → mean/median $= y_q$

[ logistic reg → classfn
Linear reg → reg

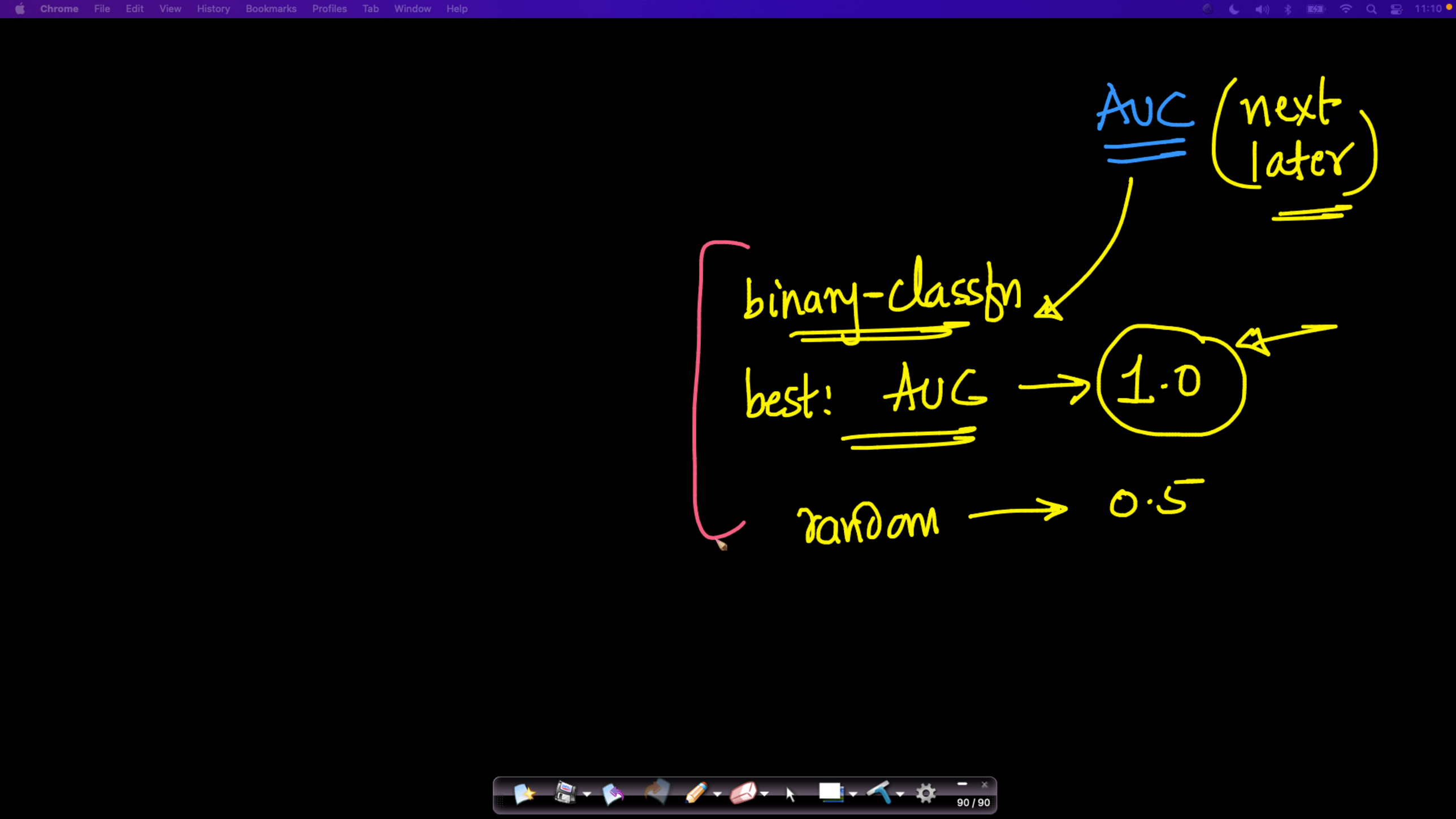
$x_i$'s

$y_i \in \mathbb{R}$

# Decision Trees:

K-NN → locality

cl: logistic Reg
reg: Linear Reg
→ Linear planes/fns


AUC (next later)
binary-classfn
best: AUC → 1.0
random → 0.5

Compute the area under the ROC curve.

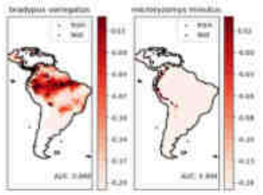
**average_precision_score**
Compute average precision from prediction scores.

**precision_recall_curve**
Compute precision-recall pairs for different probability thresholds.

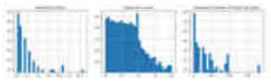
### Examples

```
>>> import numpy as np
>>> from sklearn import metrics
>>> y = np.array([1, 1, 2, 2])
>>> pred = np.array([0.1, 0.4, 0.35, 0.8])
>>> fpr, tpr, thresholds = metrics.roc_curve(y, pred, pos_label=2)
>>> metrics.auc(fpr, tpr)
0.75
```

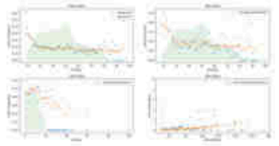
## Examples using sklearn.metrics.auc



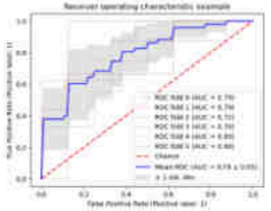
Species distribution modeling



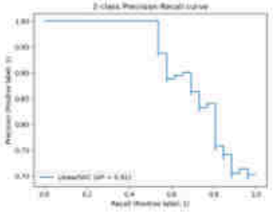
Poisson regression and non-normal loss



Tweedie regression on insurance claims



Receiver Operating Characteristic (ROC)



Precision-Recall

s validation

scikit learn
Prev Up Next
scikit-learn 1.0.2
Other versions
Please cite us if you use the software.
sklearn.metrics.auc
Examples using sklearn.metrics.auc

Compute the area under the ROC curve.

**average_precision_score**
Compute average precision from prediction scores.

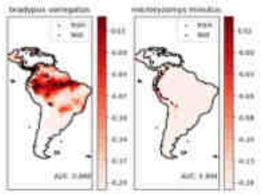
**precision_recall_curve**
Compute precision-recall pairs for different probability thresholds.

### Examples

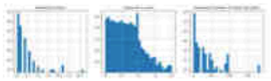
```
>>> import numpy as np
>>> from sklearn import metrics
>>> y = np.array([1, 1, 2, 2])
>>> pred = np.array([0.1, 0.4, 0.35, 0.8])
>>> fpr, tpr, thresholds = metrics.roc_curve(y, pred, pos_label=2)
>>> metrics.auc(fpr, tpr)
0.75
```

1.0 0.5 0 X

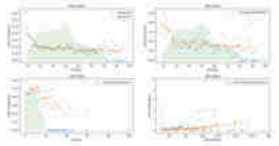
## Examples using sklearn.metrics.auc



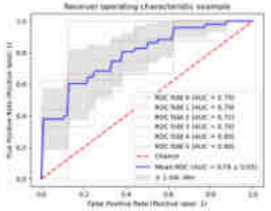
Species distribution modeling



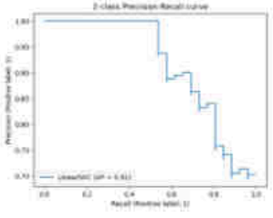
Poisson regression and non-normal loss



Tweedie regression on insurance claims



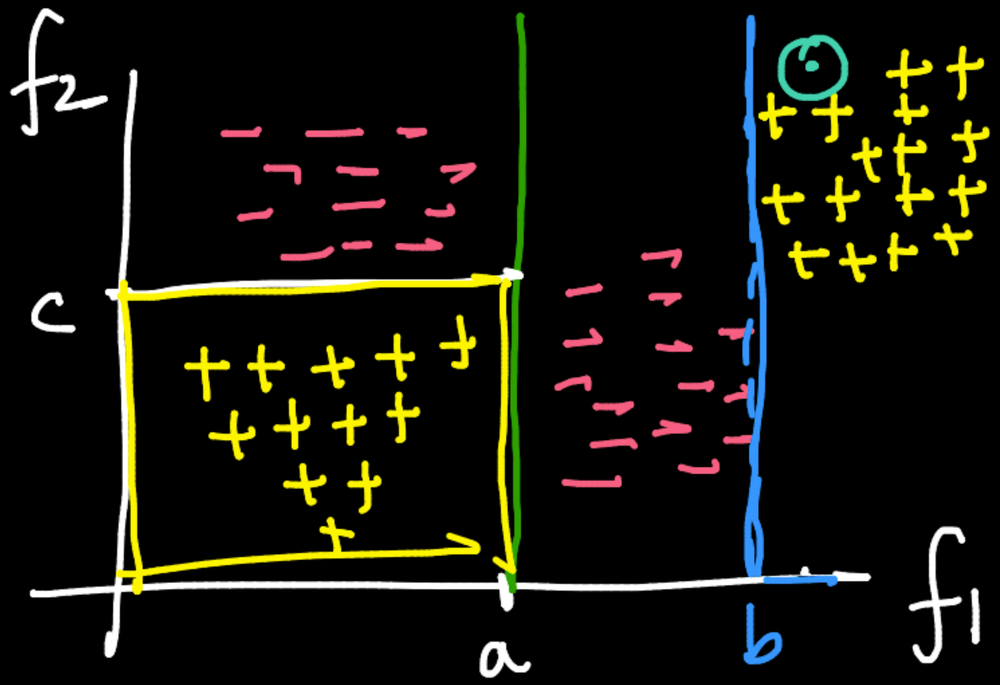
Receiver Operating Characteristic (ROC) s validation

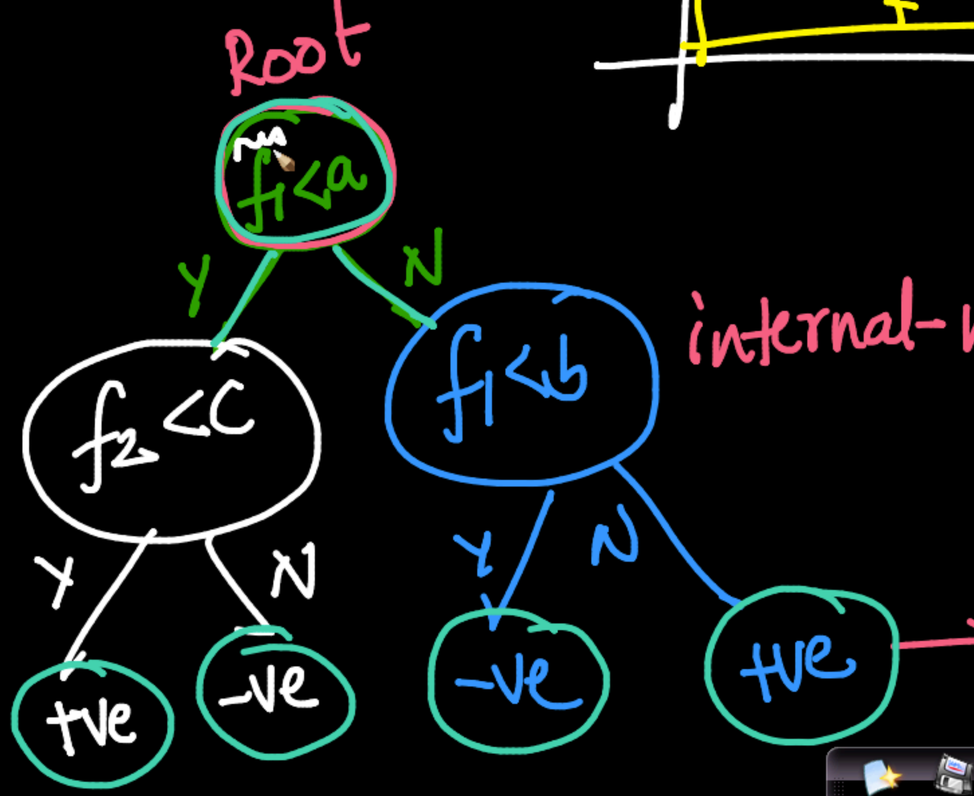


Precision-Recall

# Decision Tree

if-else

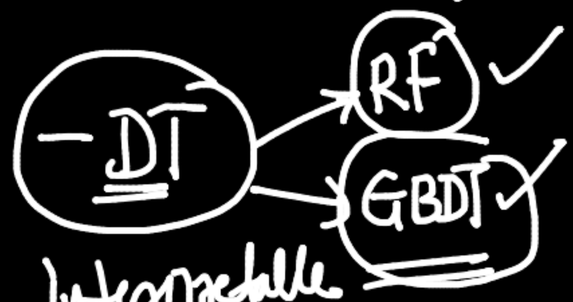



internal-nodes

→ leaf-nodes

- KNN
- Logistic-Regr (poly-features)

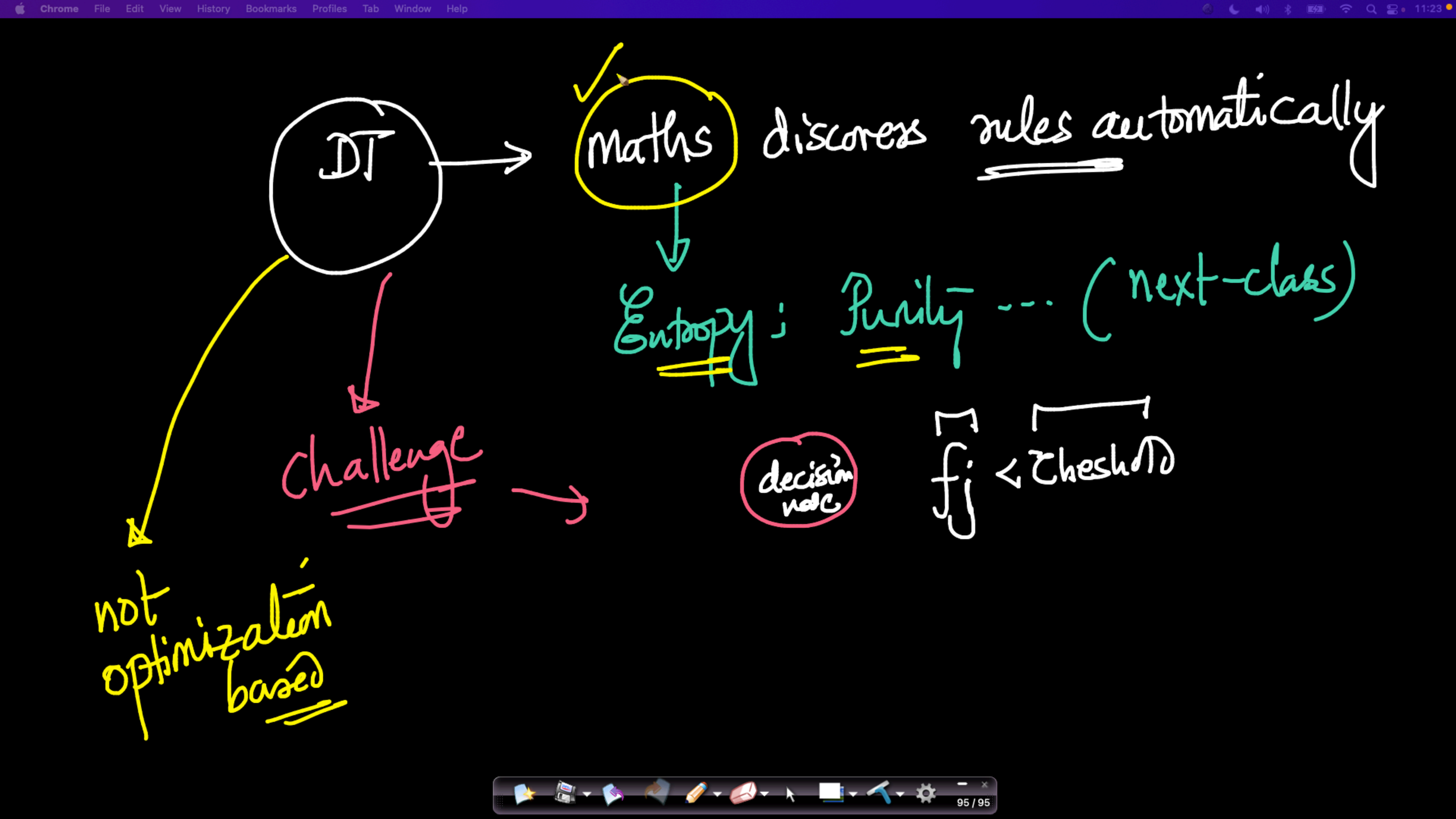


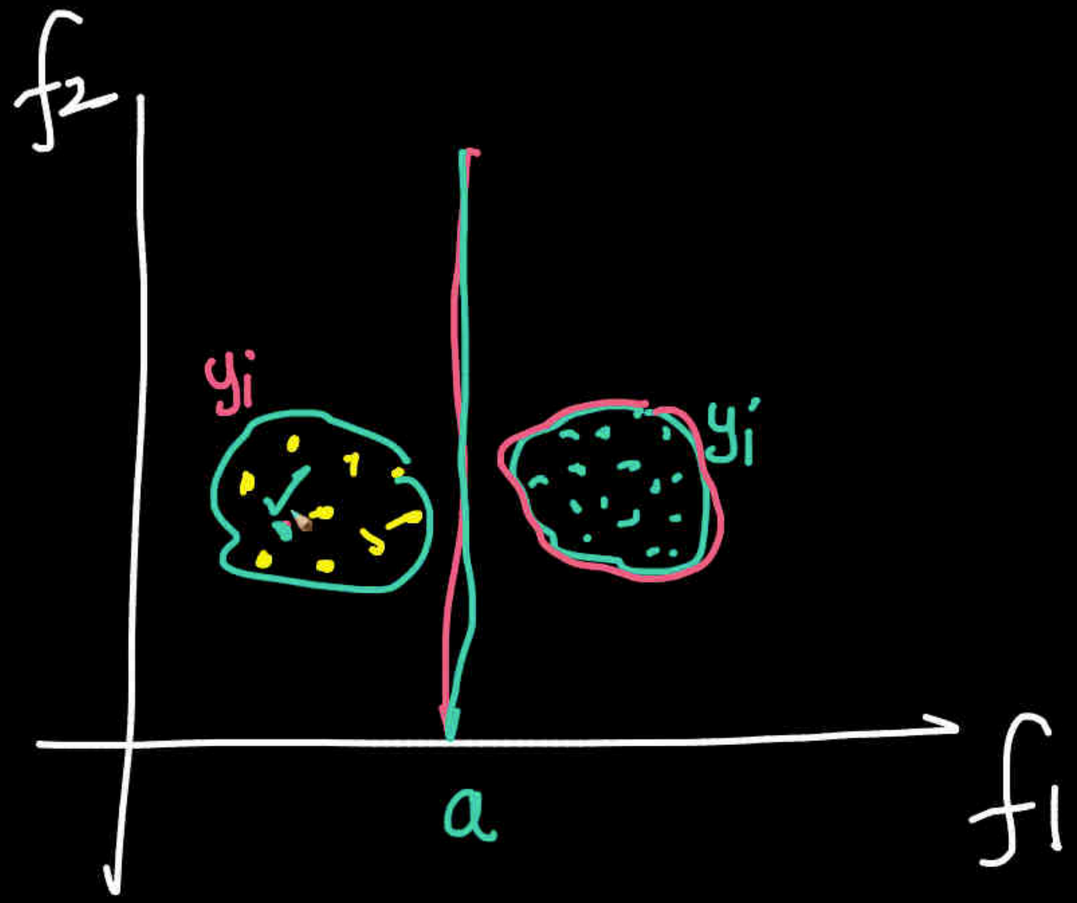

nested if---else

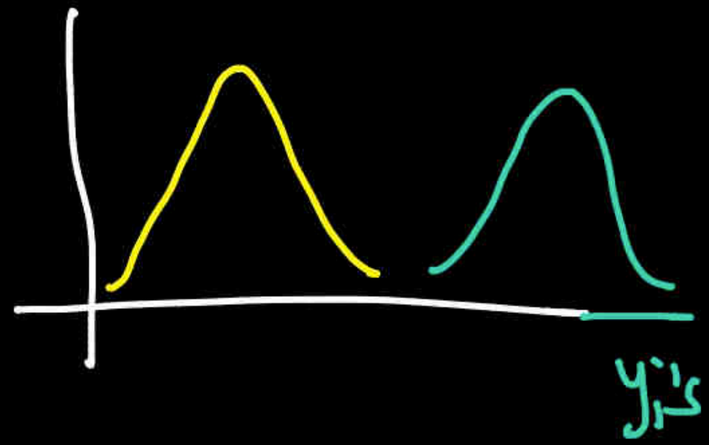
Rule-based system → nested if ... else (prog)

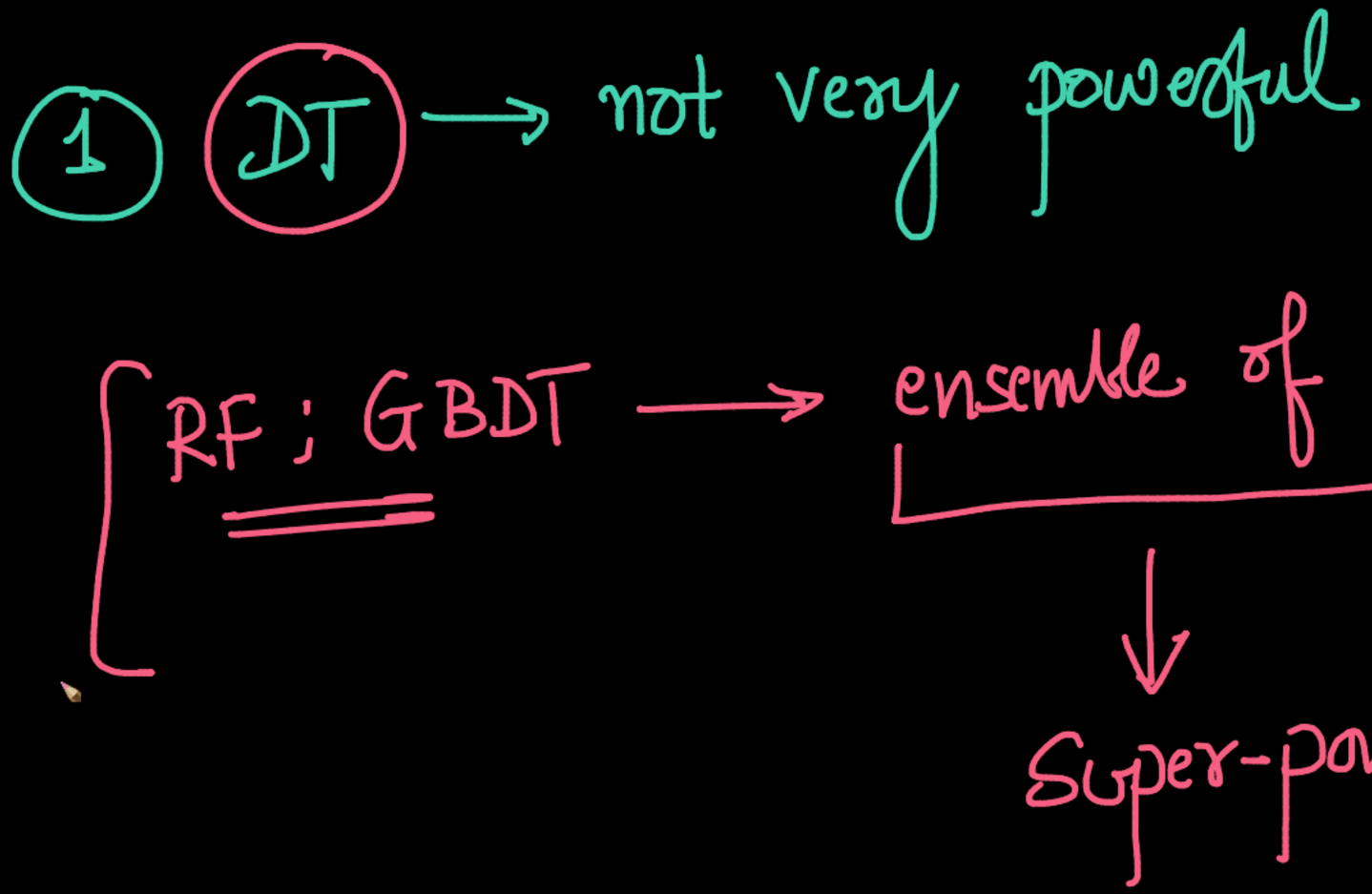
manually

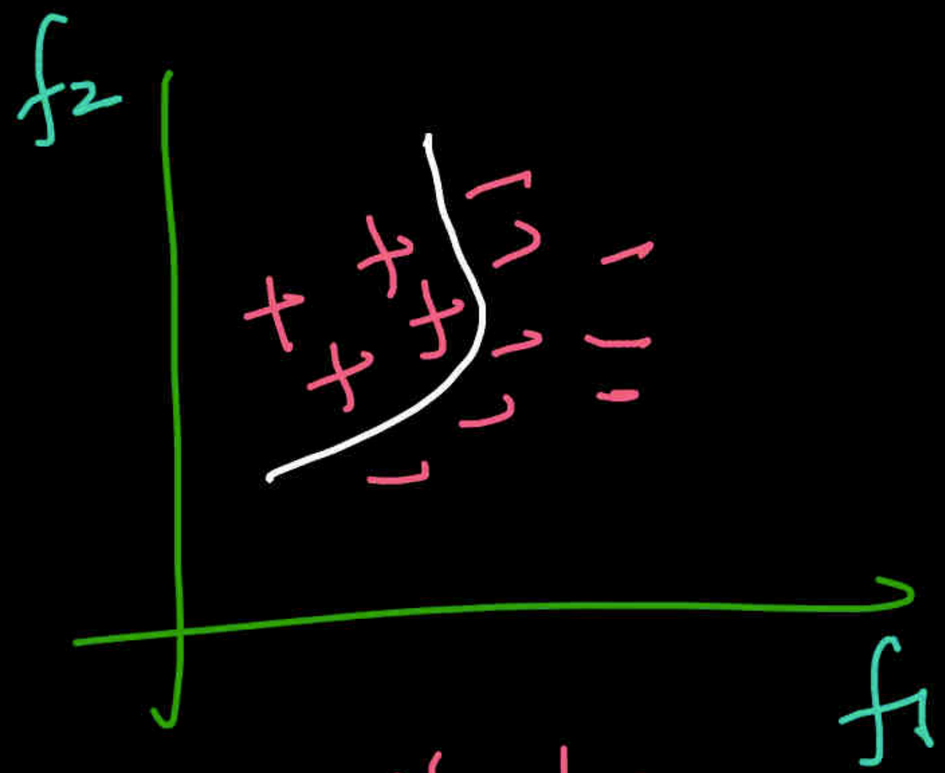
geom: DT is a collection of axis || hyperplanes


DT
maths
discoress
rules automatically
Entropy : Purity ... (next-class)
challenge
decision node
fj < Threshold
not optimization based

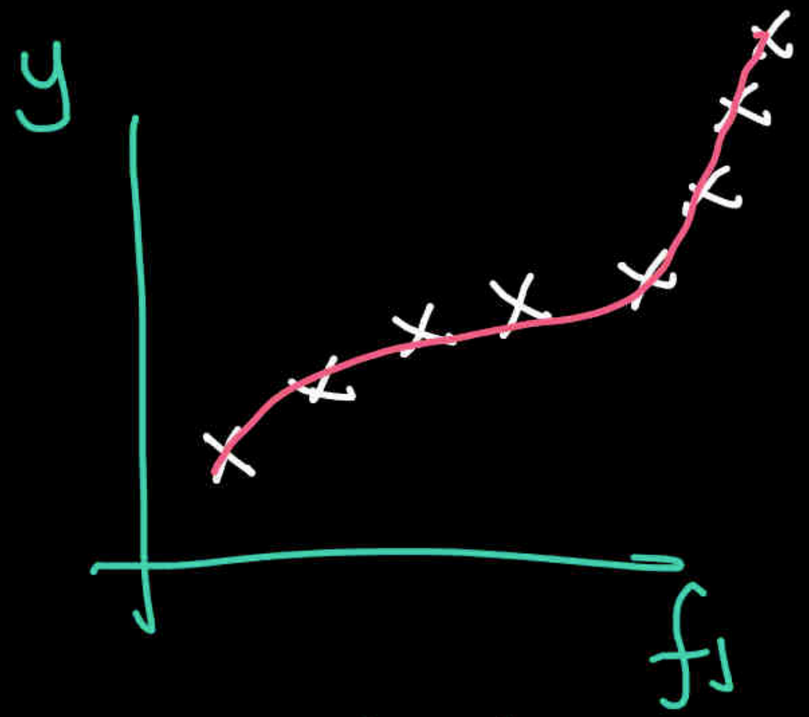



1
DT
not very powerful
RF ; GBDT
ensemble of DT
Super-powerful



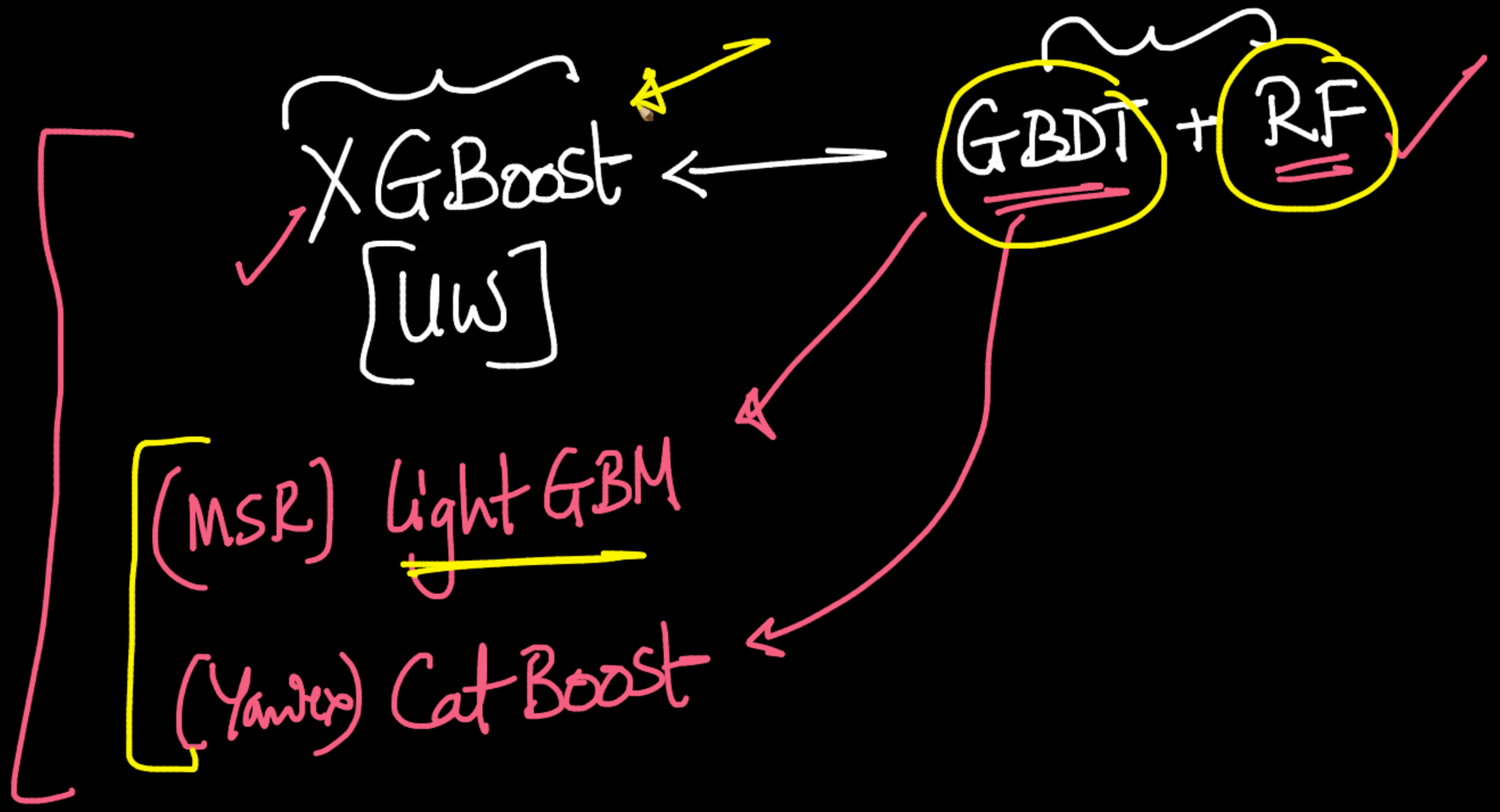

Classfn:

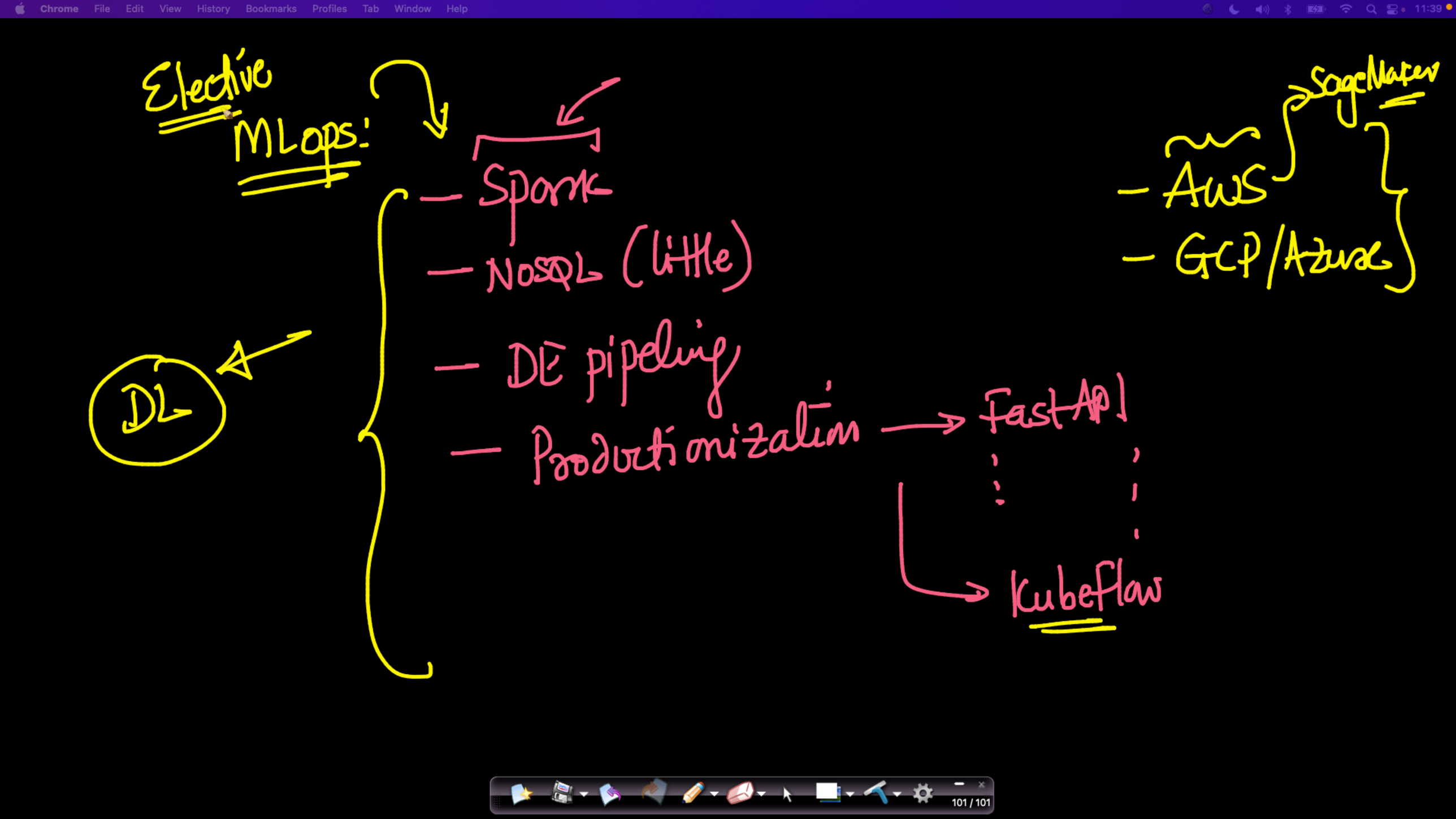



regr

# Next class:

- → Entropy: build a DT
- → AUC / ROC
- → ensembles: RF


XGBoost
[UW]
GBDT + RF
(MSR) Light GBM
(Yandex) Cat Boost


Elective
MLops:
- Spark
- NoSQL (little)
- DE pipeline
- Productionization
DL
- AWS
- GCP/Azure
SageMaker
FastAPI
Kubeflow

[ Tableau : SQL

DA → 

1

2

3

4

# LightGBM: A Highly Efficient Gradient Boosting Decision Tree

**Guolin Ke**[1], **Qi Meng**[2], **Thomas Finley**[3], **Taifeng Wang**[1],
**Wei Chen**[1], **Weidong Ma**[1], **Qiwei Ye**[1], **Tie-Yan Liu**[1]
[1]Microsoft Research [2]Peking University [3] Microsoft Redmond
[1]{guolin.ke, taifengw, wche, weima, qiwye, tie-yan.liu}@microsoft.com;
[2]qimeng13@pku.edu.cn; [3]tfinely@microsoft.com;

## Abstract

Gradient Boosting Decision Tree (GBDT) is a popular machine learning algorithm, and has quite a few effective implementations such as XGBoost and pGBRT. Although many engineering optimizations have been adopted in these implementations, the efficiency and scalability are still unsatisfactory when the feature dimension is high and data size is large. A major reason is that for each feature, they need to scan all the data instances to estimate the information gain of all possible split points, which is very time consuming. To tackle this problem, we propose two novel techniques: *Gradient-based One-Side Sampling* (GOSS) and *Exclusive Feature Bundling* (EFB). With GOSS, we exclude a significant propor-


to estimate the
r gradients play
a more important role in the computation of information gain, GOSS can obtain quite accurate estimation of the information gain with a much smaller data size. With EFB, we bundle mutually exclusive features (i.e., they rarely take nonzero values simultaneously), to reduce the number of features. We prove that finding