Auto Labeling Of Datasets

DoubleVerify

Natanel Davidovits

Senior Manager, Data Science & Research

DoubleVerify

Ad measurement leader!

Clients:

Advertisers, Platforms, Publishers

AI focus:

Online content classification

Where?

social media & open web

scale:

>100M images, videos, web-pages / day

DoubleVerify

Data Challenges

Open ended domain

Extremely in the wild

“Never enough data”

DoubleVerify

What is this talk about?

Showcasing a simple, automatic method for a reliable dataset expansion, based on a diverse minimal coreset.

This method is useful for POC / MVP when there is no resources to label a big amount of data.

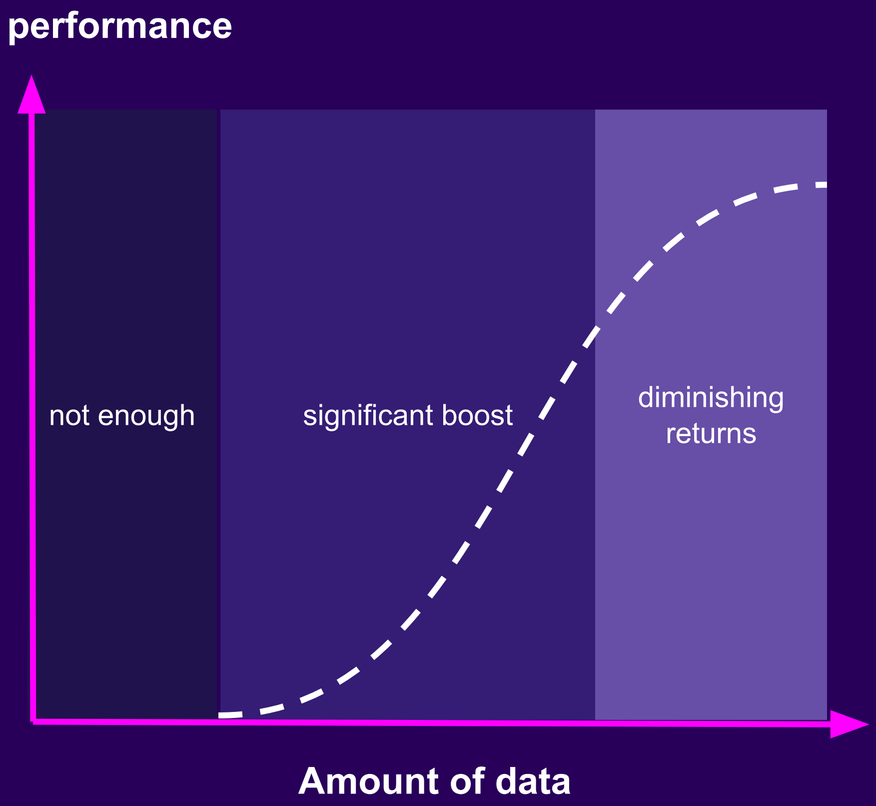
DoubleVerify

How much data is “enough”?

The bare minimum is unknown…

The classical N_class >= 1k ?

It depends…



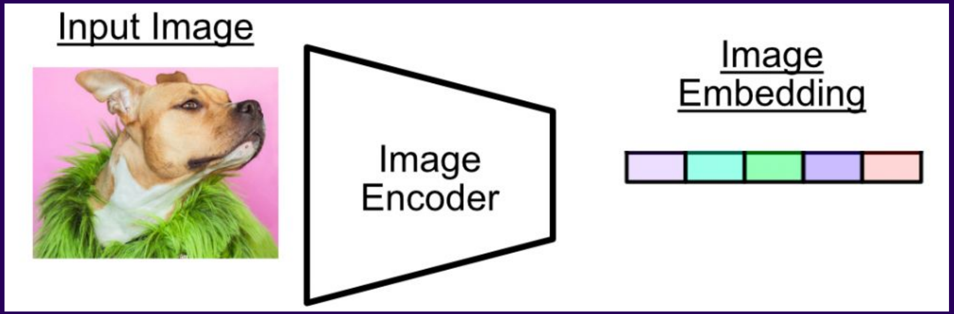
DoubleVerify

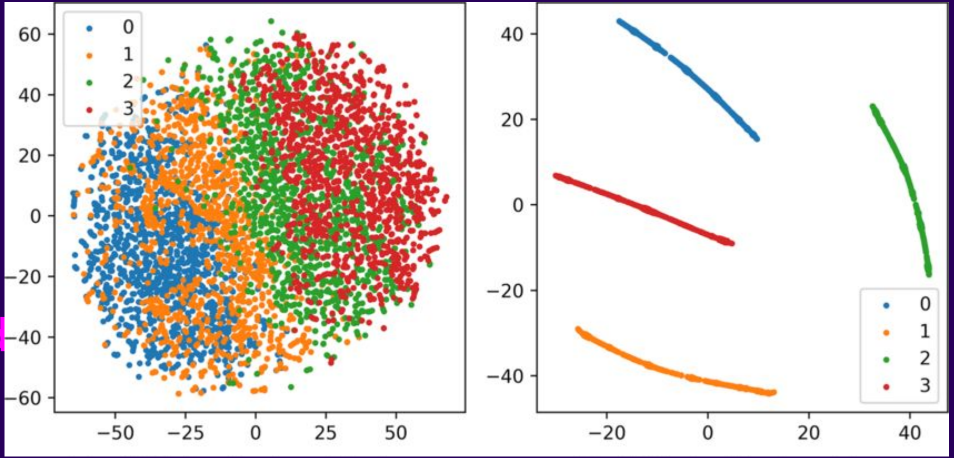
Select for your task

Business domain embedding model

HQ labeled data

Abundant, highly diverse, unlabeled d





Good diversity, bad embedding   good embedding, bad diversity

DoubleVerify

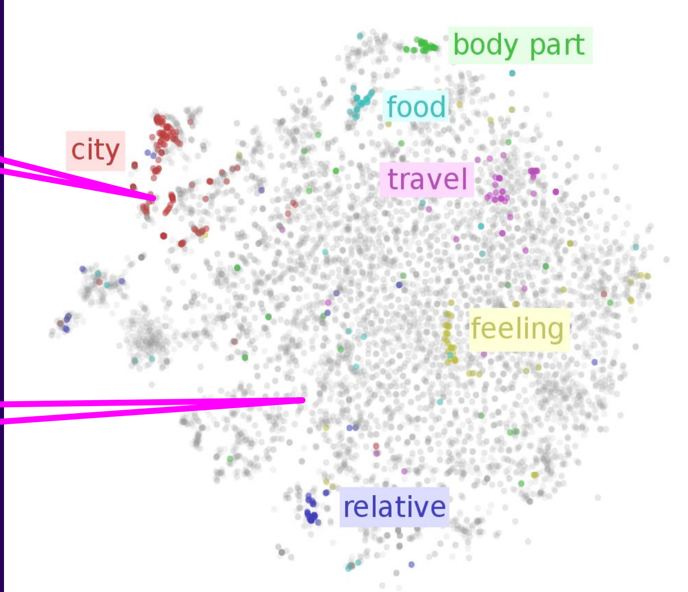
Auto labeling starting point

Select an unlabeled dataset

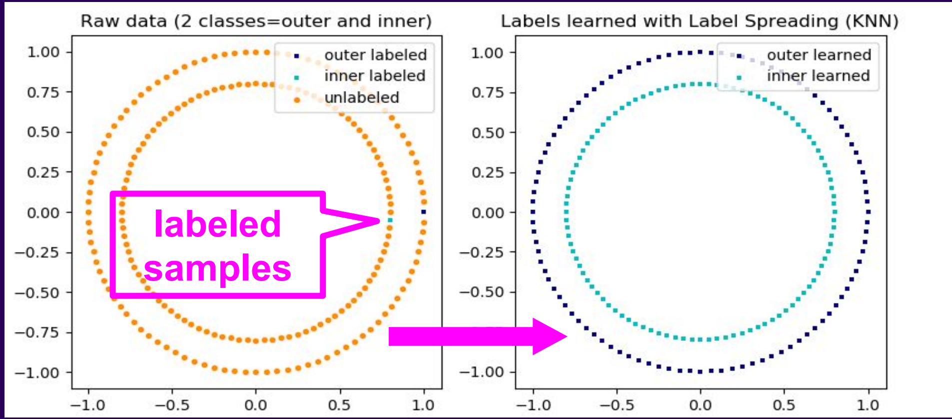
Label a few samples

some labeled data

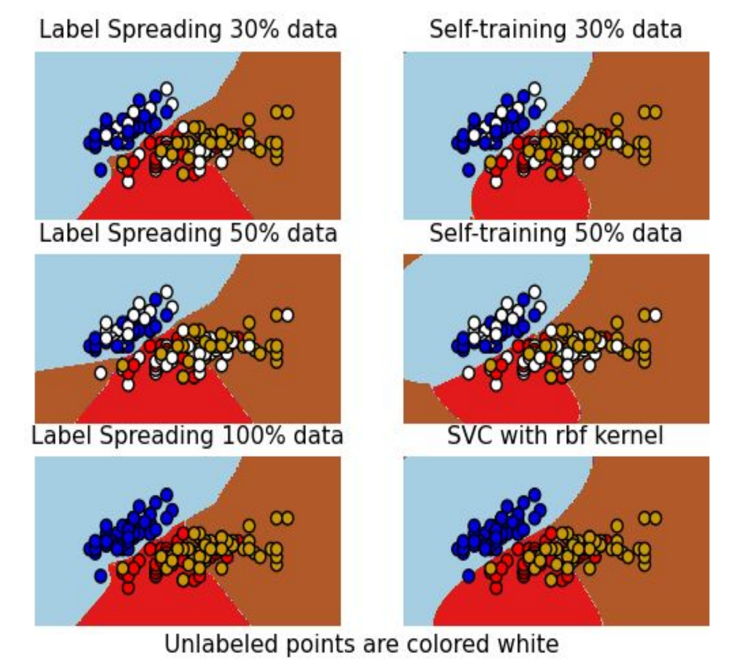
mostly unlabeled data



Rum label spreading (semi supervised) on the unlabeled data



Auto labeling




```
from sklearn.semi_supervised import LabelSpreading

# X: all input vector; Y: corresponding labels, where the unlabeled are set to value -1
spread_model = LabelSpreading()
spread_model.fit(X, Y)
Y_pred = spread_model.predict(X)

# or, when optimizing on confidence/probability:
Y_pred_conf = spread_model.predict_proba(X)
```

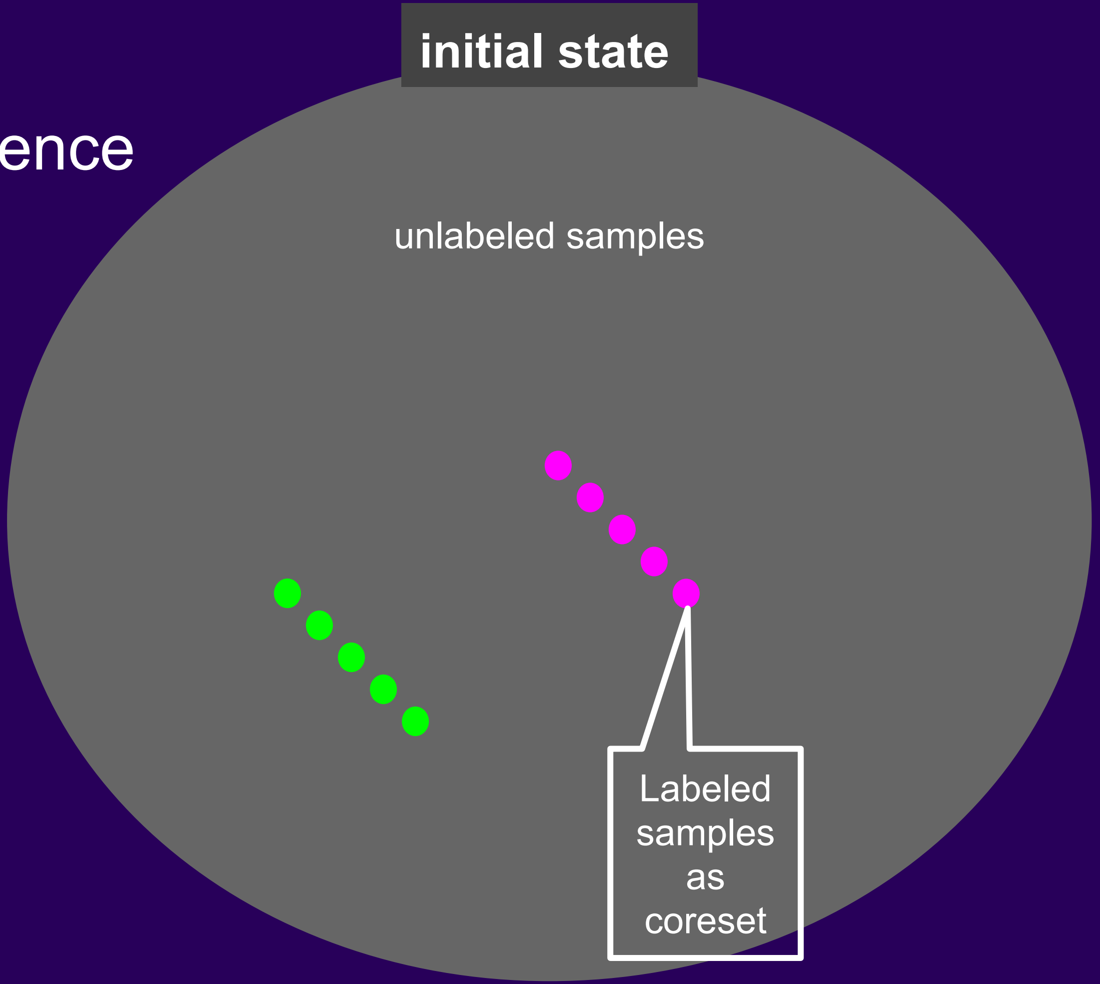
```
from sklearn.semi_supervised import SelfTrainingClassifier
from sklearn.svm import SVC

# X: all input vector; Y: corresponding labels, where the unlabeled are set to value -1
svc = SVC(probability=True)
self_training_model = SelfTrainingClassifier(svc)
self_training_model.fit(X, Y)
Y_pred = self_training_model.predict(X)
# or, when optimizing on confidence/probability:
Y_pred_conf = self_training_model.predict_proba(X)
```

DoubleVerify

Auto labeling - with confidence

"Ingredients":

- **A big unlabeled dataset**
- **Business metric**
- **Diverse core-set labeled per each class**

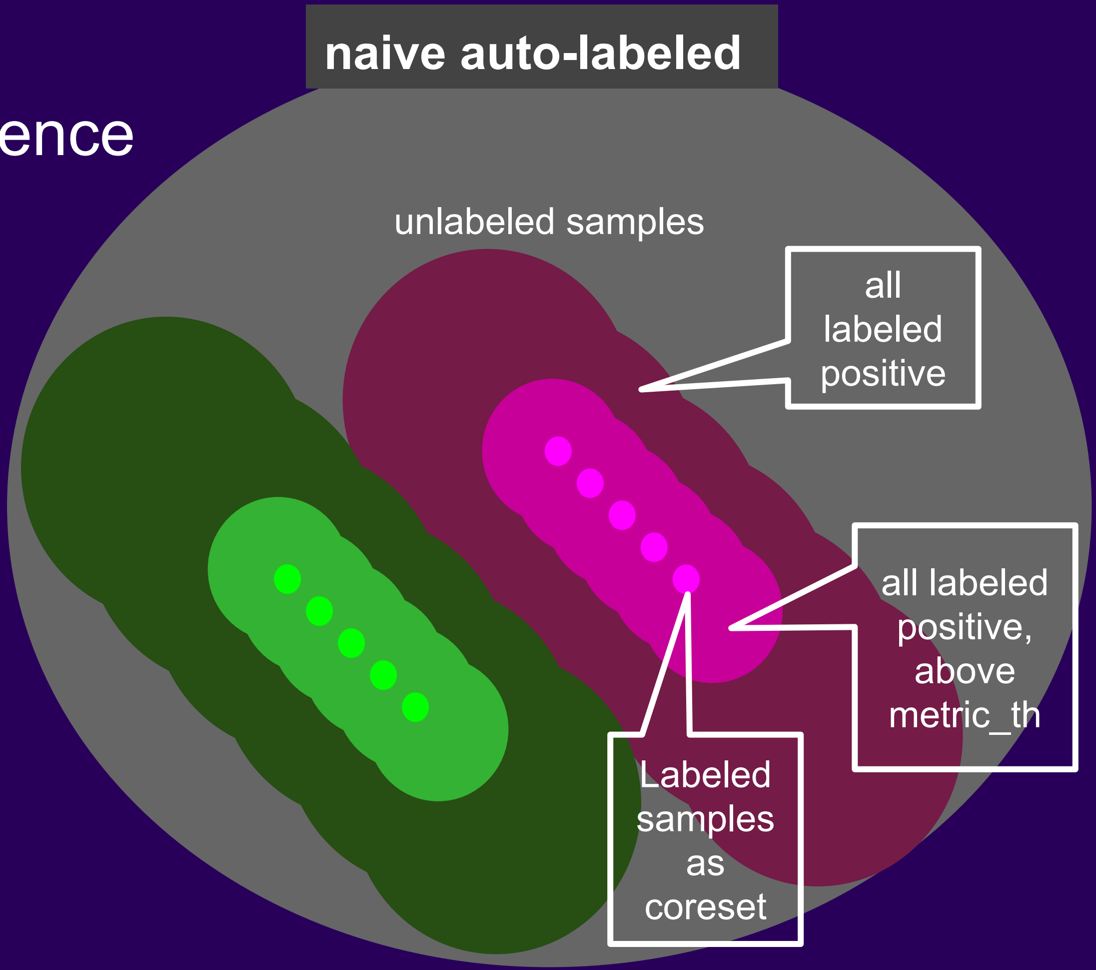


DoubleVerify

Auto labeling - with confidence

Process:

- **Run label-spreading (colored darkly)**
- **Threshold the metric to label the relevant samples (colored intermediately)**
- **Do so for stratified k-folds**

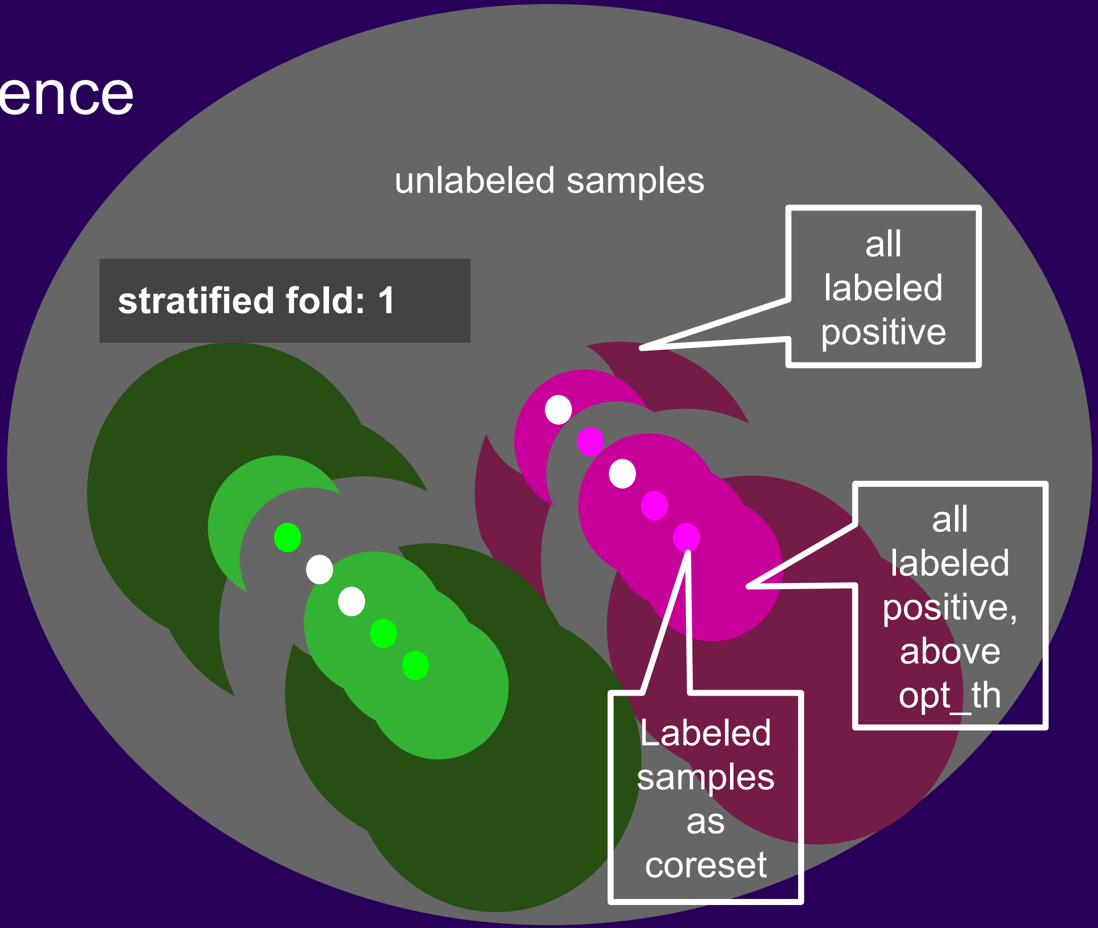


DoubleVerify

Auto labeling - with confidence

<u>Process 1:</u>

- **Run label-spreading (colored darkly)**
- **Threshold the metric to label the relevant samples (colored intermediately)**
- **Do so for stratified k-folds**

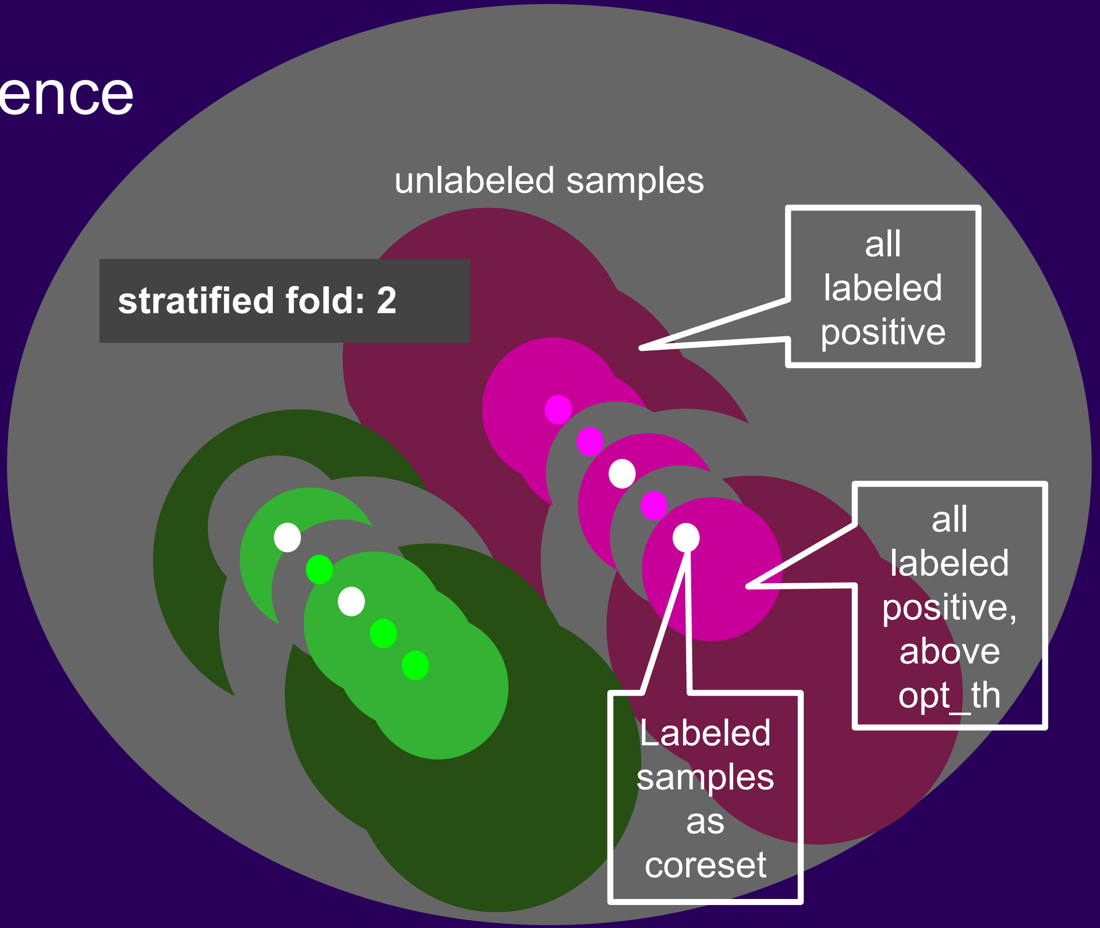


DoubleVerify

Auto labeling - with confidence

Process 1:

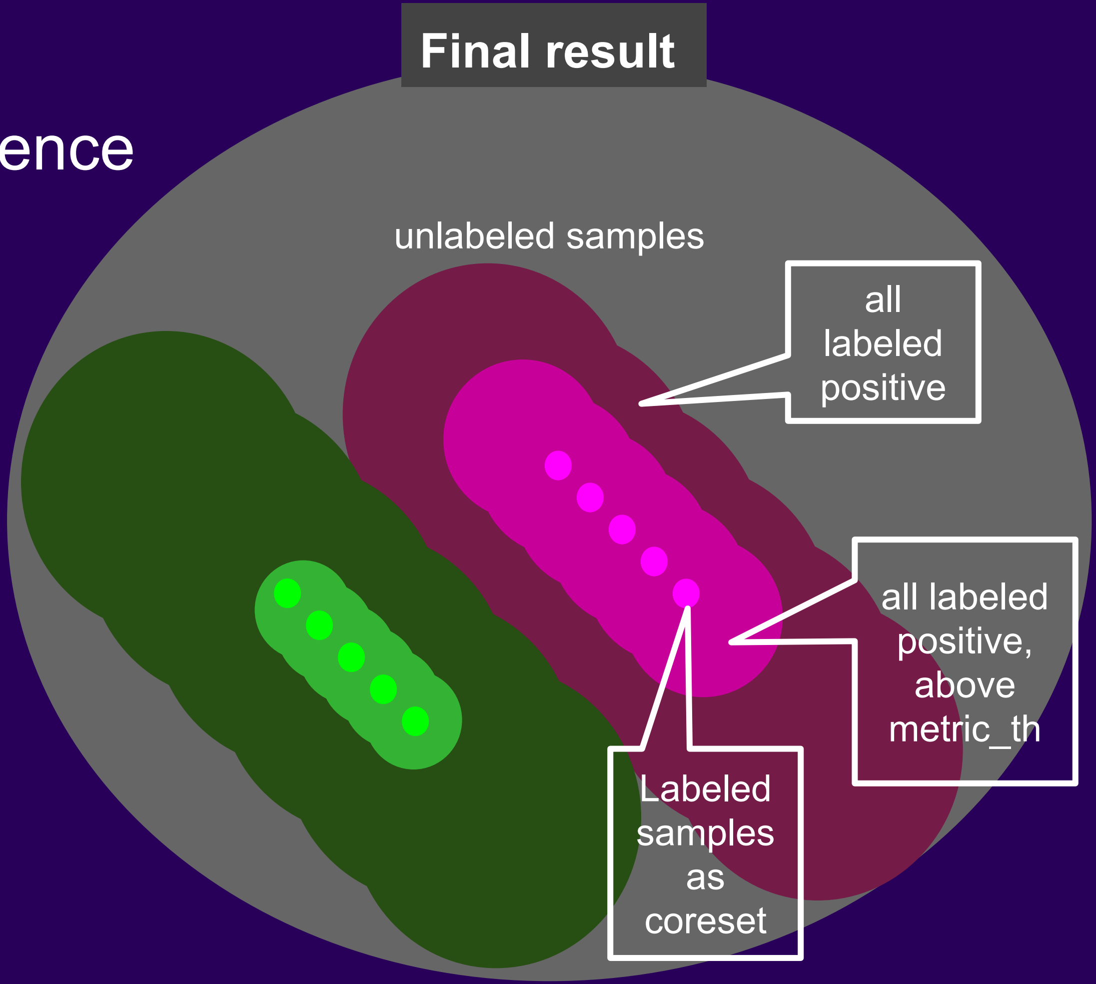
- **Run label-spreading (colored darkly)**
- **Threshold the metric to label the relevant samples (colored intermediately)**
- **Do so for stratified k-folds**



Auto labeling - with confidence

<u>Process 2:</u>

- **Categorical altogether / separately, optimize threshold/s over folds**
- **Rerun label-spreading based on the optimized threshold, for all the dataset**



```python
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.semi_supervised import SelfTrainingClassifier
from sklearn.svm import SVC

def optimal_score_threshold(X_labeled, Y_labeled, X_unlabeled, Y_unlabeled, base_classifier, dth=0.1, n_splits=5):
    splits = StratifiedKFold(n_splits=n_splits, random_state=108, shuffle=True).split(X_labeled, Y_labeled)
    th_score = []
    thresholds = np.arange(dth, 1., dth)
    for i, threshold in enumerate(thresholds): # brute/grid-search with dth increments
        self_training_model = SelfTrainingClassifier(base_classifier, threshold=threshold)
        # cross validation so that we won't get a val/train-split-skewed result:
        scores = []
        for fold, (train_index, test_index) in enumerate(splits):
            X_train, y_train = X_labeled[train_index], Y_labeled[train_index]
            X_test, y_test = X_labeled[test_index], Y_labeled[test_index]
            self_training_model.fit(np.concatenate([X_train, X_unlabeled]), np.concatenate([y_train, Y_unlabeled]))
            y_pred = self_training_model.predict(X_test)
            scores.append(business_metric(y_test, y_pred))
        th_score.append(np.mean(scores)) # we mean out folds' inconsistency
    th_score = np.array(th_score)
    return th_score.max(), thresholds[th_score.argmax()]

def optimal_auto_label(X_labeled, Y_labeled, X_unlabeled, Y_unlabeled, dth=0.1, n_splits=5):
    base_classifier = SVC(kernel='linear', probability=True, random_state=108)
    expected_score, opt_threshold = optimal_score_threshold(X_labeled, Y_labeled, X_unlabeled, Y_unlabeled,
                                                            base_classifier, dth=dth, n_splits=n_splits)
    self_training_model = SelfTrainingClassifier(base_classifier, threshold=opt_threshold)
    self_training_model.fit(np.concatenate([X_labeled, X_unlabeled]), np.concatenate([Y_labeled, Y_unlabeled]))
    y_pred = self_training_model.predict(X_unlabeled)
    # returning reliable auto-labeled samples:
    return X_unlabeled[y_pred > -1], y_pred[y_pred > -1]
```

DoubleVerify

Caveats

- **Dataset bias** - we rely on external data / foundational model
- Low-definition dataset for **your use-case**
- Heavily **relies on the chosen metric** (which may be under defined)
- **Unfit for sparse-visual** domains (medical, for example)

Takeaways

- **A small labeled dataset is not a dead end**
- **A few diverse samples can get you a long way**
- **Breach gaps by manually label a few uncertain samples**

Thanks!

Come say hi, at DoubleVerify’s booth!