W4995 Applied Machine Learning

NMF; Outlier detection

03/29/17

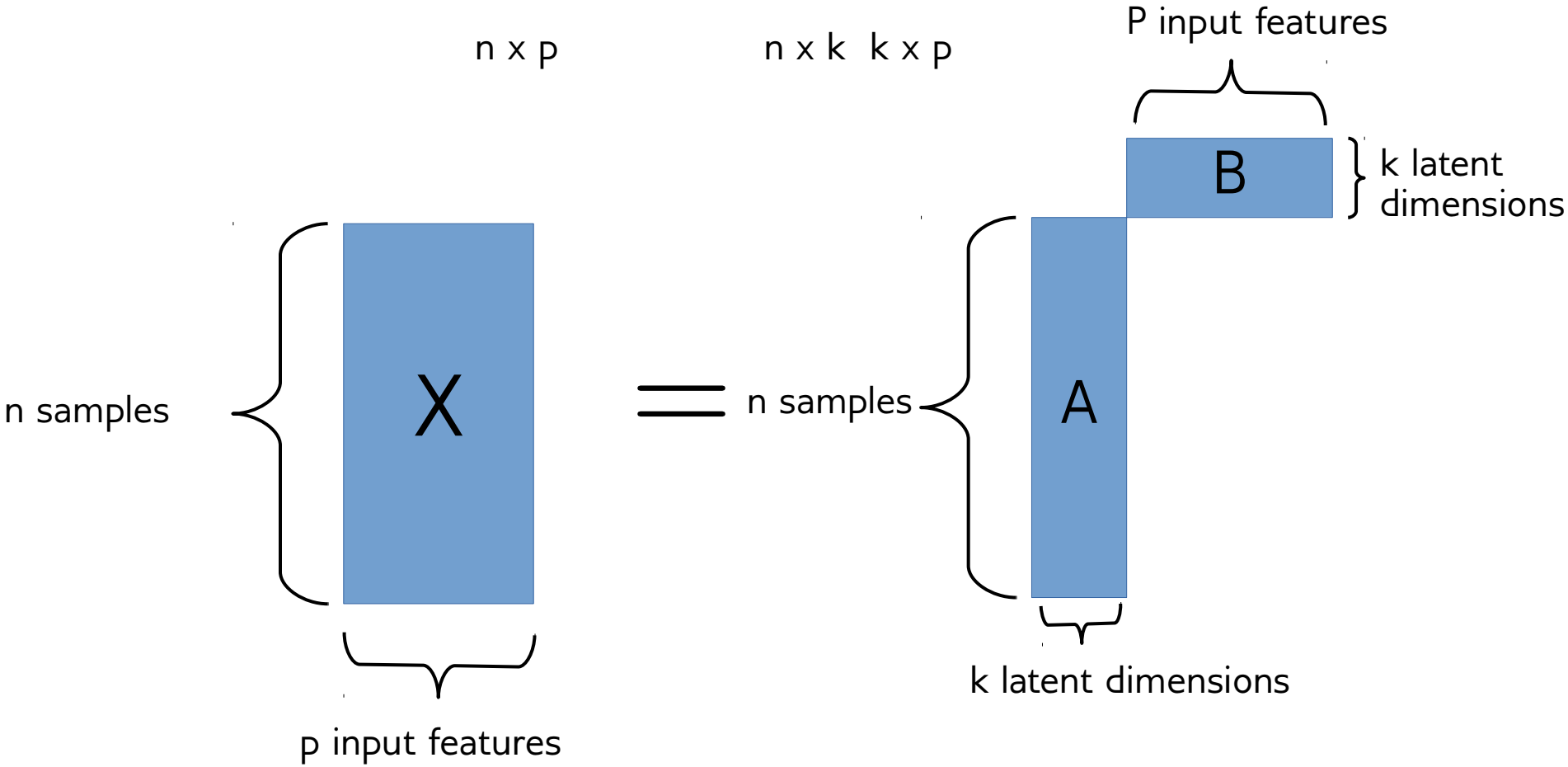
Andreas Müller

Non-Negative Matrix Factorization

Matrix Factorization

$$X = AB$$

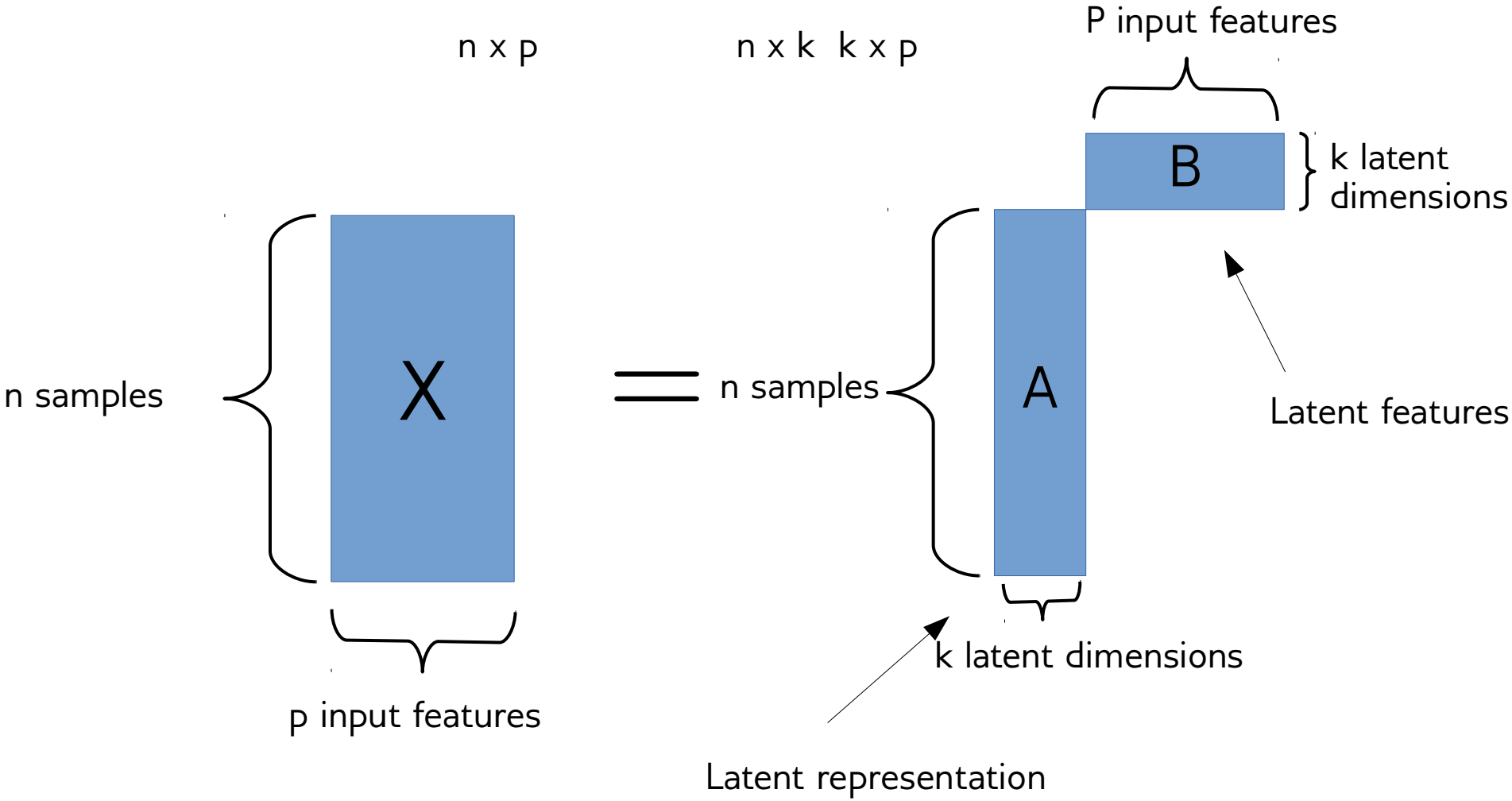
n x p n x k k x p



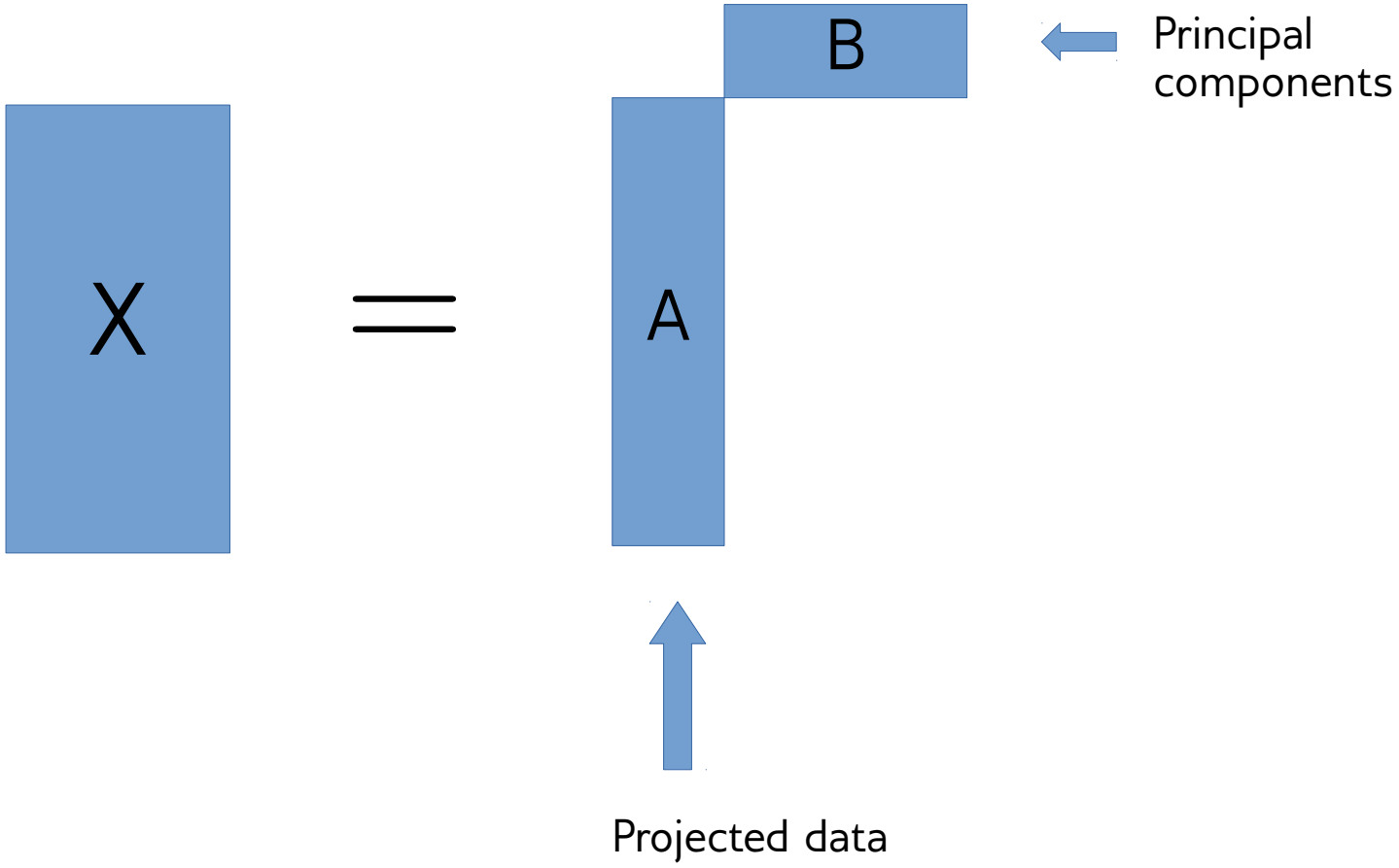
Matrix Factorization

$$X = AB$$

n x p n x k k x p



PCA

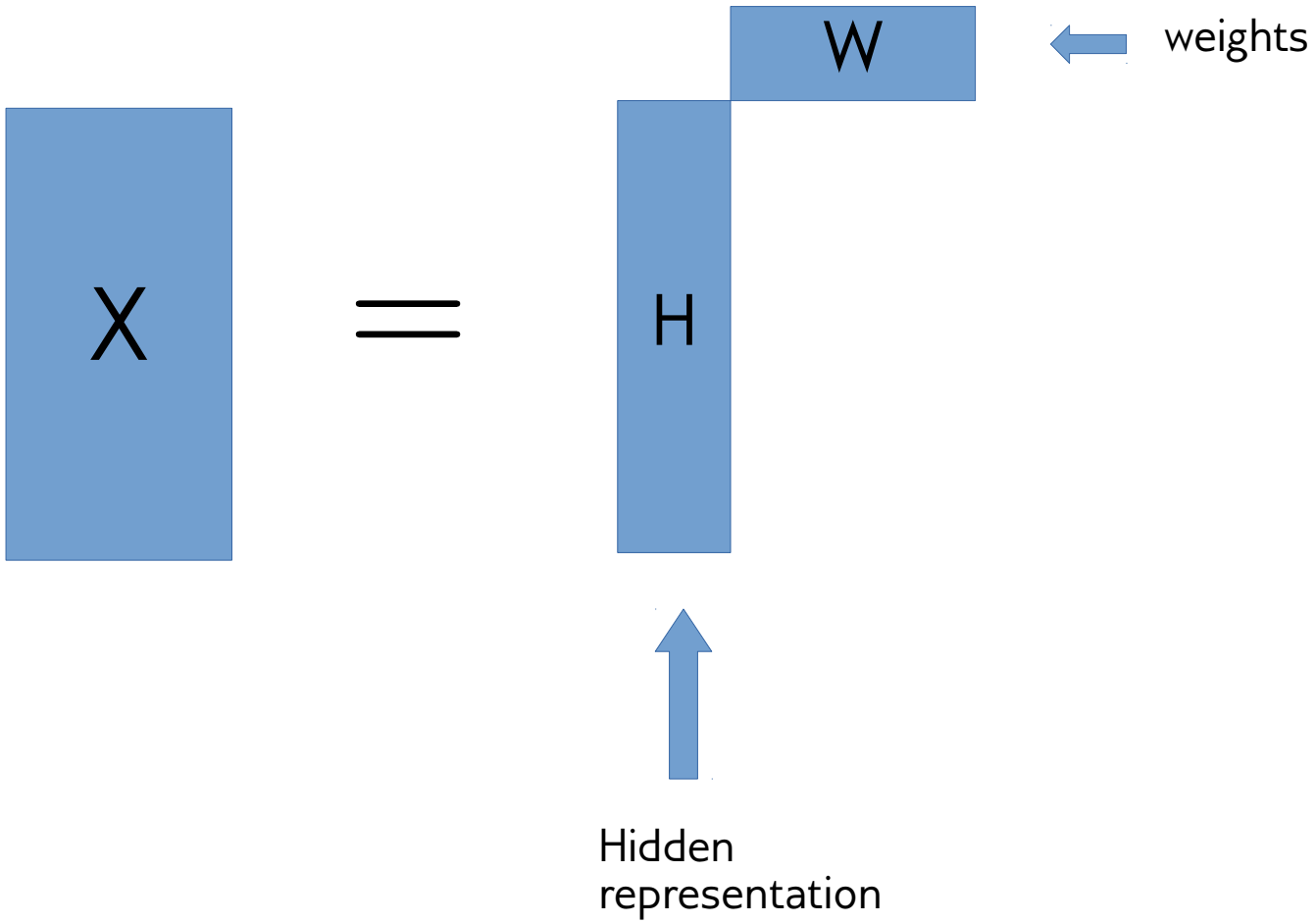


Other Matrix Factorizations

- PCA: principal components orthogonal, minimize squared loss
- Sparse PCA: components orthogonal & sparse
- ICA: independent components
- Non-negative matrix factorization (NMF):

  latent representation and latent features are non-negative.

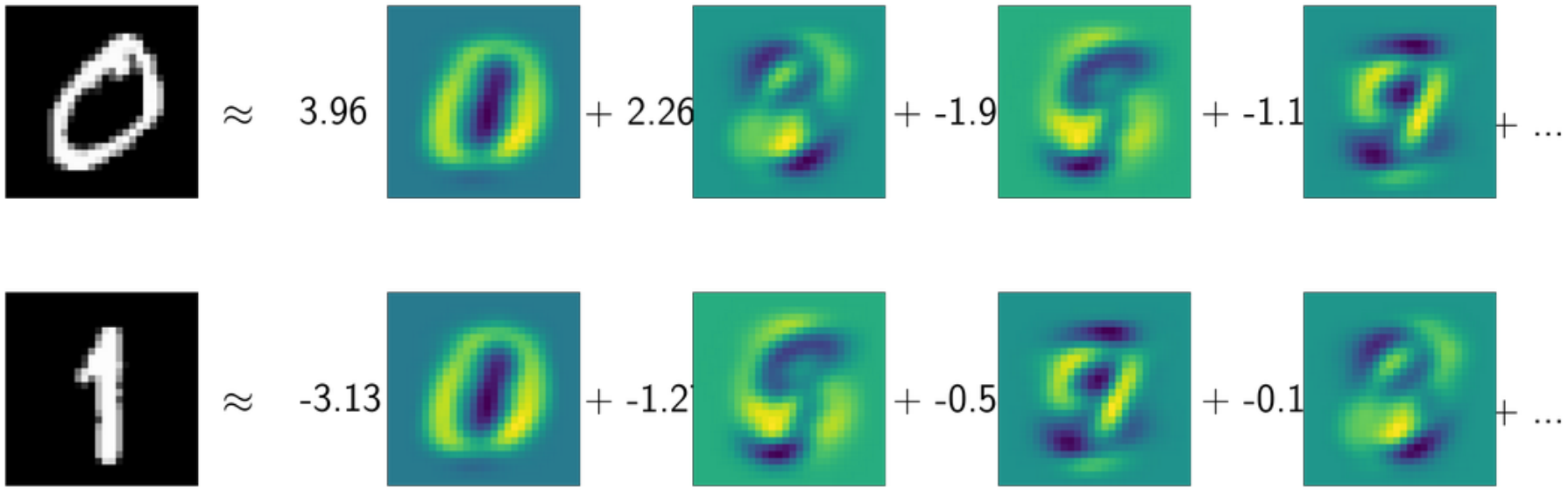
NMF



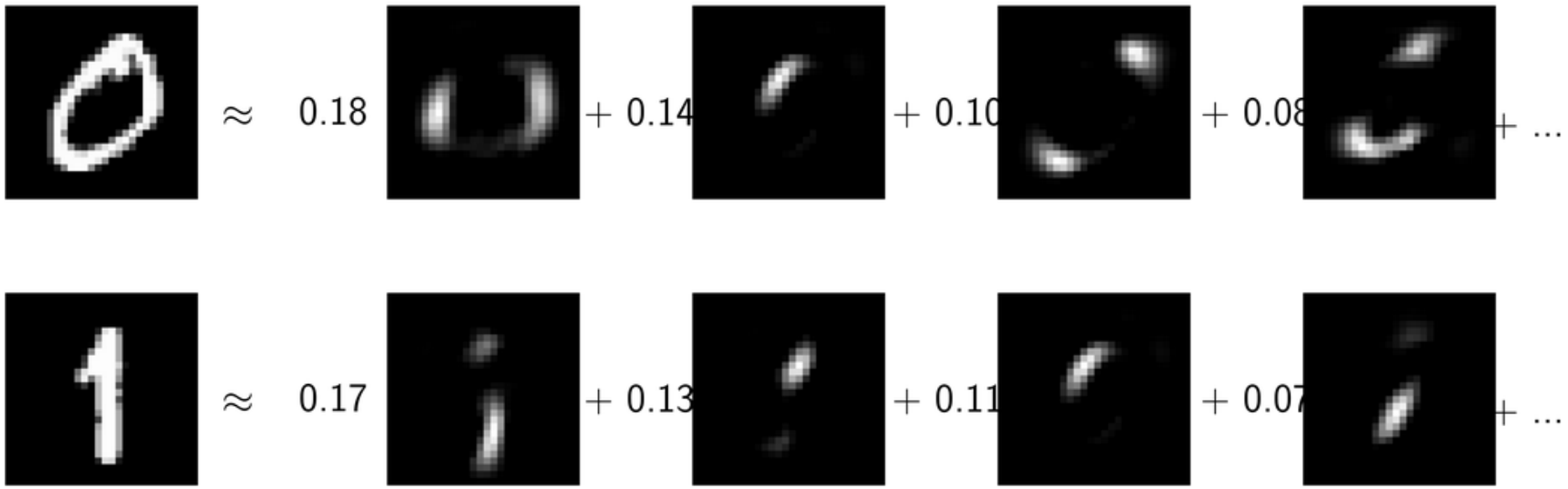
Why NMF?

- Data points are composed into positive sums
- Positive weights can be easier to interpret
- No “cancellation” like in PCA
- No sign ambiguity like in PCA
- Can learn over-complete representation (n_components > n_features) by asking for sparsity (in either W or H)
- Can be viewed as “soft clustering”: each point is positive linear combination of weights.

PCA (ordered by projection, not eigenvalue

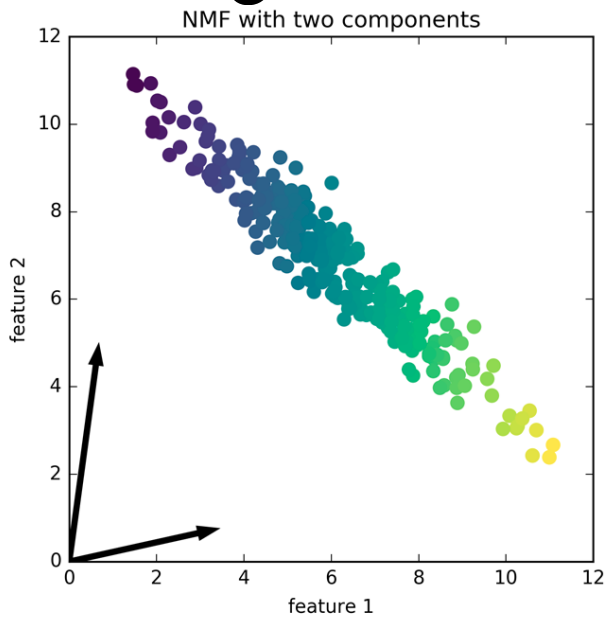


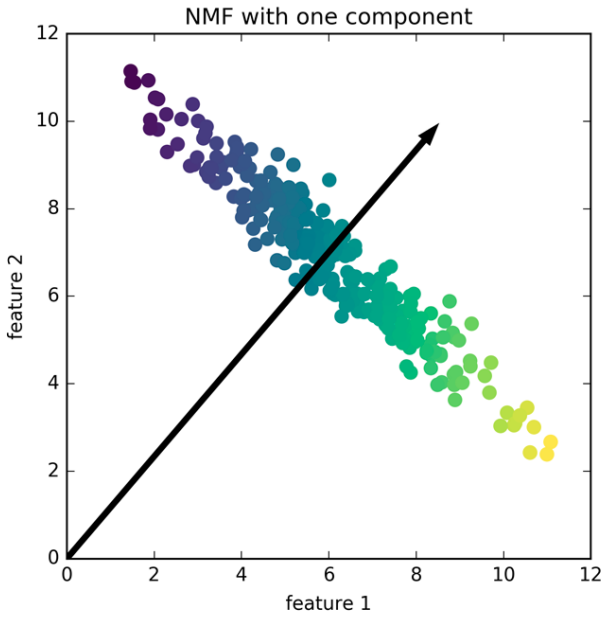
NMF (ordered by hidden representation)



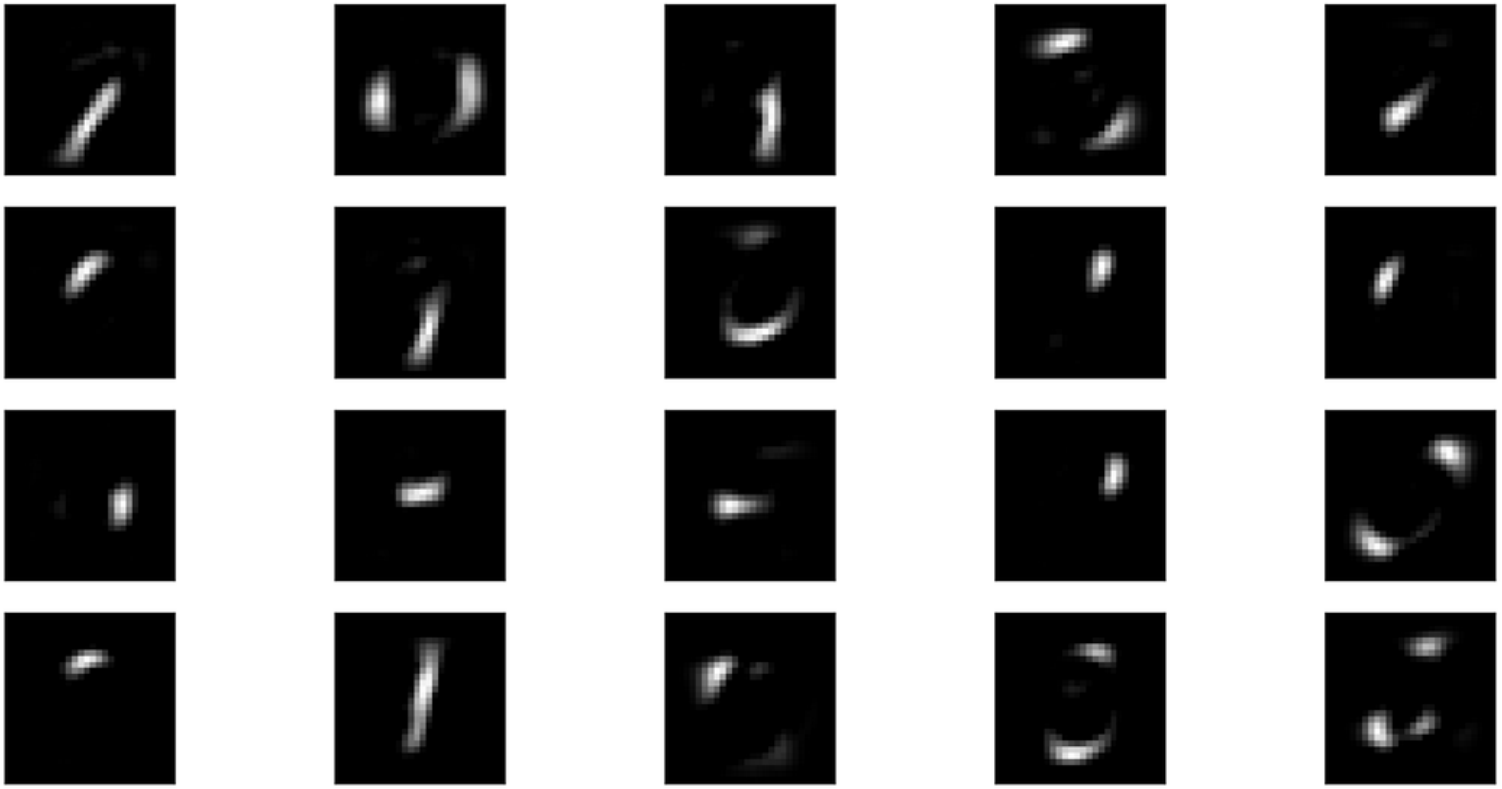
Downsides of NMF

- Can only be applied to non-negative data
- Whether components are interpretable is hit/miss
- Non-convex optimization, requires initialization
- Can be slow on large datasets
- Not orthogonal





NMF with 20 components on MNIST



NMF with 5 components on MNIST

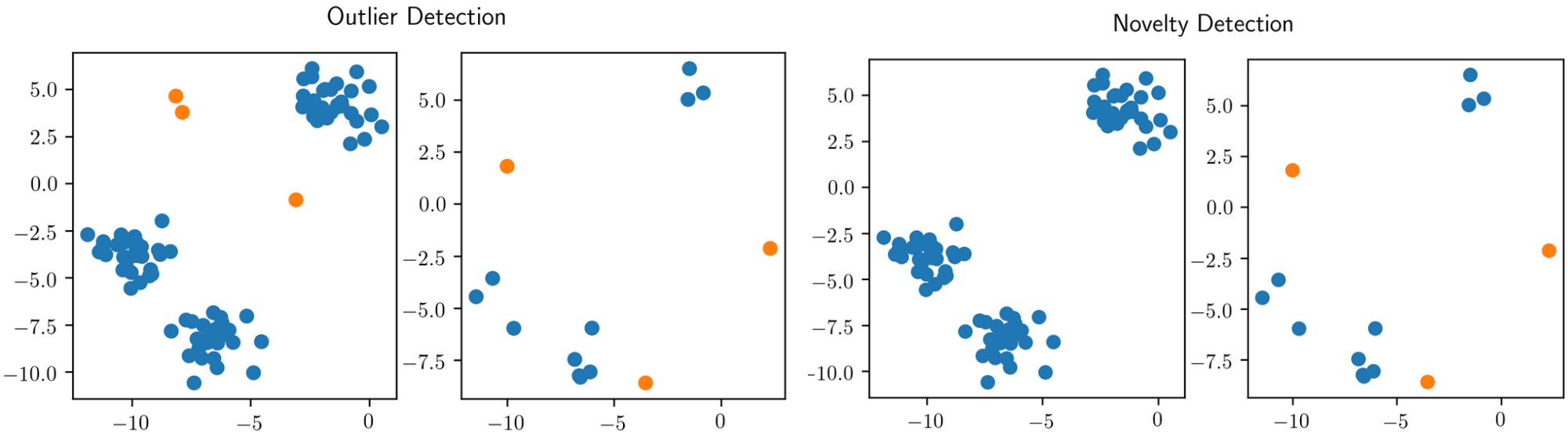
Applications of NMF

- Text analysis (next week)
- Signal processing
- Speech and Audio (see https://librosa.github.io/librosa/generated/librosa.decompose.decompose.html#librosa.decompose.decompose)
- Source separation
- Gene expression analysis

Outlier Detection

Motivation

- Find points that are “different” within the training set (and in the future).
- “Novelty detection” - no outliers in the training set.
- Outliers are not labeled! (otherwise it’s just imbalanced classification)



Often used interchangeably in practice.

Applications

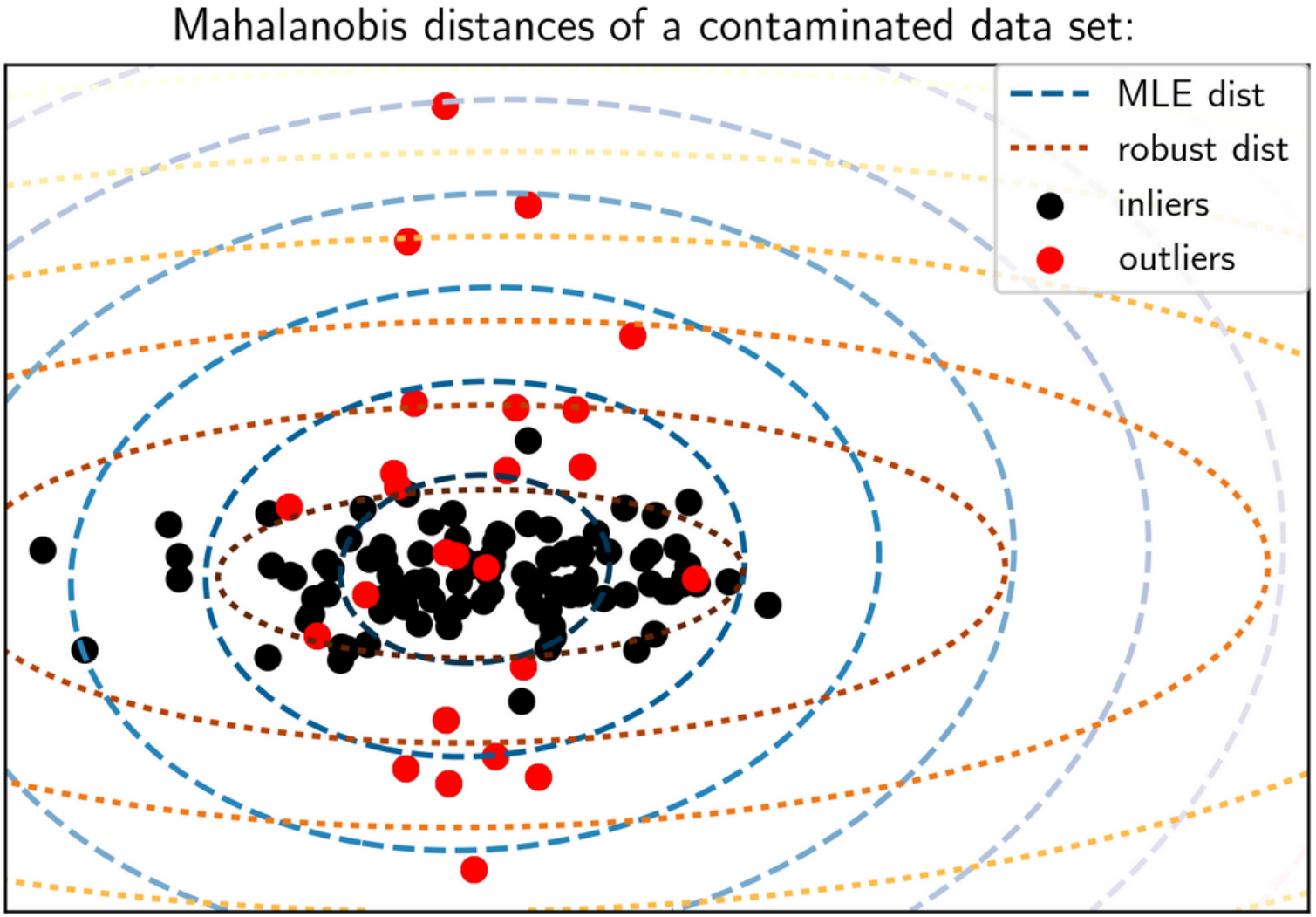
- Fraud detection (credit cards, click fraud, ...)
- Network failure detection
- Intrusion detection in networks
- Defect detection (engineering etc...)
- News? Intelligence?

Basic idea

- Model data distribution
- For outlier detection: be robust in modelling
- Nonparametric or parametric method
- Apply at test-time, threshold likelihood.
- Task is generally ill-defined (unless you know the real data distribution).

Elliptic Envelope

- Gaussian model
- Fit robust covariance matrix and mean

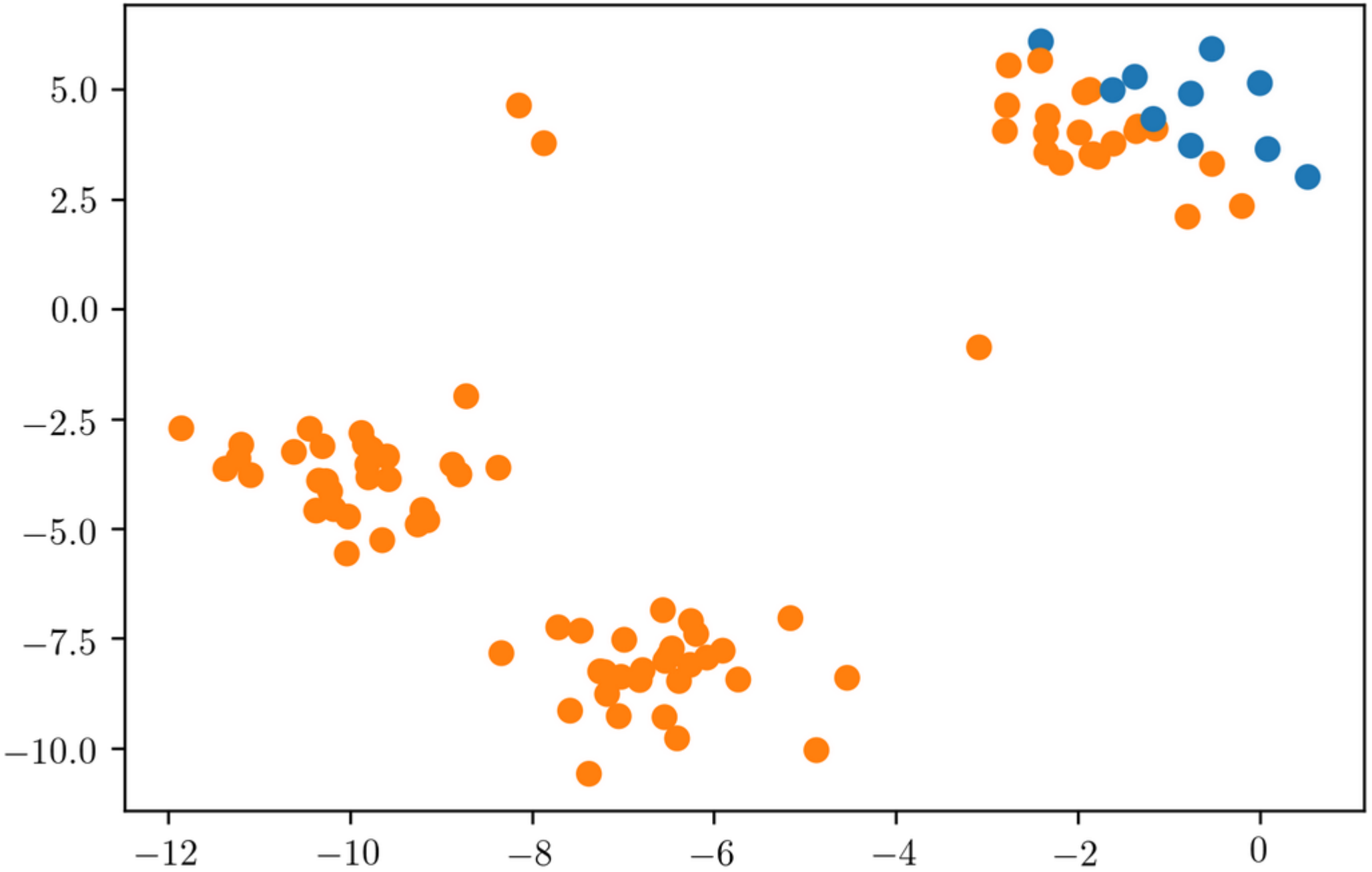


Elliptic Envelope

- Only works if Gaussian assumption is reasonable
- Preprocessing with PCA might help sometimes.

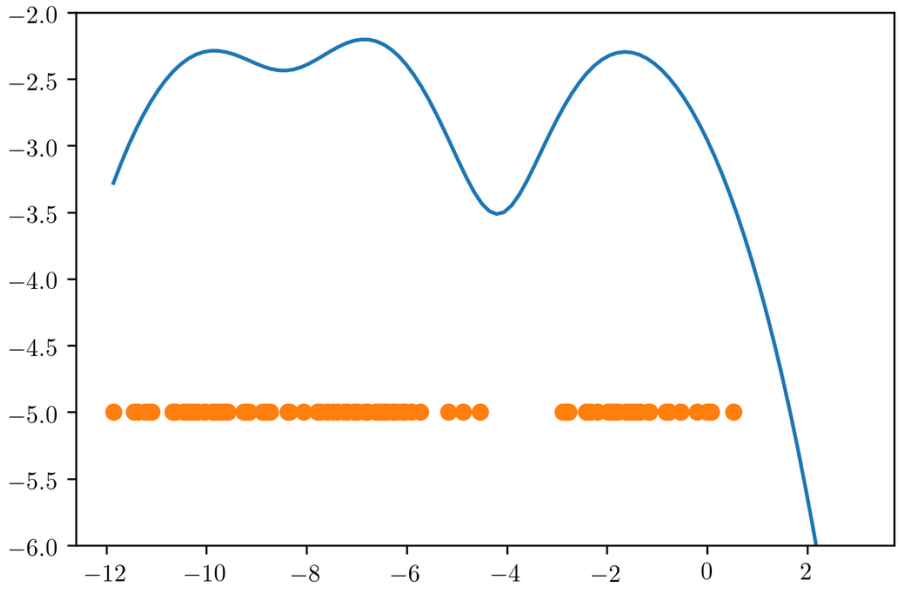
```
from sklearn.covariance import EllipticEnvelope
ee = EllipticEnvelope(contamination=.1).fit(X)
pred = ee.predict(X)
print(pred)
print(np.mean(pred == -1))
```

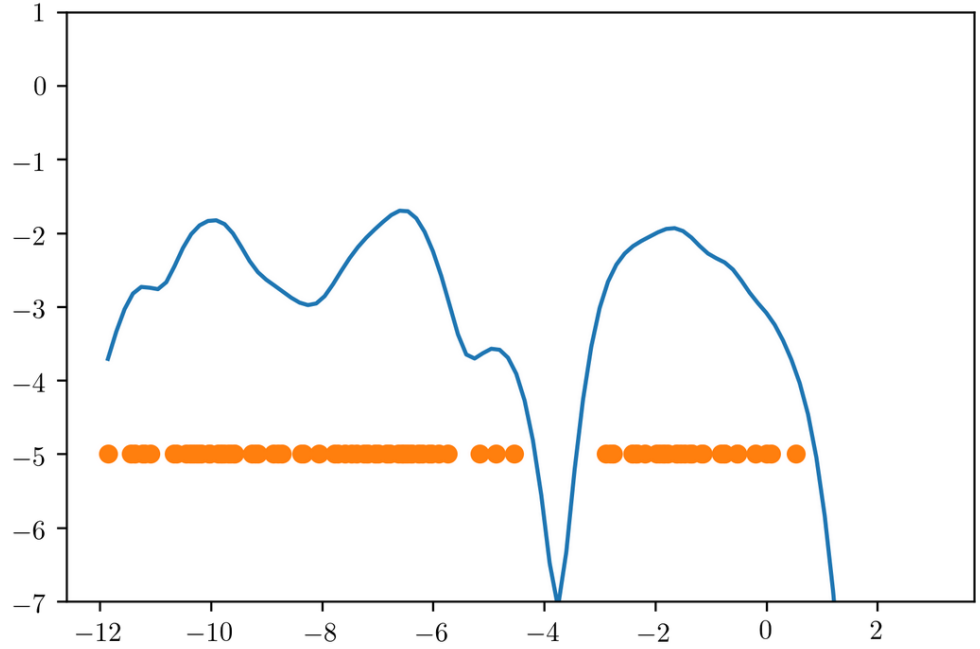
```
[ 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
 -1  1  1  1 -1 -1  1 -1  1  1 -1  1 -1 -1 -1  1  1 -1 -1 -1 -1  1 -1  1  1]
0.104
```


5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−12
−10
−8
−6
−4
−2
0

Kernel Density

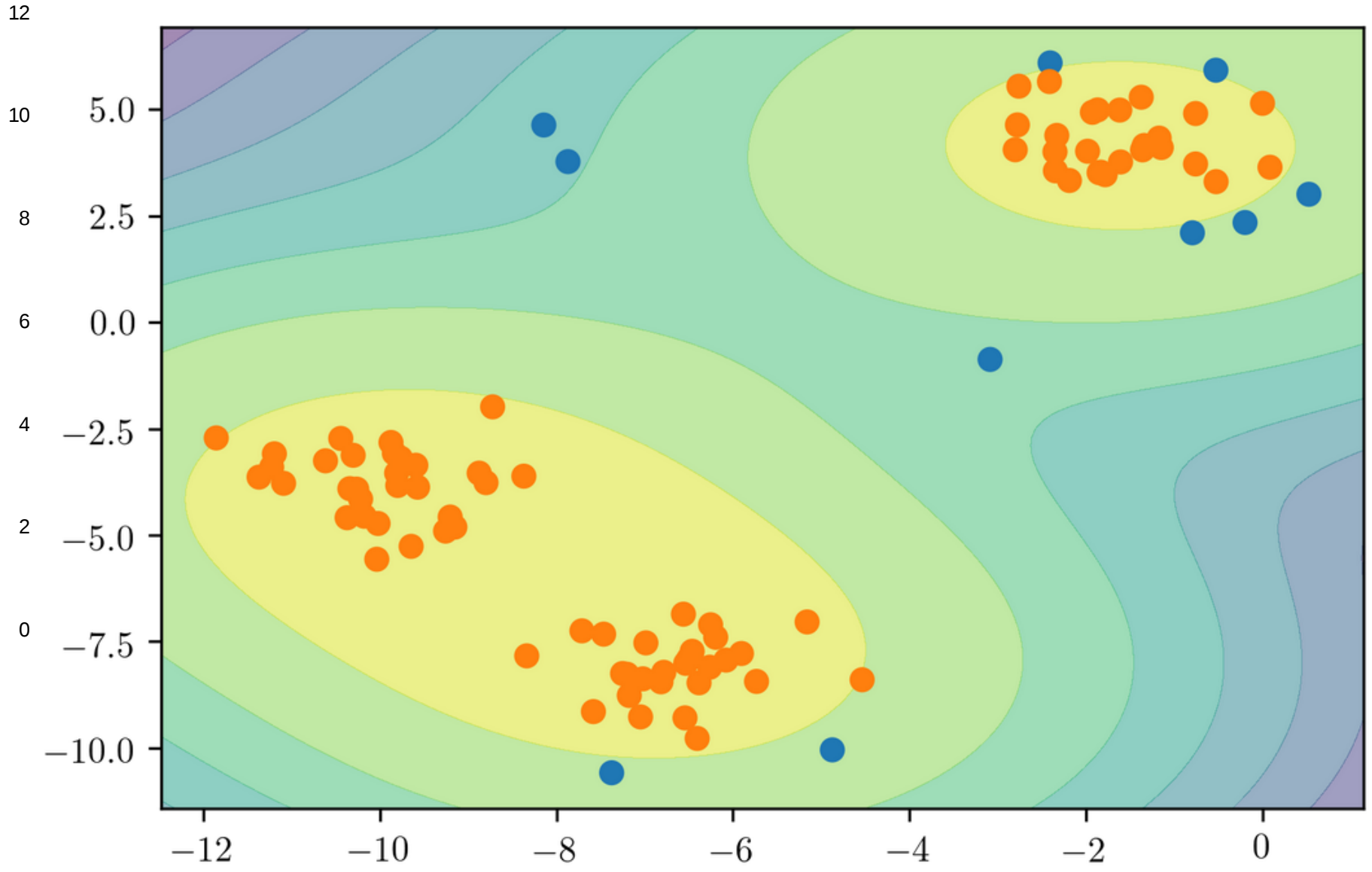
- Non-parametric density model
- Gaussian blob on each data point
- Need to adjust kernel bandwidth
- Doesn't work well in high dimensions





Unsupervised model, so how to pick?

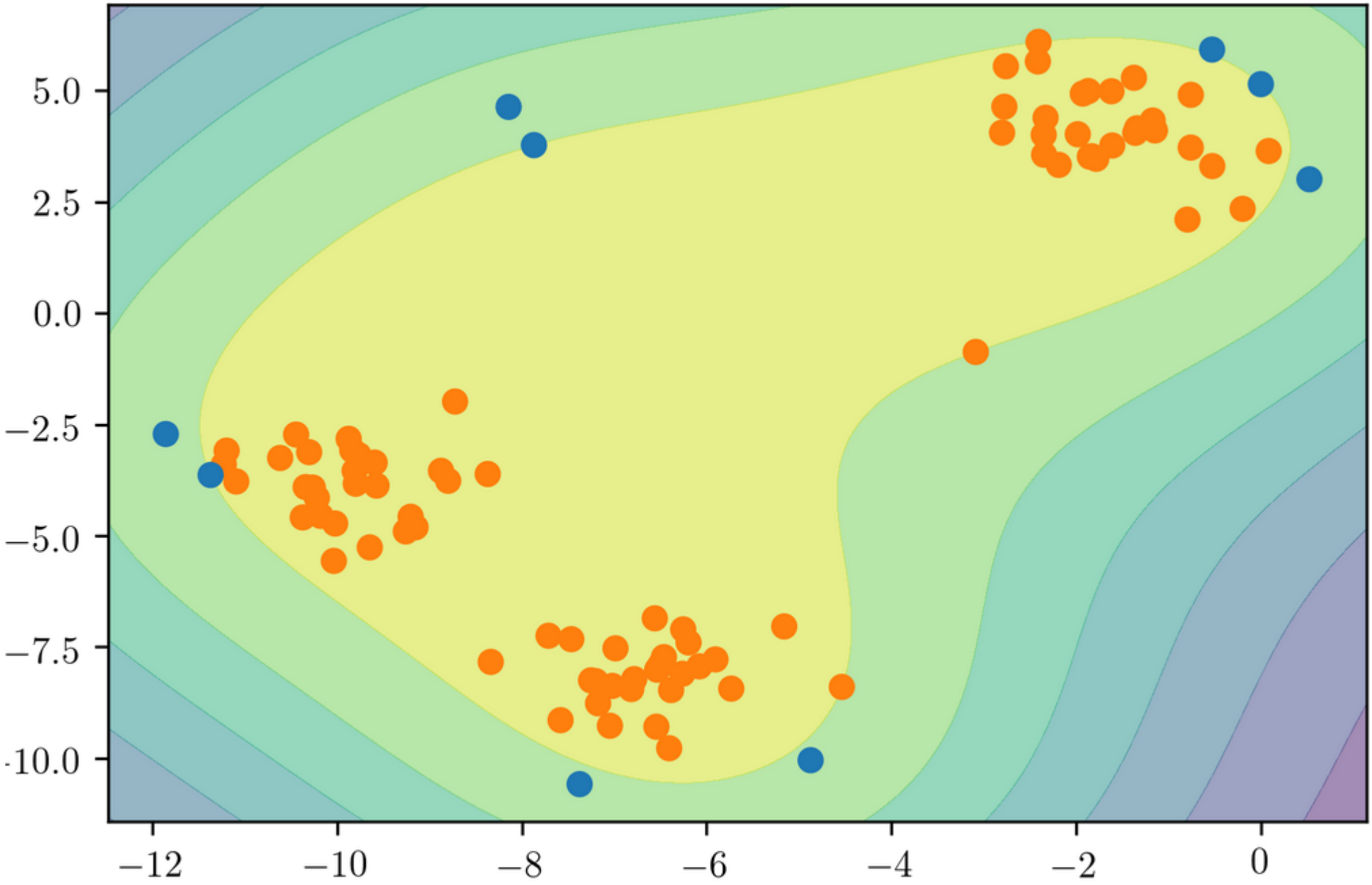
```
kde = KernelDensity(bandwidth=3)
kde.fit(X_train_noise)
pred = kde.score_samples(X_train_noise)
pred = (pred > np.percentile(pred, 10)).astype(int)
```



One Class SVM

- Also uses Gaussian kernel to cover data.
- Only select support vectors (not all points)
- Need to select gamma
- Specify outlier ratio (contamination) via nu (actually “fraction of training mistakes”)

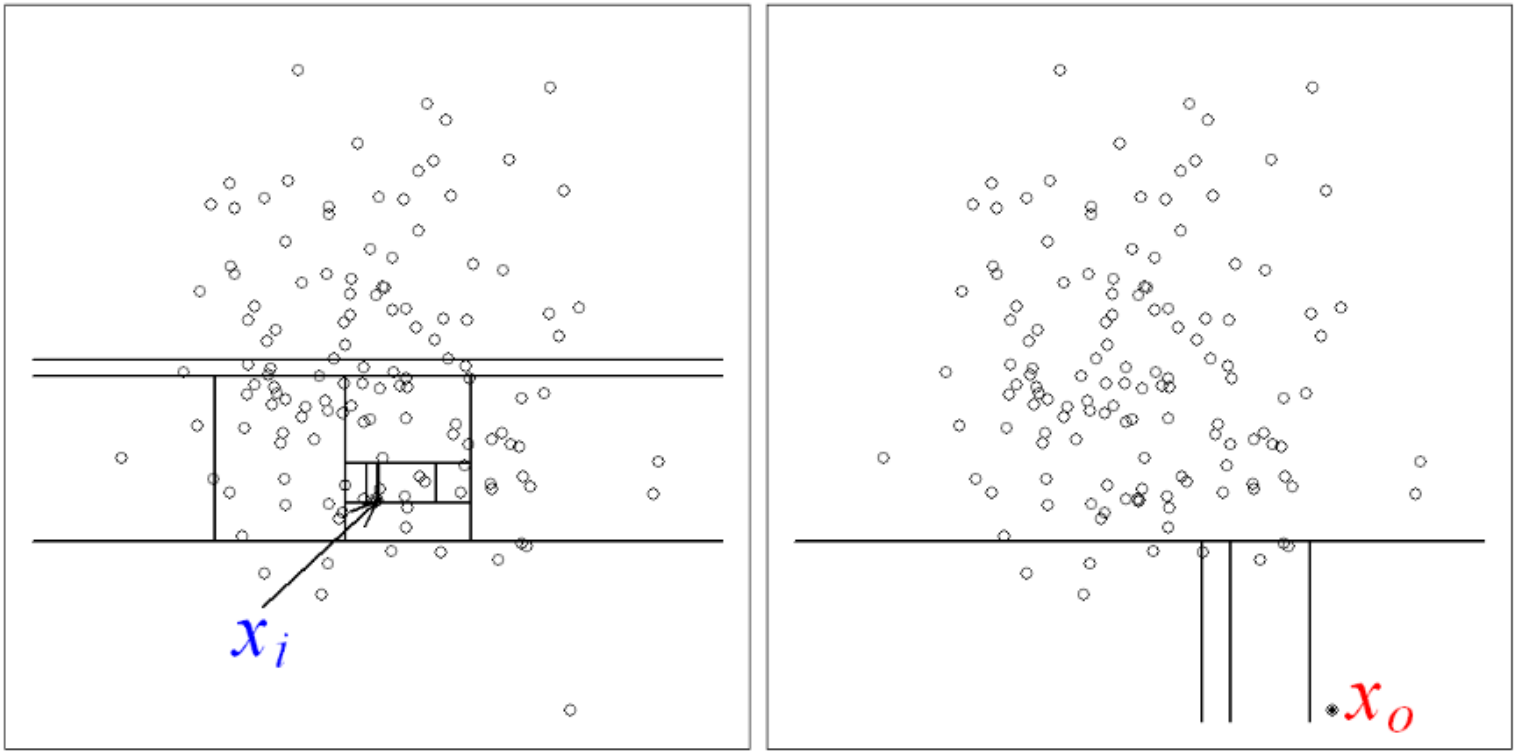
```
from sklearn.svm import OneClassSVM
scaler = StandardScaler()
X_train_noise_scaled = scaler.fit_transform(X_train_noise)
oneclass = OneClassSVM(nu=.1).fit(X_train_noise_scaled)
pred = oneclass.predict(X_train_noise_scaled).astype(np.int)
```


5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−12
−10
−8
−6
−4
−2
0

Isolation Forests

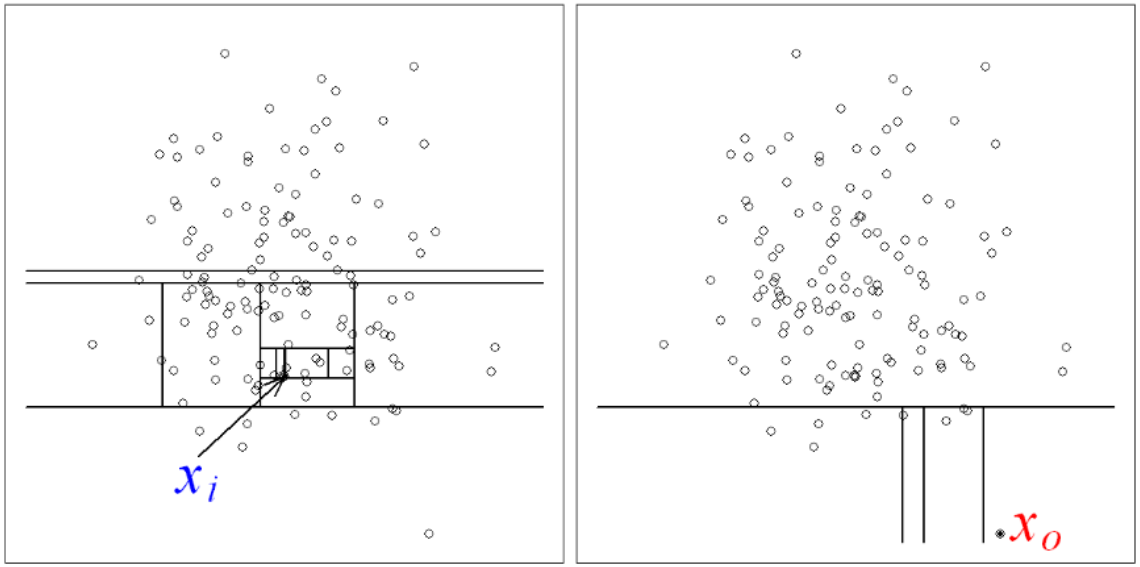
Idea

Build random trees
Outliers are easier to isolate from the rest

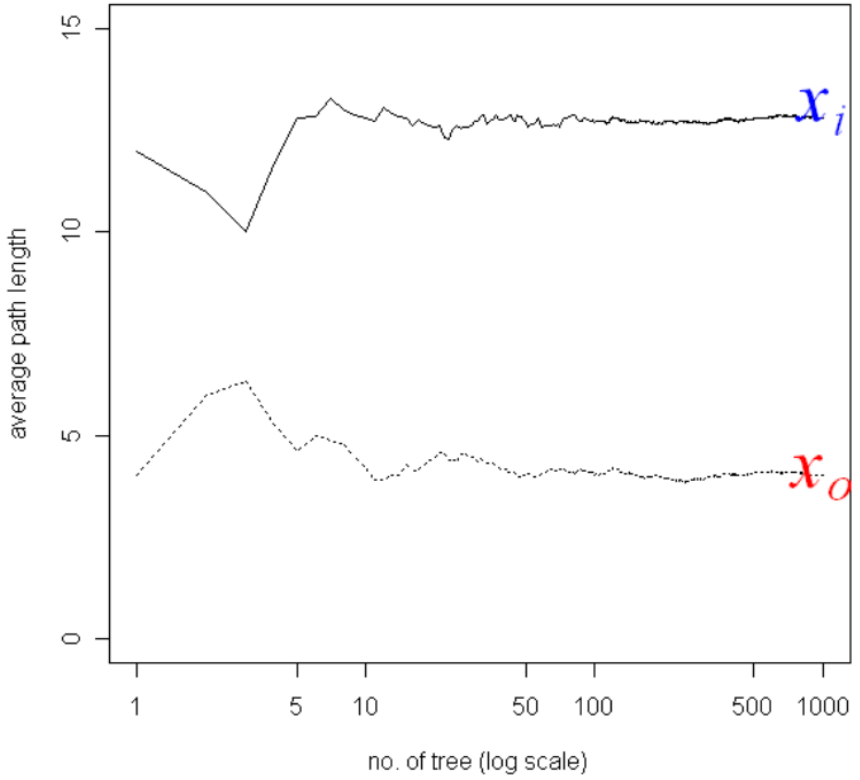


(a) Isolating x_i (b) Isolating x_o

Measure as Path length!



(a) Isolating x_i (b) Isolating x_o



Normalizing the Path Length

- Average path length of unsuccessful search in Binary Search Tree:

$$c(n) = 2H(n-1) - (2(n-1)/n),$$

H = Harmonic number

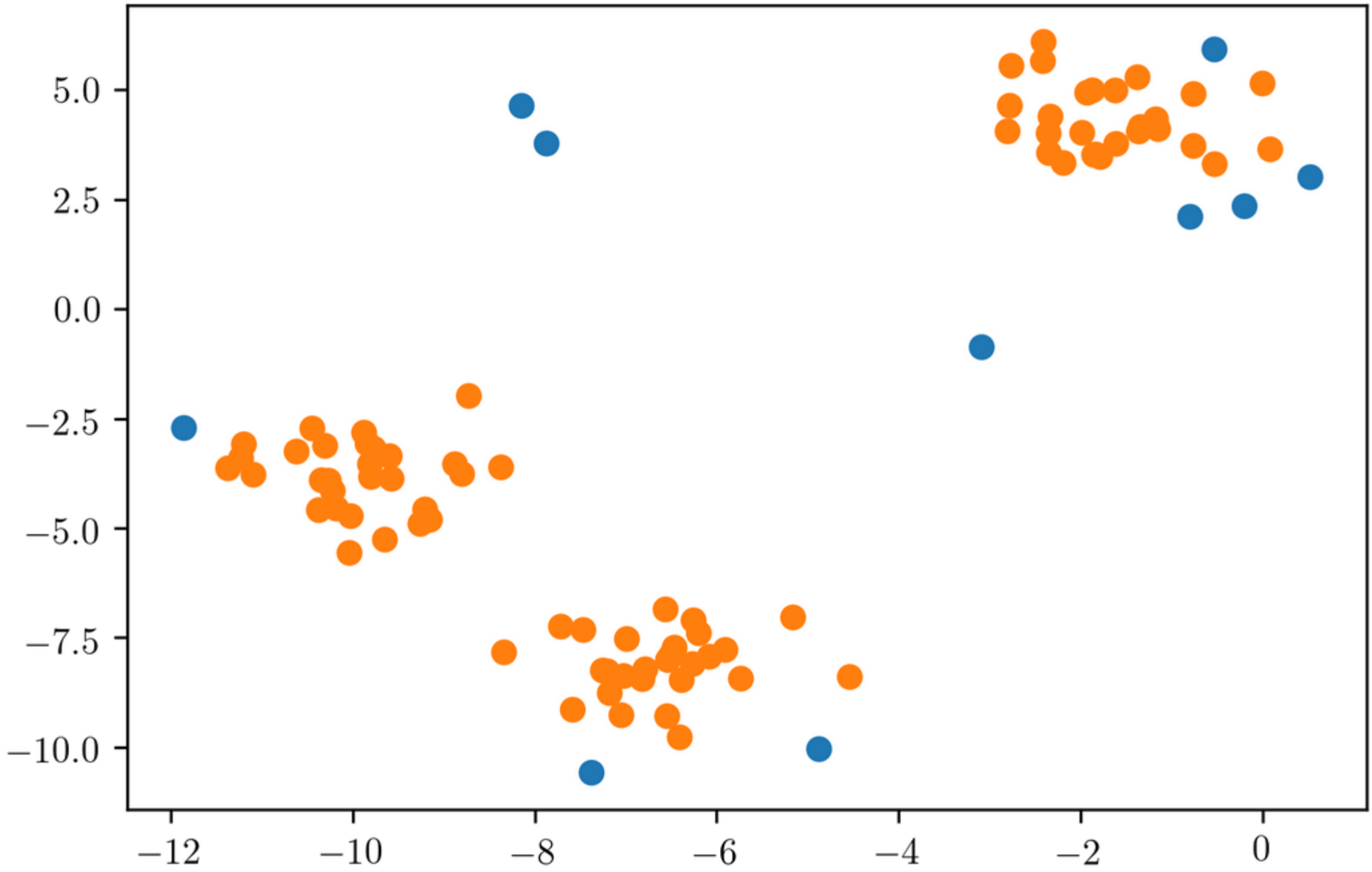
$$s(x,n) = 2^{-\frac{E(h(x))}{c(n)}},$$

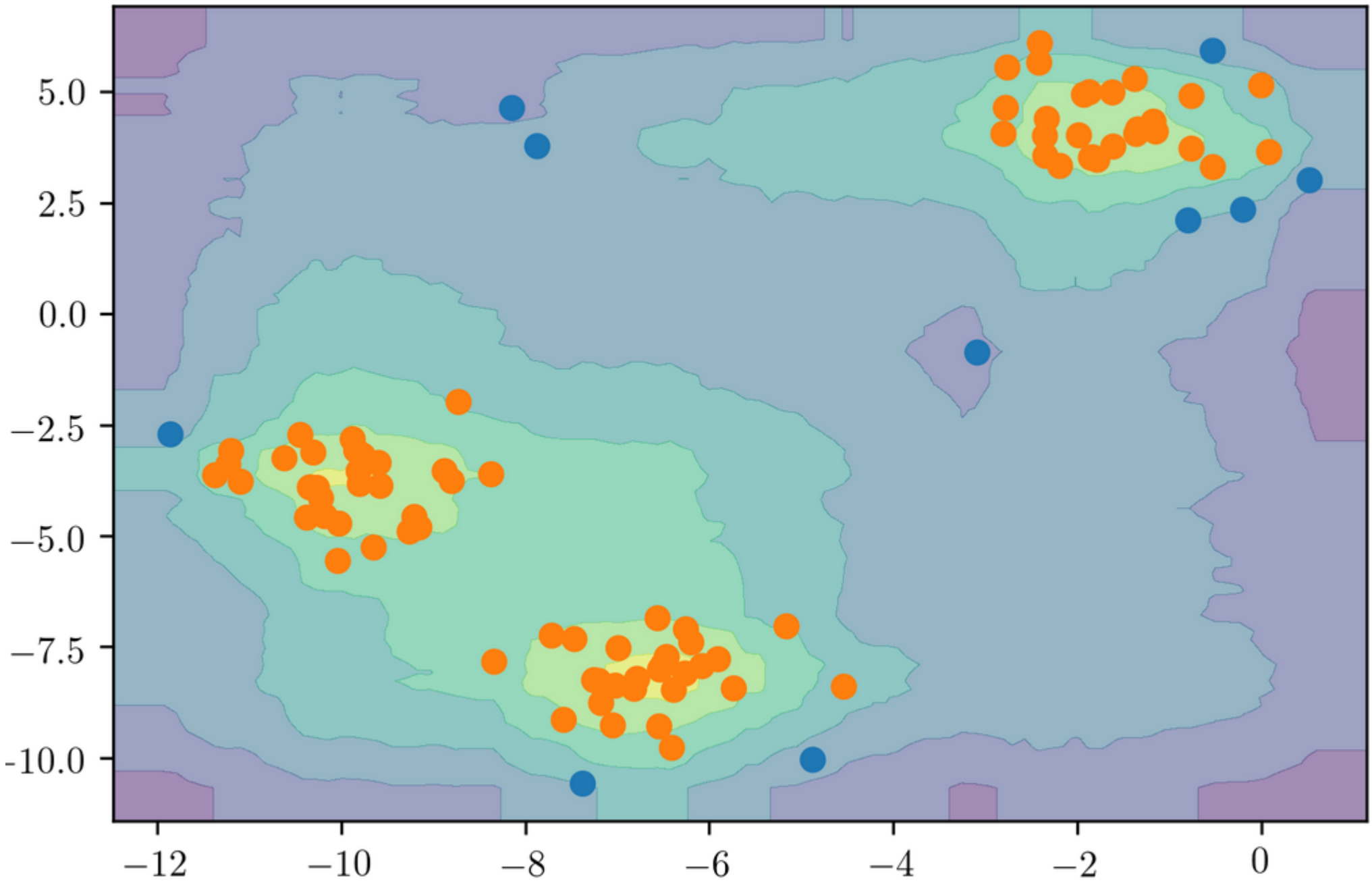
h = depth in three

s < .5: definite inlier
s close to 1: outlier

Building the forest

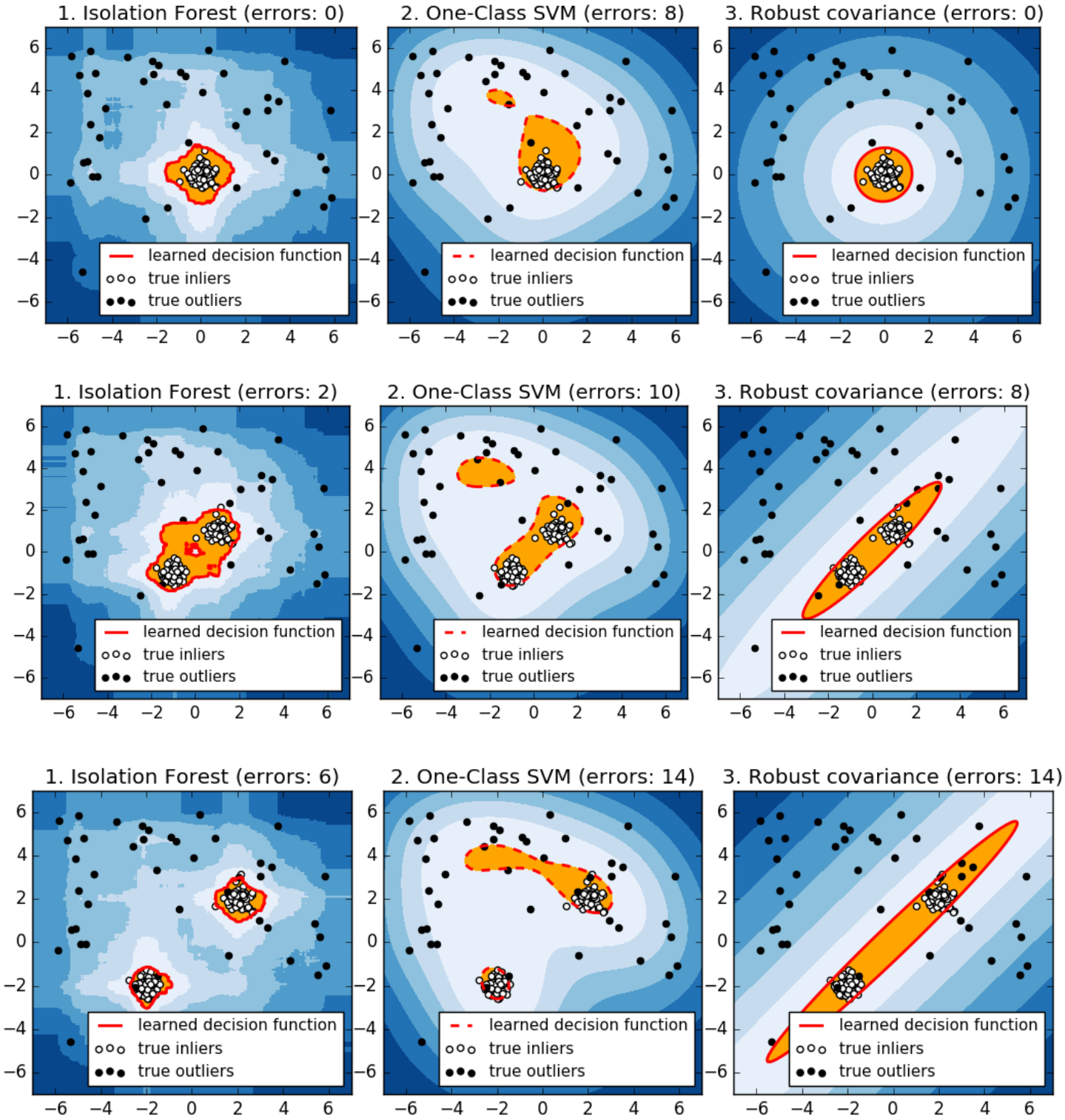
- Subsample dataset for each tree
- Default sample size of 256 works surprisingly well
- Stop growing tree at depth log_2(sample size) – so 8
- No bootstrapping usually
- More trees are better – default 100
- Need to specify contamination rate


5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−12
−10
−8
−6
−4
−2
0


5.0
2.5
0.0
−2.5
−5.0
−7.5
-10.0
−12
−10
−8
−6
−4
−2
0

Other density based-models

- PCA
- GMMs
- Robust PCA (not in sklearn :-()
- Any other probabilistic model - “robust” is better.


1. Isolation Forest (errors: 0)
2. One-Class SVM (errors: 8)
3. Robust covariance (errors: 0)
1. Isolation Forest (errors: 2)
2. One-Class SVM (errors: 10)
3. Robust covariance (errors: 8)
1. Isolation Forest (errors: 6)
2. One-Class SVM (errors: 14)
3. Robust covariance (errors: 14)
learned decision function
true inliers
true outliers

Summary

- Isolation Forest works great!
- Density models are great if they are correct.
- Estimating bandwidth can be tricky in the unsupervised setting.
- Validation of results requires manual inspection.