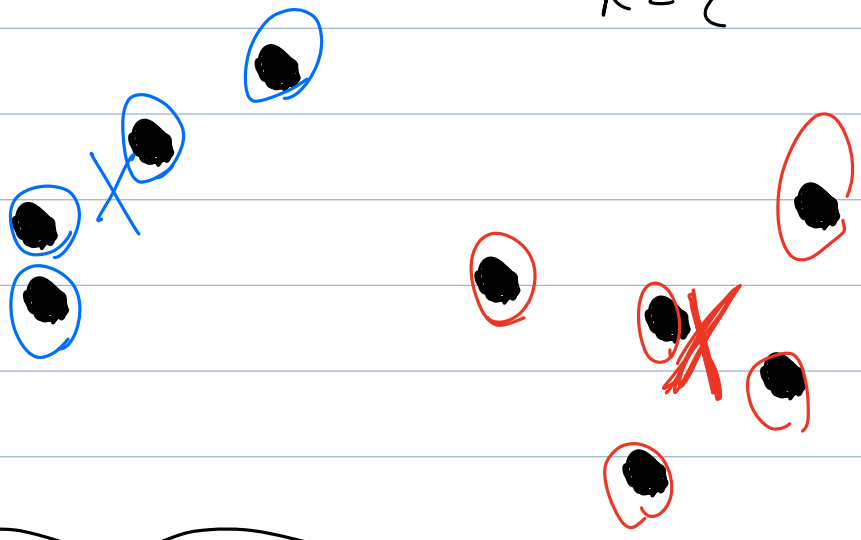
Supervised ← given label

Unsupervised ← not given

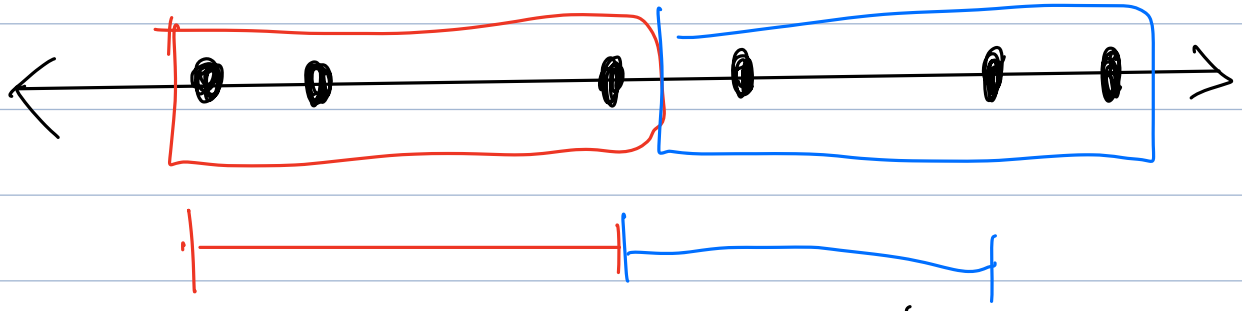
K-Means Clustering

↑ number

K = 2



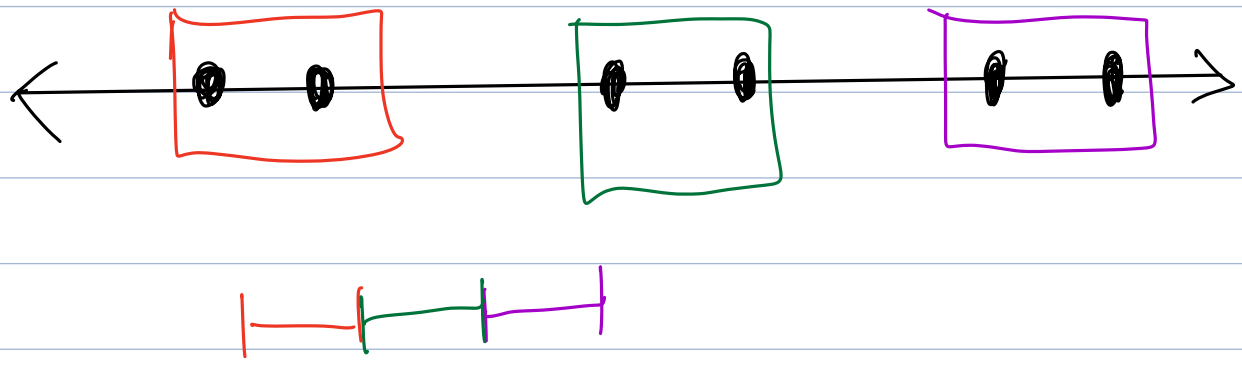
K = 2



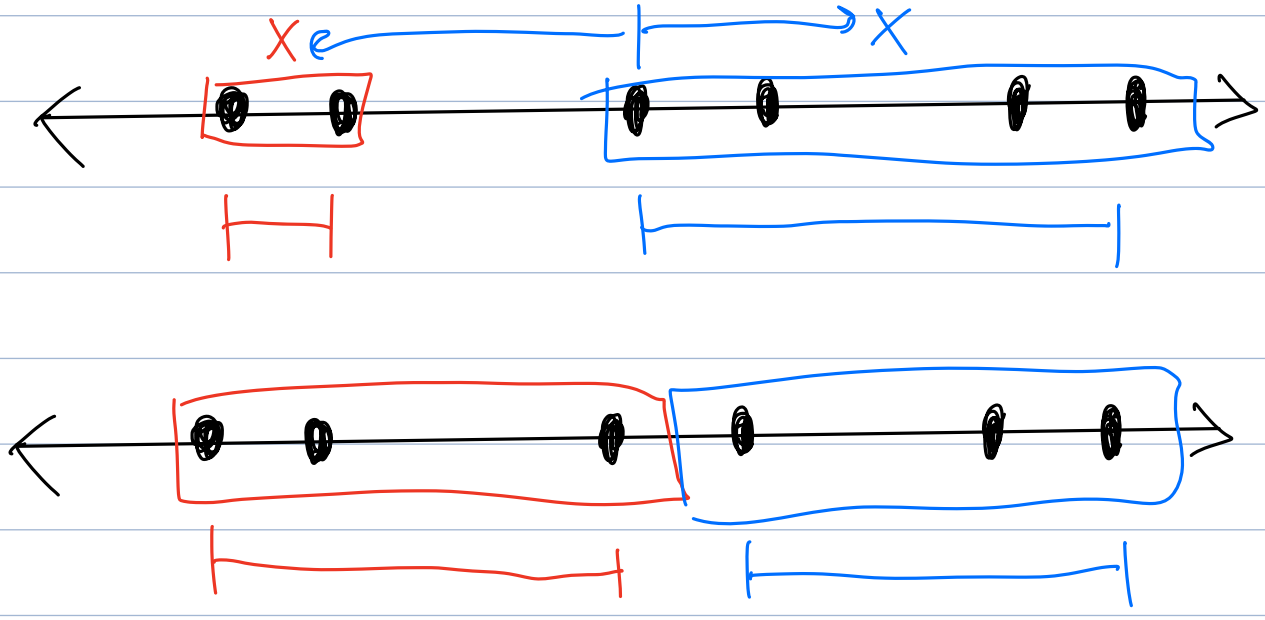
different k

k = 3

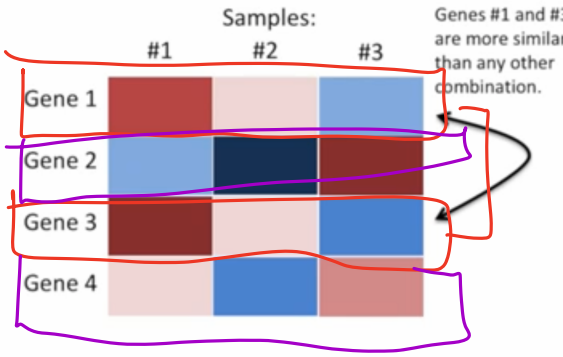
variation



Random initial points (centre)

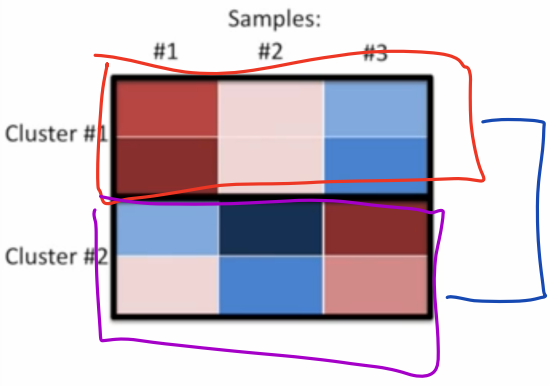


Hierachical Clustering



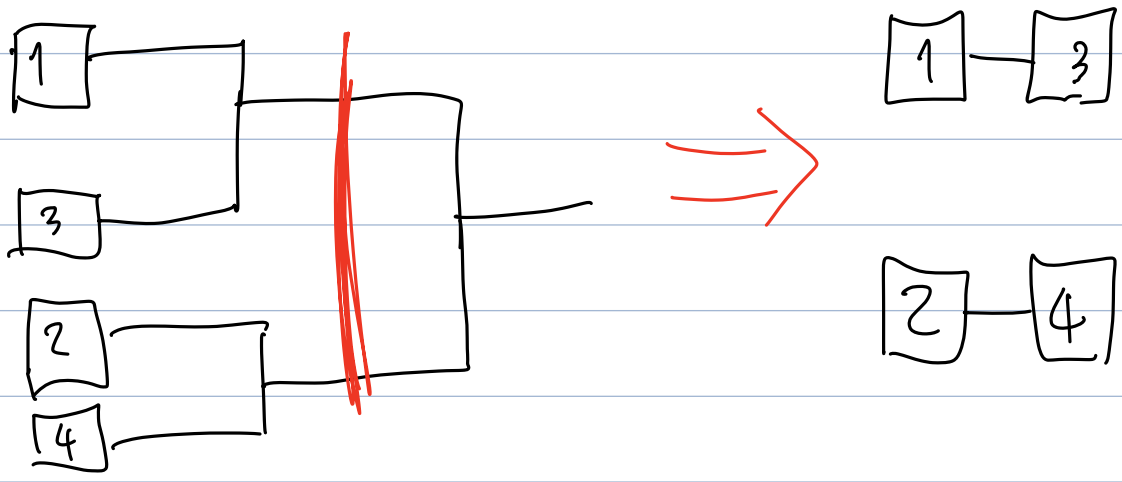
Conceptually...

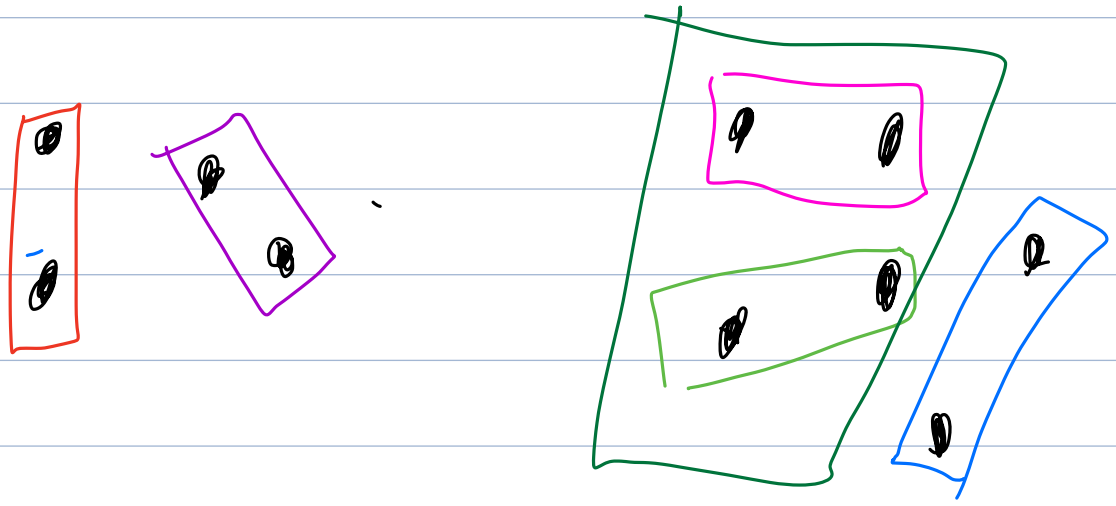
1) Figure out which gene is most similar to gene #1.
2) Figure out which genes is most similar to gene #2... (and then #3 and then #4).
3) Of the different combinations, figure out which two genes are the most similar. Merge them into a cluster.



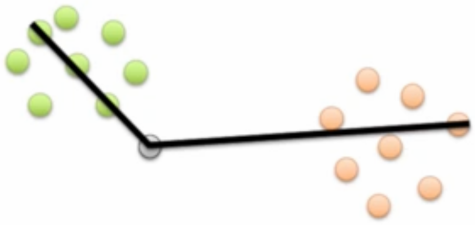
Conceptually...

1) Figure out which gene is most similar to cluster #1.
2) Figure out which genes is most similar to gene #2... (and then #4).
3) Of the different combinations, figure out which two genes are the most similar. Merge them into a cluster.
4) Go back to step 1.



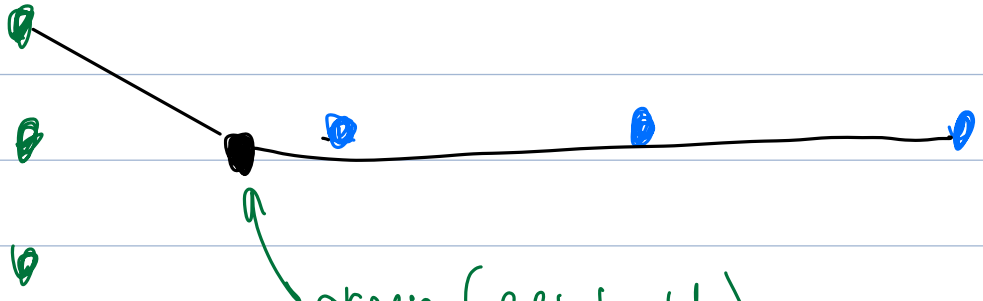


For the sake of visualizing how the different methods work, imagine our data was spread out on an X-Y plane.



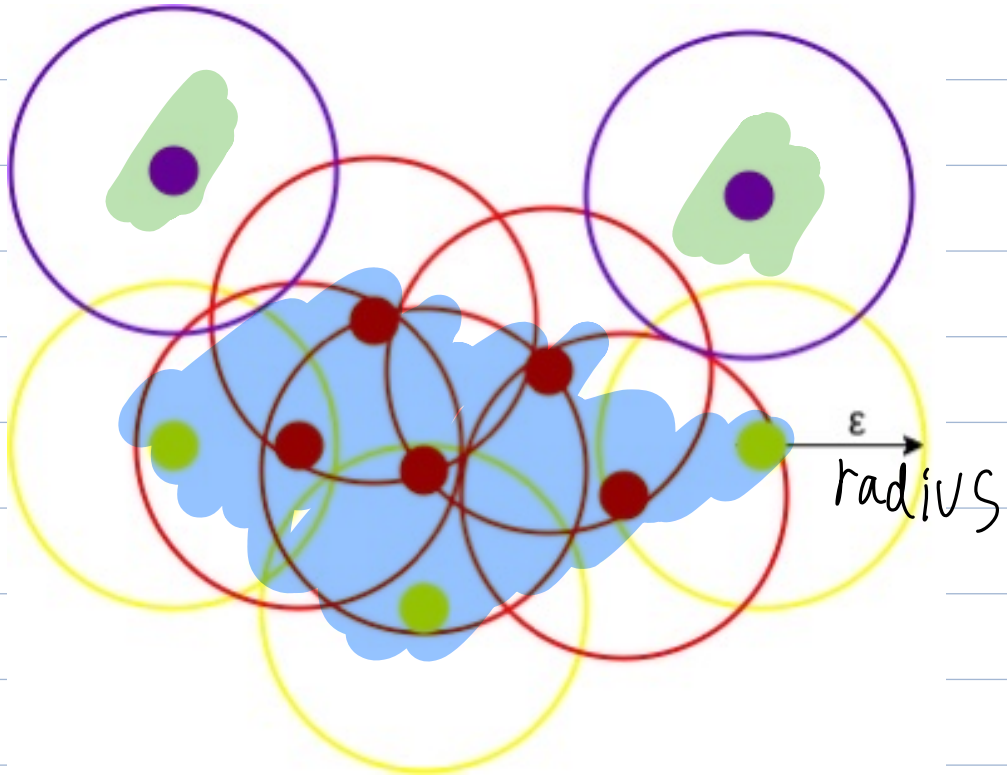
We can compare that point to...

1) The average of each cluster (this is called the "centroid")
2) The closest point in each cluster (this is called "single-linkage")
3) The furthest point in each cluster (this is called "complete-linkage")



green (centroid)
blue (closest)
green (furthest)

DBscan



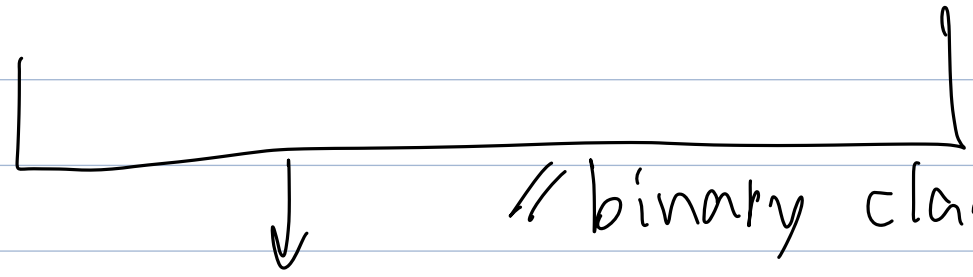
1) Core
2) border
3) Noise

Specify 1) radius (ε) 2
2) Min points 4

4 If there are points(data) $\geq$ MIN_POINTS : assign as "core"

4 If there are 1 < points(data) $\leq$ MIN_POINTS : assign as "border"

If there are points(data) == 1 : assign as "Noise"



don't include when scaling & one-hot encoding

Rules of thumb: Always OHE

but if data is in binary then OHE is optional

k_list = [1, 3, 5, 100]

```
for k in k_list:
    print(k) → 1
               3
               5
               100
```

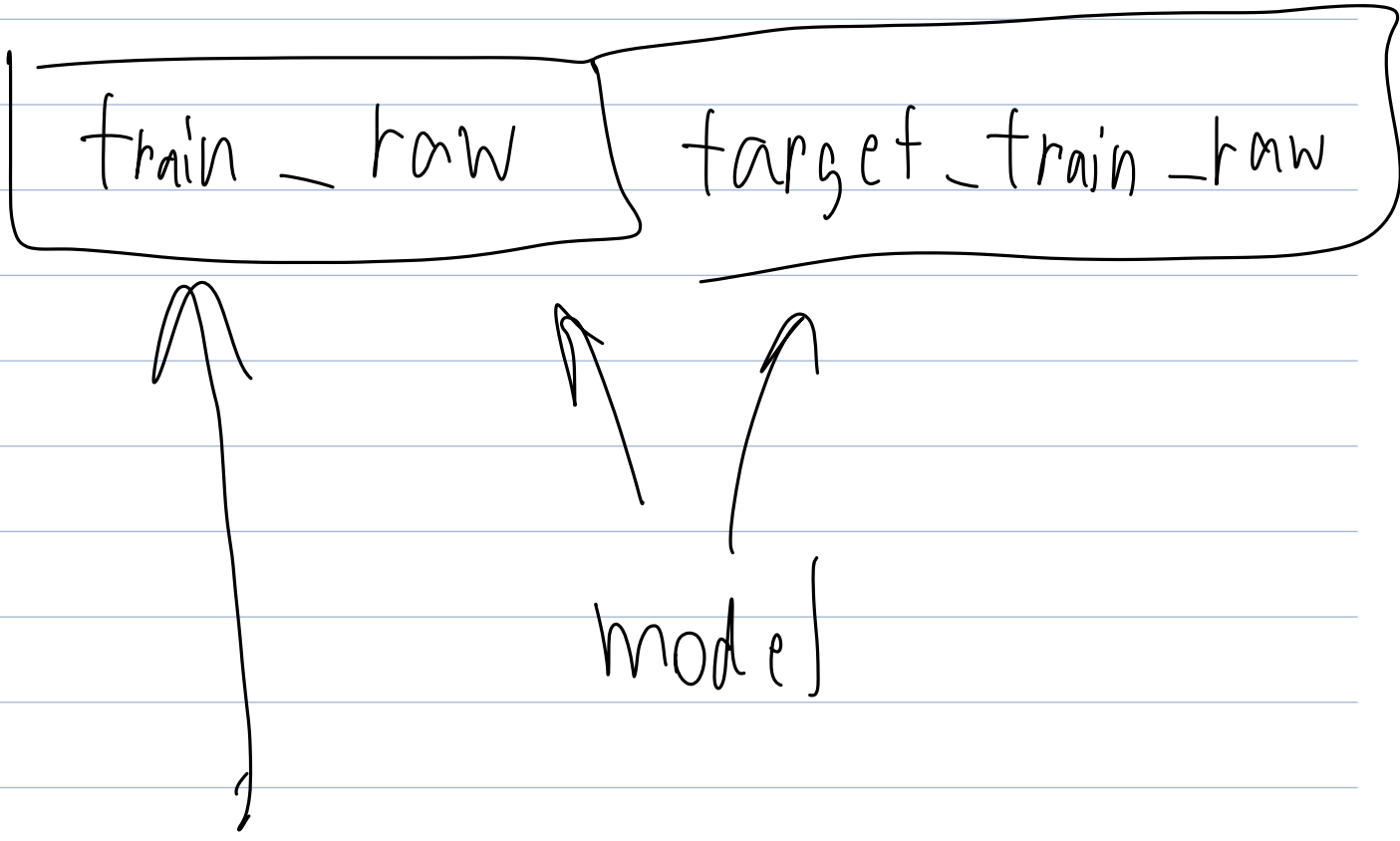
range(start, end)

↳ range(0, 10)

list(range(0, 10))

↳ [0, 1, 2, 3, ..., 10]

```
for i in range(0, 10)
    print(i) → 0 1 ...
    print(k_list[i]) → 1 3
```



testing prediction:

build the model

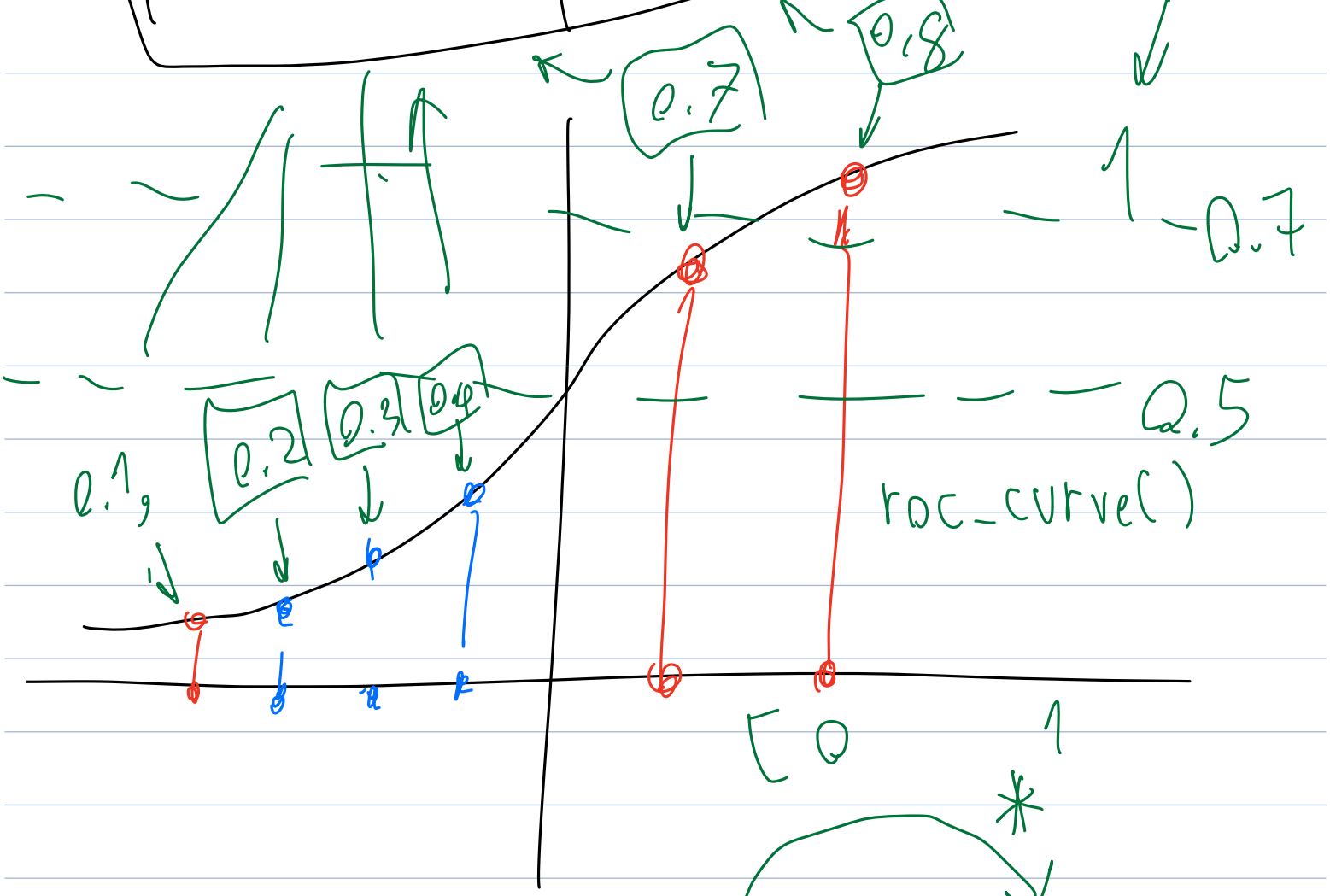
predict & compare

X_train, X_test, Y_train, Y_test =

train_test_split()

predicted_proba()

predict()



num_pipeline.
fit_transform(heart_numerical_data)

numerical
- imp
- Avg
- Sca

raw_data

= categorical — encode

full_pipeline.fit_transform(raw_data)

[data]

compare

train, test, target, test_target = train_split()

penalty='none'
'l1'

LinearRegression()

lin_reg

fit(train)

→ lin_reg.fit(train, target)

→ predicted = lin_reg.predict(test)

→ print_4_..(predicted, ...)