

# Breast Cancer Classification

Human Ensemble

J.M. Park\_S.N.U  
J.H. Park\_S.N.U  
H.W. Jung\_I.H.U  
H.Y. Choi\_C.N.U

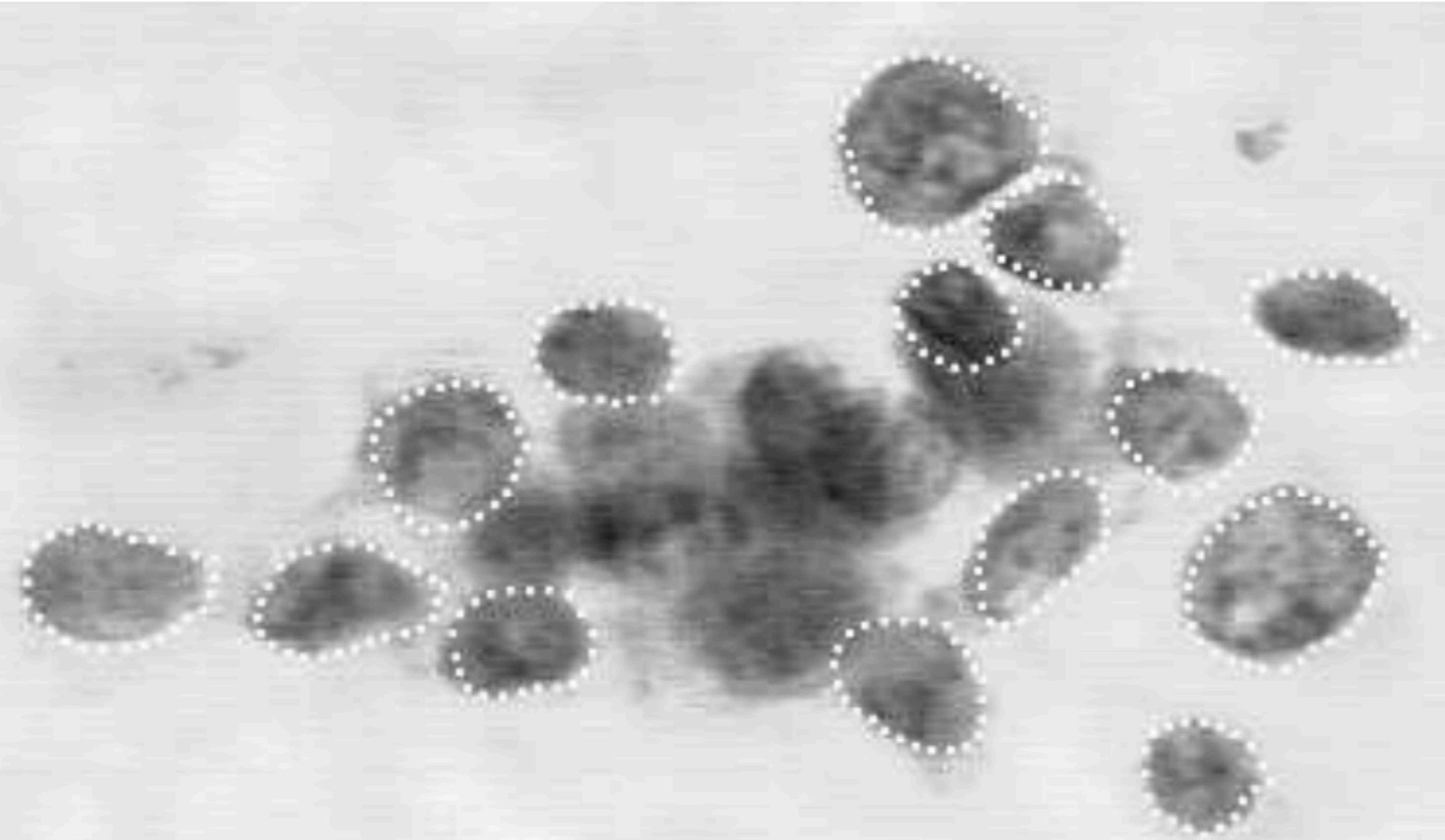
# Search Prominent Kernels

이미 모범 답안 처럼

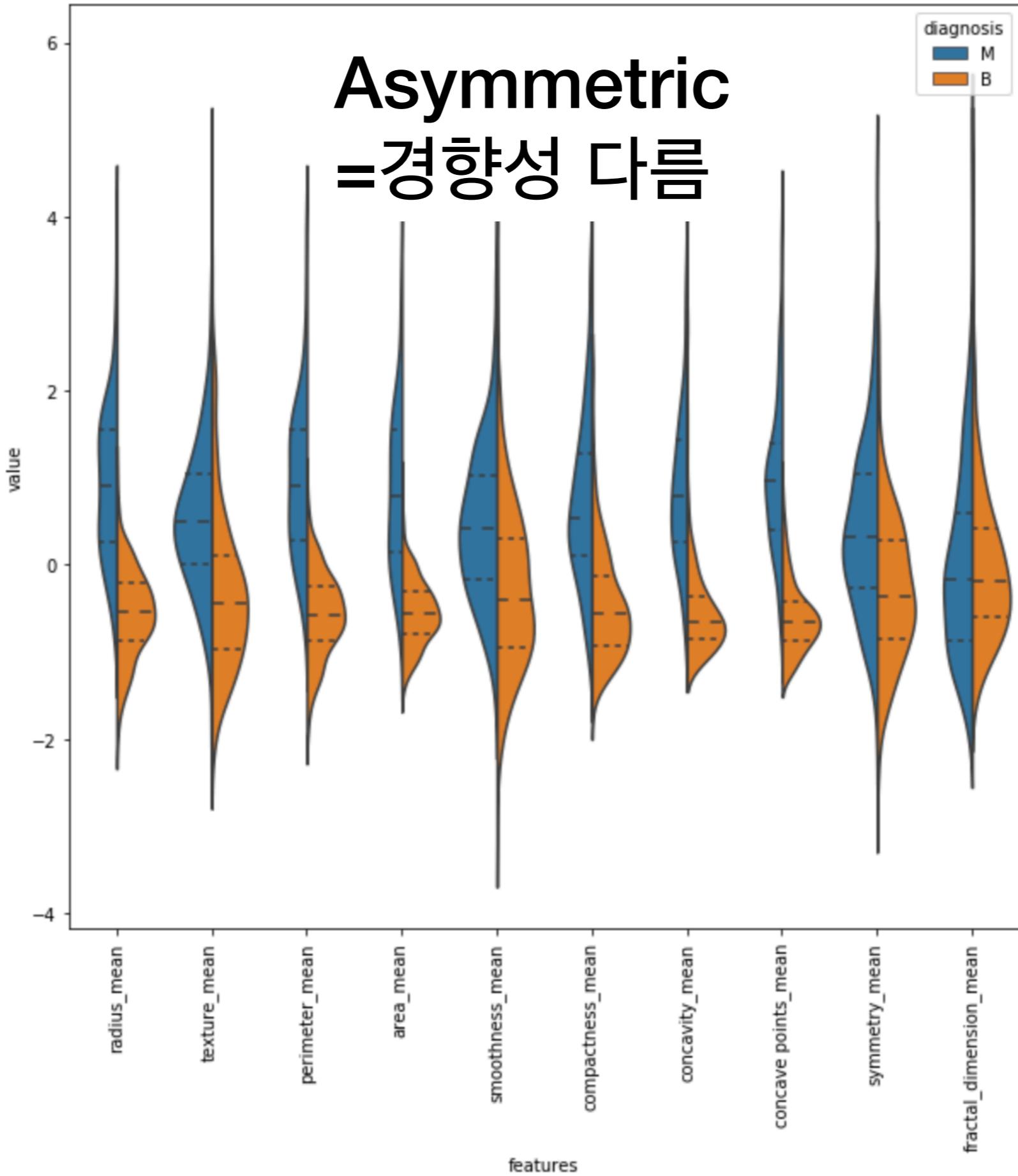
**Features Accuracy**

최적화 돼있음

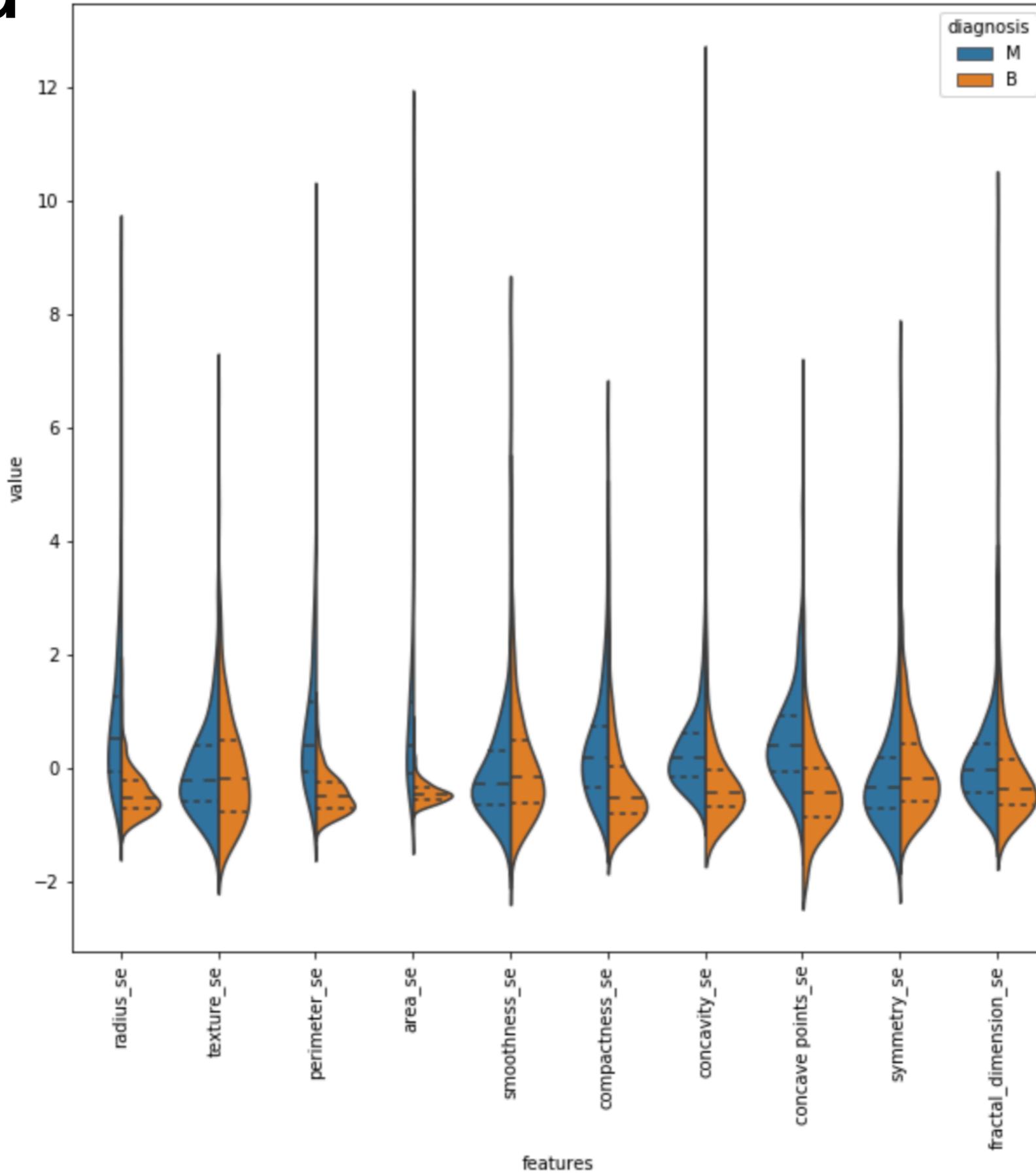
# Search Paper



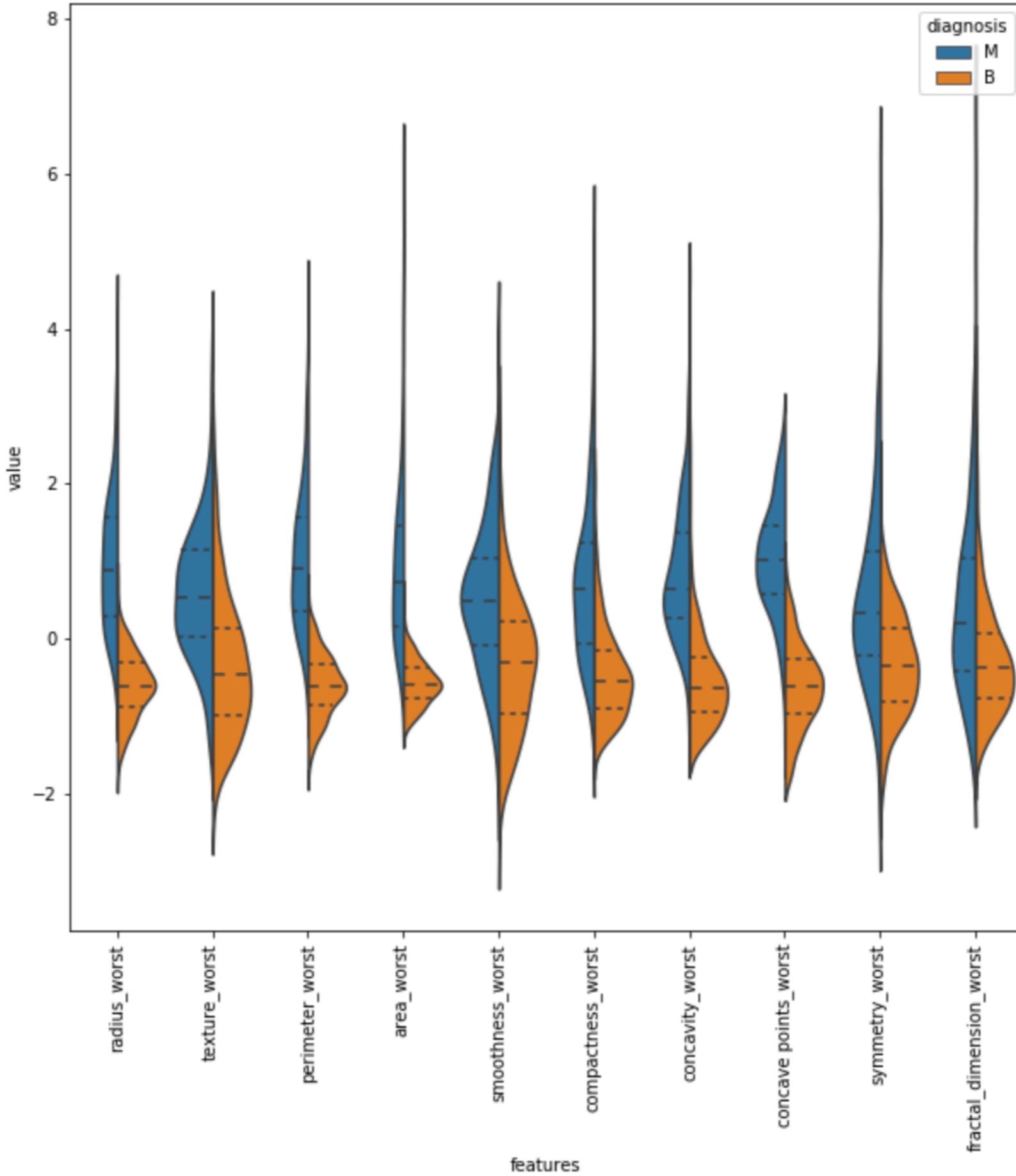
# Mean



# Standard Error



# Worst



# What to Do

Feature  
Dimension  
Reduction

VS

Performance  
Improvement

# First Trial

## Performance Improvement

### 1. Data Augment for Original models

By using **Standard error**

Perturb Mean & Worst Data

**Statically Stable?**

**Not much different  
from noise!**

$\pi_\pi$

## Performance Improvement

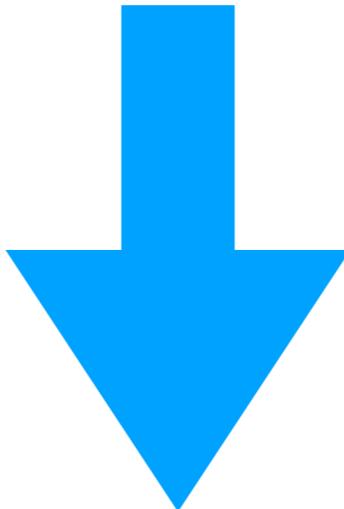
### 2. Build Robust Model (Accuracy 100%)

Difficult to Adopt to  
New Features

# Hypothesis

If, 독립적인 Feature 찾으면

Then, Optimized Feature 찾을 수 있을 것이다



최소 Feature로 최대 정보 표현

# Goal

# Optimized Feature

Feature Selection Method

Dominant Feature

# Dimension reduction

## 1. PCA, NMF 결과 확인

15~17 개의 Feature들로도 Accuracy 높음

**NEW  
METHOD!**

# Feature Normalize

Feature data를 각각  
Normalize **WHY?**

크기가 다르다고해도  
큰 의미가 없기 때문에  
각도만 재면 된다

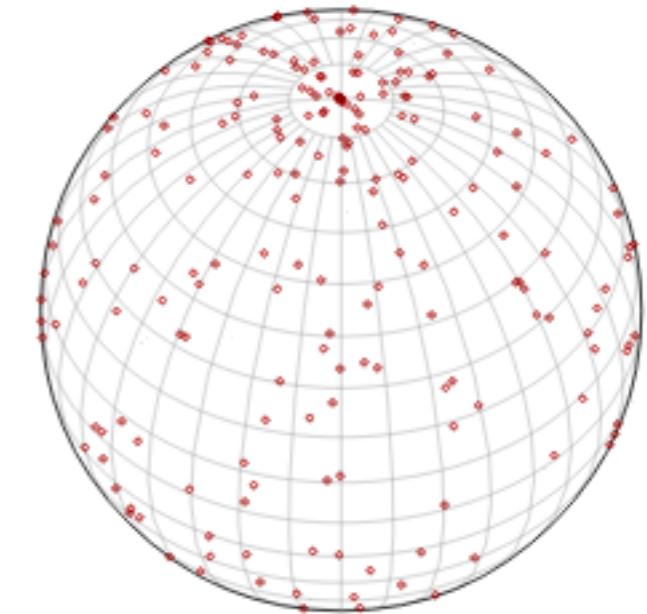
# Max Min Selector

$f$ : feature vector

Normalize

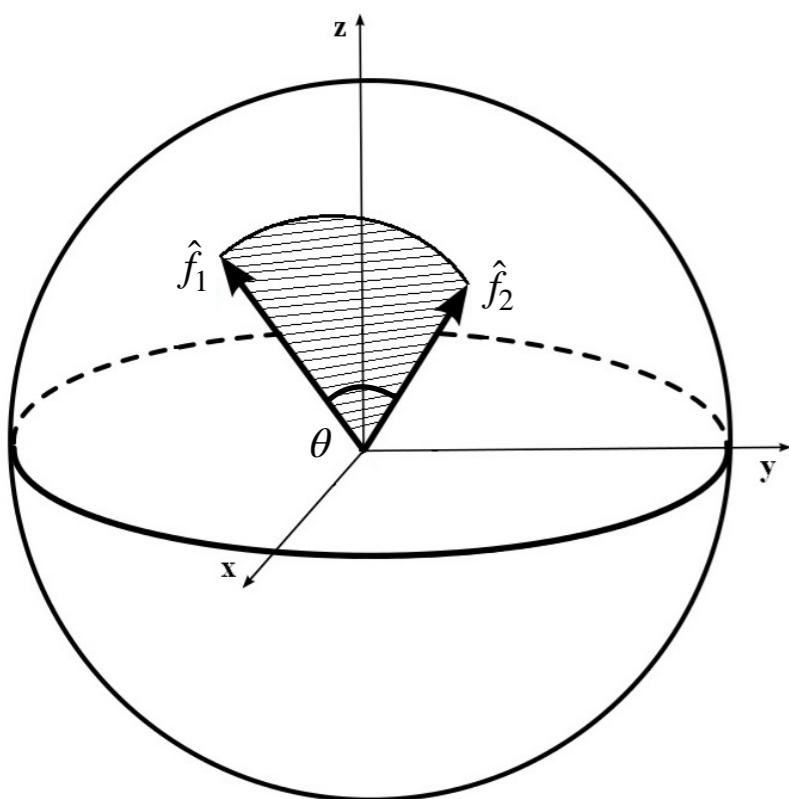
$$\hat{f} = \frac{f - \mu_f}{\sigma_f}$$

Normalize

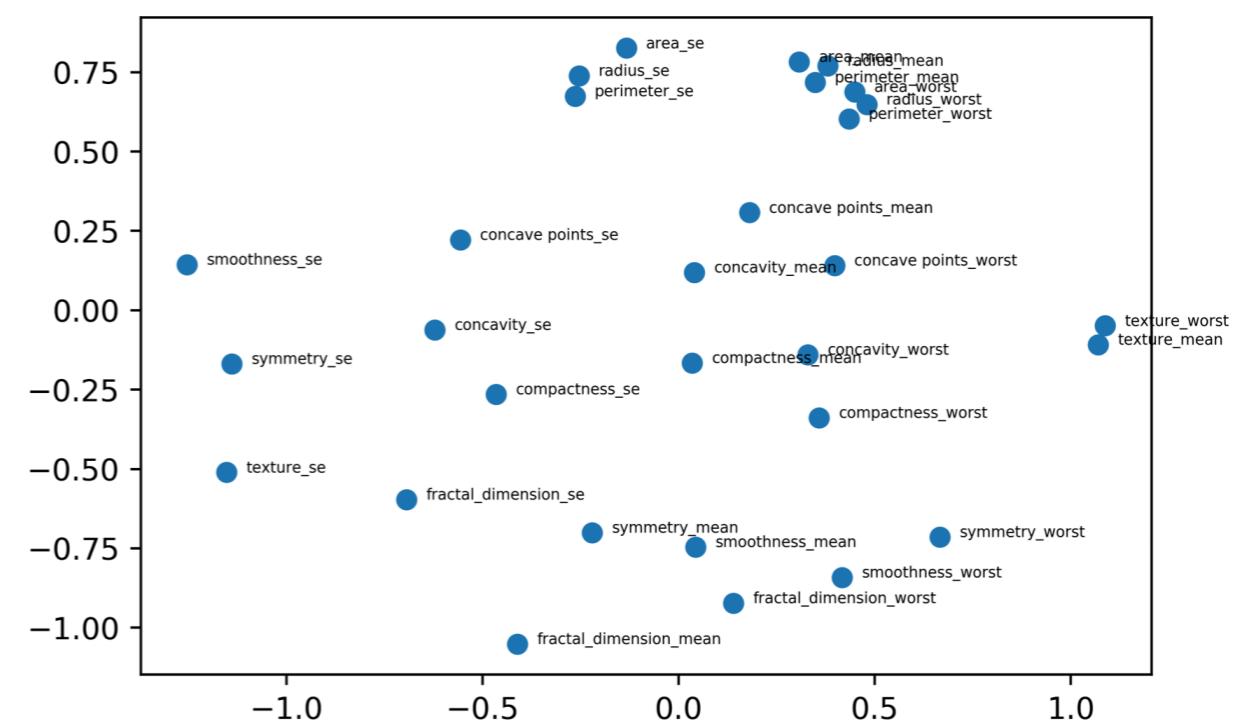


$$\text{corr}(f_1, f_2) = \langle \hat{f}_1, \hat{f}_2 \rangle$$

$$d(\hat{f}_1, \hat{f}_2) = \theta = \arccos(\langle \hat{f}_1, \hat{f}_2 \rangle) = \arccos(\text{corr}(f_1, f_2))$$



MDS



# Dimension reduction

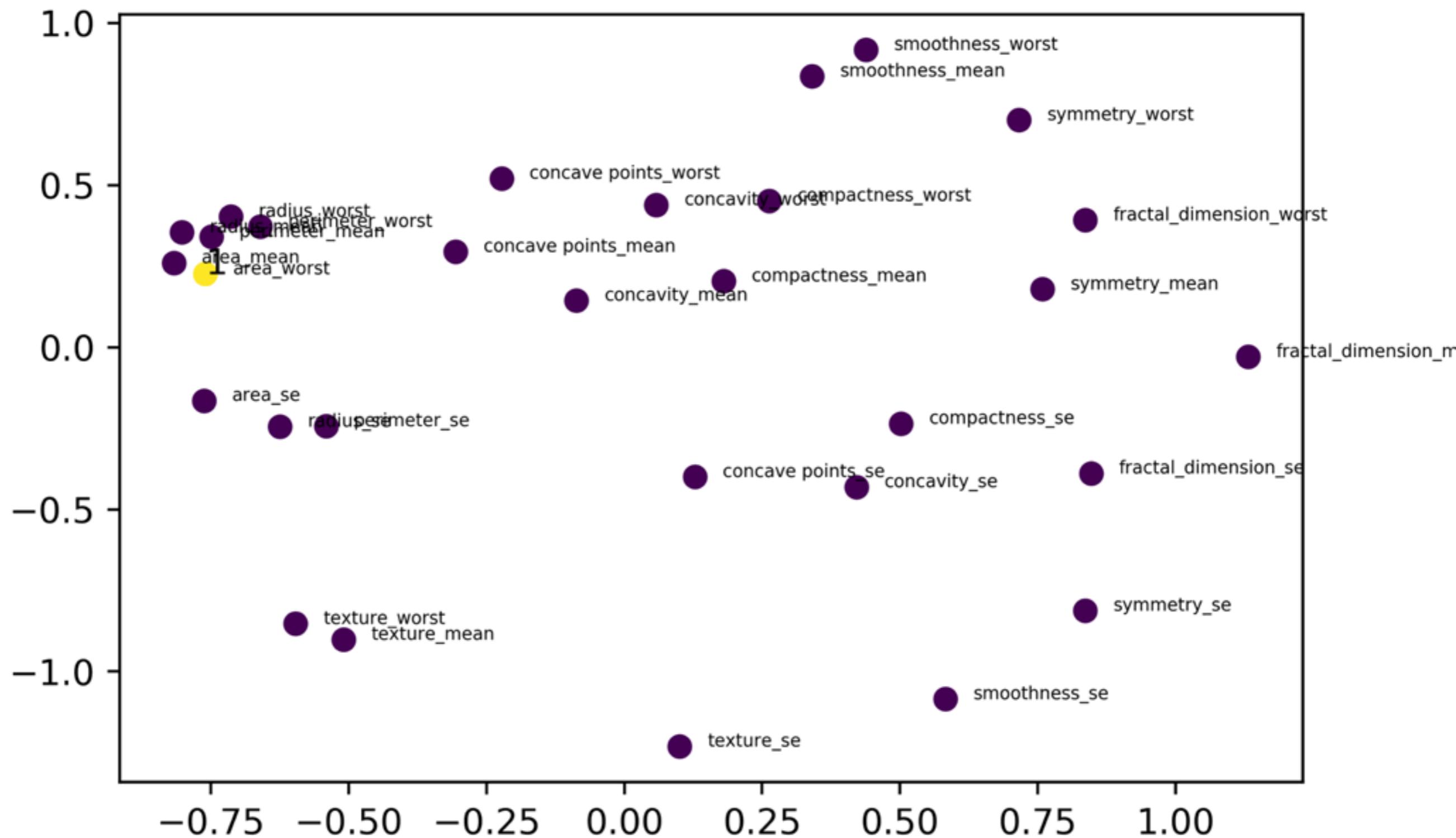
## Max Min Selector

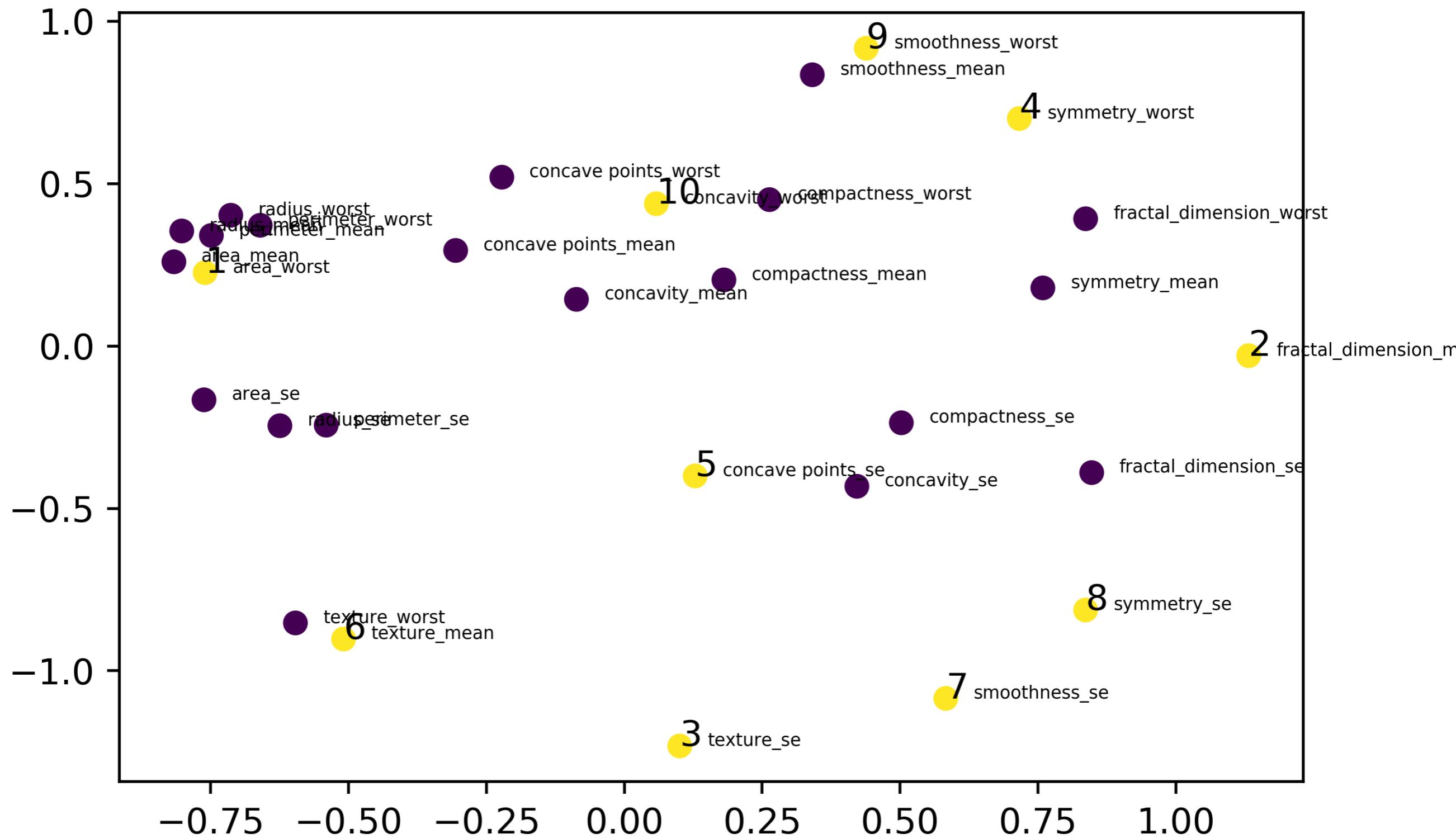
### Selector Part

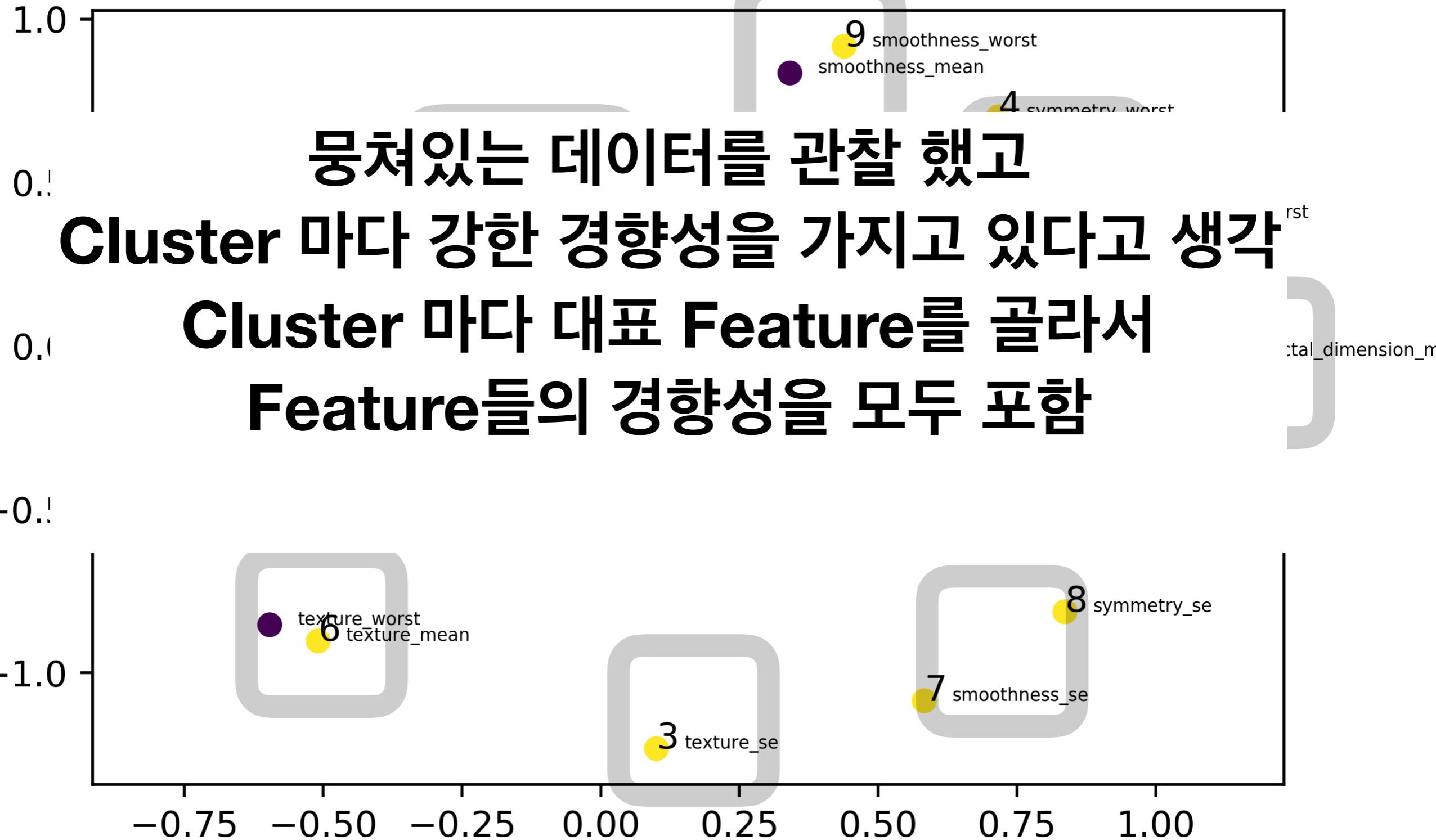
```
def maxminSelector(data, n, initial):
    data = (data-data.mean())/data.std()
    dissmat = pd.DataFrame(np.arccos(np.corrcoef(data.values.T))),index=features,columns=features
    result = [initial]
    n-=1
    while(n!=0):
        n -= 1
        result += [dissmat[result].min(axis=1).idxmax()]
    return result
```

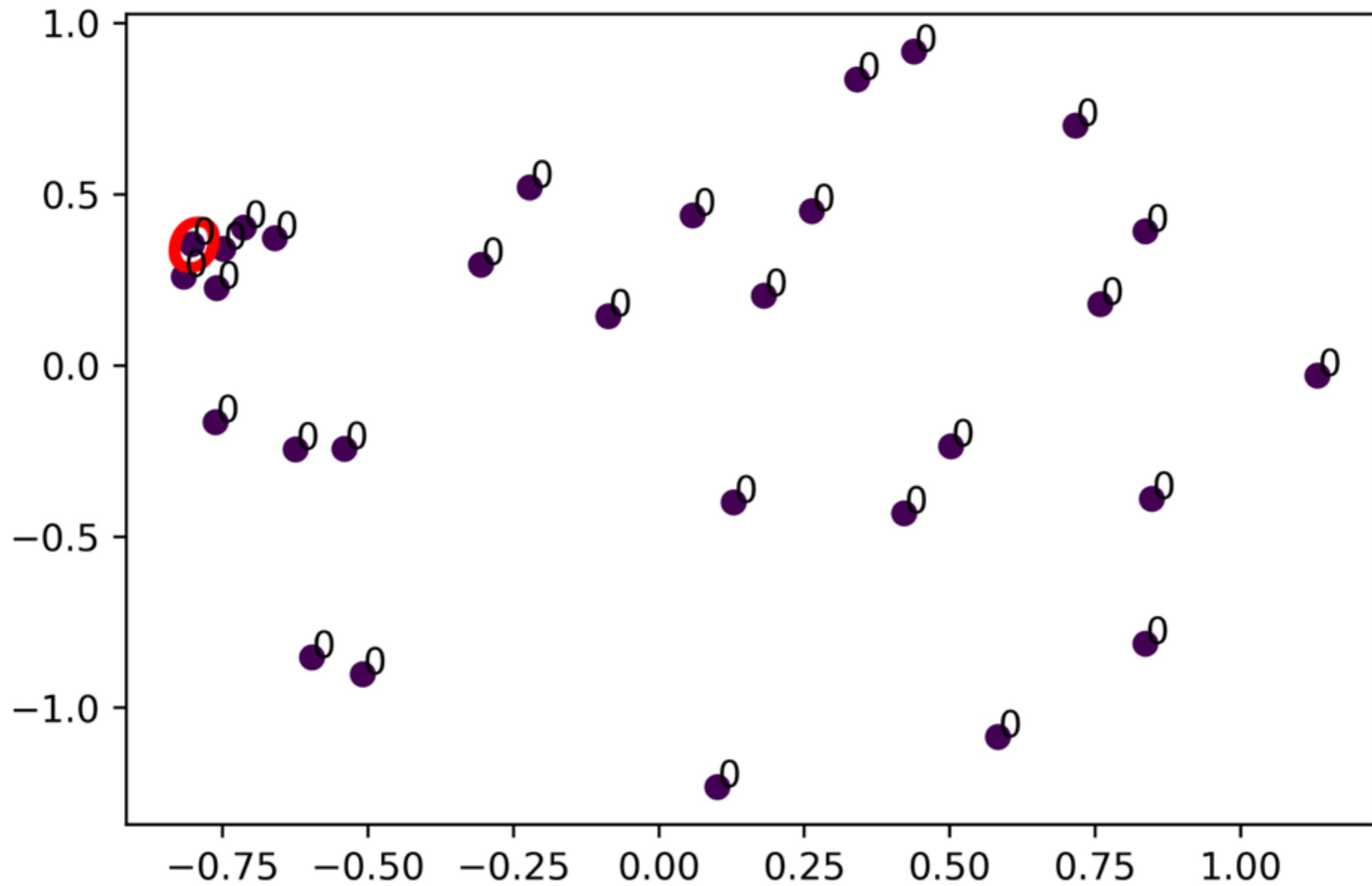
```
def divide_date(data, n, initial):
    choosed_data = maxinSelector(data, n, initial)
    reseted_data = data.drop([[choosed_data]], axis =1)
    return choosed_data, reseted_data
```

### Slicing Part

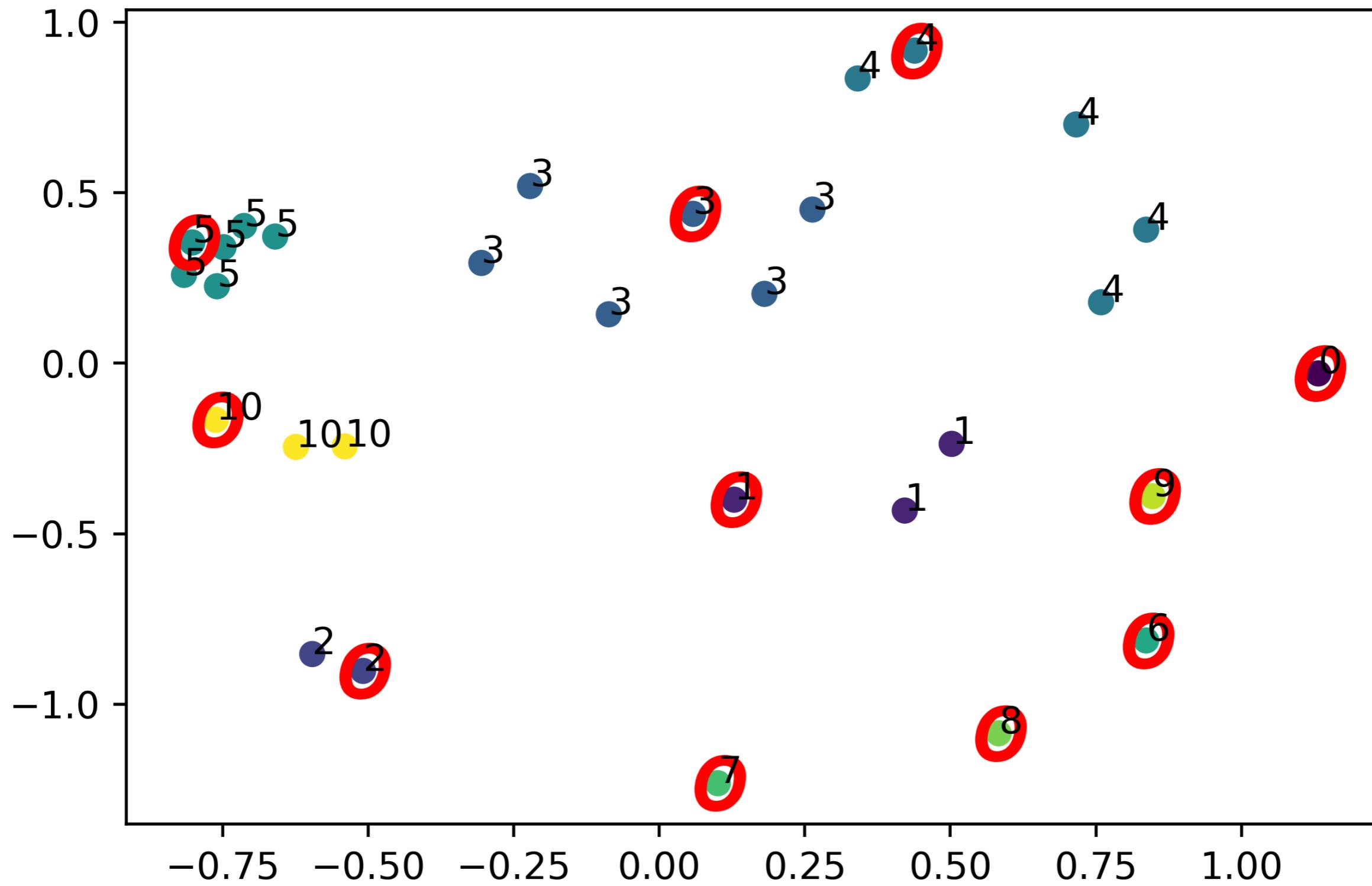








# Clustering Selector



**INITIAL  
FEATURE!**

# The Very First Feature

# Orthodox Method

A 4x4 heatmap showing correlation values between different features. The features are listed on the y-axis: 'concavity\_se', 'smoothness\_se', 'texture\_se', and 'clump\_se'. The x-axis represents the same features. The values range from 0.01 to 1.00. The heatmap is overlaid with large red text: 'NO' in the top-left, 'JAM' in the bottom-right, and 'NO' in the bottom-right.

	concavity_se	smoothness_se	texture_se	clump_se
concavity_se	0.01	0.40	0.19	0.41
smoothness_se	0.40	0.01	0.11	0.08
texture_se	0.19	0.11	0.40	0.11
clump_se	0.41	0.08	0.11	0.40



# Independency

Chi square

Specs	Score
concave points_mean	52.405743
concave points_worst	46.341648
concavity_mean	46.186395
area_worst	35.043882
perimeter_worst	34.438091
radius_worst	34.124937
concavity_worst	31.563031
area_mean	29.328594
perimeter_mean	26.528902
radius_mean	24.897293
compactness_worst	20.992541
compactness_mean	20.353176
area_se	19.676975
radius_se	17.324128
perimeter_se	16.044344
texture_worst	8.741628
texture_mean	6.394071
concave points_se	5.781996
smoothness_worst	5.675733
symmetry_worst	5.560093
compactness_se	5.056762

Laplacian

Feature	Score
symmetry_se	0.389442
smoothness_se	0.352192
texture_se	0.294414
symmetry_worst	0.280945
fractal_dimension_se	0.275966
symmetry_mean	0.261482
concavity_se	0.230877
concave points_se	0.227426
radius_se	0.209268
perimeter_se	0.204753
smoothness_mean	0.200331
texture_mean	0.200033
compactness_se	0.198070
fractal_dimension_worst	0.188476
fractal_dimension_mean	0.183858
smoothness_worst	0.181468
area_se	0.170987
texture_worst	0.168997
compactness_worst	0.111559
concavity_worst	0.108950
compactness_mean	0.093500
concavity_mean	0.067730
radius_mean	0.064922
concave points_worst	0.063395

~Related ~

Feature	Score
concave points_worst	1.700856
perimeter_worst	1.583676
concave points_mean	1.519711
radius_worst	1.518134
perimeter_mean	1.229692
area_worst	1.166843
radius_mean	1.141060
area_mean	1.010689
concavity_mean	0.941434
concavity_worst	0.770180
compactness_mean	0.552439
compactness_worst	0.536757
radius_se	0.474145
perimeter_se	0.447791
area_se	0.429721
texture_worst	0.263839
smoothness_worst	0.216002
symmetry_worst	0.209630
texture_mean	0.208282
concave points_se	0.199758
smoothness_mean	0.147533
symmetry_mean	0.122623
fractal_dimension_worst	0.117185
compactness_se	0.093911
concavity_se	0.068809
fractal_dimension_se	0.006117
smoothness_se	0.004511

Fisher

# Ranking System

$$X^2 + \frac{1}{2}(Laplacian + Fisher)$$

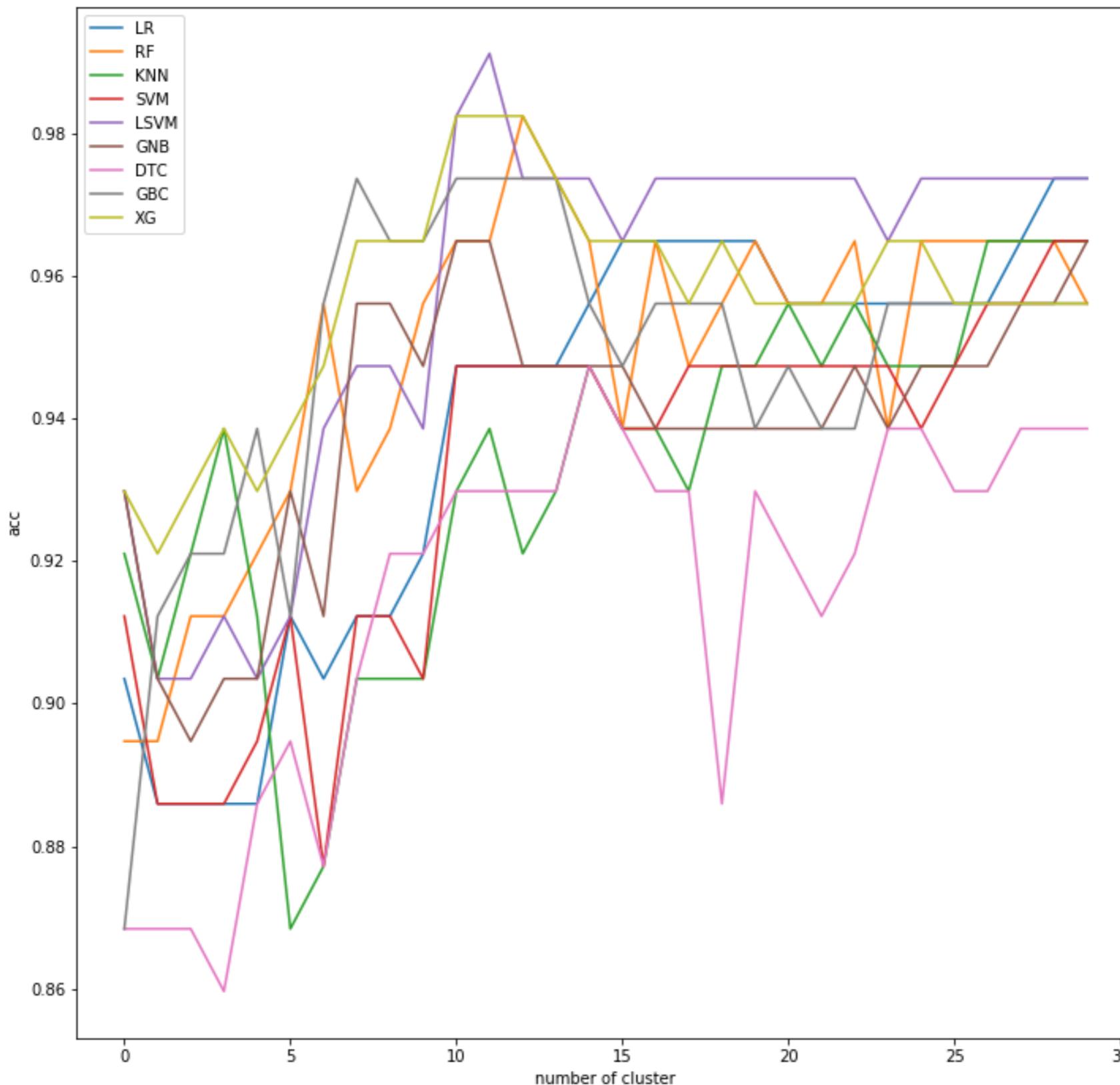
2

*Best = Area Worst*

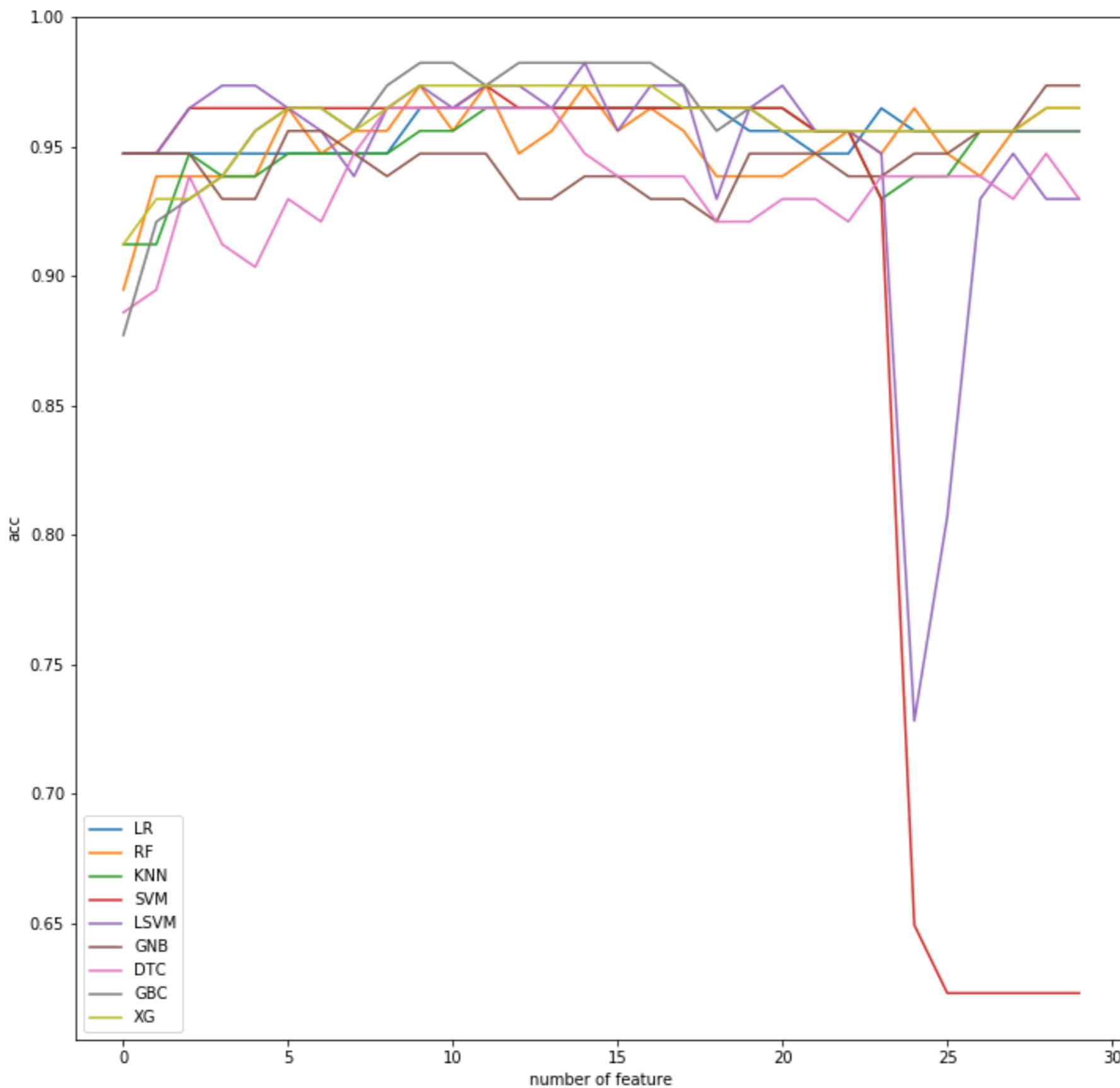
*Worst = Smoothness Standard Error*

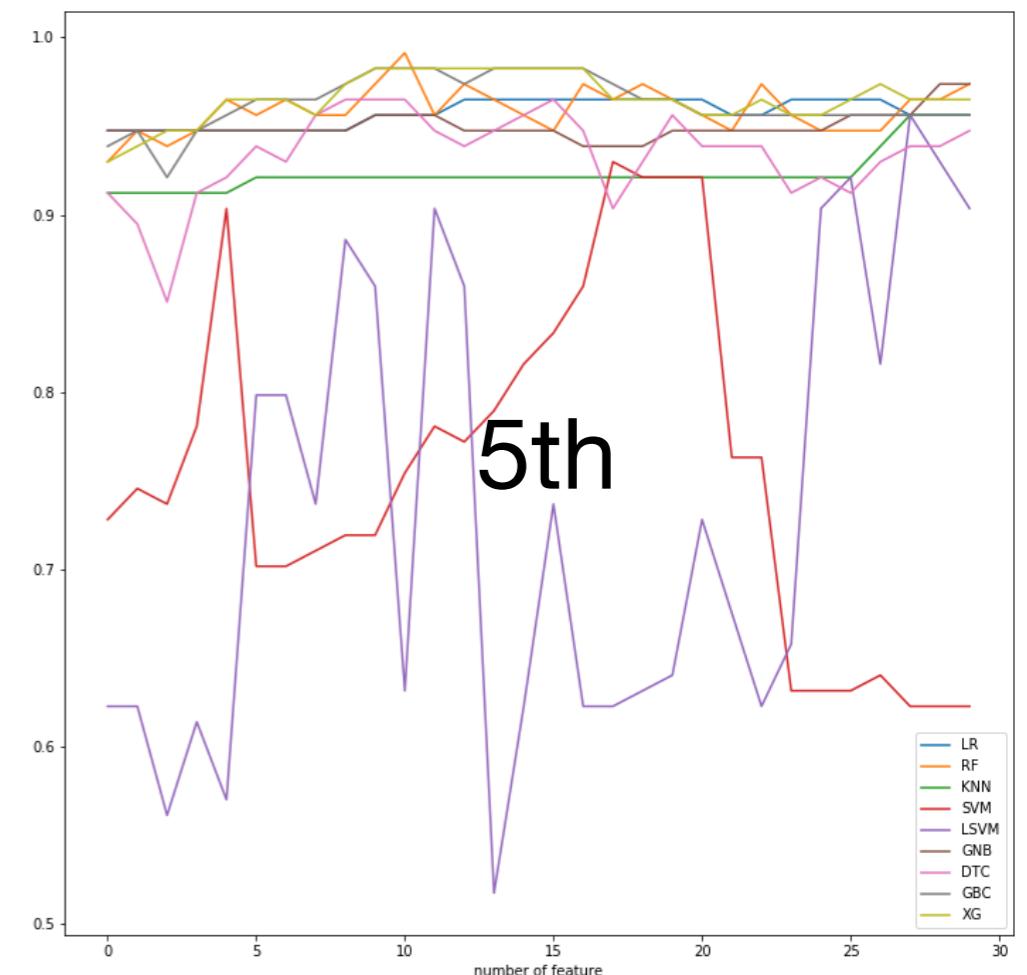
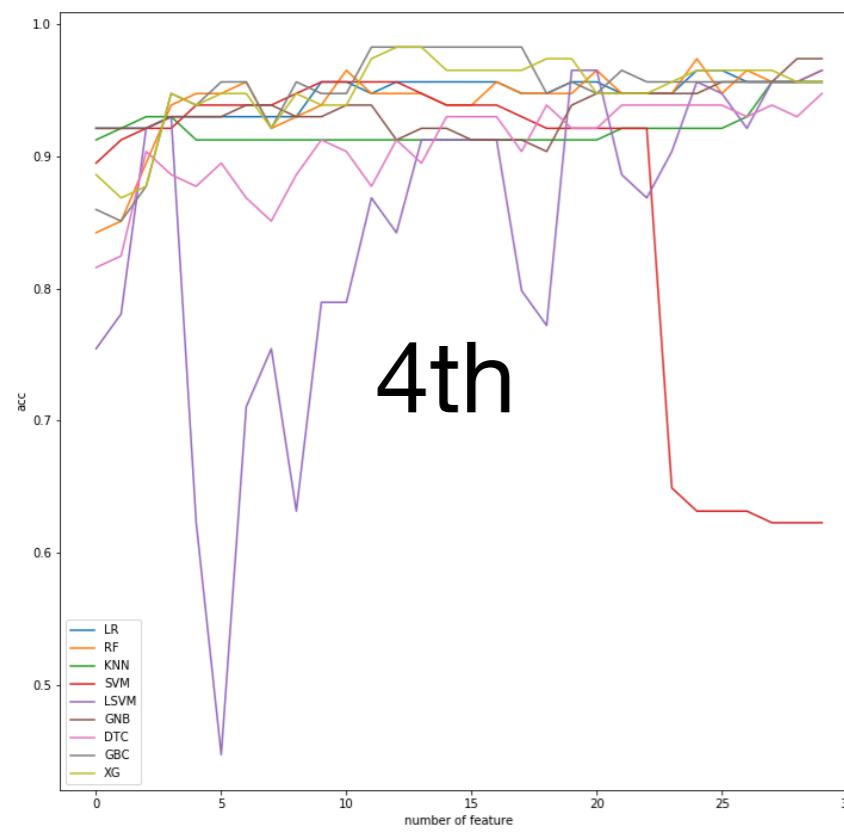
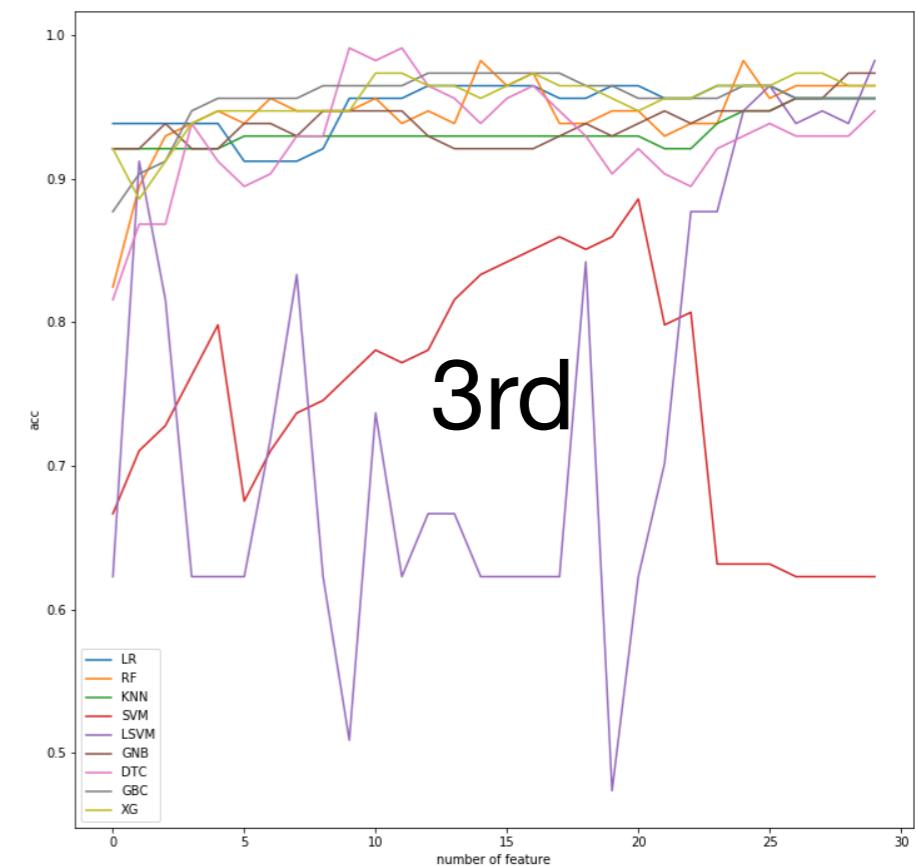
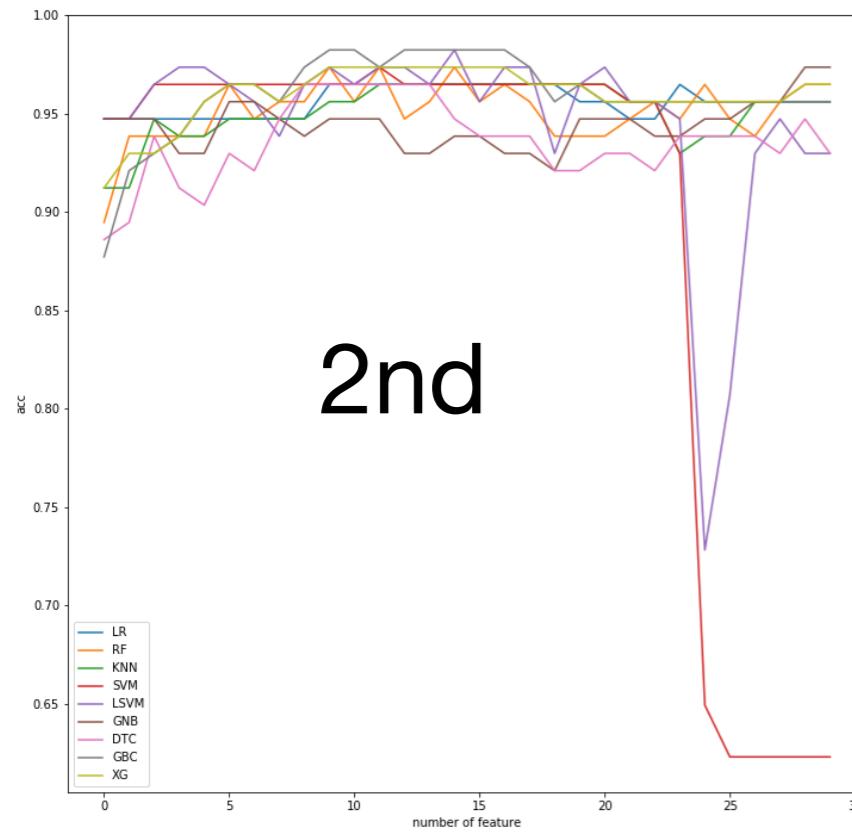
	Feature	ranking
23	area_worst	2.689655
20	radius_worst	2.655172
3	area_mean	2.551724
2	perimeter_mean	2.517241
0	radius_mean	2.310345
7	concave_points_mean	2.310345
6	concavity_mean	2.000000
13	area_se	1.965517
26	concavity_worst	1.965517
22	perimeter_worst	1.827586
25	compactness_worst	1.758621
5	compactness_mean	1.758621
21	texture_worst	1.758621
12	perimeter_se	1.586207
10	radius_se	1.448276
1	texture_mean	1.413793
27	concave_points_worst	1.275862
24	smoothness_worst	1.241379
15	compactness_se	0.931034
28	symmetry_worst	0.896552
29	fractal_dimension_worst	0.862069
17	concave_points_se	0.827586
4	smoothness_mean	0.793103
16	concavity_se	0.724138
8	symmetry_mean	0.655172
9	fractal_dimension_mean	0.551724
19	fractal_dimension_se	0.344828
11	texture_se	0.206897
14	smoothness_se	0.172414

# Accuracy by Cluster Method

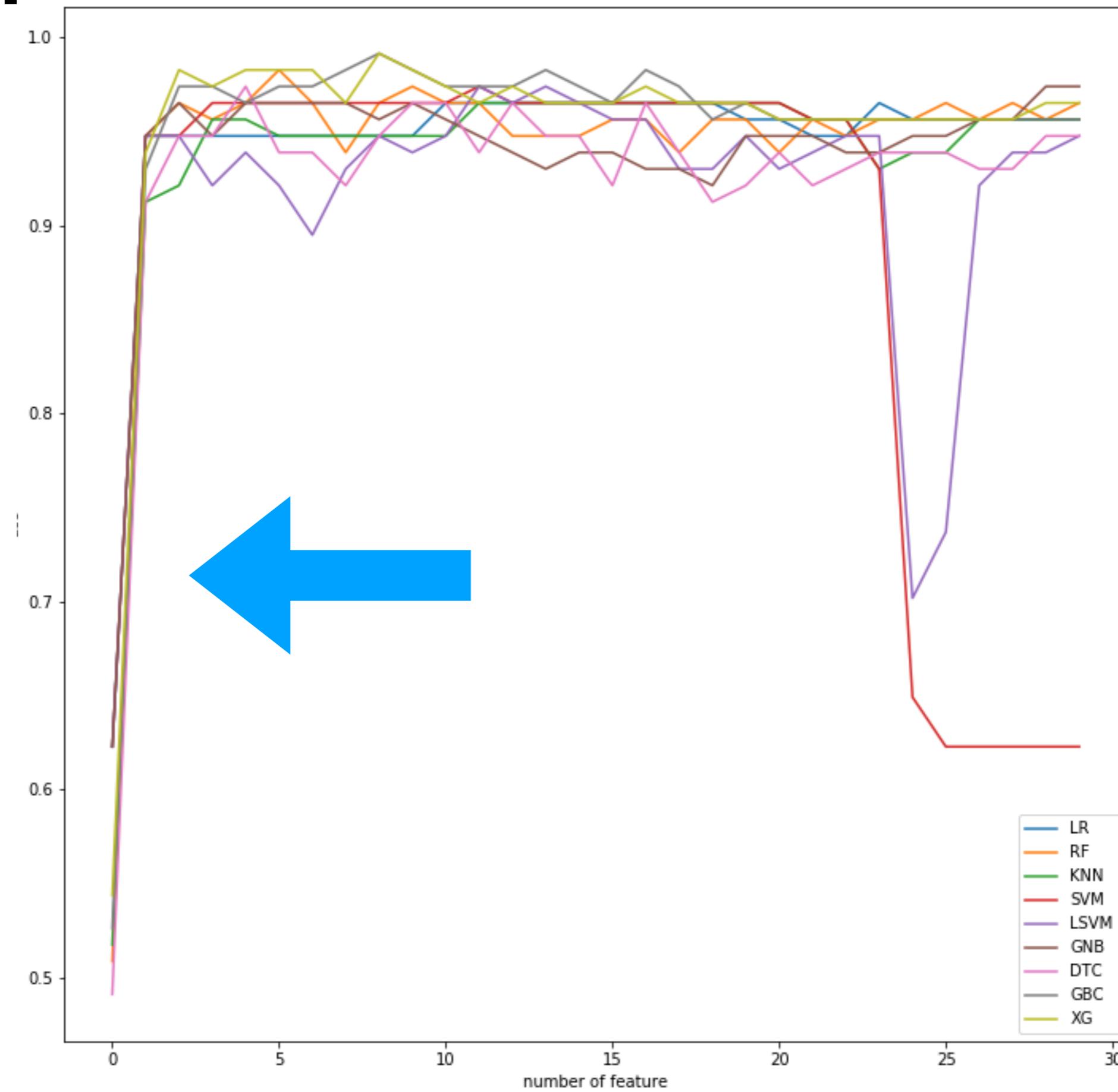


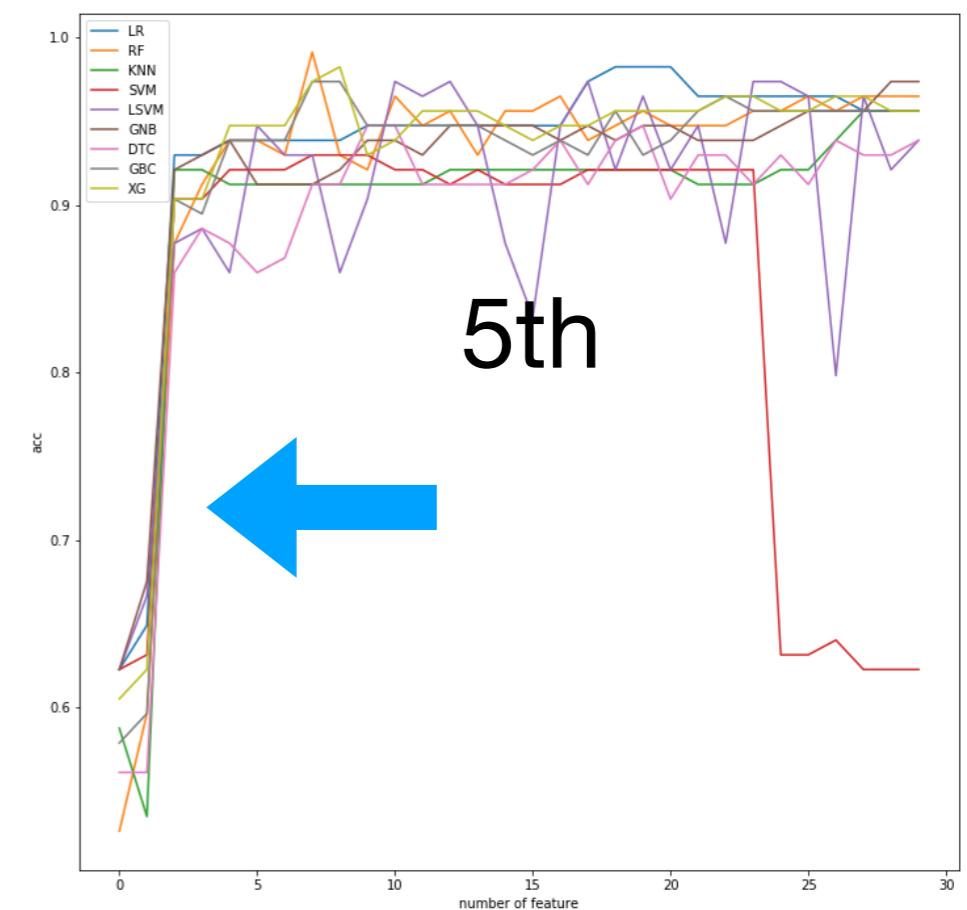
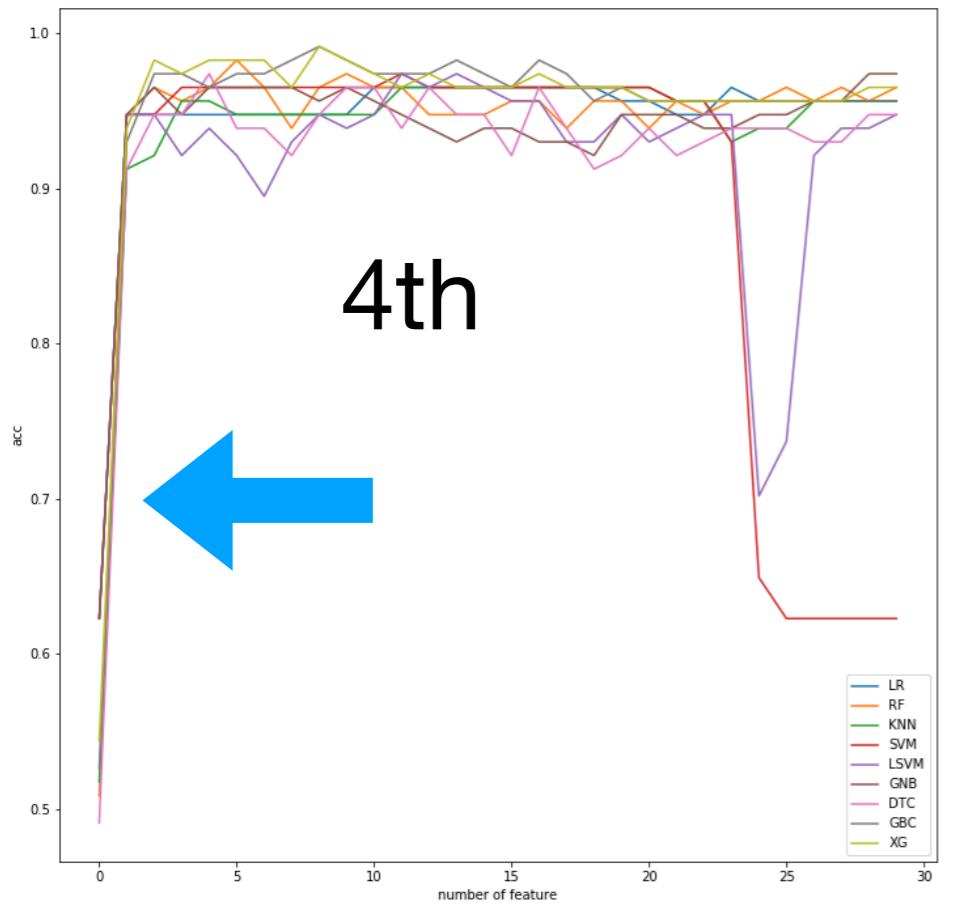
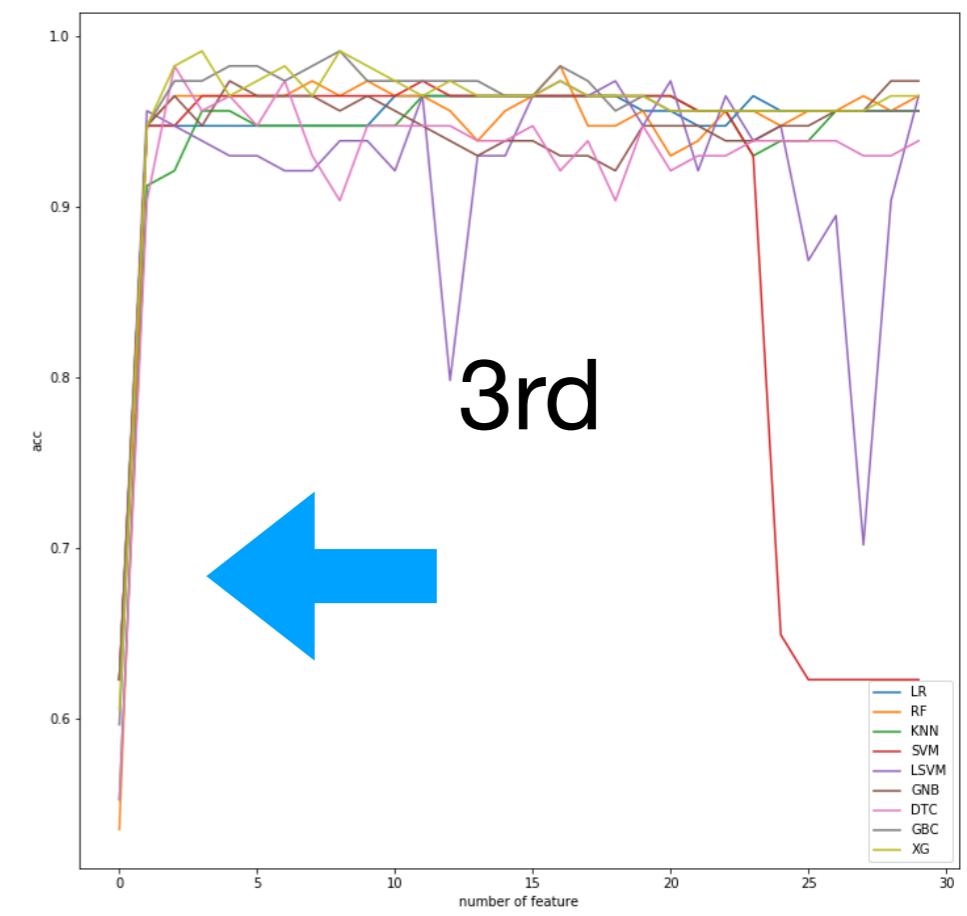
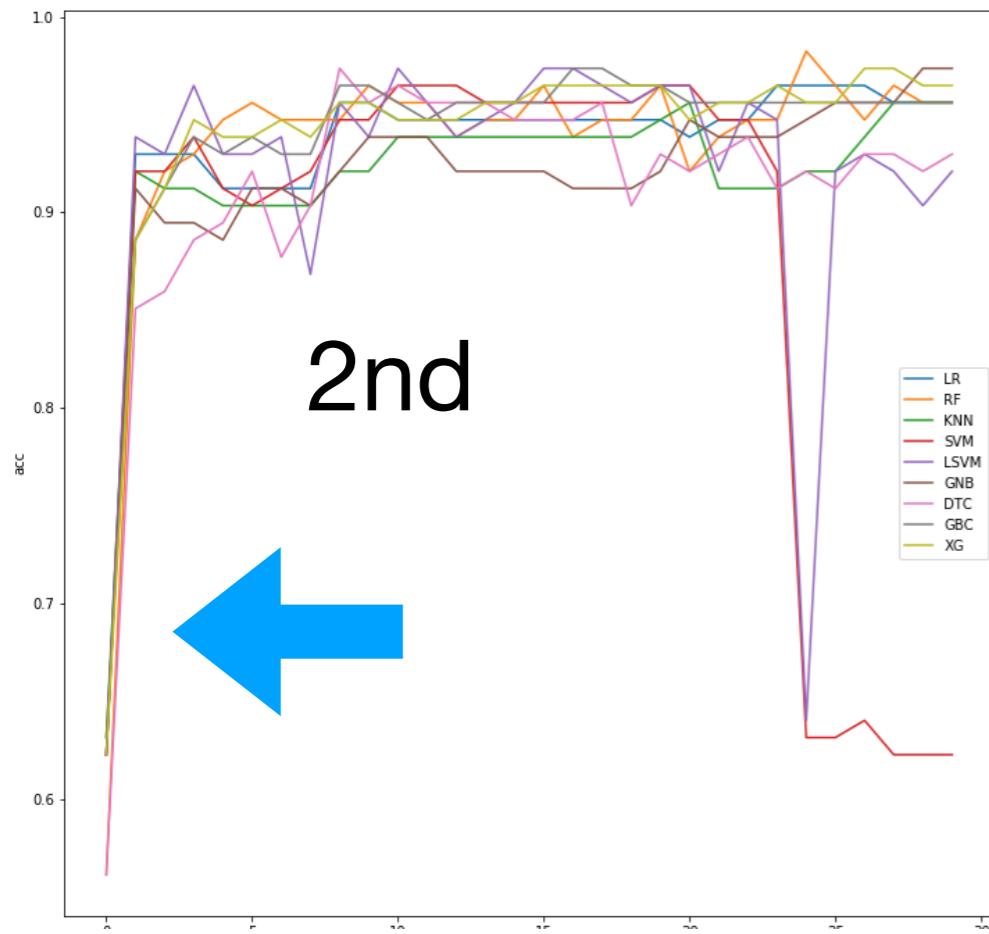
# Compare Models with best



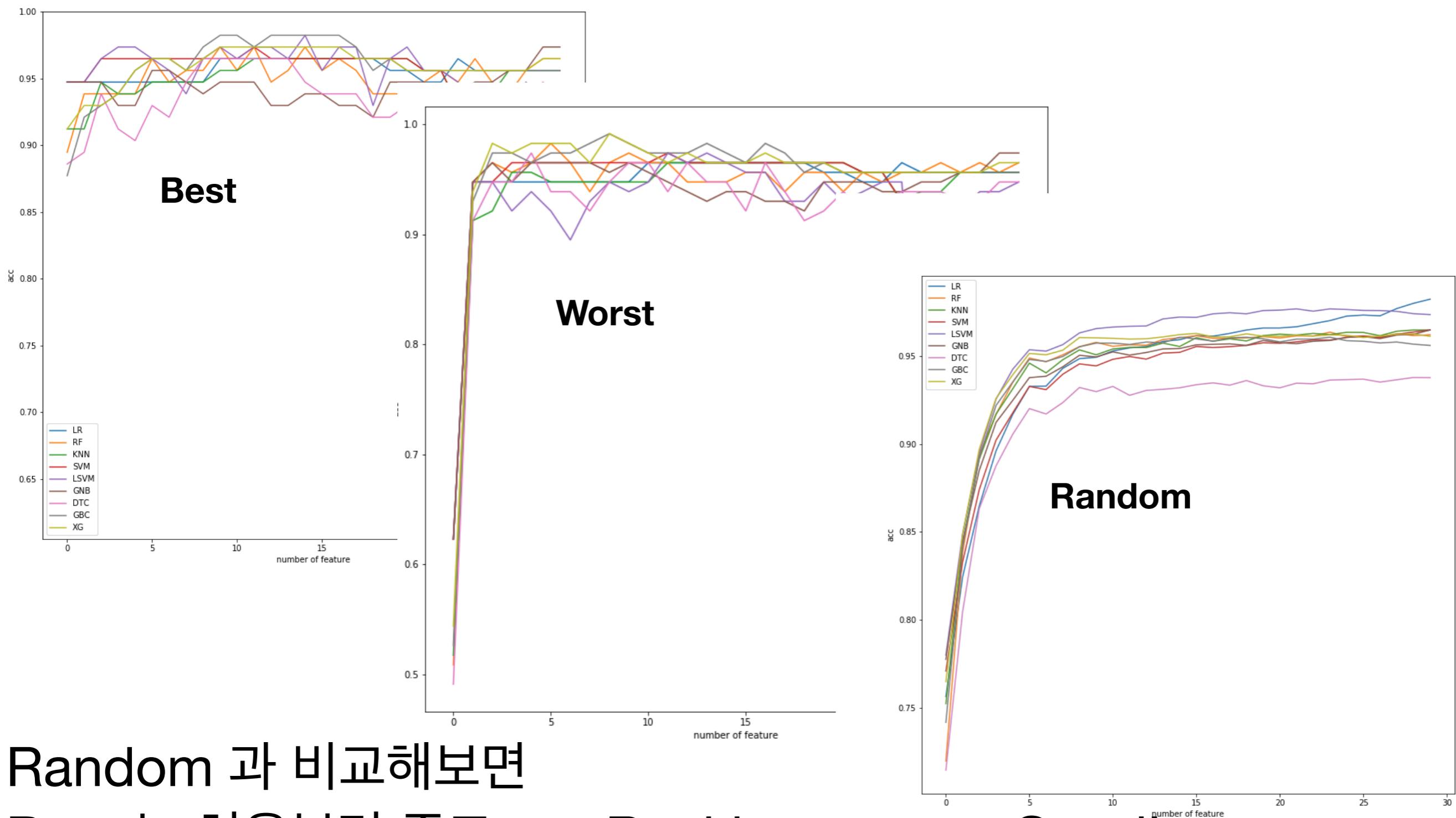


# Compare Models with worst





# Compare Models (Random selection)



Random 과 비교해보면

Best 는 처음부터 좋고 => Ranking system Good!

Worst는 두번째부터 좋아짐 => Feature selection Good!

# Future Work

```
x_input = x
```

```
w1 = tf.Variable(tf.random_normal([20, 64]))  
b1 = tf.Variable(tf.random_normal([64]))  
y1 = tf.math.sigmoid(tf.linalg.matmul(x, w1)+b1)  
#y1 = tf.nn.relu(tf.linalg.matmul(x, w1)+b1)
```

```
w2 = tf.Variable(tf.random_normal([64, 128]))  
b2 = tf.Variable(tf.random_normal([128]))  
y2 = tf.math.sigmoid(tf.linalg.matmul(y1, w2)+b2)  
#y2 = tf.nn.relu(tf.linalg.matmul(y1, w2)+b2)
```

```
w3 = tf.Variable(tf.random_normal([128, 128]))  
b3 = tf.Variable(tf.random_normal([128]))  
y3 = tf.math.sigmoid(tf.linalg.matmul(y2, w3)+b3)  
#y3 = tf.nn.relu(tf.linalg.matmul(y2, w3)+b3)
```

```
w4 = tf.Variable(tf.random_normal([128, 128]))  
b4 = tf.Variable(tf.random_normal([128]))  
y4 = tf.math.sigmoid(tf.linalg.matmul(y3, w4)+b4)  
#y4 = tf.nn.relu(tf.linalg.matmul(y3, w4)+b4)
```

```
w5 = tf.Variable(tf.random_normal([128, 64]))  
b5 = tf.Variable(tf.random_normal([64]))  
y5 = tf.math.sigmoid(tf.linalg.matmul(y4, w5)+b5)  
#y5 = tf.nn.relu(tf.linalg.matmul(y3, w4)+b4)
```

```
w6 = tf.Variable(tf.random_normal([64, 20]))  
b6 = tf.Variable(tf.random_normal([20]))  
y6 = tf.math.sigmoid(tf.linalg.matmul(y5, w6)+b6)  
#y6 = tf.nn.relu(tf.linalg.matmul(y5, w6)+b6)
```

```
w7 = tf.Variable(tf.random_normal([20, 1]))  
b7 = tf.Variable(tf.random_normal([1]))  
y7 = tf.math.sigmoid(tf.linalg.matmul(y6, w7)+b7)  
#y7 = tf.nn.relu(tf.linalg.matmul(y6, w7)+b7)
```

```
y_pred = y7
```

# Waste Processing

**Simple Int with 7 layers**

**Loss int = MSE**

**Optimizer = Adam**

**Train : Validation : Test = 8 : 2 : 2**

어제는 3개의 layers에서 0.625625 정확도

**Test Accuracy**

=0.690265

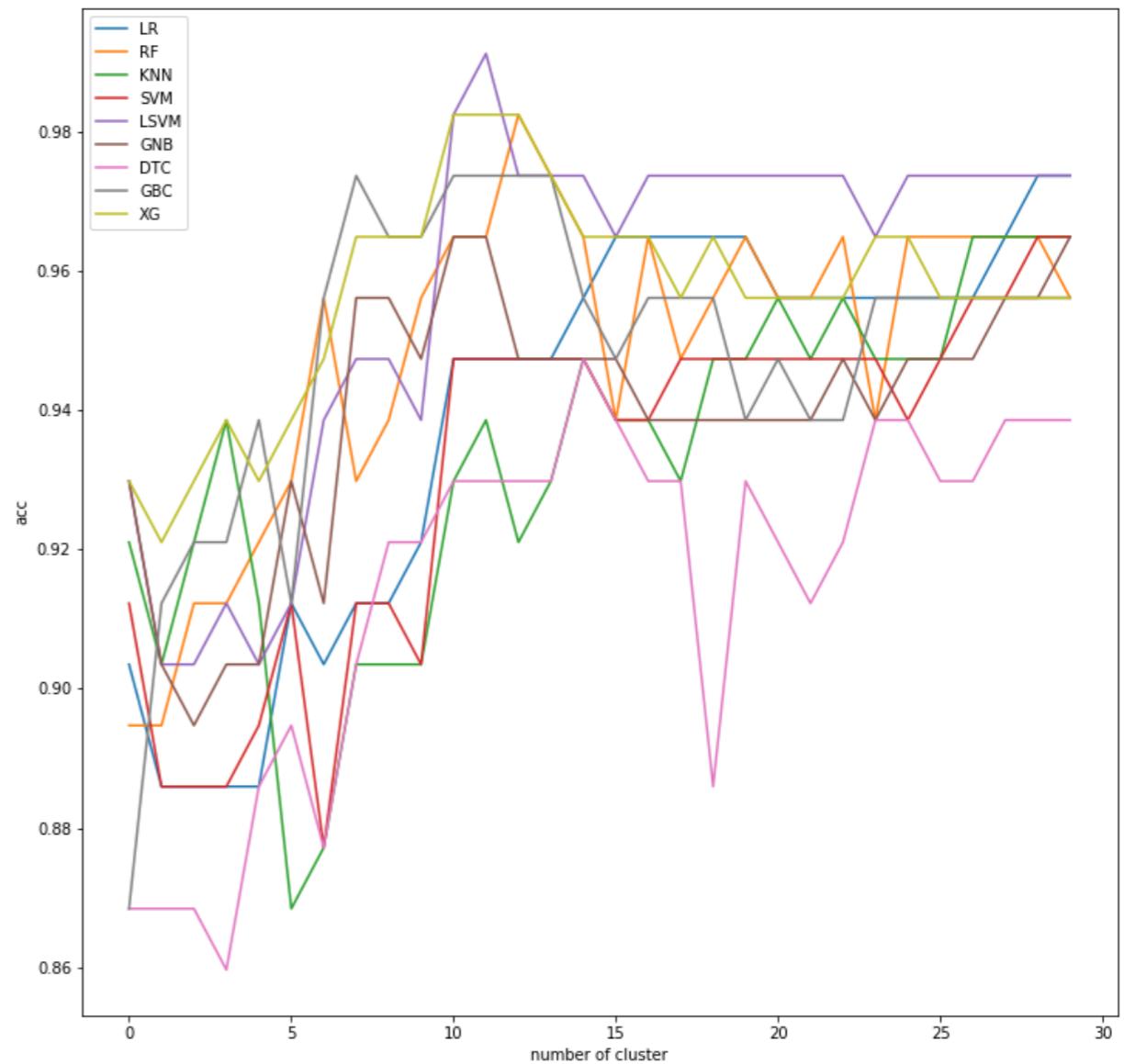
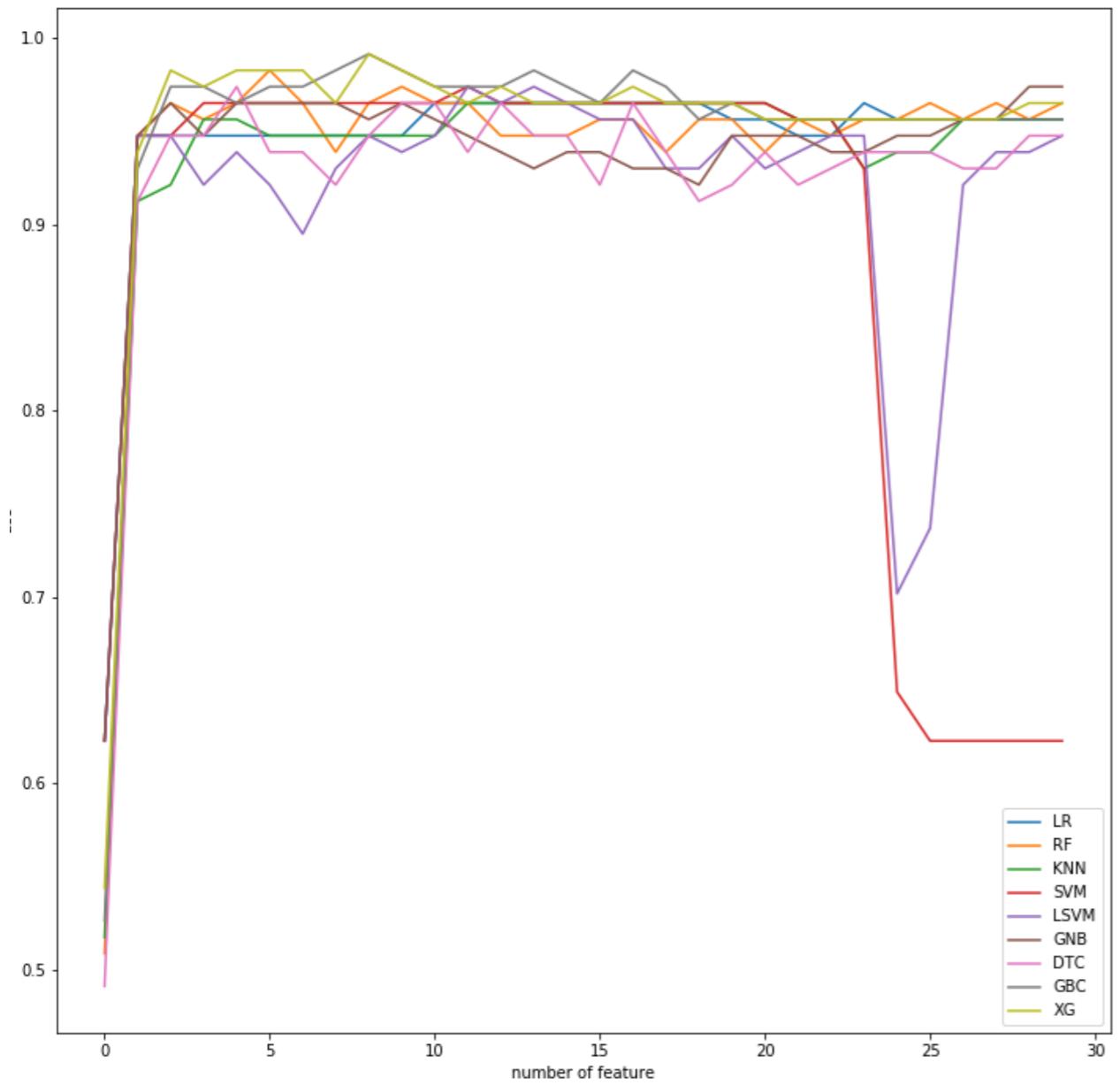
# Git & Kaggle

# Summary

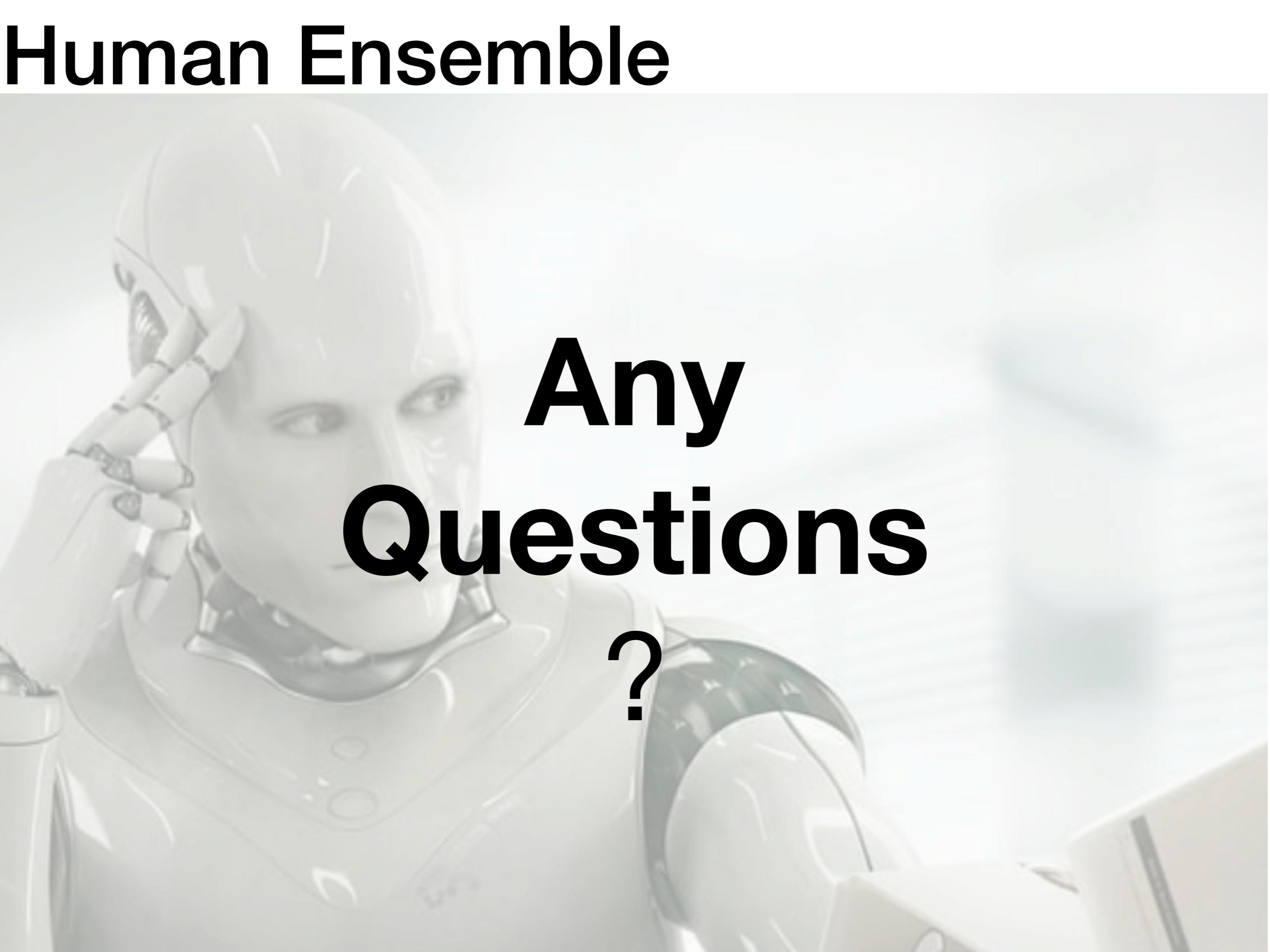
## Optimized Feature

Dominant Feature  
-Ranking System

Feature Selection Method  
-MMS, Cluster



# Human Ensemble



**Any  
Questions  
?**

# Human Ensemble

Thank  
You

!

