박테리아 종 예측하기

Contents

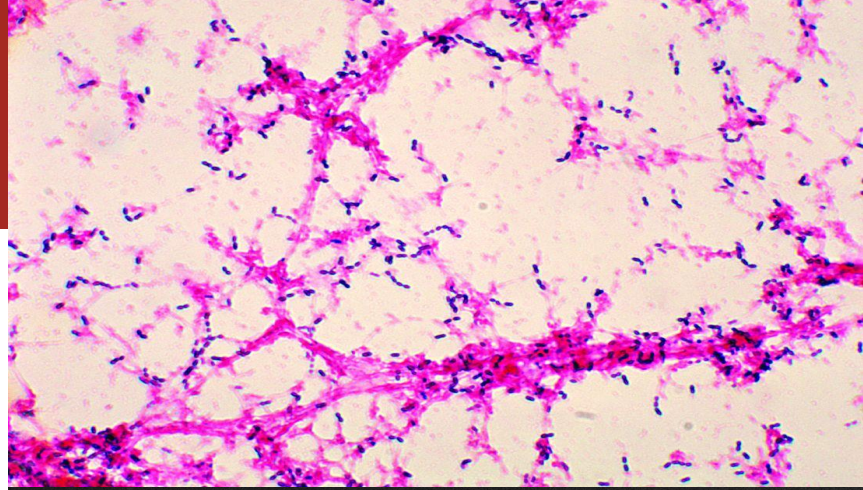
1. 소개 및 변수

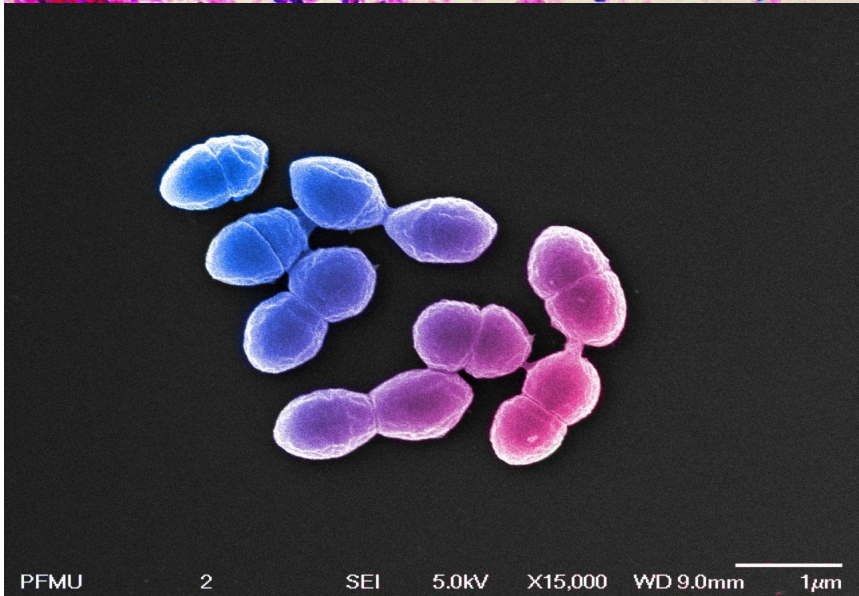
게놈 분석 기술 데이터를 이용해 10가지의 다른 박테리아종을 분류 하고 특정 값이 불연속적인 이유와 이 정보로 할 수 있는 작업은 무엇이 있는지 확인 해 봅니다.

오류율 결정 방법과
테스트 데이터가 훈련 데이터와 어떻게 다른지 결과를 봅니다.

변수

row_id - target
200000 - Streptococcus_pneumoniae
200001 - Enterococcus_hirae
etc.





2. 코드 (모듈)

Numpy, Pandas, Matplotlib
Scikit-learn

In [30]:

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4
5 from math import factorial
6
7 from sklearn.decomposition import PCA
8 from sklearn.preprocessing import StandardScaler, LabelEncoder
9 from sklearn.cluster import KMeans
```

200000 - Streptococcus_pneumoniae
200001 - Enterococcus_hirae

데이터 삽입

In [31]:
```
1 train_df = pd.read_csv("./data/train.csv")
2 train_df.shape
```

Out[31]: (200000, 288)

In [32]:
```
1 test_df = pd.read_csv("./data/test.csv")
2 test_df.shape
```

Out[32]: (100000, 287)

박테리아 이름을 숫자로 변경

In [33]:

```
elements = [e for e in train_df.columns if e != 'row_id' and e != 'target']

# Convert the 10 bacteria names to the integers 0 .. 9
le = LabelEncoder()
train_df['target_num'] = le.fit_transform(train_df.target)

train_df.shape, test_df.shape
```

Out[33]: ((200000, 289), (100000, 287))

```
In [34]:  1  np.unique(train_df.A0T0G2C8)
```

```
Out[34]: array([-4.29153442e-05, -4.19153442e-05, -4.09153442e-05, -3.99153442e-05,
                -3.89153442e-05, -3.79153442e-05, -3.69153442e-05, -3.59153442e-05,
                -3.49153442e-05, -3.39153442e-05, -3.29153442e-05, -3.19153442e-05,
                -3.09153442e-05, -2.99153442e-05, -2.89153442e-05, -2.79153442e-05,
                -2.69153442e-05, -2.59153442e-05, -2.49153442e-05, -2.39153442e-05,
                -2.29153442e-05, -2.19153442e-05, -2.09153442e-05, -1.99153442e-05,
                -1.89153442e-05, -1.79153442e-05, -1.69153442e-05, -1.59153442e-05,
                -1.49153442e-05, -1.39153442e-05, -1.29153442e-05, -1.19153442e-05,
                -1.09153442e-05, -9.91534424e-06, -8.91534424e-06, -7.91534424e-06,
                -6.91534424e-06, -5.91534424e-06, -4.91534424e-06, -3.91534424e-06,
                -2.91534424e-06, -1.91534424e-06, -9.15344238e-07,  8.46557617e-08,
                 1.08465576e-06,  2.08465576e-06,  3.08465576e-06,  4.08465576e-06,
                 5.08465576e-06,  6.08465576e-06,  7.08465576e-06,  8.08465576e-06,
                 9.08465576e-06,  1.00846558e-05,  1.10846558e-05,  1.20846558e-05,
                 1.30846558e-05,  1.40846558e-05,  1.50846558e-05,  1.60846558e-05,
                 1.70846558e-05,  1.80846558e-05,  1.90846558e-05,  2.00846558e-05,
                 2.10846558e-05,  2.20846558e-05,  2.30846558e-05,  2.40846558e-05,
                 2.50846558e-05,  2.60846558e-05,  2.70846558e-05,  2.80846558e-05,
                 2.90846558e-05,  3.00846558e-05,  3.10846558e-05,  3.20846558e-05,
                 3.30846558e-05,  3.40846558e-05,  3.50846558e-05,  3.60846558e-05,
                 3.70846558e-05,  3.80846558e-05,  3.90846558e-05,  4.00846558e-05,
                 4.10846558e-05,  4.20846558e-05,  4.30846558e-05,  4.40846558e-05,
                 4.50846558e-05,  4.60846558e-05,  4.70846558e-05,  4.80846558e-05,
                 4.90846558e-05,  5.00846558e-05,  5.10846558e-05,  5.20846558e-05,
                 5.30846558e-05,  5.40846558e-05,  5.50846558e-05,  5.60846558e-05,
                 5.70846558e-05,  5.80846558e-05,  5.90846558e-05,  6.00846558e-05,
                 6.10846558e-05,  6.20846558e-05,  6.30846558e-05,  6.40846558e-05,
                 6.50846558e-05,  6.60846558e-05,  6.70846558e-05,  6.80846558e-05,
                 6.90846558e-05,  7.00846558e-05,  7.10846558e-05,  7.20846558e-05,
                 7.30846558e-05,  7.40846558e-05,  7.50846558e-05,  7.60846558e-05,
                 7.70846558e-05,  7.80846558e-05,  7.90846558e-05,  8.00846558e-05,
                 8.20846558e-05,  8.40846558e-05,  8.70846558e-05,  9.70846558e-05,
                 1.07084656e-04,  1.17084656e-04,  1.27084656e-04,  1.37084656e-04,
                 1.47084656e-04,  1.57084656e-04,  1.67084656e-04,  1.77084656e-04,
                 1.97084656e-04,  9.57084656e-04,  9.95708466e-03])
```

중복된 요소 찾기

값의 불연속성이 확인이 됩니다.

값을 보니

부동 소수점 숫자이지만 고유 값이 200000개가 아니라 약 1000개 정도입니다.
마지막 몇 자리는 항상 동일하다는걸 알 수 있습니다.

(1.00846558e-05부터 9.70846558e-05까지 항상 0846558로 끝남)

부동 소수점 숫자 정수로 변경

In [35]:

```
def bias(w, x, y, z) :
    return factorial(10) / (factorial(w) * factorial(x) * factorial(y) * factorial(z) * 4 **10)
def bias_of(s) :
    w = int(s[1:s.index("T")])
    x = int(s[s.index("T")+1:s.index("G")])
    y = int(s[s.index("G")+1:s.index("C")])
    z = int(s[s.index("C")+1:])
    return factorial(10) / (factorial(w) * factorial(x) * factorial(y) * factorial(z) * 4 **10)

train_i = pd.DataFrame({col: ((train_df[col] + bias_of(col)) * 1000000).round().astype(int)
                        for col in elements})
test_i = pd.DataFrame({col: ((test_df[col] + bias_of(col)) * 1000000).round().astype(int)
                       for col in elements})
```

In [36]:

```
train_i.sample(10)
```

Out[36]:

	A0T0G0C10	A0T0G1C9	A0T0G2C8	A0T0G3C7	A0T0G4C6	A0T0G5C5	A0T0G6C4	A0T0G7C3	A0T0G8C2	A0T0G9C1	...	A8T0G0C2	A8T0G1C1	A8T
134593	0	0	0	0	0	0	0	0	0	0	...	0	0	
154979	0	0	7	19	33	41	33	13	3	0	...	247	517	
7815	1	4	31	121	231	251	222	155	31	2	...	114	222	
178351	2	7	103	611	1345	1718	1279	543	76	7	...	71	157	
149758	0	0	10	20	0	30	10	10	0	0	...	450	1240	
133151	0	0	0	0	0	0	0	0	0	0	...	0	0	
122368	0	8	114	655	1473	1913	1465	626	87	10	...	90	155	
153561	0	0	0	0	0	0	0	0	0	0	...	0	1000	
52721	0	0	0	0	0	0	10	0	0	0	...	330	890	
82559	0	20	60	110	170	270	160	140	0	0	...	60	190	

10 rows × 286 columns

1, 10, 1000, 100000
최대 공약수 활용

In [37]:
```
train_i.sum(axis=1).min(), train_i.sum(axis=1).max()
```

Out[37]:
```
(1000000, 1000000)
```

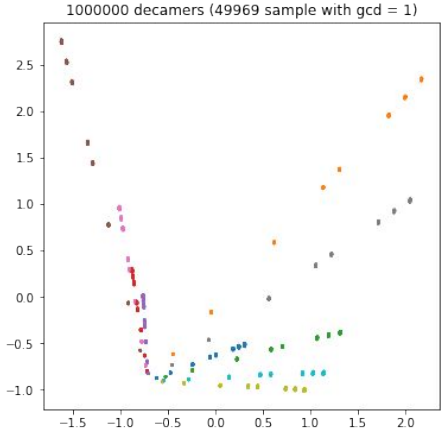
In [38]:
```
train_df['gcd'] = np.gcd.reduce(train_i[elements], axis=1)
test_df['gcd'] = np.gcd.reduce(test_i[elements], axis=1)

np.unique(train_df['gcd'], return_counts=True)
np.unique(test_df['gcd'], return_counts=True)
```

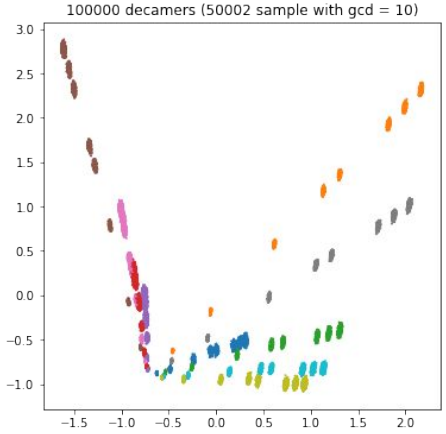
Out[38]:
```
(array([    1,    10,  1000, 10000]),
 array([25208, 24951, 24930, 24911], dtype=int64))
```

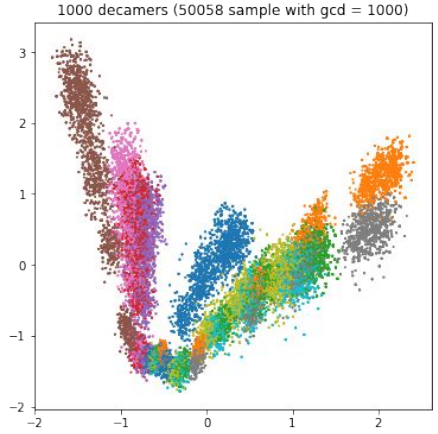
1000000, 100000, 1000, 100 데카머를 넣고 머신러닝을합니다.
A0T0G0C10에서 A10T0G0C0까지 286가지 유형이 모두 몇 번 발생하는지 계산합니다.
모든 카운트를 행 합으로 나누고 바이어스를 빼서 스펙트럼을 정규화합니다.
100으로 갈 수록 정보가 없고 예측이 어려운걸 볼 수 있습니다.

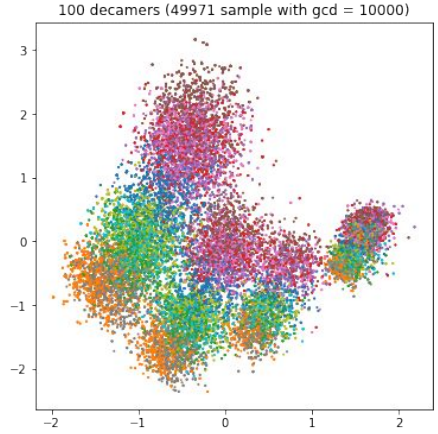
- gcd = 1의 경우 높은정확도 예측
- gcd=10000경우 낮은 정확도가 보입니다.

In [39]:
```
for scale in np.sort(train_df['gcd'].unique()) :
    pca = PCA(whiten=True, random_state=1)
    pca.fit(train_i[elements][train_df['gcd'] == scale])

    Xt_train = pca.transform(train_i[elements][train_df['gcd'] == scale])
    Xt_test = pca.transform(test_i[elements][test_df['gcd'] == scale])

    plt.figure(figsize=(6,6))
    plt.scatter(Xt_train[:,0], Xt_train[:, 1], c=train_df.target_num[train_df['gcd'] == scale], cmap = 'tab10', s=1)
    plt.title(f"{1000000 // scale} decamers ({(train_df['gcd'] == scale).sum()} sample with gcd = {scale})")
    plt.show()
```









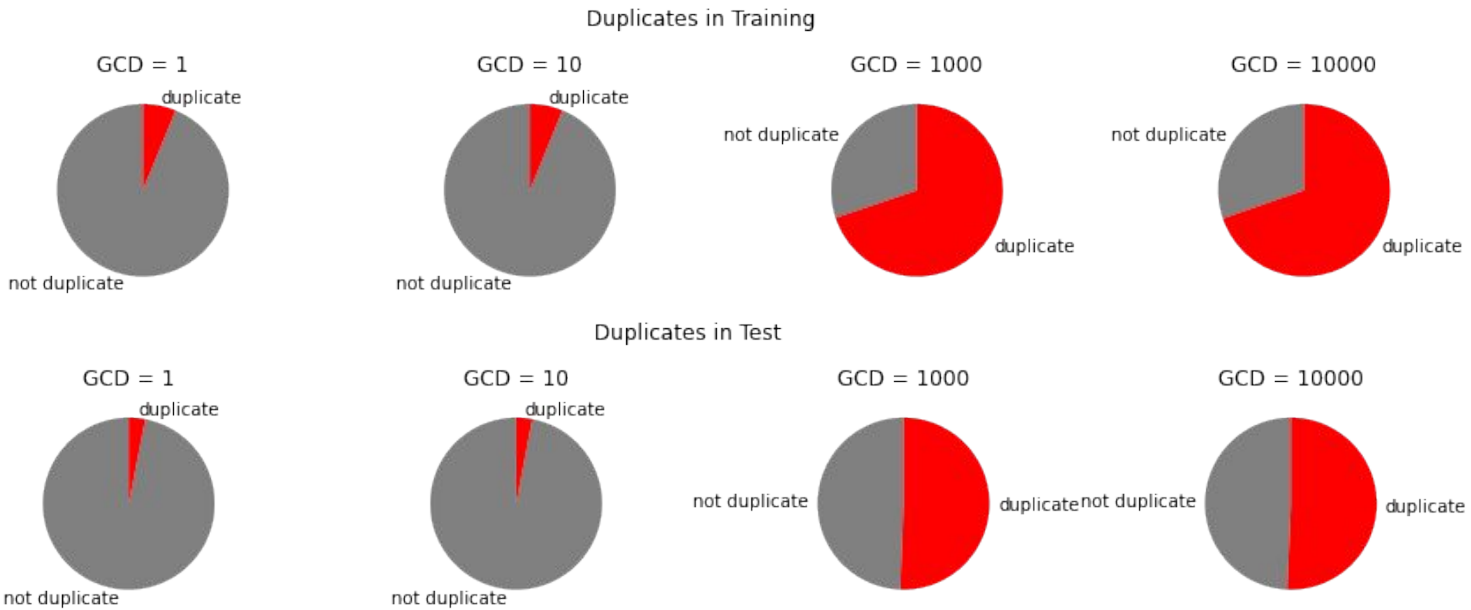
4개의 GCD 값에 대한 중복을 개별적으로 계산하면 대부분의 중복이 높은 GCD 값에 대해 발생한다는 것을 알 수 있습니다.

100개의 데카머가

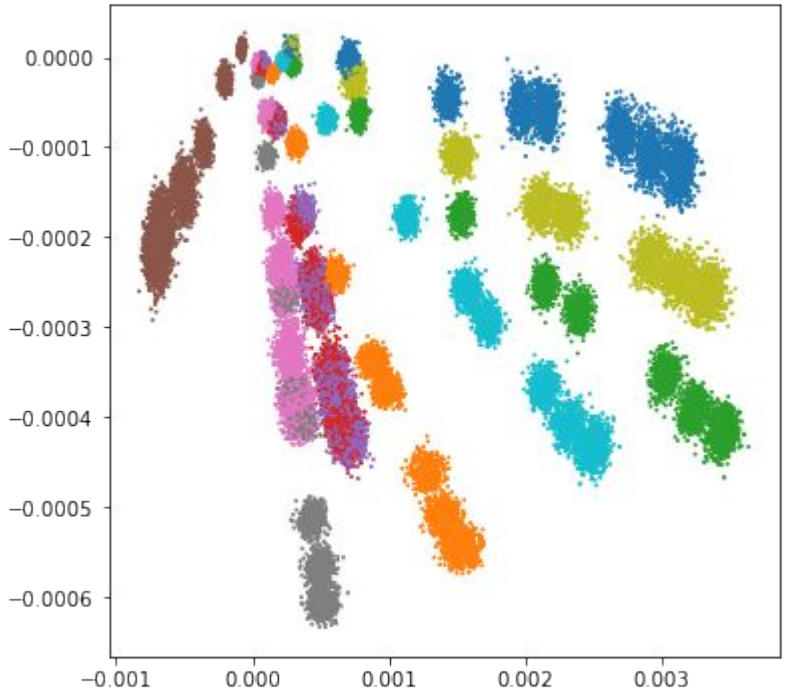
1000000개의 데카머 286개의 빈에 들어가는 것보다 더 많은 중복을 예상합니다.

In [42]:

```
def plot_duplicates_per_gcd(df, title):
    plt.figure(figsize=(14, 3))
    plt.tight_layout()
    for i, gcd in enumerate(np.unique(df.gcd)) :
        plt.subplot(1, 4, i+1)
        duplicates = df[df.gcd == gcd][elements].duplicated().sum()
        non_duplicates = len(df[df.gcd == gcd]) - duplicates
        plt.pie([non_duplicates, duplicates], labels=['not duplicate', 'duplicate'],
                colors = ['gray', 'r'], startangle=90)
        plt.title(f"GCD = {gcd}")
    plt.subplots_adjust(wspace=0.8)
    plt.suptitle(title)
    plt.show()

plot_duplicates_per_gcd(train_df, title="Duplicates in Training")
plot_duplicates_per_gcd(test_df, title="Duplicates in Test")
```



가장 높은 정밀도(1000000 데카머)로 두 개의 임의의 특징을 구성합니다.
모든 클래스의 포인트가 8개의 클러스터로
그룹화되고 8개의 클러스터가 원점에서 교차한다는 것을 알 수 있습니다.



In [43]:

```
plt.figure(figsize=(6,6))
plt.scatter(train_df.iloc[:, 240][train_df['gcd']==1],
            train_df.iloc[:, 181][train_df['gcd'] ==1],
            c = train_df.target_num[train_df['gcd'] == 1],
            cmap = 'tab10', s=1)
plt.show()
```

오류가 많을 수록 클러스터가 원점에 가까워지고 겹치기 시작합니다.
다음 히스토그램은 8개의 스케일링(8개의 오류율에 해당)이 있고 스케일링이 10개의 박테리아 모두에 대해 동일함을 보여줍니다. 히스토그램에서 8개 값을 읽거나 1차원 k-means 클러스터링 알고리즘 으로 결정할 수 있습니다. 축소된 7개의 클러스터는 크기가 550이고 내부에 중복이 없습니다.
확장되지 않은 클러스터의 크기는 1100입니다(이 중 800은 고유하고 300은 중복)

In [44]:

```
v = train_df[elements].abs().sum(axis=1)
chosen_gcd = 1

plt.figure(figsize=(18,14))
plt.tight_layout()
for t in range(10) :
    plt.subplot(5, 2, t+1)
    plt.title(le.inverse_transform([t])[0])

    vt = v[(train_df['gcd'] == chosen_gcd) & ( train_df['target_num'] == t)]
    km = KMeans(n_clusters=8)
    km.fit(vt.values.reshape(-1, 1))
    cluster_max = km.cluster_centers_.max()
    print(le.inverse_transform(np.array([t]))[0])
    print('Cluster centers:', sorted((km.cluster_centers_ / cluster_max).ravel().round(2)))
    print('Cluster sizes:', np.unique(km.predict(vt.values.reshape(-1, 1)), return_counts=True)[1][np.argsort(km.cluster_centers_.
    print('Cluster unique elements:',
          train_df[elements][(train_df['gcd'] == chosen_gcd) & (train_df['target_num'] == t)].
          groupby(km.predict(vt.values.reshape(-1, 1))).apply(lambda df: np.unique(df.values, axis=0).shape[0]).values[np.argsort(k
    print()

    plt.hist(vt / cluster_max, bins=np.linspace(0, (vt / cluster_max).max(), 200), color='m', density=True)
    plt.xticks(ticks=(km.cluster_centers_ / cluster_max).round(2))
    plt.xlabel('scale')
    plt.ylabel('density')
plt.subplots_adjust(hspace=0.5)
plt.show()
```

Bacteroides_fragilis

Cluster centers: [0.1, 0.24, 0.46, 0.64, 0.7, 0.88, 0.94, 1.0]

Cluster sizes: [556 576 559 557 568 528 585 1145]

Cluster unique elements: [556 576 559 557 568 528 585 824]

Enterococcus_hirae

Cluster centers: [0.09, 0.23, 0.46, 0.64, 0.7, 0.88, 0.94, 1.0]

Cluster sizes: [534 542 565 533 556 560 555 1111]

Cluster unique elements: [534 542 565 533 556 560 555 804]

scherichia_fergusonii

Cluster centers: [0.11, 0.24, 0.46, 0.64, 0.7, 0.88, 0.94, 1.0]

Cluster sizes: [547 559 543 542 553 545 566 1127]

Cluster unique elements: [547 559 543 542 553 545 566 796]

Salmonella_enterica

Cluster centers: [0.1, 0.23, 0.46, 0.64, 0.7, 0.88, 0.94, 1.0]

Cluster sizes: [534 574 562 539 558 554 552 1147]

Cluster unique elements: [534 574 562 539 558 554 552 823]

Streptococcus_pneumoniae

Cluster centers: [0.09, 0.23, 0.46, 0.64, 0.7, 0.88, 0.94, 1.0]

Cluster sizes: [548 563 580 585 543 548 568 1098]

Cluster unique elements: [548 563 580 585 543 548 568 792]

mpylobacter_jejuni

Cluster centers: [0.09, 0.23, 0.46, 0.64, 0.7, 0.88, 0.94, 1.0]

Cluster sizes: [550 542 538 578 566 560 565 1083]

Cluster unique elements: [550 542 538 578 566 560 565 789]

Escherichia_coli

Cluster centers: [0.11, 0.24, 0.46, 0.64, 0.7, 0.88, 0.94, 1.0]

Cluster sizes: [552 524 538 545 549 540 553 1095]

Cluster unique elements: [552 524 538 545 549 540 553 796]

Klebsiella_pneumoniae

Cluster centers: [0.09, 0.23, 0.46, 0.64, 0.7, 0.88, 0.94, 1.0]

Cluster sizes: [532 590 548 552 584 561 556 1109]

Cluster unique elements: [532 590 548 552 584 561 556 808]

Staphylococcus_aureus

Cluster centers: [0.09, 0.23, 0.46, 0.64, 0.7, 0.88, 0.94, 1.0]

Cluster sizes: [545 576 549 575 548 556 553 1101]

Cluster unique elements: [545 576 549 575 548 556 553 790]

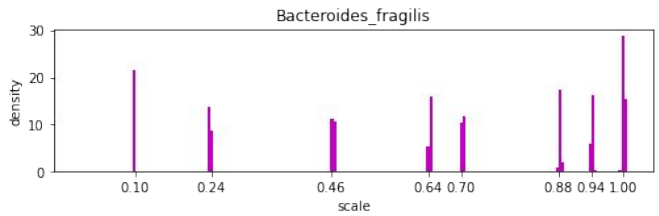
Streptococcus_pyogenes

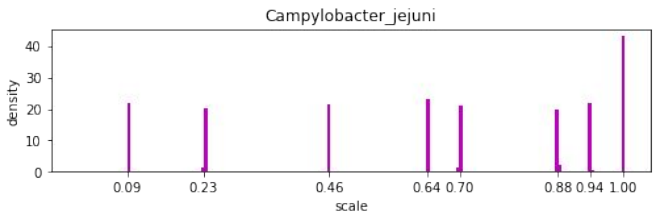
Cluster centers: [0.09, 0.23, 0.46, 0.64, 0.7, 0.88, 0.94, 1.0]

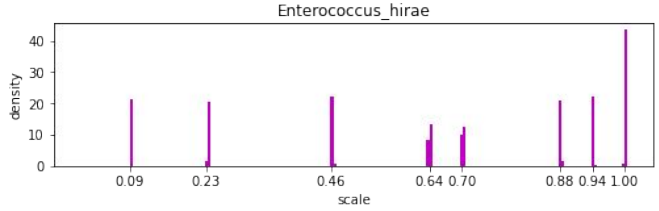
Cluster sizes: [558 558 546 583 550 511 556 1129]

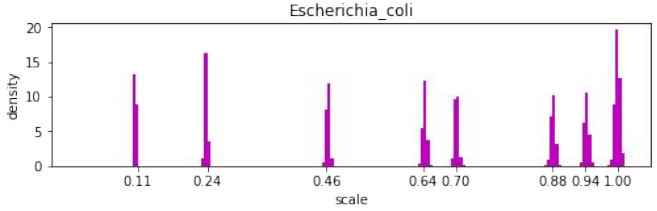
Cluster unique elements: [558 558 546 583 550 511 556 808]

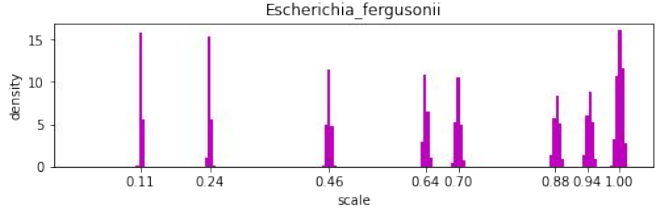
Data Visualization

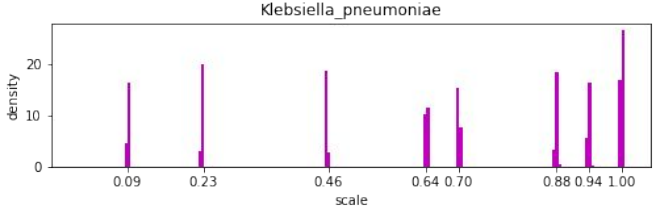


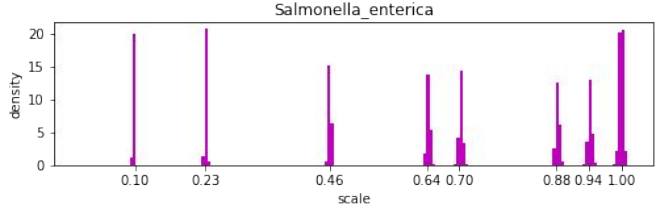


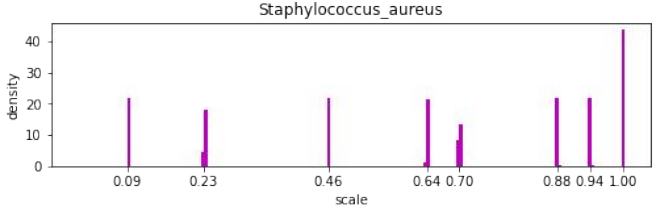


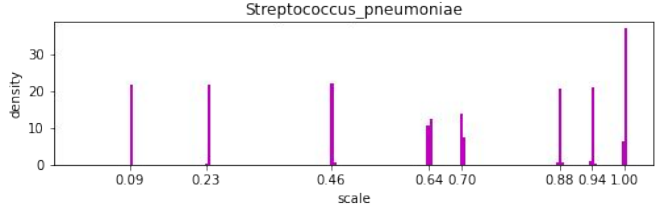


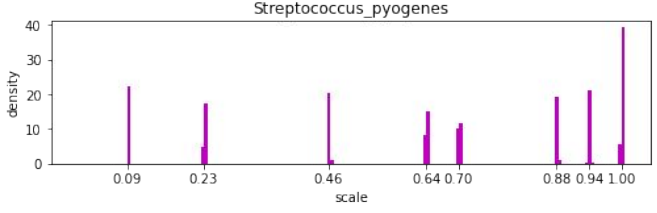










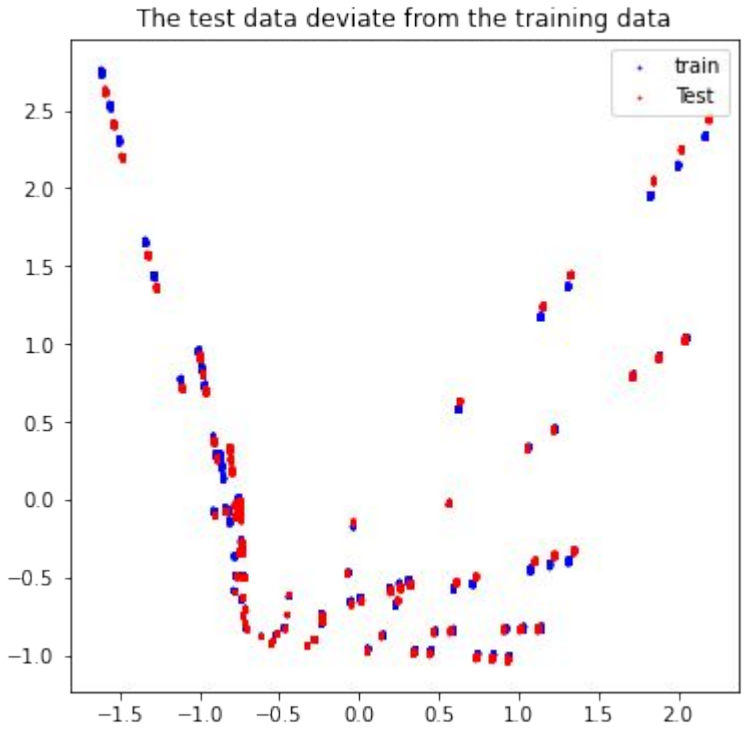


Train, Test 비교

박테리아 DNA를 훈련을 시킨 결과 Test set의 박테리아는 다른 DNA를 갖는 결과가 나왔습니다.

In [32]:

```
scale = 1

pca = PCA(whiten=True, random_state=1)
pca.fit(train_i[elements][train_df['gcd'] == scale])

Xt_tr = pca.transform(train_i[elements][train_df['gcd'] == scale])
Xt_te = pca.transform(test_i[elements][test_df['gcd'] == scale])

plt.figure(figsize=(6,6))
plt.scatter(Xt_tr[:, 0], Xt_tr[:, 1], c='b', s=1, label='train')
plt.scatter(Xt_te[:, 0], Xt_te[:, 1], c='r', s=1, label='Test')
plt.title("The test data deviate from the training data")
plt.legend()
plt.show()
```



THANK YOU ☺️