实验与分析

实验目的

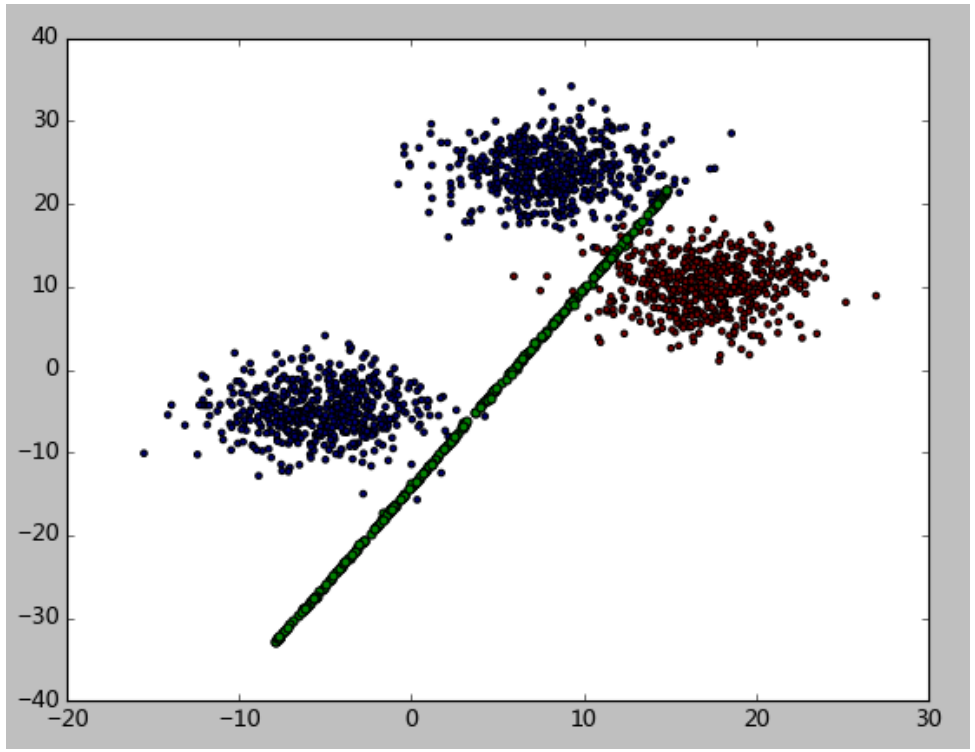
Active Learning(linear scan)

MMD ANN

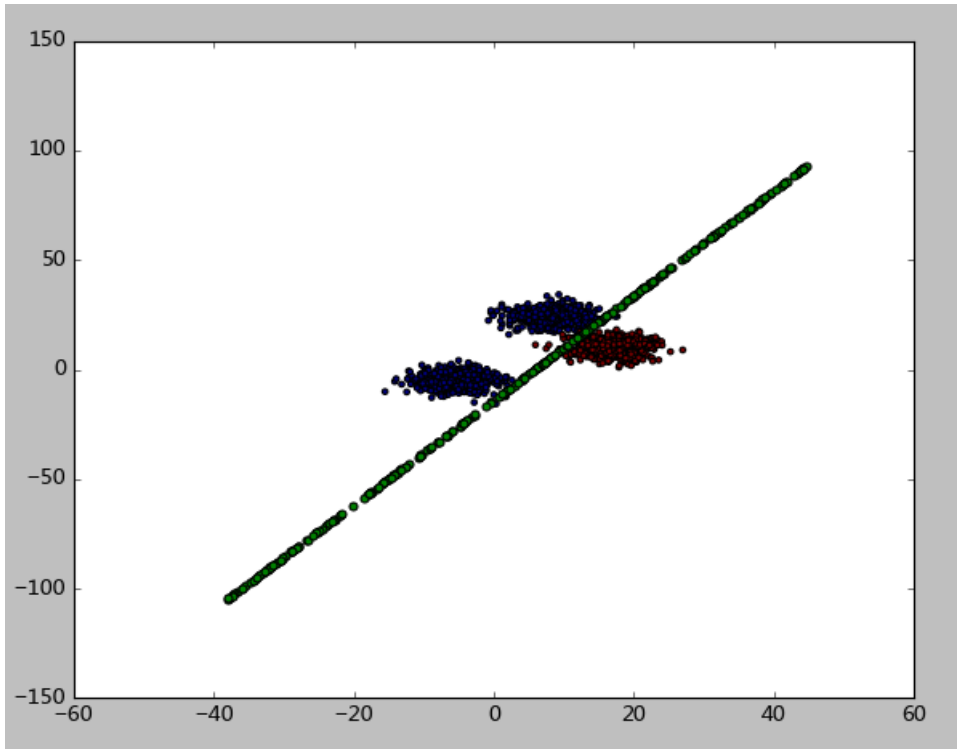
Active Learning(Generative model)

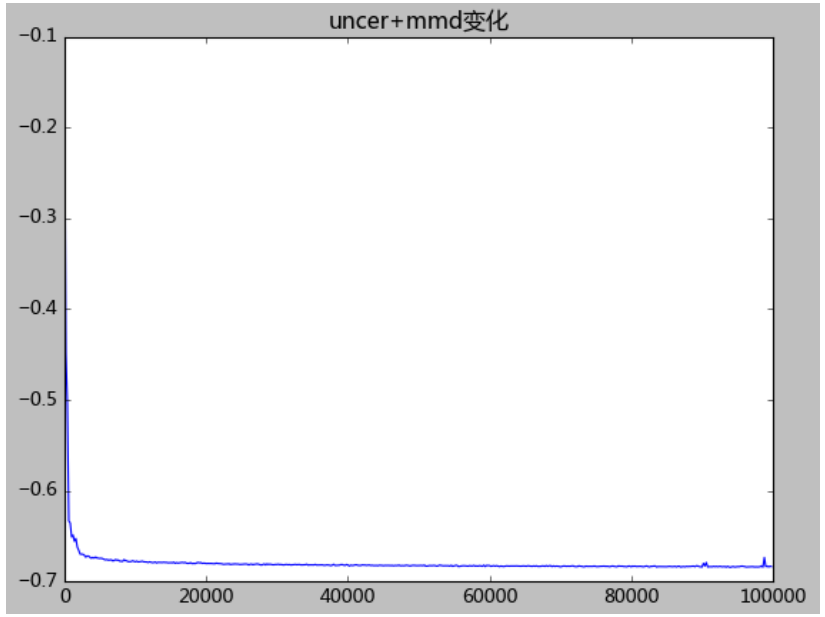
2017.11.20

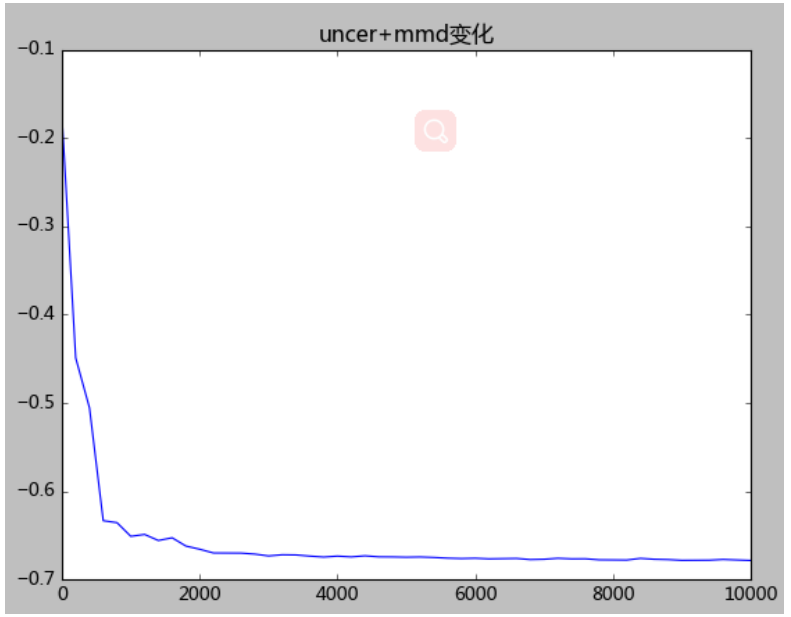
2500代

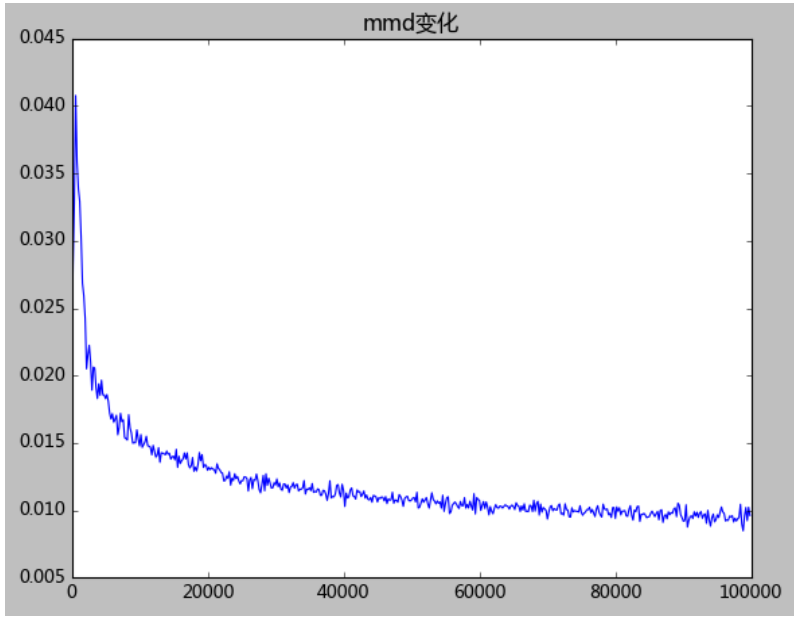


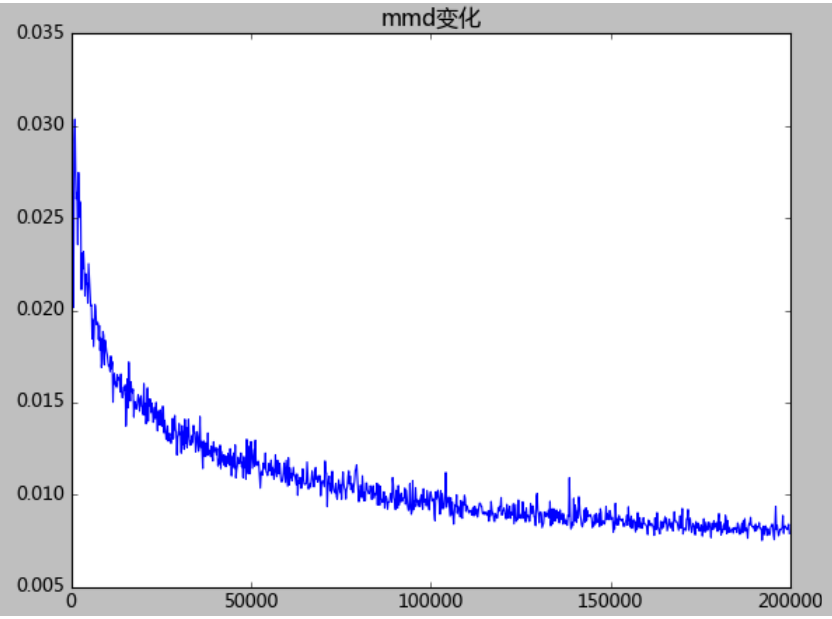
99999代

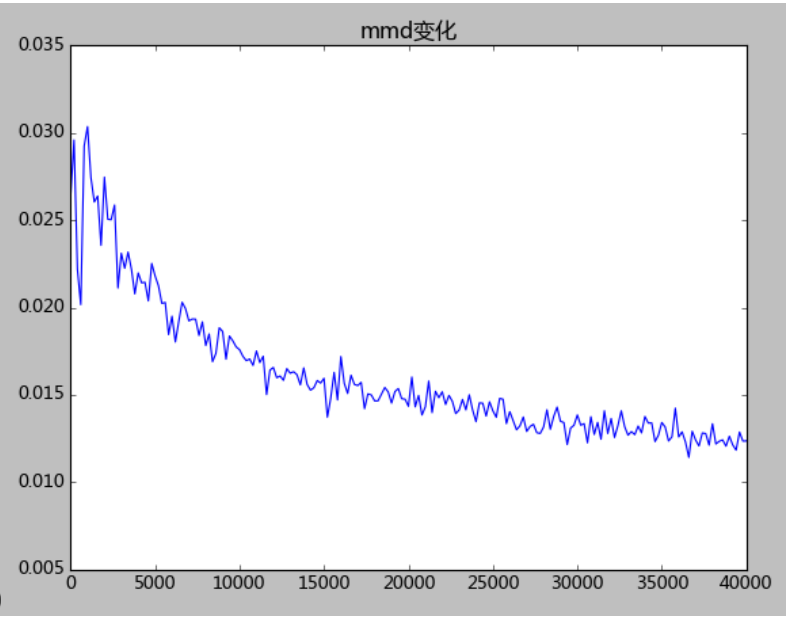


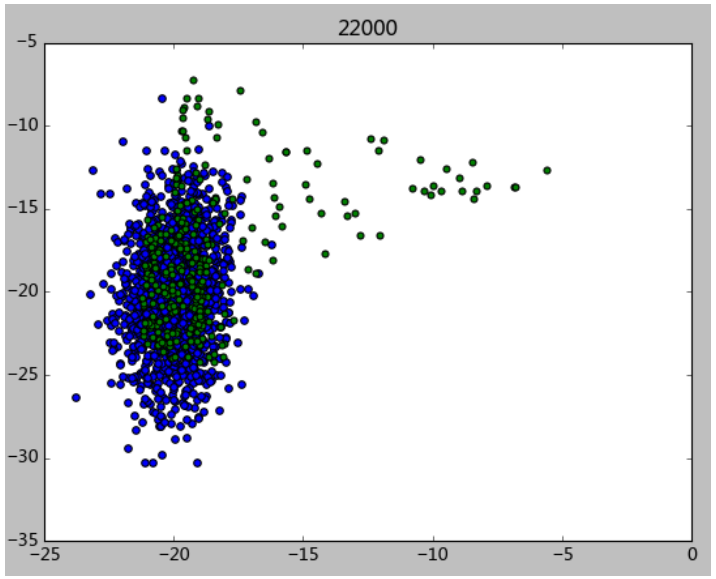

uncer+mmd变化
−0.1
−0.2
−0.3
−0.4
−0.5
−0.6
−0.7
0
20000
40000
60000
80000
100000

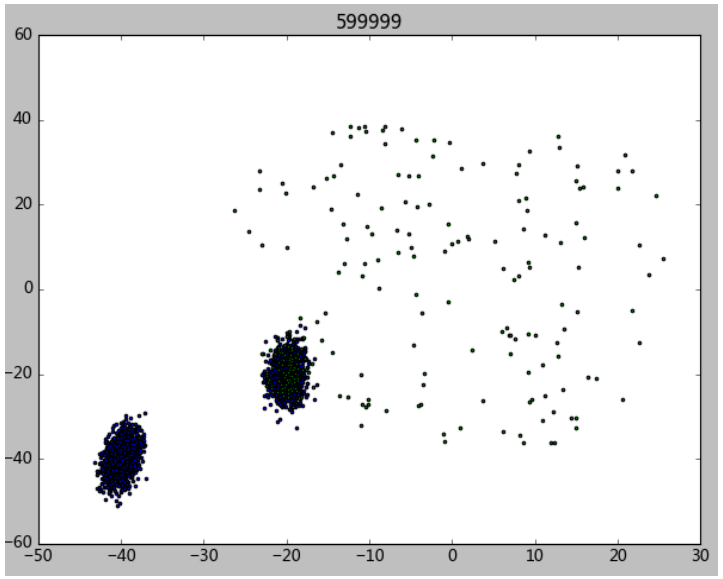

uncer+mmd变化
−0.1
−0.2
−0.3
−0.4
−0.5
−0.6
−0.7
0
2000
4000
6000
8000
10000

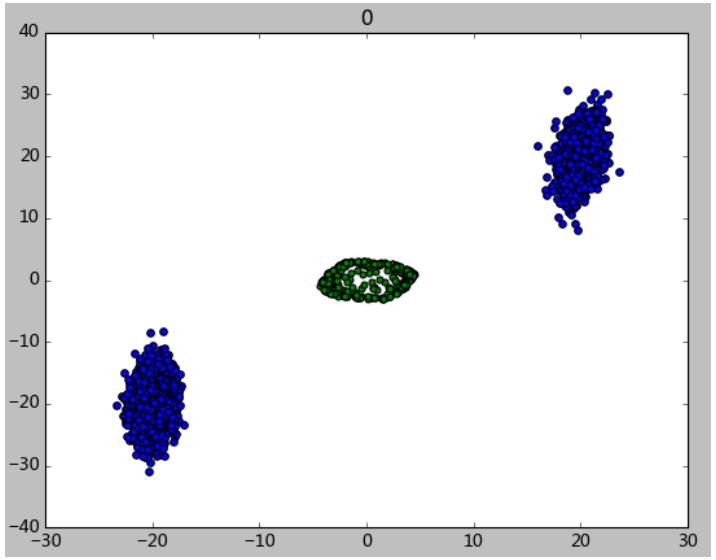

mmd变化
0.045
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0
20000
40000
60000
80000
100000


mmd变化
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0
50000
100000
150000
200000

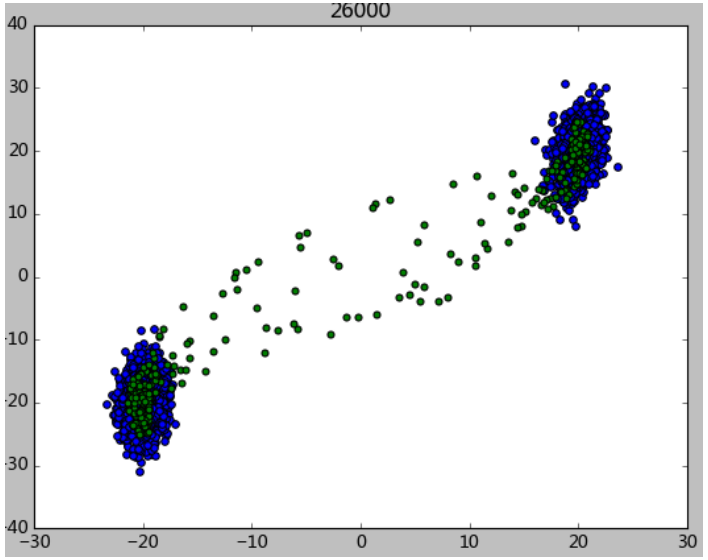

mmd变化
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0
5000
10000
15000
20000
25000
30000
35000
40000

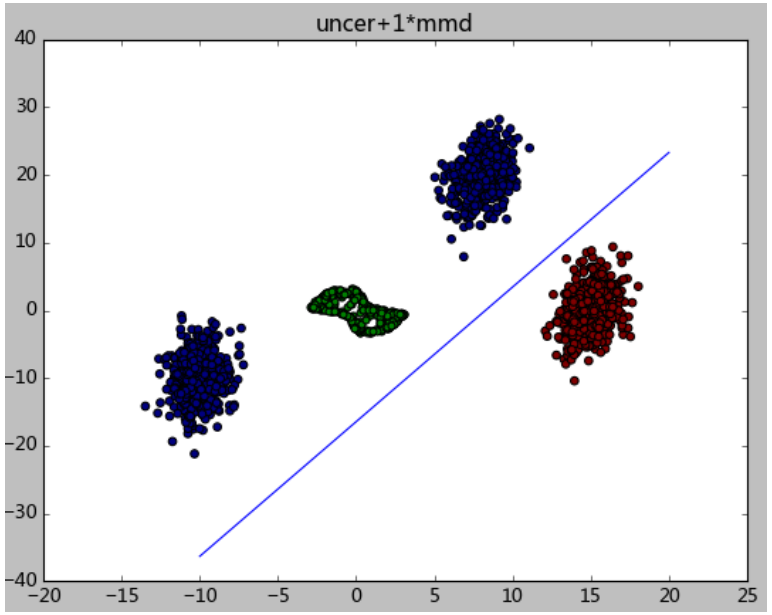




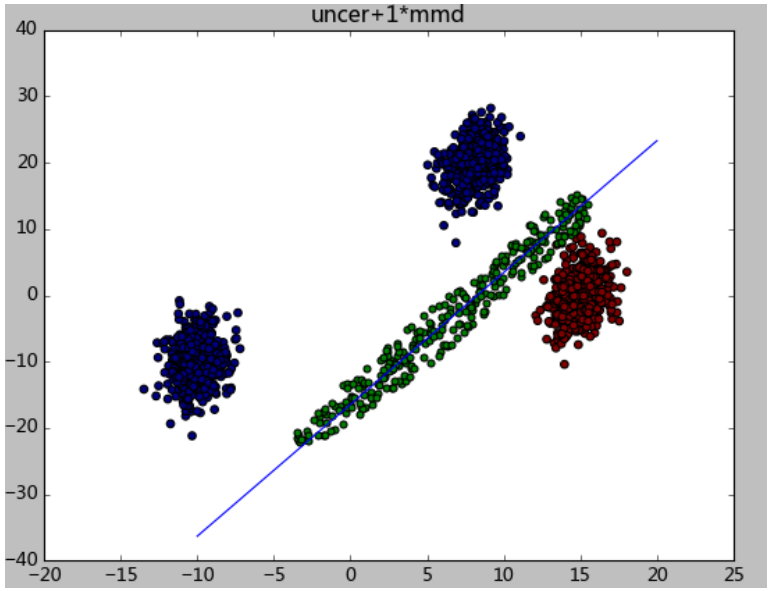


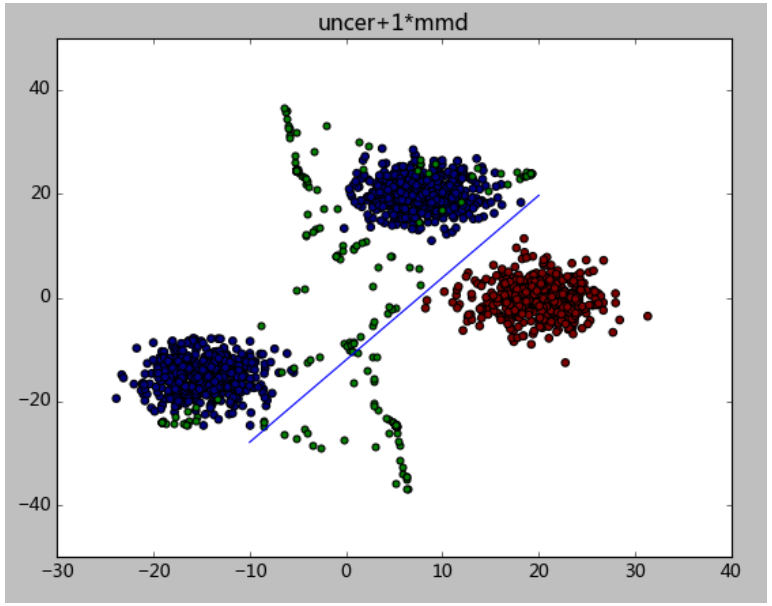
26000代



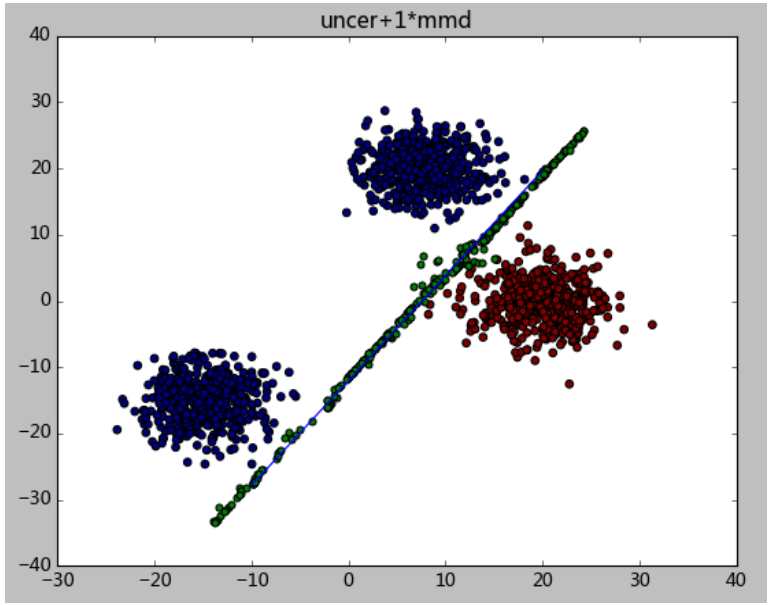


2000代后

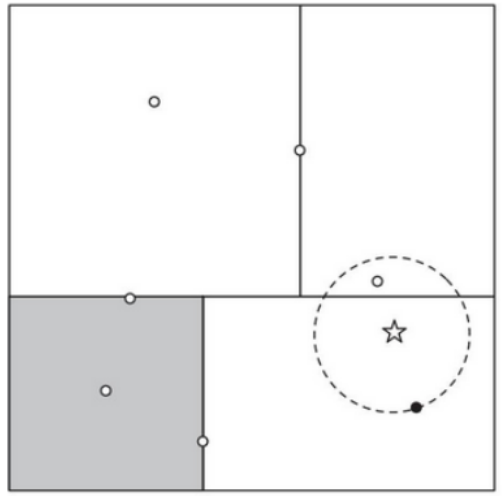


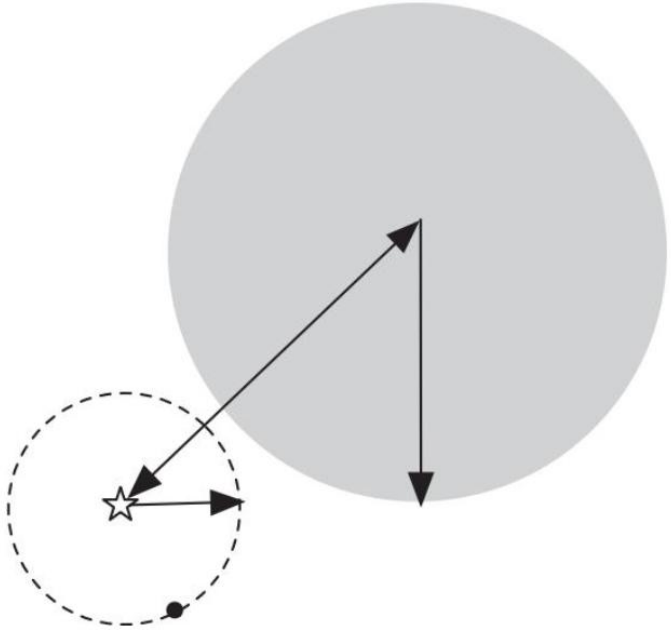


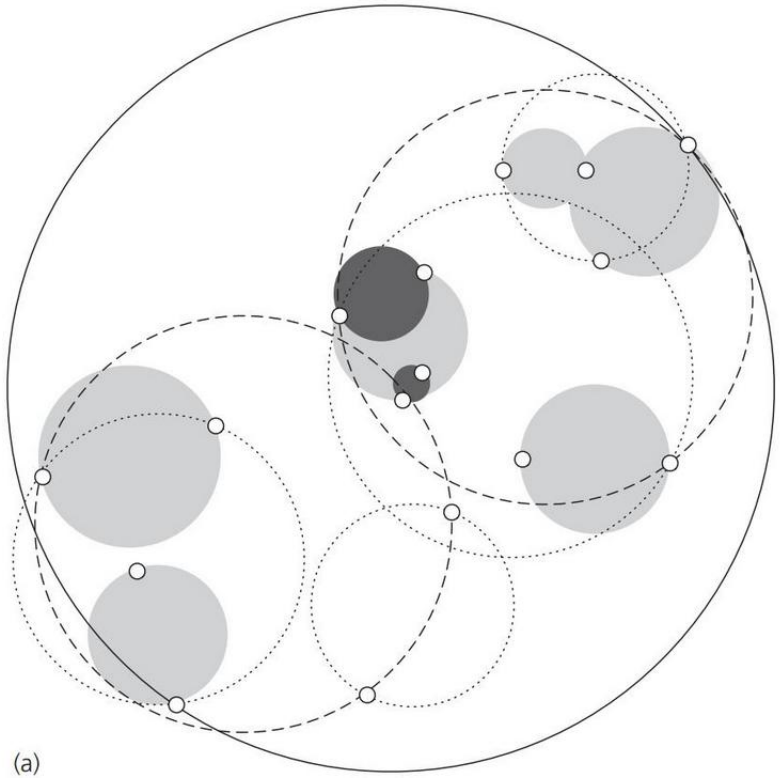
2000代后



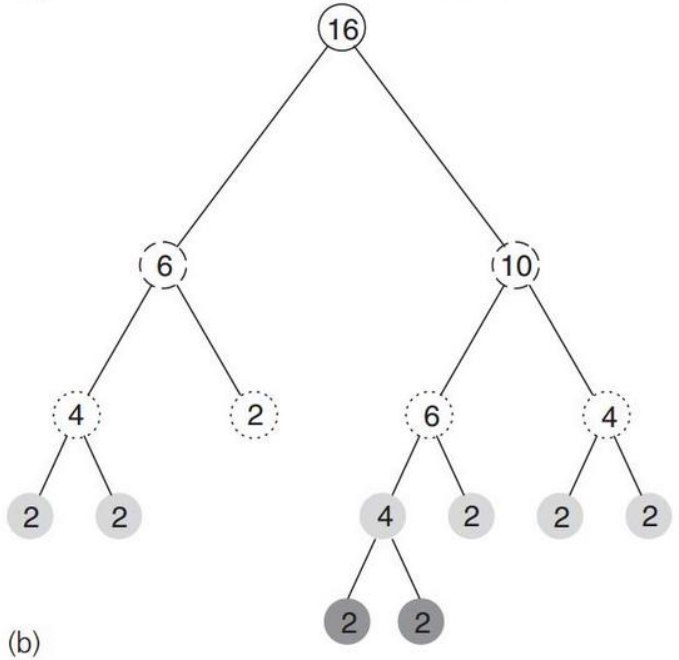
Tree







(a)

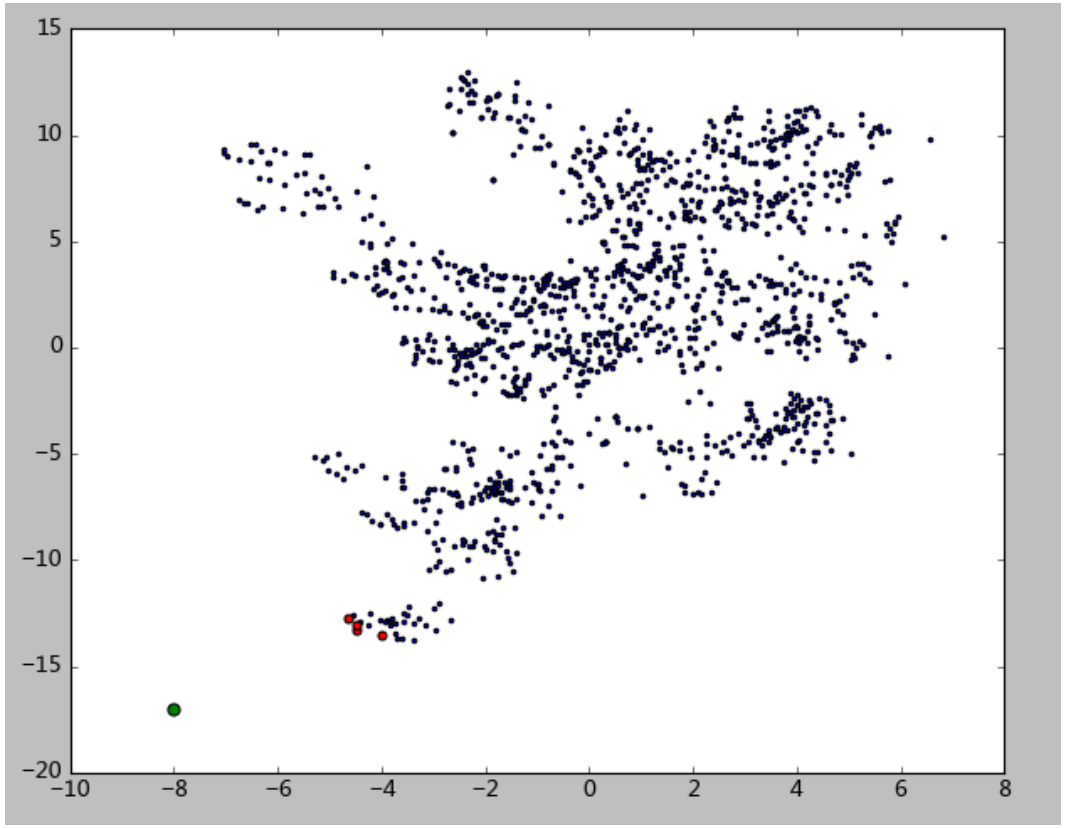


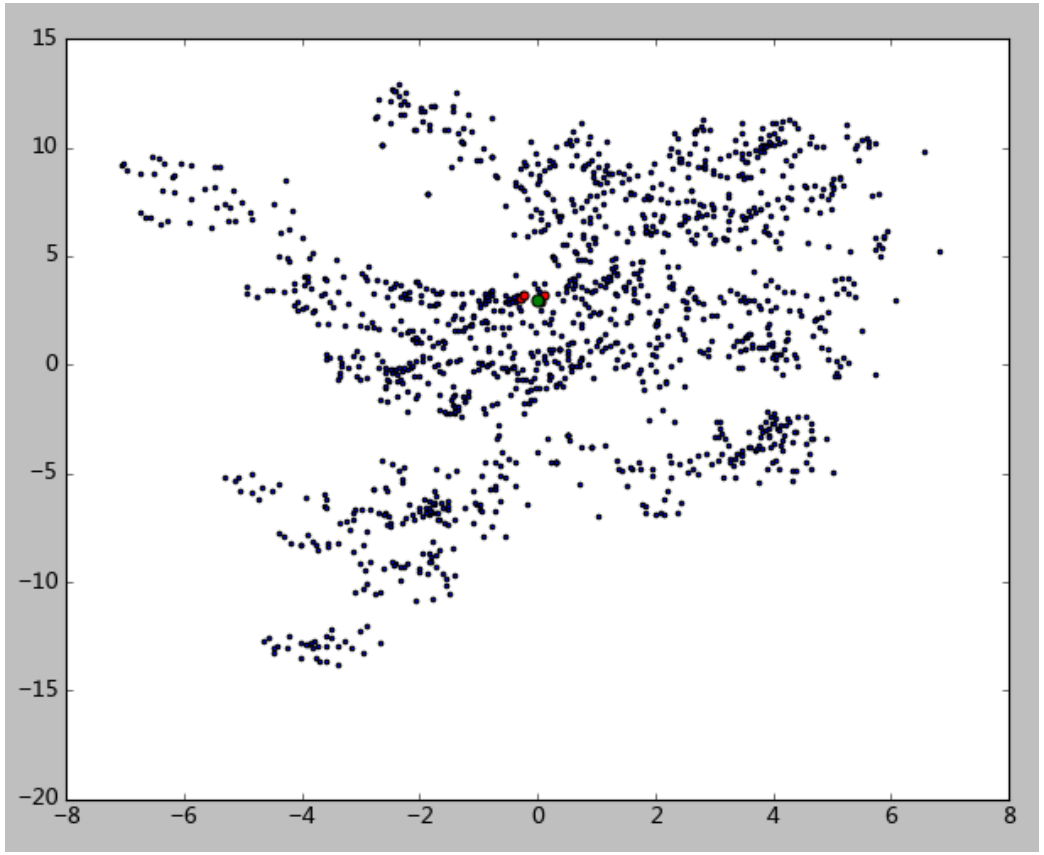
(b)

1.6.4.4. Choice of Nearest Neighbors Algorithm

The optimal algorithm for a given dataset is a complicated choice, and depends on a number of factors:

- number of samples N (i.e. `n_samples`) and dimensionality D (i.e. `n_features`).
  - *Brute force* query time grows as $O[DN]$
  - *Ball tree* query time grows as approximately $O[D\log(N)]$
  - *KD tree* query time changes with D in a way that is difficult to precisely characterise. For small D (less than 20 or so) the cost is approximately $O[D\log(N)]$, and the KD tree query can be very efficient. For larger D, the cost increases to nearly $O[DN]$, and the overhead due to the tree structure can lead to queries which are slower than brute force.


15
10
5
0
−5
−10
−15
−20
−10
−8
−6
−4
−2
0
2
4
6
8


15
10
5
0
−5
−10
−15
−20
−8
−6
−4
−2
0
2
4
6
8

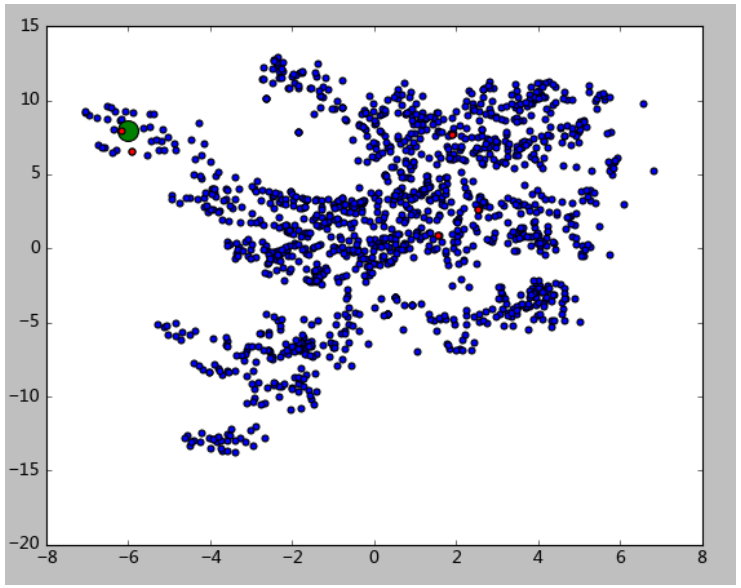
p-stable LSH

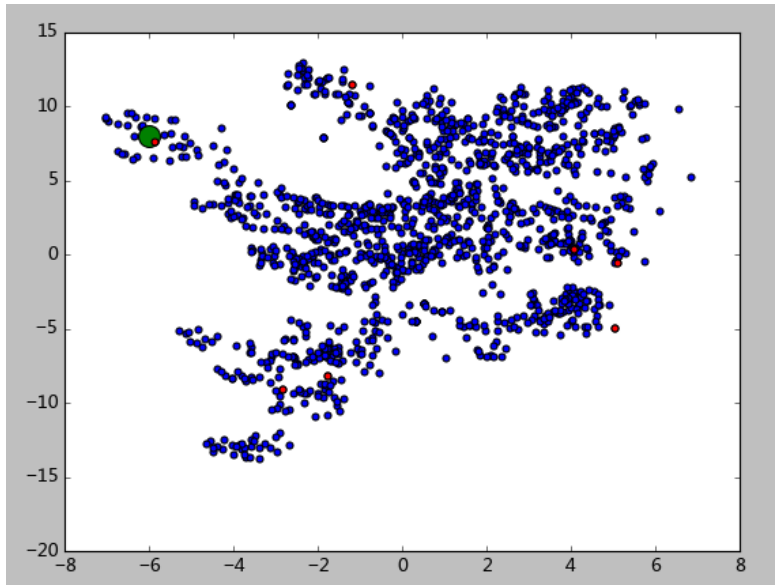
p-stable

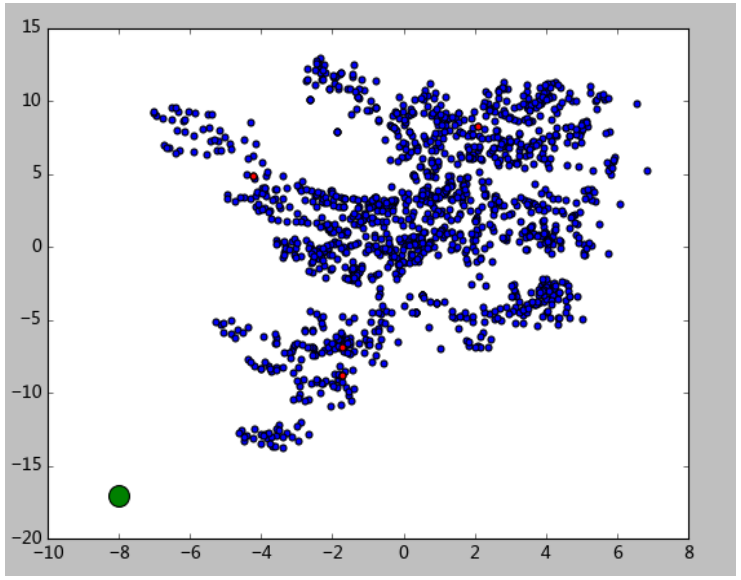
Stable Distribution: A distribution $\mathcal{D}$ over $\Re$ is called *p-stable*, if there exists $p \geq 0$ such that for any n real numbers $v_1 \ldots v_n$ and i.i.d. variables $X_1 \ldots X_n$ with distribution $\mathcal{D}$, the random variable $\sum_i v_i X_i$ has the same distribution as the variable $(\sum_i |v_i|^p)^{1/p} X$, where X is a random variable with distribution $\mathcal{D}$.

$$(a \cdot v_1 - a \cdot v_2) \longleftrightarrow \|v_1 - v_2\|_p X$$

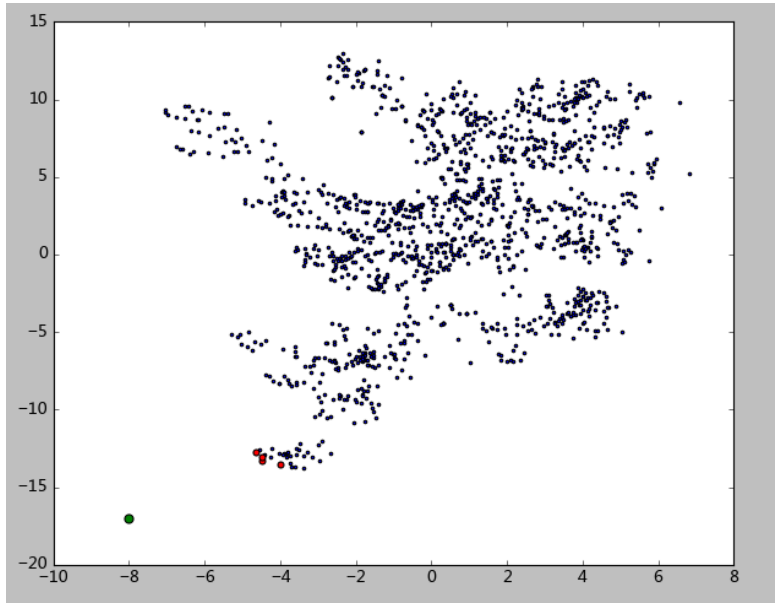
$$h_{a,b}(v) = \left\lfloor \frac{a \cdot v + b}{r} \right\rfloor \qquad b \sim \mathrm{U}(0, r)$$


15
10
5
0
−5
−10
−15
−20
−8
−6
−4
−2
0
2
4
6
8

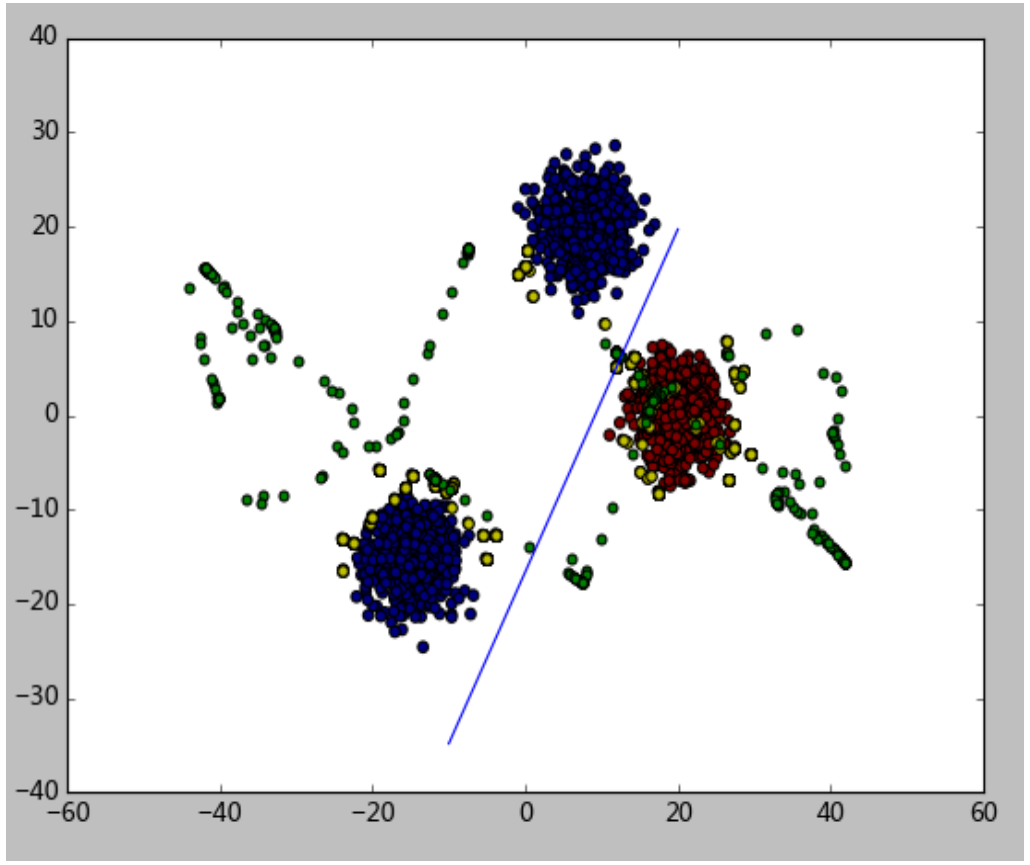

15
10
5
0
−5
−10
−15
−20
−8
−6
−4
−2
0
2
4
6
8


15
10
5
0
−5
−10
−15
−20
−10
−8
−6
−4
−2
0
2
4
6
8

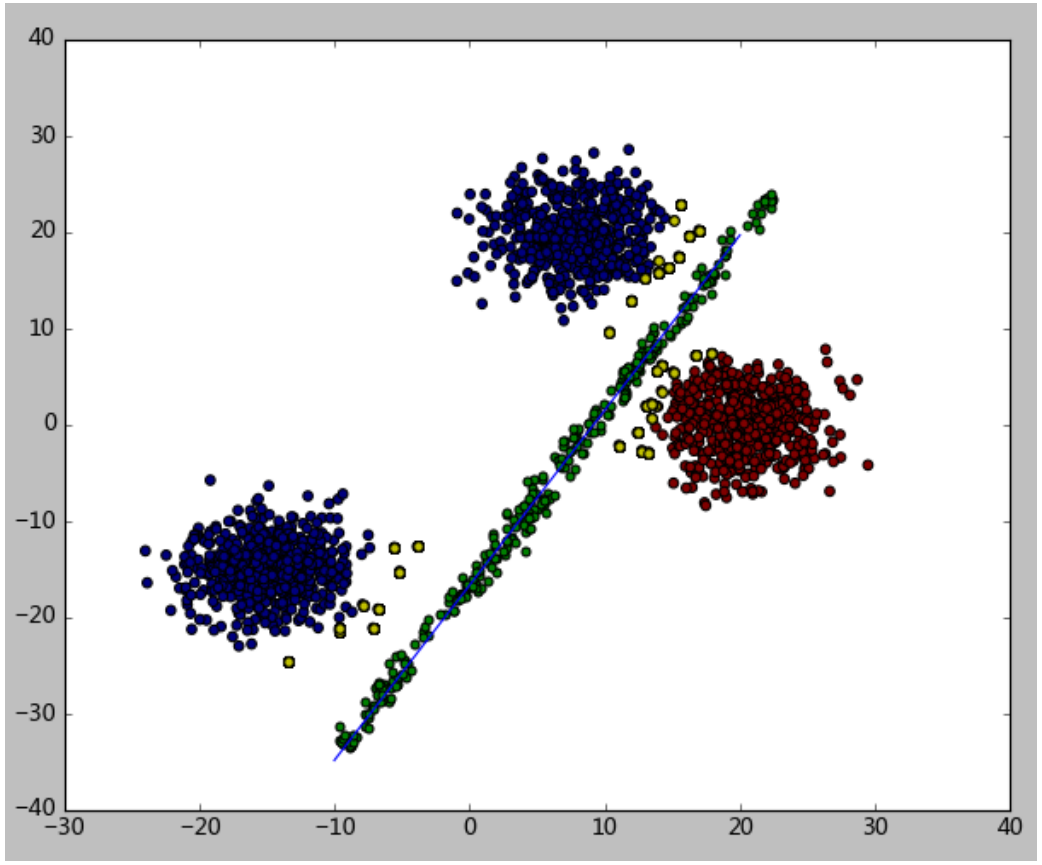
[-8, -17]

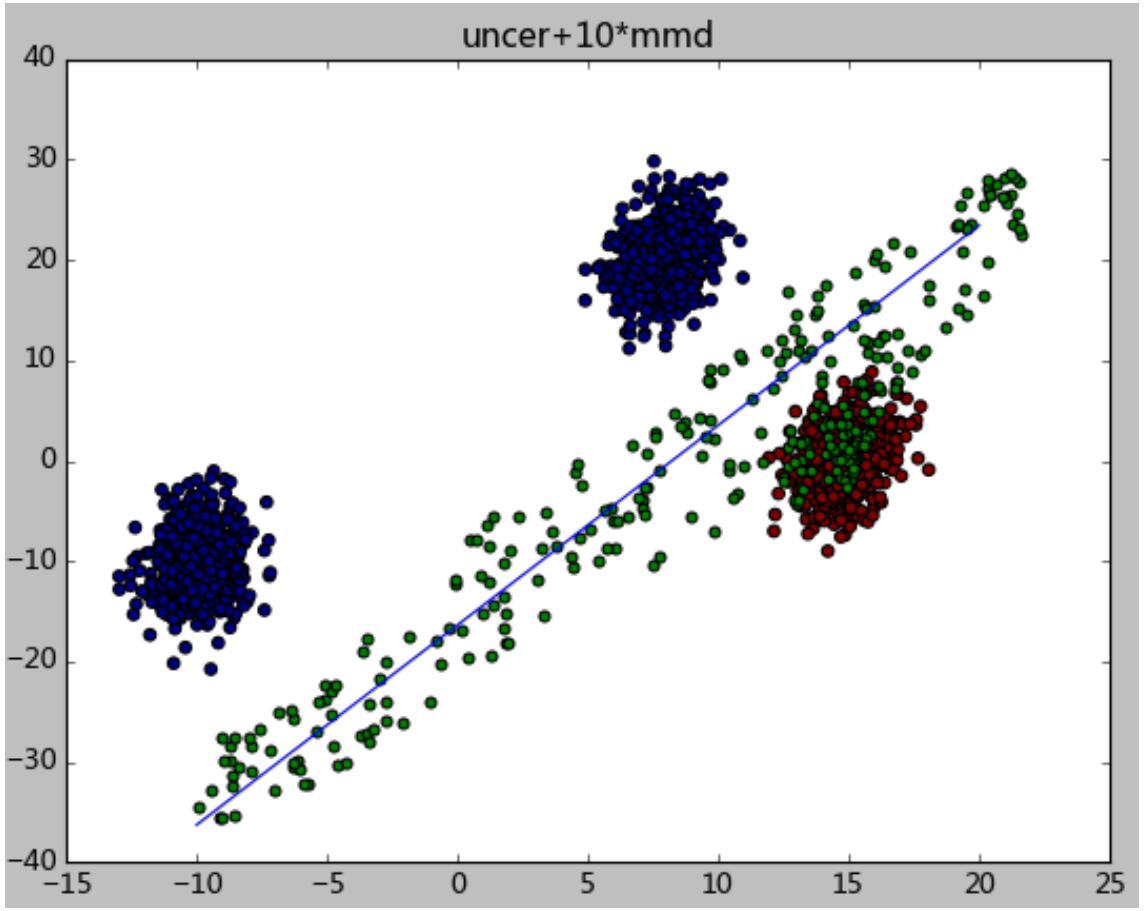

15
10
5
0
−5
−10
−15
−20
−10
−8
−6
−4
−2
0
2
4
6
8

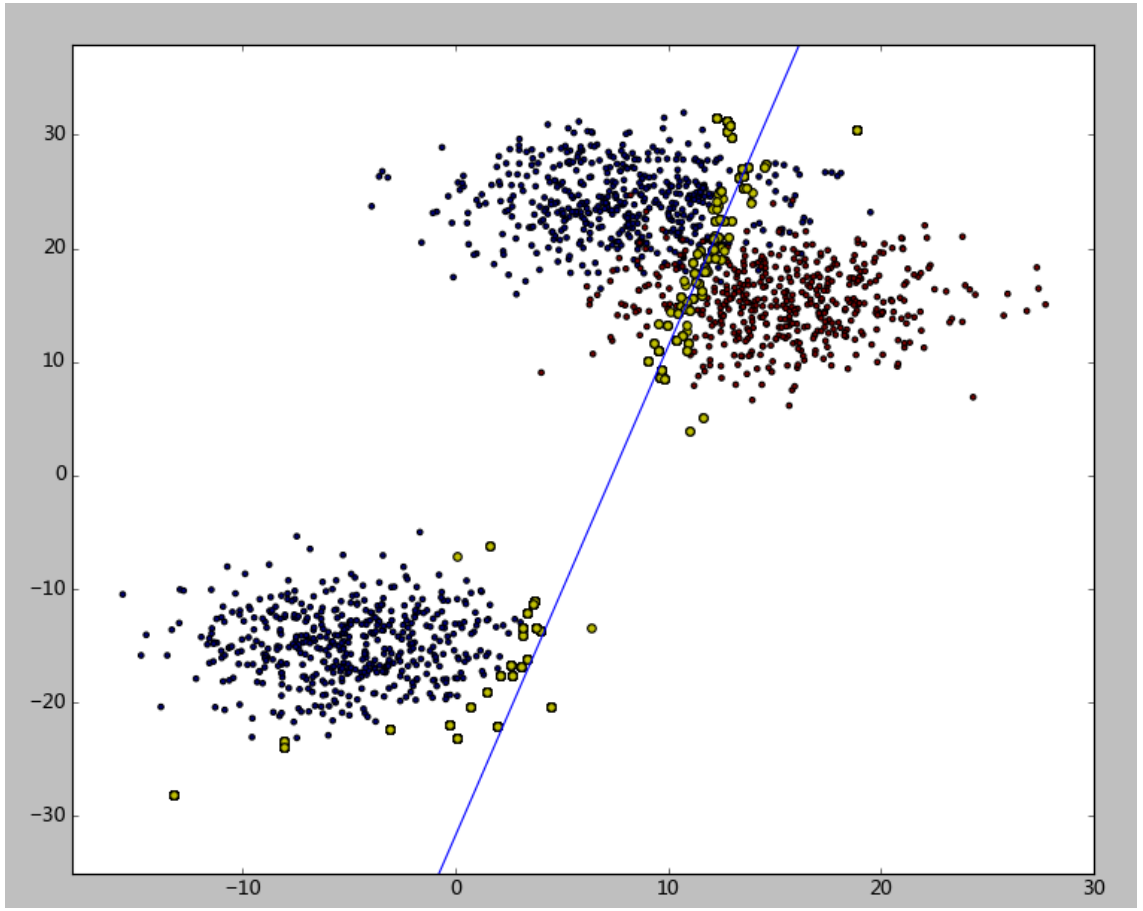
初始状态


40
30
20
10
0
−10
−20
−30
−40
−60
−40
−20
0
20
40
60

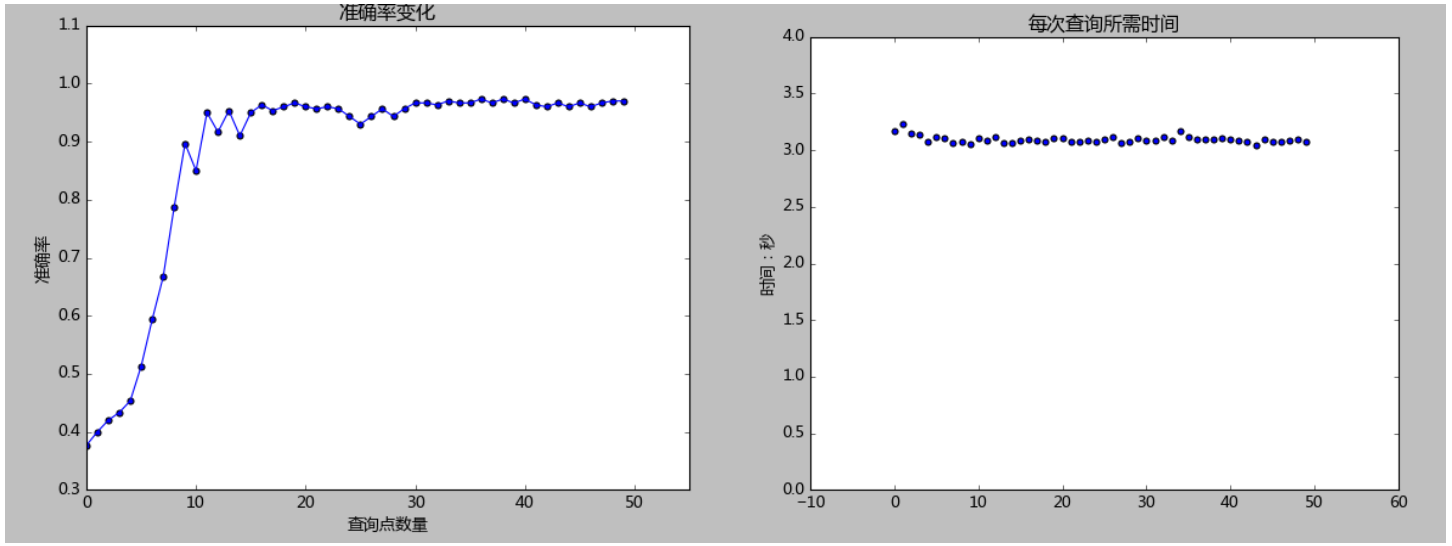
若干代之后


40
30
20
10
0
−10
−20
−30
−40
−30
−20
−10
0
10
20
30
40


uncer+10*mmd
40
30
20
10
0
−10
−20
−30
−40
−15
−10
−5
0
5
10
15
20
25

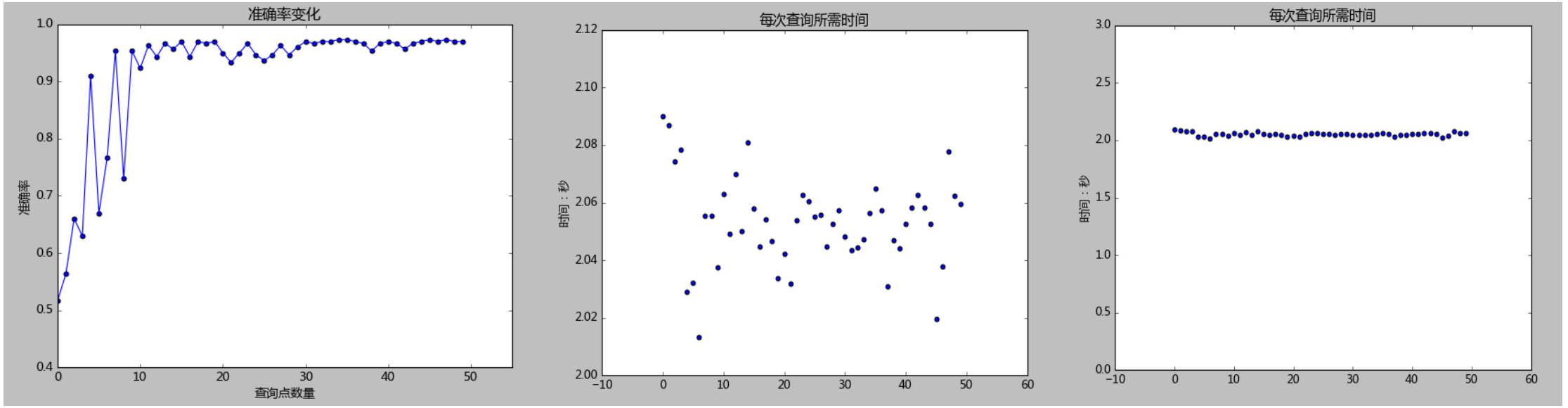

30
20
10
0
−10
−20
−30
−10
0
10
20
30

linear scan



300万

450万



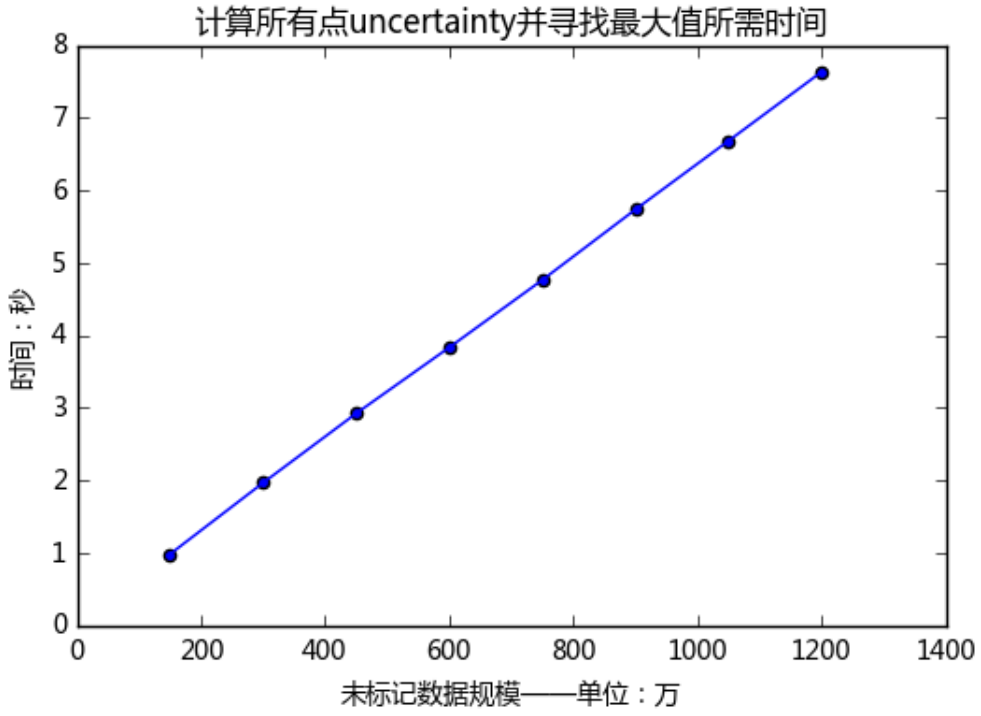
```
start = time.clock()
uncer = uncertainty(coef, intercept, X)
# print(type(uncer))
# id = np.argmax(uncer)
id = max_id(uncer)
# max_value = np.max(uncer)
# id = np.argwhere(uncer==max_value)
mid = (time.clock() - start)
```

```
def max_id(x):
    max = 0;
    id = -1;
    for i in range(len(x)):
        if(x[i]>max):
            max = x[i]
            id = i
    return id
```

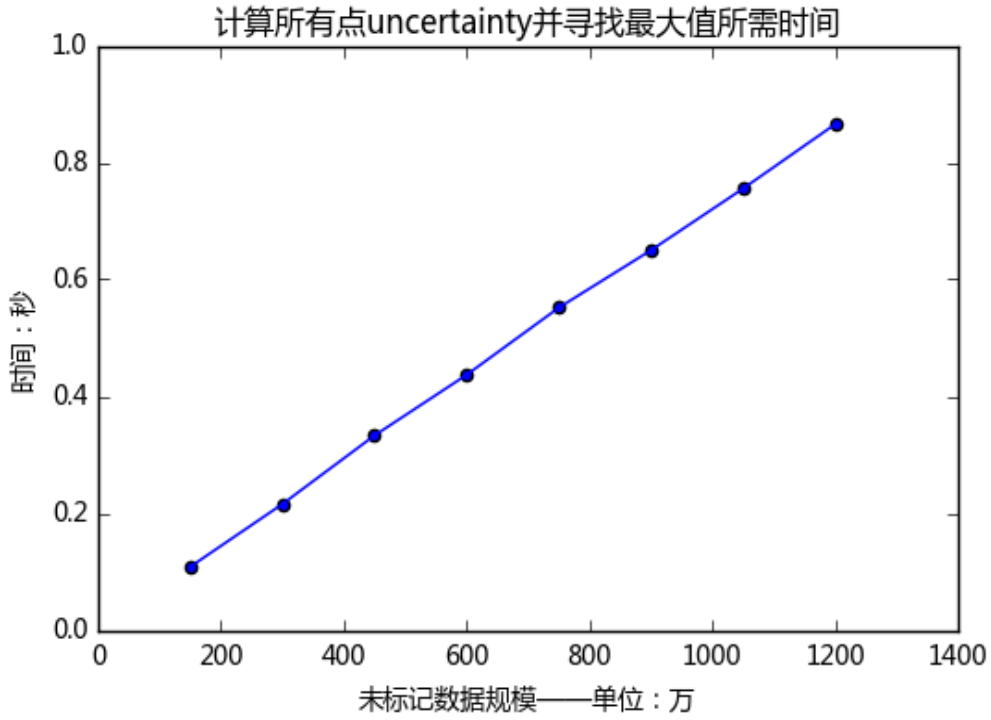
```
start = time.clock()
for i in range(num_it):
    x_data = np.random.uniform(-4, 4, size=[300, 2])
    sess.run(train_step, feed_dict={xs: x_data, coef_tf: coef})

    if i == num_it - 1:
        prediction_value = sess.run(prediction, feed_dict={xs: x_data, coef_tf: coef})
        dist, ind = tree_copy.query(prediction_value, k=3)
        # plt.scatter(train_u[:, 0], train_u[:, 1], c=train_z, s=10)
        # plt.scatter(train_u[ind, 0], train_u[ind, 1], c='y', s=25)
        ind = np.unique(ind)
        train_candidates = train_u[ind]
        uncer_candidates = uncertainty(coef, intercept, train_candidates)
        max_id = np.argmax(uncer_candidates)
        id = ind[max_id]
        # plt.scatter(prediction_value[:, 0], prediction_value[:, 1], c='g')
        # plt.scatter(train_u[id, 0], train_u[id, 1], c='r')
        # plt.plot(norm, norm_y)
        # plt.show()
mid_time = (time.clock() - start)
time_cost.append(mid_time)
```

```python
def uncertainty(coef, intercept, x_pred):
    middle = np.dot(x_pred, np.array(coef)) + intercept
    p1 = np.exp(middle) / (1.+np.exp(middle))
    p2 = 1 / (1+np.exp(middle))
    uncer = -1 * np.multiply(p1, np.log(p1)) - np.multiply(p2, np.log(p2))
    return uncer
```

max_id



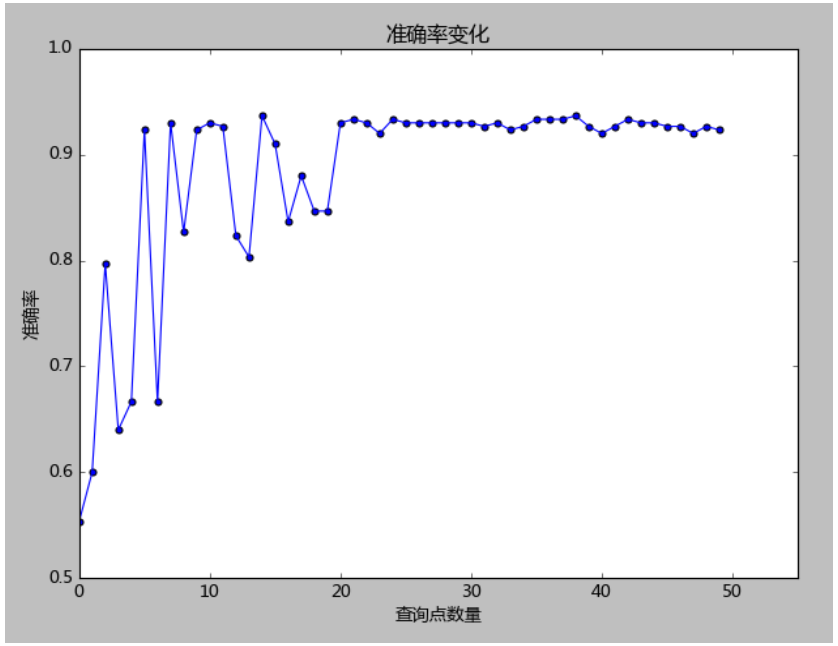
argmax



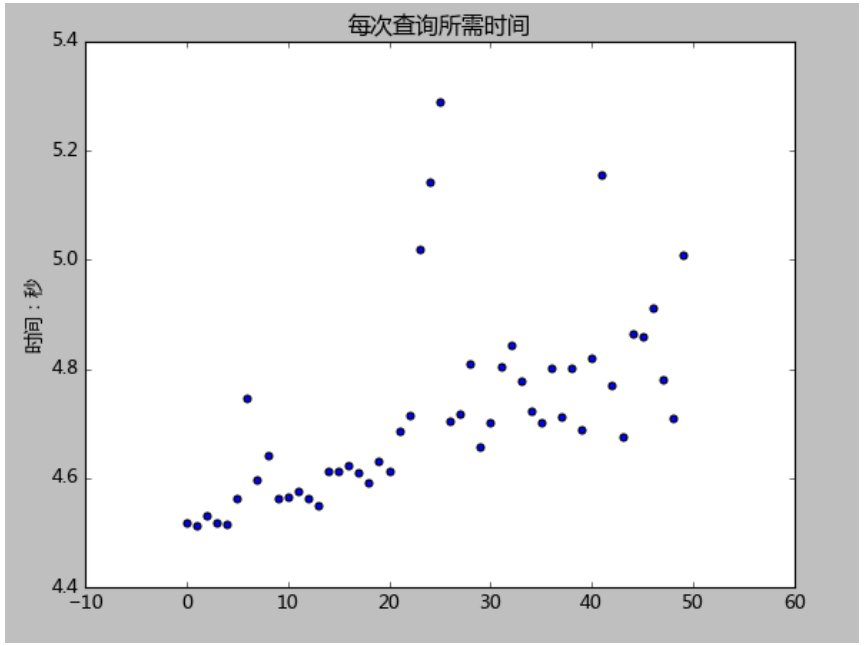
```
    return _wrapfunc(a, 'argmax', axis=axis, out=out)
```

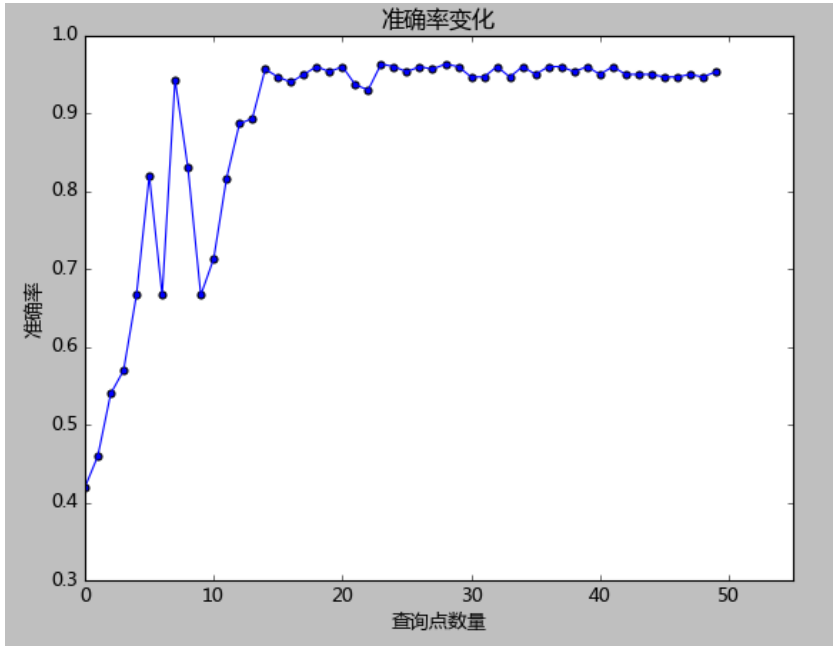
```
def _wrapfunc(obj, method, *args, **kwds):
    try:
        return getattr(obj, method)(*args, **kwds)

    # An AttributeError occurs if the object does not have
    # such a method in its class.

    # A TypeError occurs if the object does have such a method
    # in its class, but its signature is not identical to that
    # of NumPy's. This situation has occurred in the case of
    # a downstream library like 'pandas'.
    except (AttributeError, TypeError):
        return _wrapit(obj, method, *args, **kwds)
```

```
getattr(object, name, default=None): # known special case of getattr
"""
getattr(object, name[, default]) -> value

Get a named attribute from an object; getattr(x, 'y') is equivalent to x.y.
When a default argument is given, it is returned when the attribute doesn't
exist; without it, an exception is raised in that case.
"""
pass
```

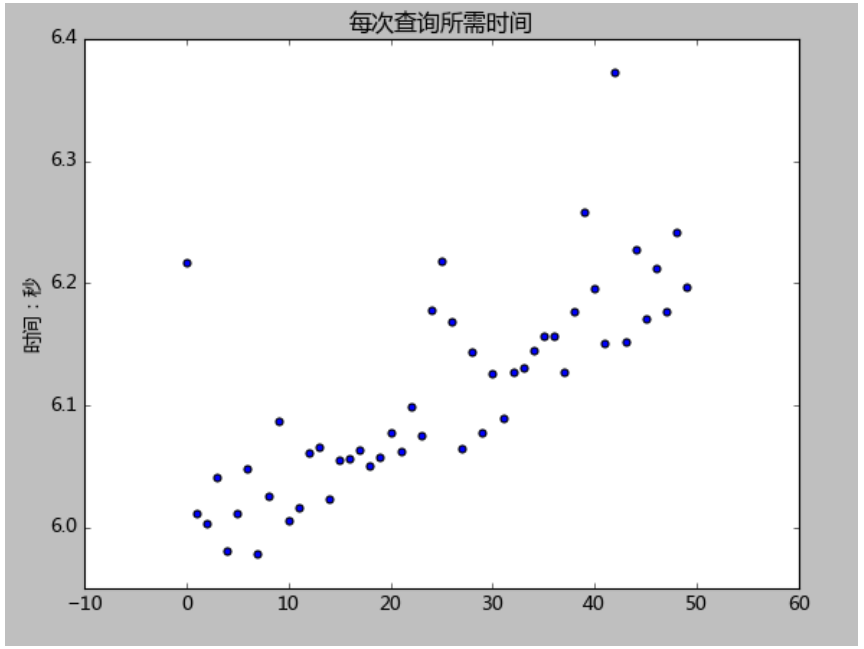


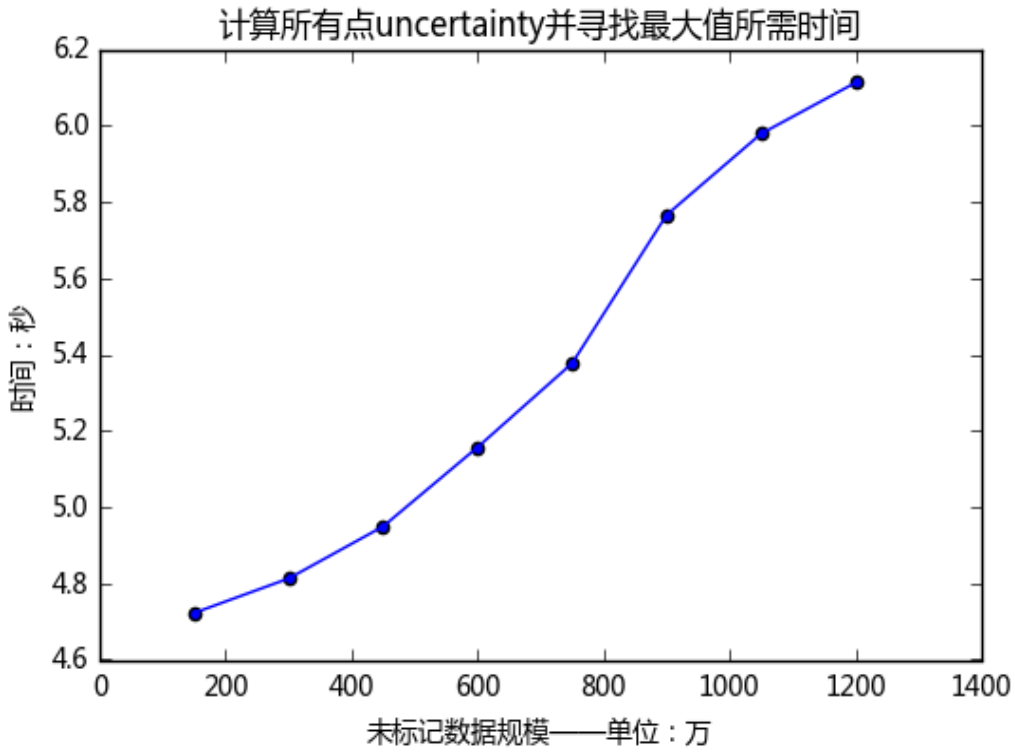
150万

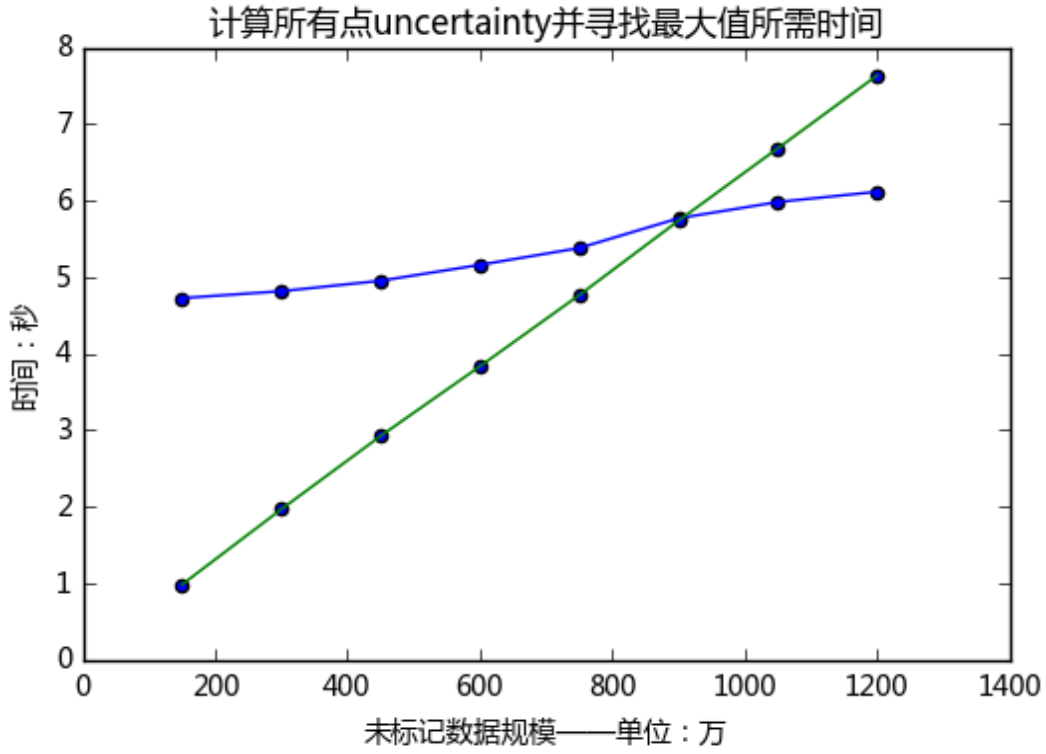




1200万




计算所有点uncertainty并寻找最大值所需时间
时间：秒
未标记数据规模——单位：万
6.2
6.0
5.8
5.6
5.4
5.2
5.0
4.8
4.6
0
200
400
600
800
1000
1200
1400


计算所有点uncertainty并寻找最大值所需时间
时间：秒
未标记数据规模——单位：万
8
7
6
5
4
3
2
1
0
0
200
400
600
800
1000
1200
1400

实验总结

1. 生成数据点在分类界面上的延伸
2. MMD训练不完全重合与阻挡
3. 网络初值选取——覆盖范围广
4. 采用BallTree作为ANN方法
5. 大uncertainty小MMD
6. 查询时间随查询点的增多而增加
7. 是否采用numpy函数
8. 网络权值初始化
9. 网络终止条件的设置

Next Dataset和Model选择