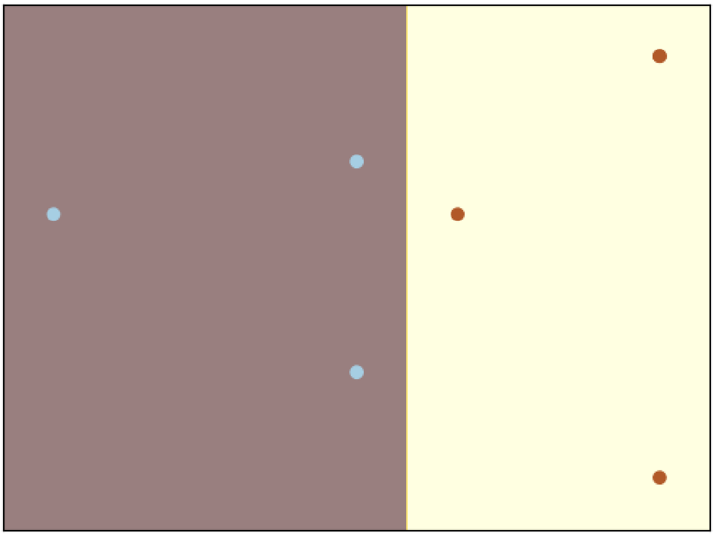
a(i)



```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm

# (i)
X = [[1, 0], [0, 1], [0, -1], [-1, 0], [0, 2], [0, -2], [-2, 0]]
Y = [-1, -1, -1, 1, 1, 1, 1]

X_t = []
for i in range(7):
    x1 = 2*X[i][1]**2 - 4*X[i][0] + 1
    x2 = X[i][0]**2 - 2*X[i][1] - 3
    X_t.append([x1,x2])
```

```python
X = np.array(X)
Y = np.array(Y)
X_t = np.array(X_t)
clf = svm.SVC(C=7, kernel='linear')
clf.fit(X_t, Y)
h = 0.01
x_min, x_max = X_t[:, 0].min() - 1, X_t[:, 0].max() + 1
y_min, y_max = X_t[:, 1].min() - 1, X_t[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),np.arange(y_min, y_max, h))
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.xticks(())
plt.yticks(())
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap='hot', alpha=0.5)
plt.scatter(X_t[:, 0], X_t[:, 1], c=Y, cmap=plt.cm.Paired)
plt.show()
```

a(ii)

```
the indices of my identified support vectors:
[[ 3. -5.]
 [ 3. -1.]
 [ 5. -2.]]
estimated values:
[-1 -1 -1  1  1  1  1]
error:
0.0
```

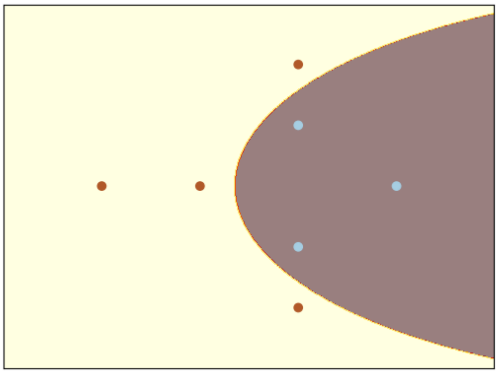
```
# (ii)
print('the indices of my identified support vectors:')
print(clf.support_vectors_)
print('estimated values:')
print(clf.predict(X_t))
print('error:')
print(1-clf.score(X_t, Y))
```

a(iii)

```
the indices of my identified support vectors:
[[ 0.  1.]
 [ 0. -1.]
 [-1.  0.]
 [ 0.  2.]
 [ 0. -2.]]
estimated values:
[-1 -1 -1  1  1  1  1]
error:
0.0
```

```python
# (iii)
clf = svm.SVC(C=7, kernel='poly', degree=2, coef0=2)
clf.fit(X, Y)
print('the indices of my identified support vectors:')
print(clf.support_vectors_)
print('estimated values:')
print(clf.predict(X))
print('error:')
print(1-clf.score(X, Y))
```

a(iv)The two methods completely different models, one is a linear classification while the other is a polynomial approach where both fit the training data exactly, but the polynomial kernel approach has more support vectors in it, thus I think the answer in (iii) is better than (ii).



b

(i) for every X_i (i in 1 to n)
find a h_j (j in 1 to T) ~~for~~
to make $rMSE(h_j)=0$, so j is what we want
so the queries is T^n
The larger the n, the larger the scale of the algorithm

(ii) $h(X)=(h(X_1),h(X_2),\cdots,h(X_n))^T$

$y=(y_1,y_2,\cdots,y_n)^T$

$RMSE=\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i-h(x_i))^2}$

$\Rightarrow n\,RMSE^2=\|y-h(x)\|_2^2$

so using RMSE and $h(x)$ to compute y and then compute $y^Th(x)$
smallest number of querles $=n$.

(iii) $H=\{h_1,\cdots,h_K\}$

① using RMSE and $h(x)$ to compute y and then compute $y^Th(x)$
queries number $=kn$
② obtain the optimal weights $\alpha_1,\cdots,\alpha_K$