Abdul Qayyum

Lecturer at University of Burgundy, France

- Postdoc in Electrical and Informatics Engineering
- PhD in Electrical & Electronics Engineering
- Masters in Electronics Engineering
- Bachelor in Computer Engineering

Collaborations & Expertise:

Topic: Naive Bayes and Support Vector Machine Classifiers

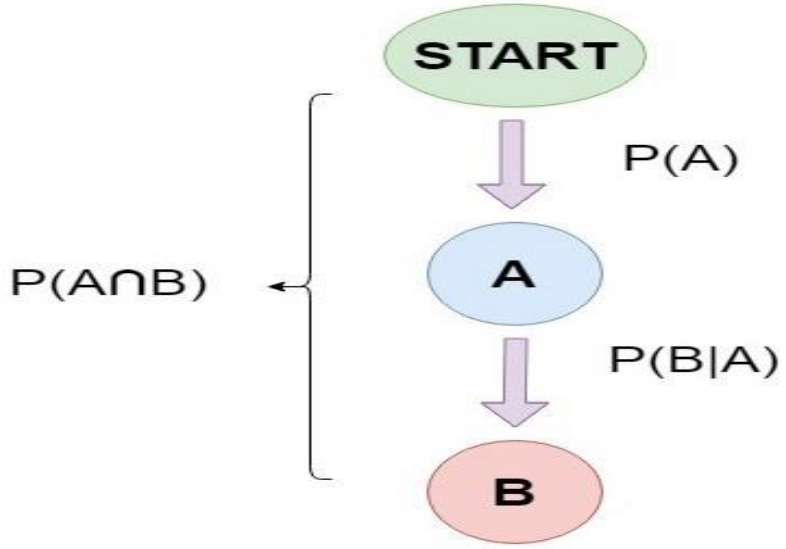
Instructor: Abdul Qayyum, PhD

Class: MSCV

University of Burgundy, France

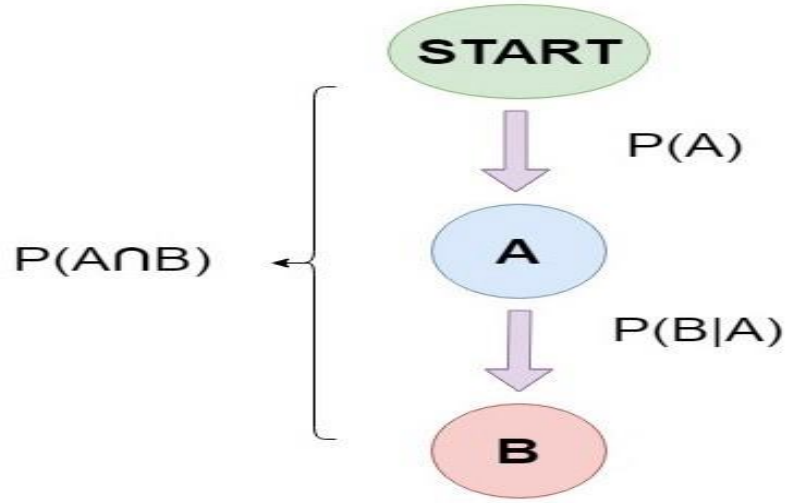
Bayes' Theorem?

- **What is Conditional Probability?**
- Conditional probability is a measure of the probability of an event (some particular situation occurring) given that (by assumption, presumption, assertion or evidence) another event has occurred. If the event of interest is A and the event B is known or assumed to have occurred, "the conditional probability of A given B", or "the probability of A under the condition B", is usually written as P(A|B), or sometimes P(A/B).
- So now let us try to interpret it visually by a new approach



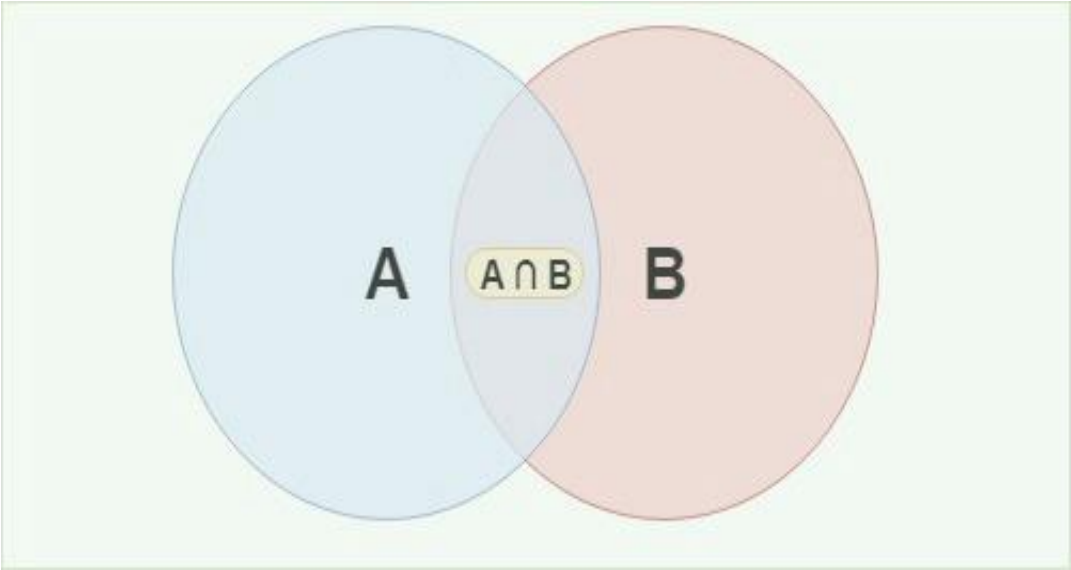
Bayes' Theorem?

- **What is Conditional Probability?**
- Let us assume that we start our observations within the timeline from START. There is a possibility P(A) that event A occurs after we start observing the timeline. There is also a possibility for another event B to occur after A and the odds of that are denoted by P(B|A).
- Since both events occur successively, the probability for this whole timeline to come about(i.e. A and B both occur and B takes place after A) is
- P(A)· P(B|A)



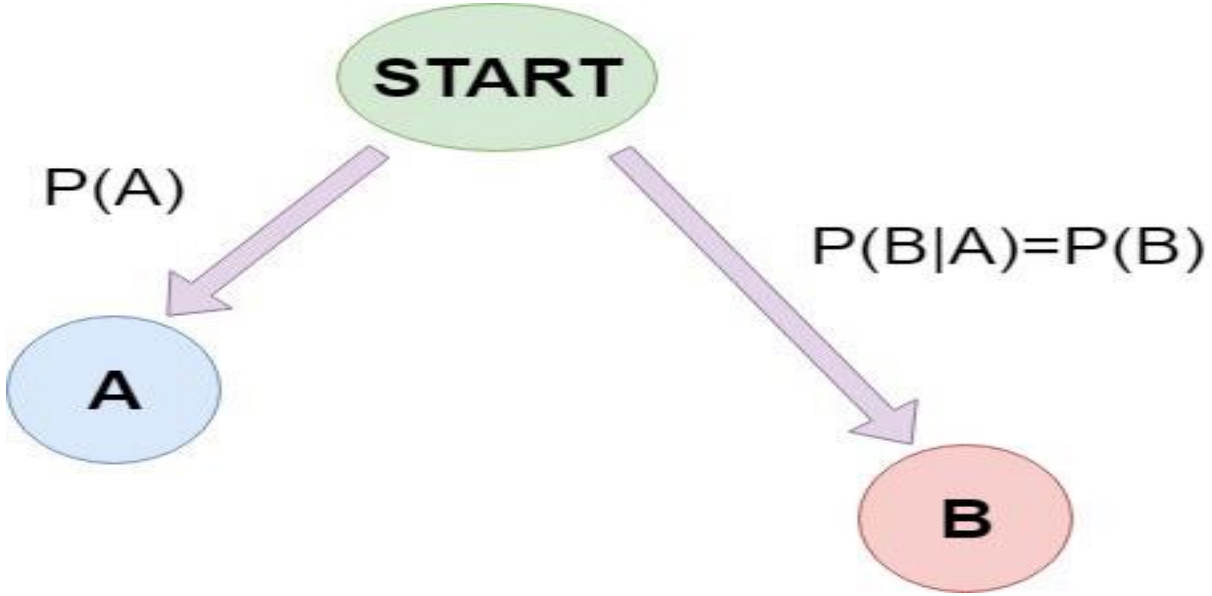
Bayes' Theorem?

- **What is Conditional Probability?**
- Since we are considering the probability for A and B both to occur, it can also be interpreted as $P(A \cap B)$.
- Hence,
- $P(A \cap B) = P(A) \cdot P(B|A)$
- Here $P(B|A)$ is known as the conditional probability and consequently, can be simplified to
- $P(B|A) = P(A \cap B)/P(A)$, Assuming that $P(A) \neq 0$



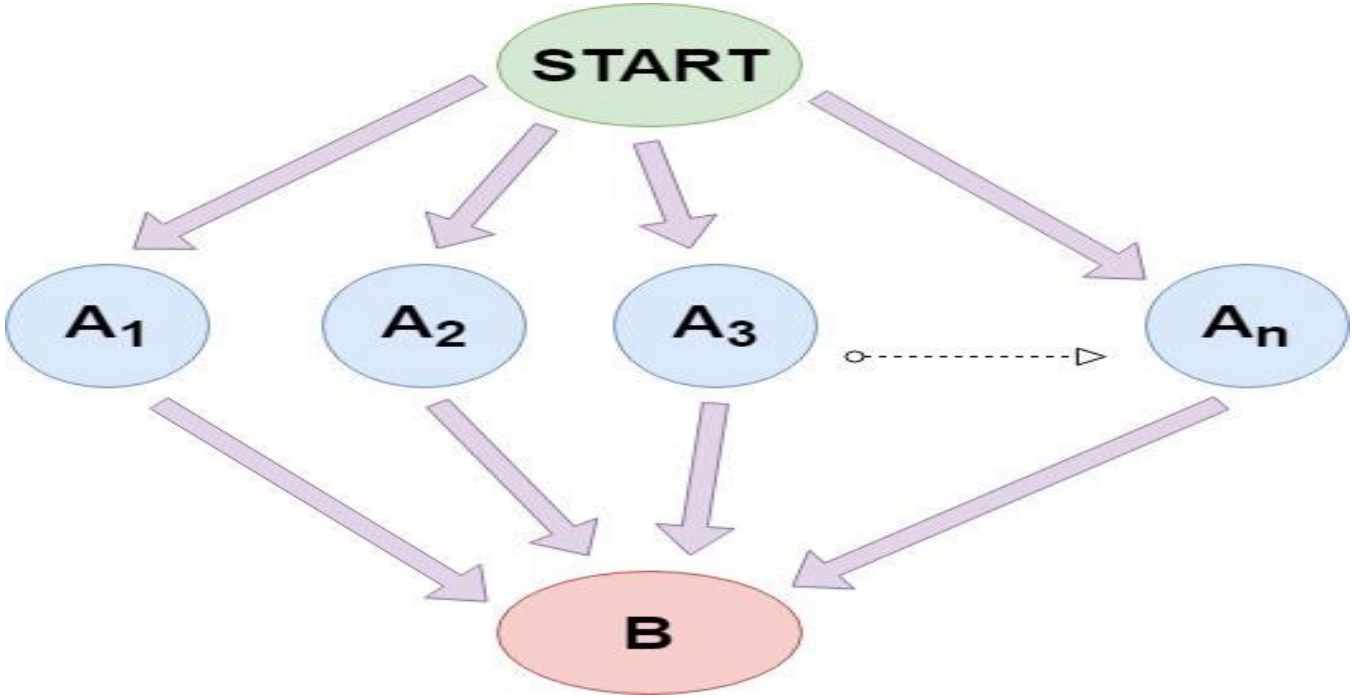
Bayes' Theorem?

- **What is Conditional Probability?**
- Note that the above case is only valid if the events occur successively and are codependent on each other.
- There can also be a possibility that A doesn't influence B, If so, then these events are independent of each other and are called Independent Events.
- In the case of Independent events, the odds of A to occur doesn't affect the odds of B to occur, therefore.
- P(B|A) = P(B)



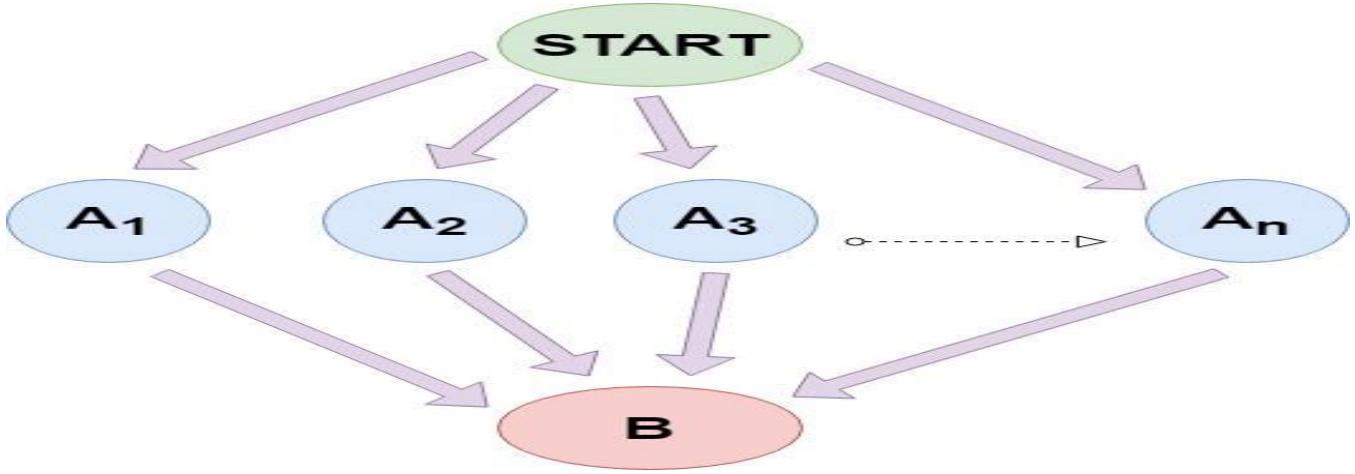
Bayes' Theorem?

- **Law of Total Probability?**
- The Law of total probability divides the calculations into different parts. It is used to find the probability of an event which is codependent on two or more events that occur prior to the former event.
- Too abstract? Let's try a visual approach



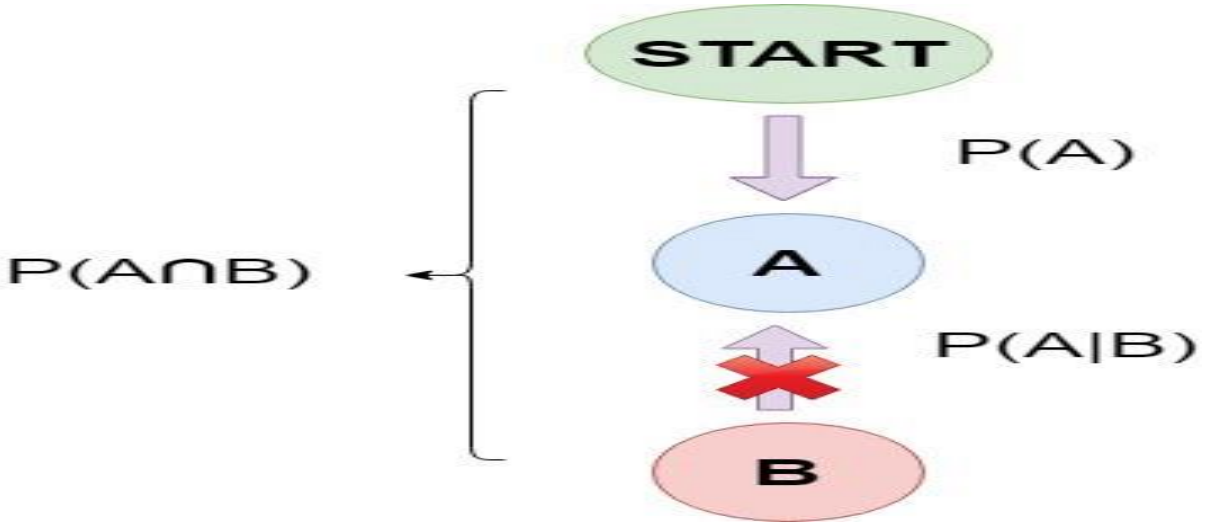
Bayes' Theorem?

- **Law of Total Probability?**
- Let B be an event which can occur after any of the "n" events(A1, A2, A3,…….. An). As defined above $P(Ai \cap B) = P(Ai) \cdot P(B|Ai)\ \forall\ i \in [1,n]$.
- Since the events A1, A2, A3, …. An are mutually exclusive and cannot occur at the same time and we can reach B either through A1 or A2 or A3 or…… or An. Therefore the rule of sum dictates.
- $P(B) = P(A1 \cap B) + P(A2 \cap B) + P(A3 \cap B) + \ldots.. + P(An \cap B)$
- $P(B) = P(A1) \cdot P(B|A1) + P(A2) \cdot P(B|A2) + \ldots.. + P(An) \cdot P(B|An)$



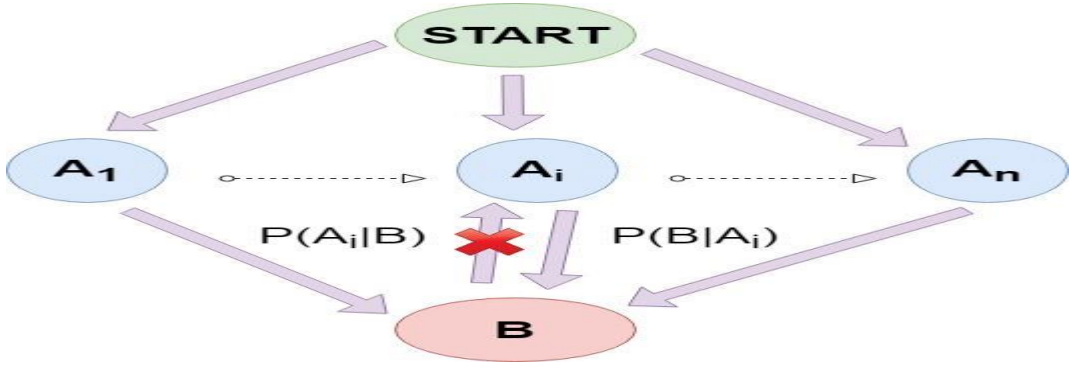
Bayes' Theorem?

- **Bayes' Theorem?**
- Bayes' Theorem is a method of predicting the origin or source based on the prior knowledge of certain probabilities.
- We already know $P(B|A) = P(A \cap B)/P(A)$, Assuming that $P(A) \neq 0$ for two codependent events.
- Ever wonder what $P(A|B) = ?$, Semantically it doesn't make any sense since B occurs after A and the timeline cannot be reversed (i.e we cannot travel upwards from B to START)



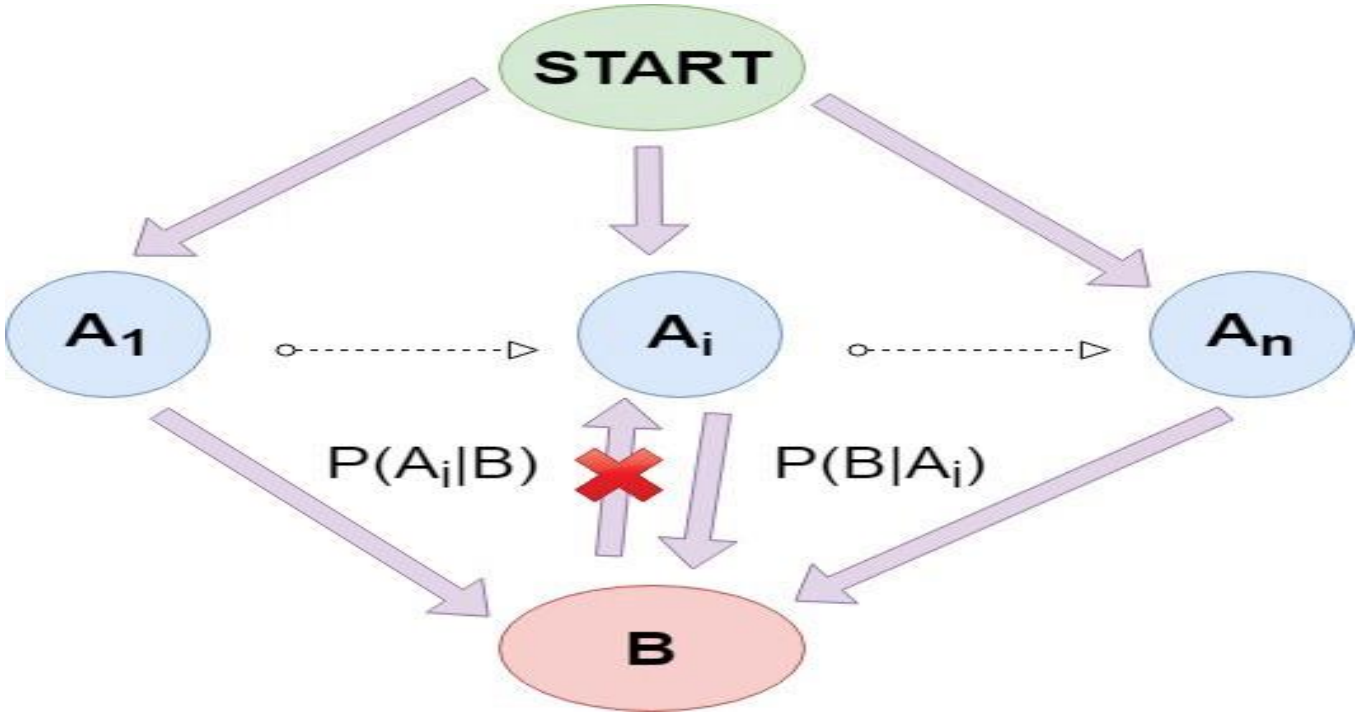
Bayes' Theorem?

- **Bayes' Theorem?**
- Mathematically we know according to the conditional probability
- $P(A|B) = P(B \cap A)/P(B)$, Assuming that $P(B) \neq 0$
- $P(A|B) = P(A \cap B)/P(B)$, as $P(A \cap B) = P(B \cap A)$
- and we know that
- $P(A \cap B) = P(B|A) \cdot P(A)$
- Substituting values we get
- **$P(A|B) = P(B|A) \cdot P(A)/P(B)$**
- This is the simplest form of Bayes' Theorem.
- Now, assume that B is codependent on multiple events that occur prior to it. Applying Total Probability Rule to the above expression we get



Bayes' Theorem?

- **Bayes' Theorem?**
- Now, assume that B is codependent on multiple events that occur prior to it. Applying Total Probability Rule to the above expression we get.
- P(Ai|B) = P(B|Ai)· P(Ai)/(P(A1)· P(B|A1) + ….. + P(An)· P(B|An))
- This is the form of Bayes' Theorem we generally use in various real-world applications.



What is Naive Bayes Classifier?

- But before you go into Naive Bayes, you need to understand what 'Conditional Probability' is and what is the 'Bayes Rule'.
- What is Conditional Probability?
- Lets start from the basics by understanding conditional probability.
- **Coin Toss and Fair Dice Example**
  - When you flip a fair coin, there is an equal chance of getting either heads or tails. So you can say the probability of getting heads is 50%.
  - Similarly what would be the probability of getting a 1 when you roll a dice with 6 faces? Assuming the dice is fair, the probability of 1/6 = 0.166.

- This is a classic example of conditional probability. So, when you say the conditional probability of A given B, it denotes the probability of A occurring given that B has already occurred.
- Mathematically, Conditional probability of A given B can be computed as:

P(A|B) = P(A AND B) / P(B)

What is Naive Bayes Classifier?

- **Let's see a another example.**
- Consider a school with a total population of 100 persons. These 100 persons can be seen either as 'Students' and 'Teachers' or as a population of 'Males' and 'Females'.

- With below tabulation of the 100 people, what is the conditional probability that a certain member of the school is a 'Teacher' given that he is a 'Male'?

- To calculate this, you may intuitively filter the sub-population of 60 males and focus on the 12 (male) teachers.
- So the required conditional probability P(Teacher | Male) = 12 / 60 = 0.2.

$$P(Teacher \setminus Male) = \frac{P(Teacher \cap Male)}{P(Male)} = 12/60 = 0.2$$

	Female	Male	Total
Teacher	8	12	20
Student	32	48	80
Total	40	60	100

What is Naive Bayes Classifier?

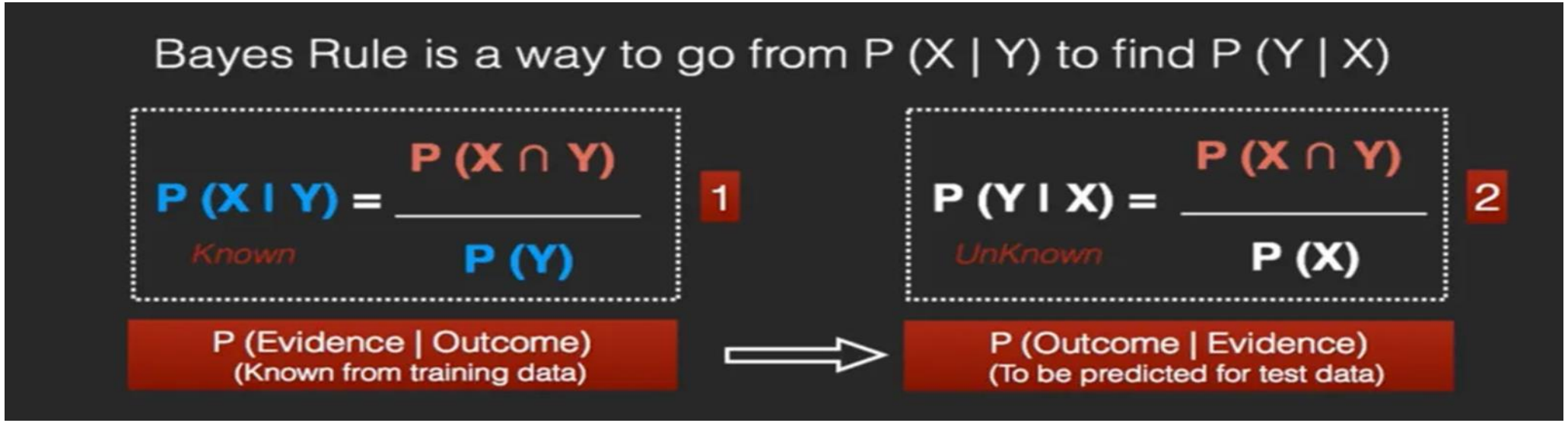
- This can be represented as the intersection of Teacher (A) and Male (B) divided by Male (B).
- Likewise, the conditional probability of B given A can be computed. The Bayes Rule that we use for Naive Bayes, can be derived from these two notations.

$$\boldsymbol{P(A \setminus B) = \frac{P(A \cap B)}{P(B)}} \qquad \textbf{(1)}$$

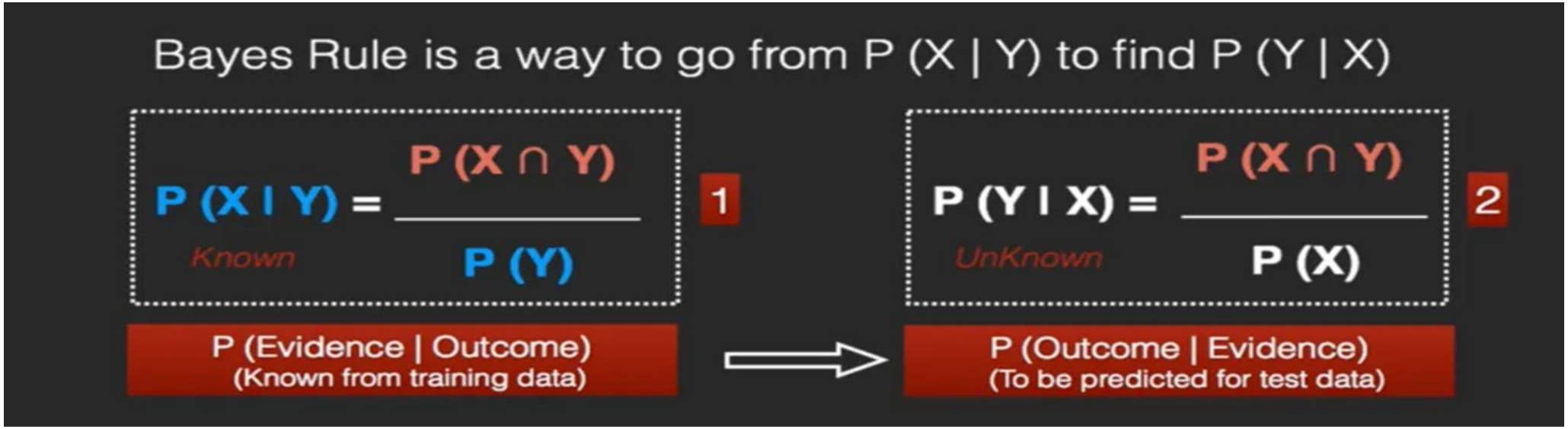
$$\boldsymbol{P(B \setminus A) = \frac{P(A \cap B)}{P(A)}} \qquad \textbf{(2)}$$

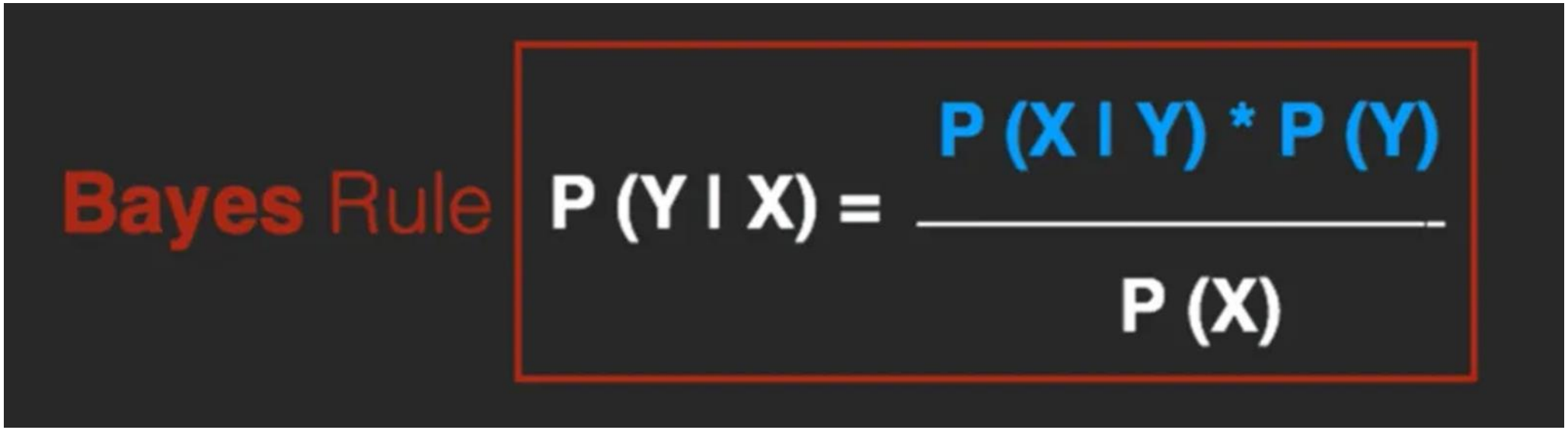
What is Naive Bayes Classifier?

- **The Bayes Rule**
- The Bayes Rule is a way of going from P(X|Y), known from the training dataset, to find P(Y|X).
- For observations in test or scoring data, the X would be known while Y is unknown. And for each row of the test dataset, you want to compute the probability of Y given the X has already happened.
- What happens if Y has more than 2 categories? we compute the probability of each class of Y and let the highest win.



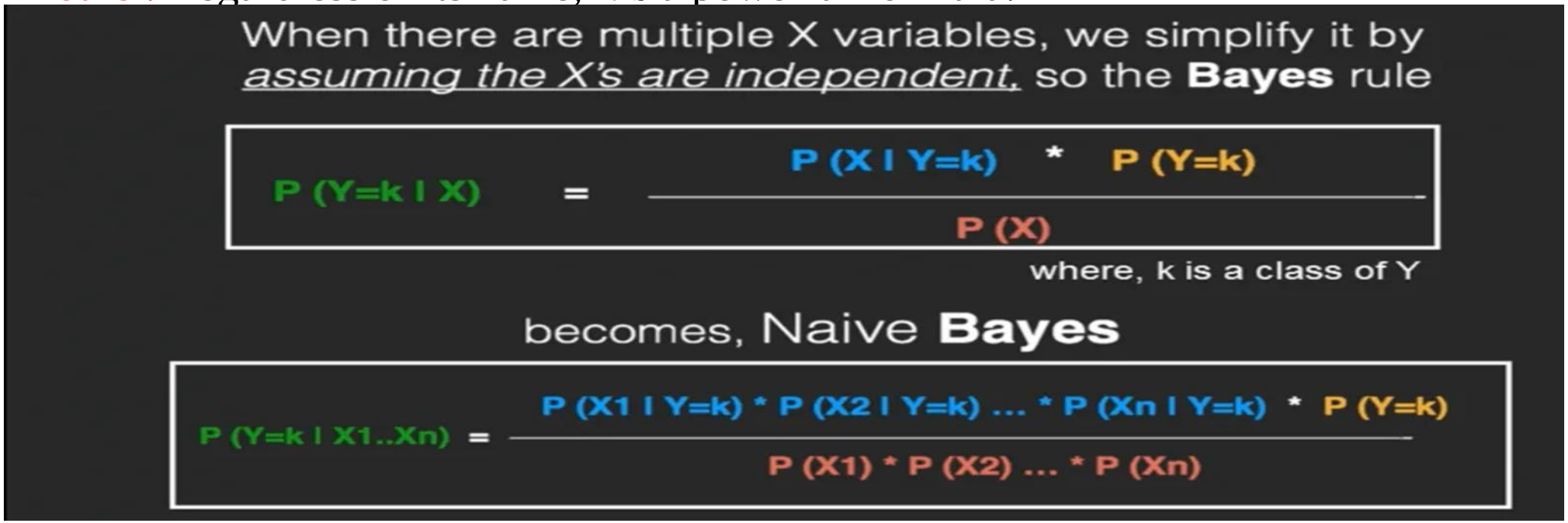
What is Naive Bayes Classifier?



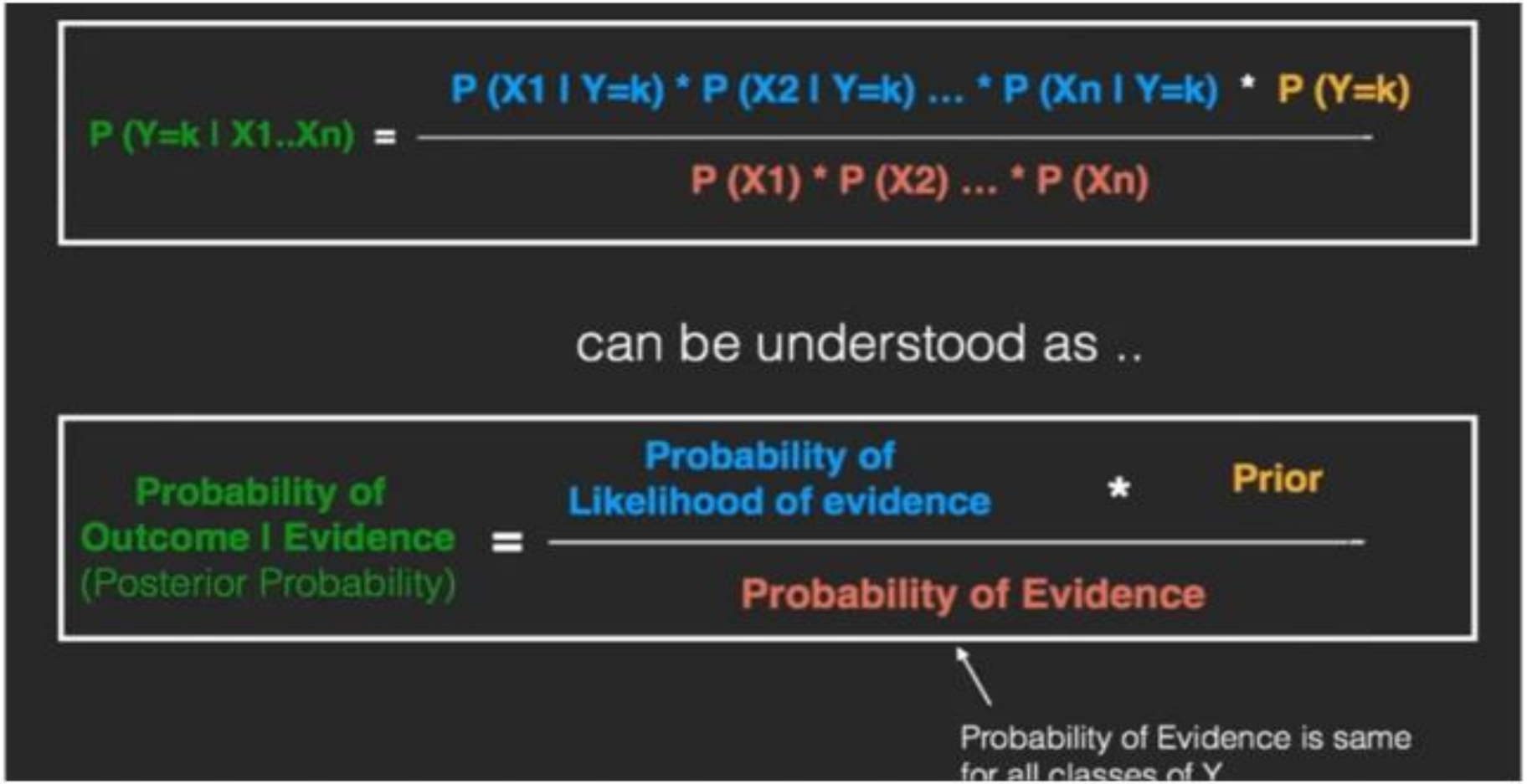


Naive Bayes Classifier?

- **The Naive Bayes**
- The Bayes Rule provides the formula for the probability of Y given X. But, in real-world problems, you typically have multiple X variables.
- When the features are independent, we can extend the Bayes Rule to what is called Naive Bayes.
- It is called 'Naive' because of the naive assumption that the X's are independent of each other. Regardless of its name, it's a powerful formula.



Naive Bayes Classifier?



Naive Bayes Classifier?

- In technical, the left-hand-side (LHS) of the equation is understood as the posterior probability or simply the posterior
- The RHS has 2 terms in the numerator.
- The first term is called the 'Likelihood of Evidence'. It is nothing but the conditional probability of each X's given Y is of particular class 'c'.
- Since all the X's are assumed to be independent of each other, you can just multiply the 'likelihoods' of all the X's and called it the 'Probability of likelihood of evidence'. This is known from the training dataset by filtering records where Y=c.
- The second term is called the prior which is the overall probability of Y=c, where c is a class of Y. In simpler terms, Prior = count(Y=c) / n_Records.

Naive Bayes Classifier?

- **Naive Bayes Example by Hand**
- Say you have 1000 fruits which could be either ‘banana’, ‘orange’ or ‘other’. These are the 3 possible classes of the Y variable.

- We have data for the following X variables, all of which are binary (1 or 0).
- **Long**
- **Sweet**
- **Yellow**
- The first few rows of the training dataset look like this:

Fruit	Long (x1)	Sweet (x2)	Yellow (x3)
Orange	0	1	0
Banana	1	0	1
Banana	1	1	1
Other	1	1	0
..	..	..	..

Naive Bayes Classifier?

- For the sake of computing the probabilities, let's aggregate the training data to form a counts table like this.

Type	Long	Not Long	Sweet	Not Sweet	Yellow	Not Yellow	Total
Banana	400	100	350	150	450	50	500
Orange	0	300	150	150	300	0	300
Other	100	100	150	50	50	150	200
Total	500	500	650	350	800	200	1000

- So the objective of the classifier is to predict if a given fruit is a 'Banana' or 'Orange' or 'Other' when only the 3 features (long, sweet and yellow) are known.
- Let's say you are given a fruit that is: Long, Sweet and Yellow, can you predict what fruit it is?

Naive Bayes Classifier?

- This is the same of predicting the Y when only the X variables in testing data are known. **Let's solve it by hand using Naive Bayes.**
- The idea is to compute the 3 probabilities, that is the probability of the fruit being a banana, orange or other.
- Whichever fruit type gets the highest probability wins.
- All the information to calculate these probabilities is present in the above tabulation.
- **Step 1: Compute the 'Prior' probabilities for each of the class of fruits.**
- That is, the proportion of each fruit class out of all the fruits from the population. You can provide the 'Priors' from prior information about the population. Otherwise, it can be computed from the training data.
- For this case, let's compute from the training data. Out of 1000 records in training data, you have 500 Bananas, 300 Oranges and 200 Others. So the respective priors are 0.5, 0.3 and 0.2.

P(Y=Banana) = 500 / 1000 = 0.50

P(Y=Orange) = 300 / 1000 = 0.30

P(Y=Other) = 200 / 1000 = 0.20

Naive Bayes Classifier?

- **Step 2: Compute the probability of evidence that goes in the denominator.**

- This is nothing but the product of P of Xs for all X. This is an optional step because the denominator is the same for all the classes and so will not affect the probabilities.

- P(x1=Long) = 500 / 1000 = 0.50

- P(x2=Sweet) = 650 / 1000 = 0.65

- P(x3=Yellow) = 800 / 1000 = 0.80

Naive Bayes Classifier?

- **Step 3: Compute the probability of likelihood of evidences that goes in the numerator.**
- It is the product of conditional probabilities of the 3 features. If you refer back to the formula, it says P(X1 |Y=k). Here X1 is 'Long' and k is 'Banana'.
- That means the probability the fruit is 'Long' given that it is a Banana. In the above table, you have 500 Bananas. Out of that 400 is long. So, P(Long | Banana) = 400/500 = 0.8.
- Here, I have done it for Banana alone.
- **Probability of Likelihood for Banana**
  - P(x1=Long | Y=Banana) = 400 / 500 = 0.80
  - P(x2=Sweet | Y=Banana) = 350 / 500 = 0.70
  - P(x3=Yellow | Y=Banana) = 450 / 500 = 0.90
- So, the overall probability of Likelihood of evidence for Banana = 0.8 * 0.7 * 0.9 = 0.504

Naive Bayes Classifier?

- **Step 4: Substitute all the 3 equations into the Naive Bayes formula, to get the probability that it is a banana.**

Step 4: If a fruit is *'Long', 'Sweet' and 'Yellow'*, what fruit is it?

$$P(\text{Banana} \mid \textit{Long, Sweet and Yellow}) = \frac{P(\text{Long} \mid \text{Banana}) * P(\text{Sweet} \mid \text{Banana}) * P(\text{Yellow} \mid \text{Banana}) \times P(\text{banana})}{P(\text{Long}) * P(\text{Sweet}) * P(\text{Yellow})}$$

$$= \frac{0.8 * 0.7 * 0.9 * 0.5}{P(\text{Evidence})} = 0.252/P(\text{Evidence})$$

$P(\text{Orange} \mid \textit{Long, Sweet and Yellow}) = 0$, because $P(\text{Long} \mid \text{Orange}) = 0$

$P(\text{Other Fruit} \mid \textit{Long, Sweet and Yellow}) = 0.01875 / P(\text{Evidence})$

Answer: Banana - Since it has highest probability amongst the 3 classes

Naive Bayes Classifier?

- Similarly, you can compute the probabilities for ‘Orange’ and ‘Other fruit’. The denominator is the same for all 3 cases, so it’s optional to compute.

- Clearly, Banana gets the highest probability, so that will be our predicted class.
- **What is Laplace Correction?**
- The value of P(Orange | Long, Sweet and Yellow) was zero in the above example, because, P(Long | Orange) was zero. That is, there were no ‘Long’ oranges in the training data.
- It makes sense, but when you have a model with many features, the entire probability will become zero because one of the feature’s value was zero. To avoid this, we increase the count of the variable with zero to a small value (usually 1) in the numerator, so that the overall probability doesn’t become zero.
- **This correction is called ‘Laplace Correction’. Most Naive Bayes model implementations accept this or an equivalent form of correction as a parameter.**

Naive Bayes Classifier?

- **What is Gaussian Naive Bayes?**
- So far we've seen the computations when the X's are categorical. But how to compute the probabilities when X is a continuous variable?

- If we assume that the X follows a particular distribution, then you can plug in the probability density function of that distribution to compute the probability of likelihoods.

- If you assume the X's follow a Normal (aka Gaussian) Distribution, which is fairly common, we substitute the corresponding probability density of a Normal distribution and call it the Gaussian Naive Bayes. You need just the mean and variance of the X to compute this formula. where mu and sigma are the mean and variance of the continuous X computed for a given class 'c' (of Y).
- That's it. Now, let's build a Naive Bayes classifier

$$P(X|Y=c) = \frac{1}{\sqrt{2\pi\sigma_c^2}} e^{\frac{-(x-\mu_c)^2}{2\sigma_c^2}}$$

What is Naive Bayes Classifier?

- **Bayesian Classification**
- Naive Bayes classifiers are built on Bayesian classification methods. These rely on Bayes's theorem, which is an equation describing the relationship of conditional probabilities of statistical quantities.
- In Bayesian classification, we're interested in finding the probability of a label given some observed features, which we can write as **P(L(label) | features).**
- **Bayes's theorem** tells us how to express this in terms of quantities we can compute more directly:

$$P(L \mid features) = P(features \mid L)P(L)P(features)$$

What is Naive Bayes Classifier?

- Bayes's theorem tells us how to express this in terms of quantities we can compute more directly:

$$P(L \mid features) = P(features \mid L)P(L)P(features)$$

- If we are trying to decide between two labels—let's call them L1(label 1) and L2(label2)—then one way to make this decision is to compute the ratio of the posterior probabilities for each label:

$$P(L1 \mid features)P(L2 \mid features) = P(features \mid L1)P(features \mid L2)P(L1)P(L2)$$

- All we need now is some model by which we can compute P(features | Li) for each label. Such a model is called a generative model because it specifies the hypothetical random process that generates the data.
- Specifying this generative model for each label is the main piece of the training of such a Bayesian classifier.
- The general version of such a training step is a very difficult task, but we can make it simpler through the use of some **simplifying assumptions about the form of this model**.

Gaussian Naive Bayes Classifier

Gaussian Naive Bayes:

- Gaussian Naive Bayes is useful when working with continuous values which probabilities can be modeled using a Gaussian distribution:

$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- The conditional probabilities P(xi|y) are also Gaussian distributed and, therefore, it's necessary to estimate mean and variance of each of them using the maximum likelihood approach.

$$L(\mu; \sigma^2; x_i|y) = log \prod_k P(x_i^{(k)}|y) = \sum_k log\ P(x_i^{(k)}|y)$$

Gaussian Naive Bayes Classifier

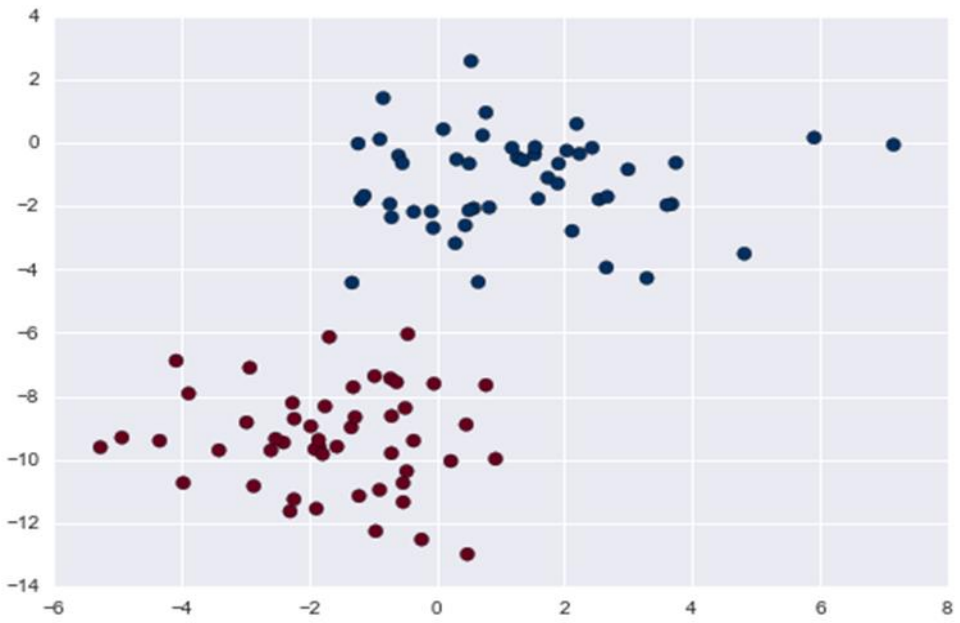
Gaussian Naive Bayes:

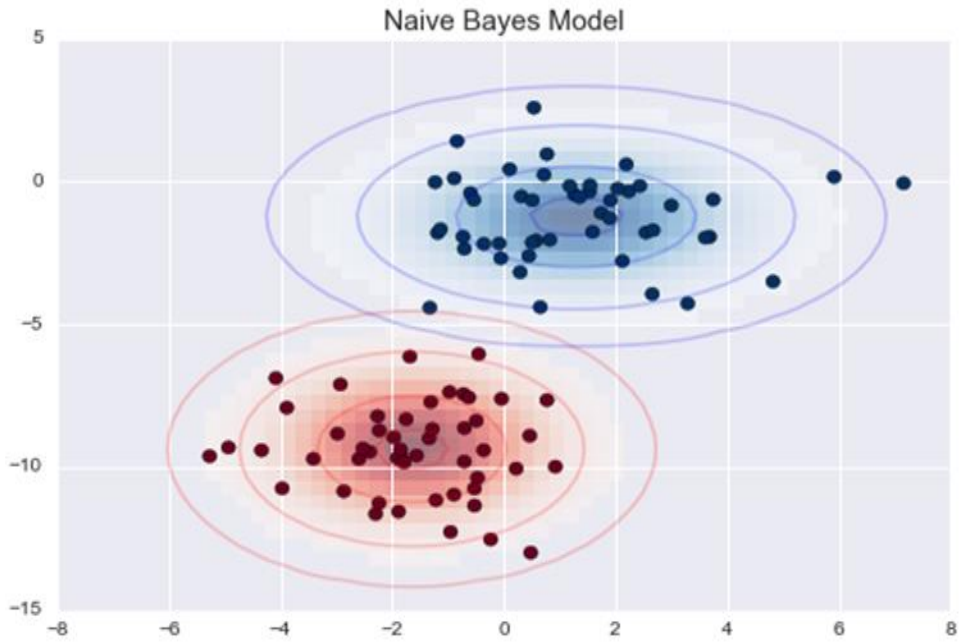
- In scikit-learn, here is simple code

```
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25)
gnb = GaussianNB()
gnb.fit(X_train, Y_train)
Y_gnb_score = gnb.predict_proba(X_test)
```

Gaussian Naive Bayes Classifier Example

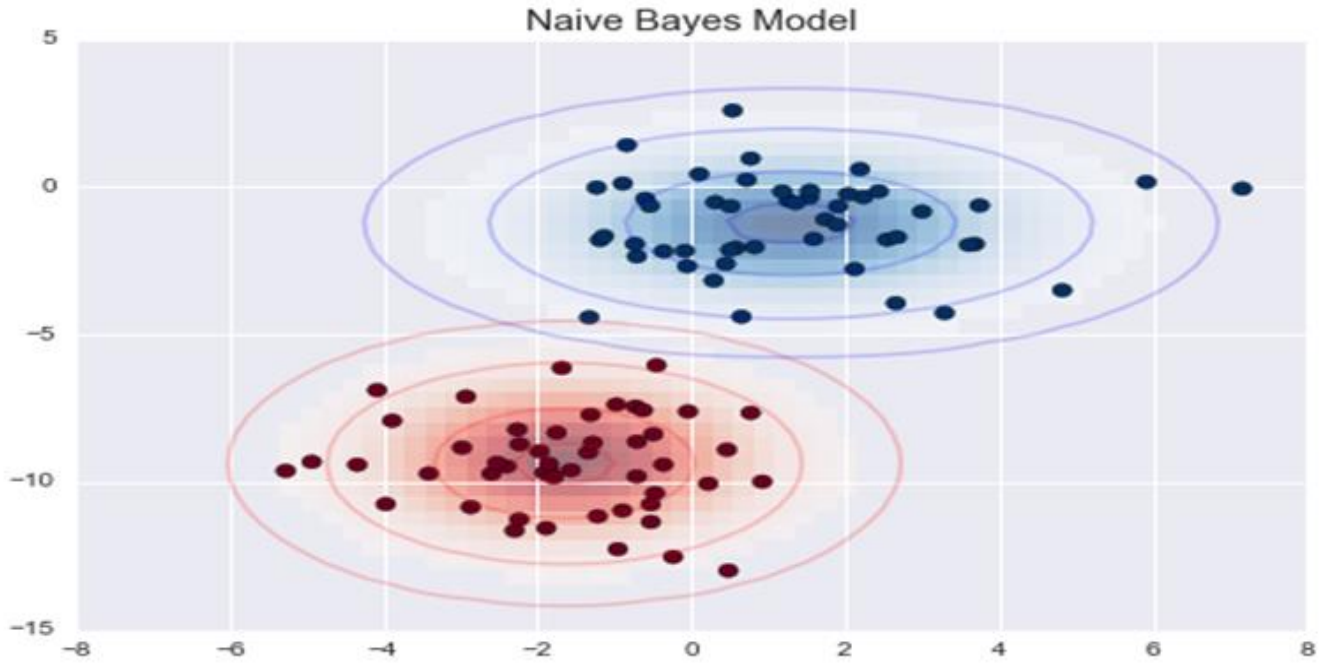
- Imagine that you have the following data
- One extremely fast way to create a simple model is to assume that the data is described by a Gaussian distribution with no covariance between dimensions.
- This model can be fit by simply finding the mean and standard deviation of the points within each label, which is all you need to define such a distribution.
- The result of this naive Gaussian assumption is shown in the following figure:





Gaussian Naive Bayes Classifier Example

- The ellipses here represent the Gaussian generative model for each label, with larger probability toward the center of the ellipses.
- With this generative model in place for each class, we have a simple recipe to compute the likelihood P(features | L1) for any data point, and thus we can quickly compute the posterior ratio and determine which label is the most probable for a given point.



Multinomial Naive Bayes Classifier

- **Multinomial naive Bayes**
- A multinomial distribution is useful to model feature vectors where each value represents, for example, the number of occurrences of a term or its relative frequency.
- If the feature vectors have n elements and each of them can assume k different values with probability pk, then:

$$P(X_1 = x_1 \cap X_2 = x_2 \cap \ldots \cap X_k = x_k) = \frac{n!}{\prod_i x_i!} \prod_i {p_i}^{x_i}$$

- The conditional probabilities P(xi|y) are computed with a frequency count (which corresponds to applying a maximum likelihood approach).

Multinomial Naive Bayes Classifier

- **Multinomial Naive Bayes**
- The Gaussian assumption just described is by no means the only simple assumption that could be used to specify the generative distribution for each label.

- Another useful example is multinomial naive Bayes, where the features are assumed to be generated from a simple multinomial distribution.

- The multinomial distribution describes the probability of observing counts among a number of categories, and thus multinomial naive Bayes is most appropriate for features that represent counts or count rates.

- The idea is precisely the same as before, except that instead of modeling the data distribution with the best-fit Gaussian, we model the data distribution with a best-fit multinomial distribution.

Multinomial Naive Bayes Classifier example

- **Example: Classifying Text**
- One place where multinomial naive Bayes is often used is in text classification, where the features are related to word counts or frequencies within the documents to be classified.

- Remember that this is nothing more sophisticated than a simple probability model for the (weighted) frequency of each word in the string; nevertheless, the result is striking.

- Even a very naive algorithm, when used carefully and trained on a large set of high-dimensional data, can be surprisingly effective.

Multinomial Naive Bayes Classifier

- Multinomial naive Bayes
- In scikit-learn code is written as

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25)
mnb = MultinomialNB()
mnb.fit(X_train, Y_train)
Y_gnb_score = mnb.predict_proba(X_test)
```

Bernoulli Naive Bayes Classifier

- If X is random variable Bernoulli-distributed, it can assume only two values (for simplicity, let's call them 0 and 1) and their probability is:

$$P(X) = \begin{cases} p & if\ X = 1 \\ q & if\ X = 0 \end{cases}$$

$$where\ q = 1 - p\ and\ 0 < p < 1$$

- Bernoulli naive Bayes expects binary feature vectors, however, the class BernoulliNB has a binarize parameter which allows specifying a threshold that will be used internally to transform the features.

Bernoulli Naive Bayes Classifier

- Bernoulli naive Bayes expects binary feature vectors, however, the class BernoulliNB has a binarize parameter which allows specifying a threshold that will be used internally to transform the features.

```
from sklearn.naive_bayes import BernoulliNB
from sklearn.model_selection import
train_test_split
X_train, X_test, Y_train, Y_test =
train_test_split(X, Y, test_size=0.25)
bnb = BernoulliNB(binarize=0.0)
bnb.fit(X_train, Y_train) >>> bnb.score(X_test,
Y_test)
```

Naive Bayes Classifier from scratch

Naïve Bayes

$$P(A\backslash \mathrm{B}) = \frac{P(B\backslash \mathrm{A}).P(A)}{P(B)}$$

In our case

$$P(y\backslash \mathrm{X}) = \frac{P(X\backslash \mathrm{y}).P(y)}{P(X)}$$

In our case with feature vector X $\mathrm{X}=((x_1, x_2, x_3, \ldots., x_n))$

Assume that all features are mutually independent

$$P(y\backslash \mathrm{X}) = \frac{P(x_1\backslash \mathrm{y})P(x_2\backslash \mathrm{y}) \ldots. P(x_n\backslash \mathrm{y}).P(y)}{P(X)}$$

Naive Bayes Classifier from scratch

Select class with highest probability

$$y = argmax_y \frac{P(x_1 \backslash y)P(x_2 \backslash y) \ldots . P(x_n \backslash y).P(y)}{P(X)}$$

$$y = argmax_y P(x_1 \backslash y)P(x_2 \backslash y) \ldots . P(x_n \backslash y).P(y)$$

$$y = argmax_y \log\big(P(x_1 \backslash y)\big) + \log(P(x_2 \backslash y)) + \cdots + \log\big(P(x_n \backslash y)\big) + \log(P(y))$$

Class conditional probability $P(x_i \backslash y)$

prior probability $P(y)$: frequency

Naive Bayes Classifier from scratch

prior probability $P(y)$: frequency

Class conditional probability $P(x_i \backslash y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} . \exp\left(-\frac{(xi-\mu y)^2}{2\sigma_y^2}\right)$

Naive Bayes Classifier from scratch

```
import numpy as np

class NaiveBayes:

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self._classes = np.unique(y)
        n_classes = len(self._classes)

        # calculate mean, var, and prior for each class
        self._mean = np.zeros((n_classes, n_features), dtype=np.float64)
        self._var = np.zeros((n_classes, n_features), dtype=np.float64)
        self._priors =  np.zeros(n_classes, dtype=np.float64)

        for c in self._classes:
            X_c = X[y==c]
            self._mean[c, :] = X_c.mean(axis=0)
            self._var[c, :] = X_c.var(axis=0)
            self._priors[c] = X_c.shape[0] / float(n_samples)

    def predict(self, X):
        y_pred = [self._predict(x) for x in X]
        return np.array(y_pred)
```

```
def _predict(self, x):
        posteriors = []

        # calculate posterior probability for each class
        for idx, c in enumerate(self._classes):
            prior = np.log(self._priors[idx])
            posterior = np.sum(np.log(self._pdf(idx, x)))
            posterior = prior + posterior
            posteriors.append(posterior)

        # return class with highest posterior probability
        return self._classes[np.argmax(posteriors)]
```

```
def _pdf(self, class_idx, x):
        mean = self._mean[class_idx]
        var = self._var[class_idx]
        numerator = np.exp(- (x-mean)**2 / (2 * var))
        denominator = np.sqrt(2 * np.pi * var)
        return numerator / denominator
```

$$\text{y} = argmax_y \log\left(P(x_1 \backslash \text{y})\right) + \log(P(x_2 \backslash \text{y})) + \cdots + \log\left(P(x_n \backslash \text{y})\right) + \log(P(y))$$

Naive Bayes Classifier from scratch

Testing Function for Naive Bayes

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import datasets
import matplotlib.pyplot as plt

from naivebayes import NaiveBayes



def accuracy(y_true, y_pred):
    accuracy = np.sum(y_true == y_pred) / len(y_true)
    return accuracy

X, y = datasets.make_classification(n_samples=1000, n_features=10, n_classes=2, random_state=123)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123)

nb = NaiveBayes()
nb.fit(X_train, y_train)
predictions = nb.predict(X_test)

print("Naive Bayes classification accuracy", accuracy(y_test, predictions))
```

Naive Bayes Classifier pros and cons

- **Because naive Bayesian classifiers make such stringent assumptions about data, they will generally not perform as well as a more complicated model. That said, they have several advantages:**
- They are extremely fast for both training and prediction
- They provide straightforward probabilistic prediction
- They are often very easily interpretable
- They have very few (if any) tunable parameters
- These advantages mean a naive Bayesian classifier is often a good choice as an initial baseline classification.
- If it performs suitably, then congratulations: you have a very fast, very interpretable classifier for your problem. If it does not perform well, then you can begin exploring more sophisticated models, with some baseline knowledge of how well they should perform.

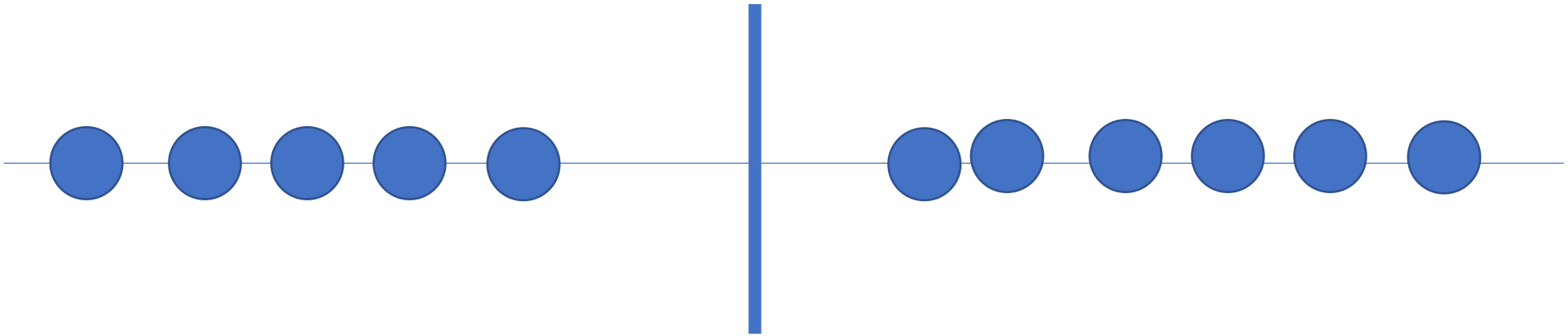
Naive Bayes Classifier pros and cons

- When the naive assumptions actually match the data (very rare in practice)
- For very well-separated categories, when model complexity is less important
- For very high-dimensional data, when model complexity is less important
- The last two points seem distinct, but they actually are related: as the dimension of a dataset grows, it is much less likely for any two points to be found close together (after all, they must be close in every single dimension to be close overall).
- This means that clusters in high dimensions tend to be more separated, on average, than clusters in low dimensions, assuming the new dimensions actually add information. For this reason, simplistic classifiers like naive Bayes tend to work as well or better than more complicated classifiers as the dimensionality grows: once you have enough data, even a simple model can be very powerful.

Support Vector Machine

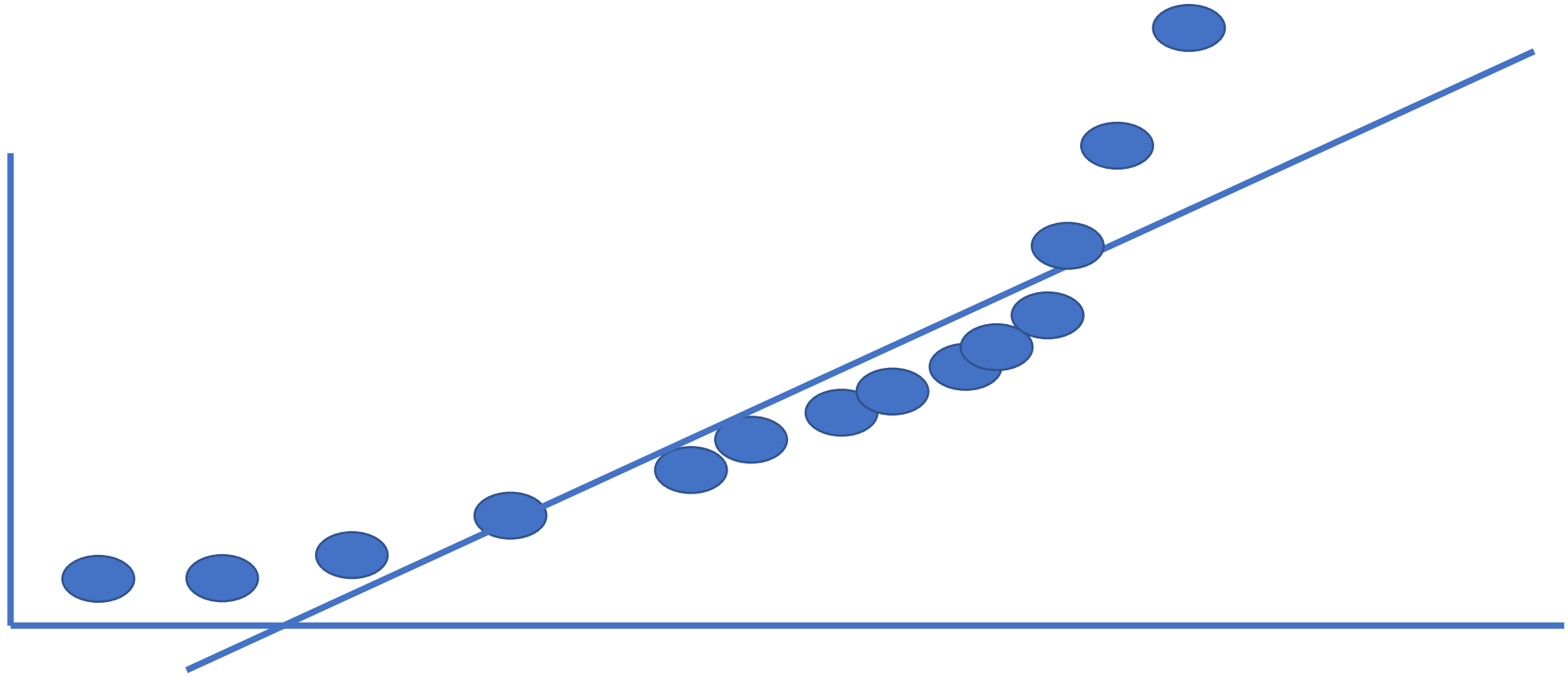
Support Vector Machine (SVM)

When the data are 1-dimensional, the support vector classifier is a single point on a 1-Dimensional number line.
A point is a **flat affine 0-dimensional subspace**



Support Vector Machine (SVM)

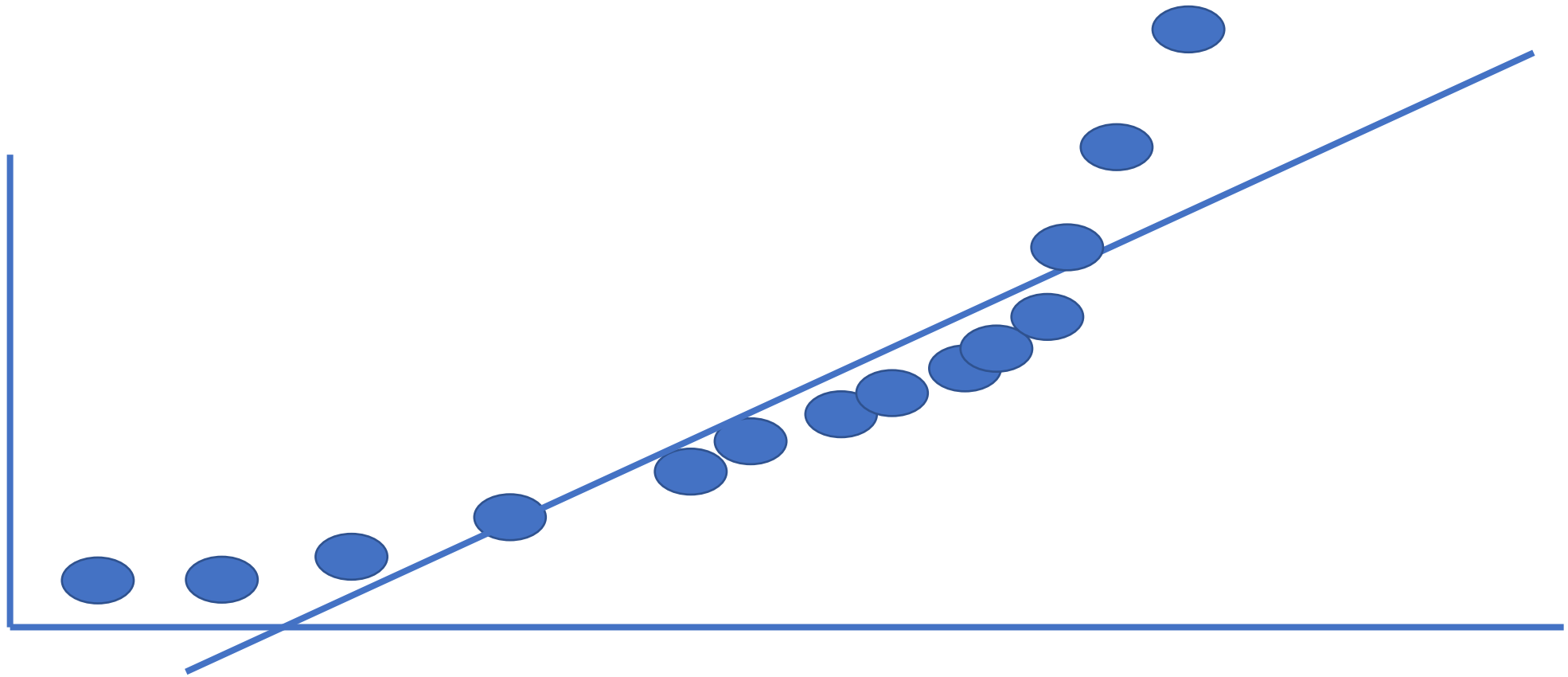
When the data are 2-dimensional, the support vector classifier is a 1-Dimensional number line in a 2-dimensional space. A line is a **flat affine 1-dimensional subspace. Technically speaking, this 1-dimensional line is a hyperplane**



Support Vector Machine (SVM)

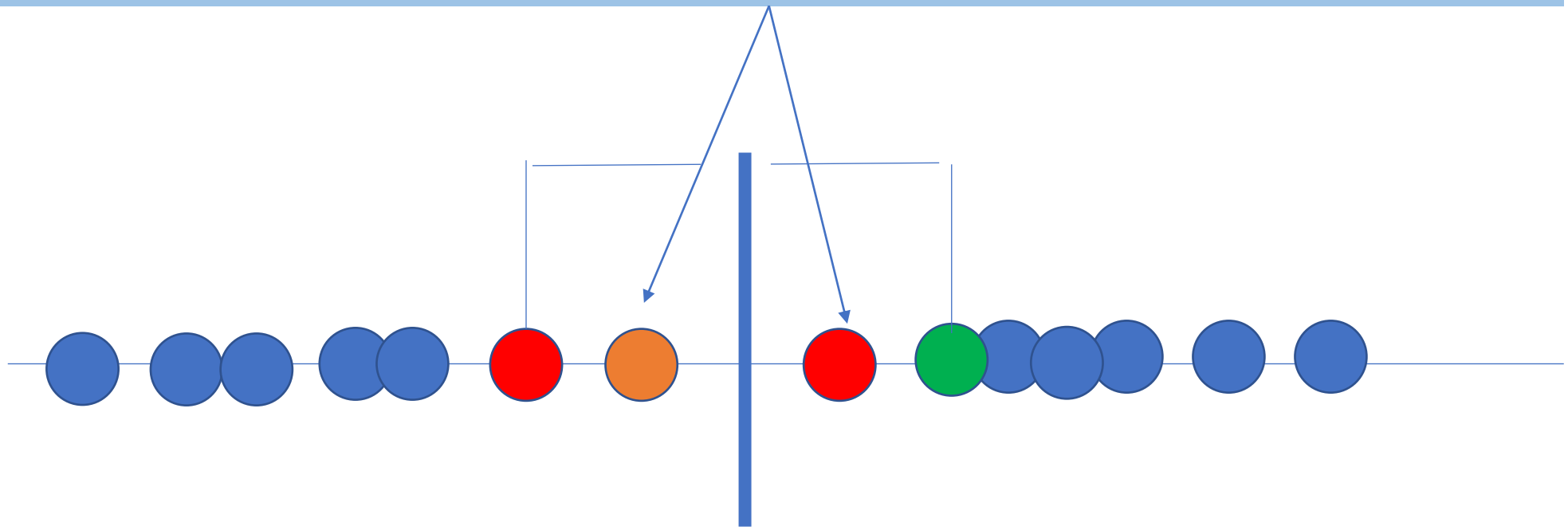
When the data are 3-dimensional, the support vector classifier is in a 2-dimensional plane in a 3-dimensional space

All flat affine subspaces are called hyperplanes

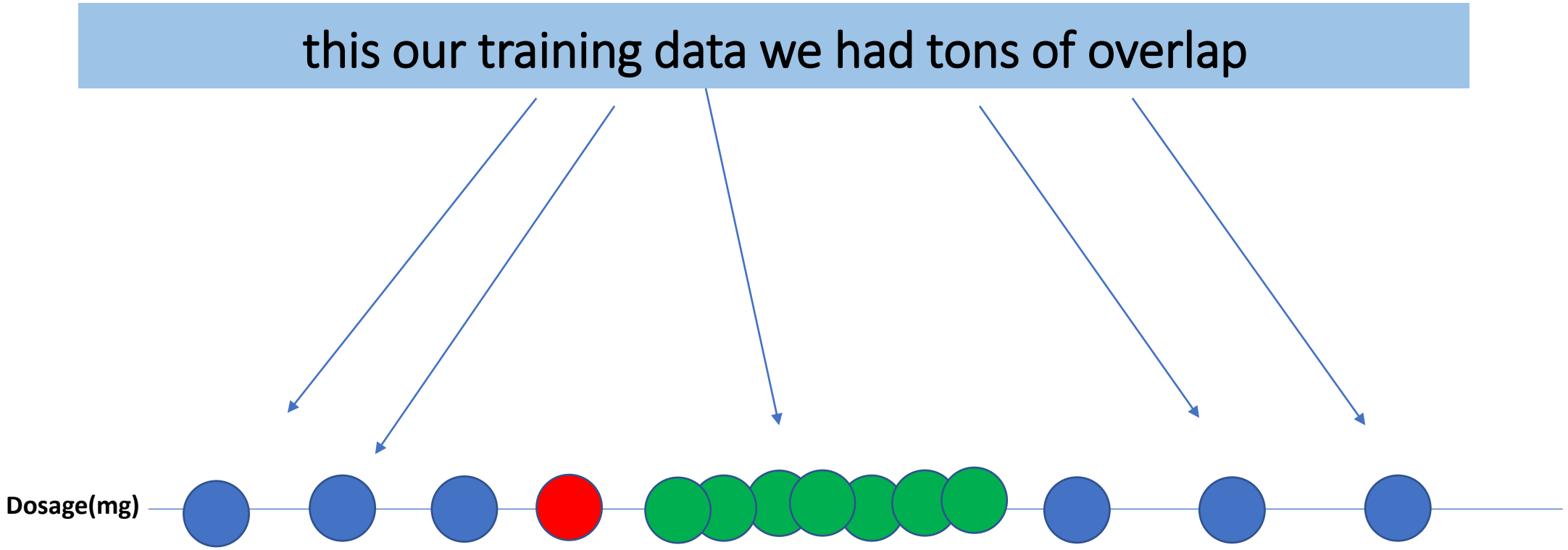


Support Vector Machine (SVM)

Support vector classifiers seem pretty cool because they can handle outlies and because they misclassifications, they can handle overlapping classification

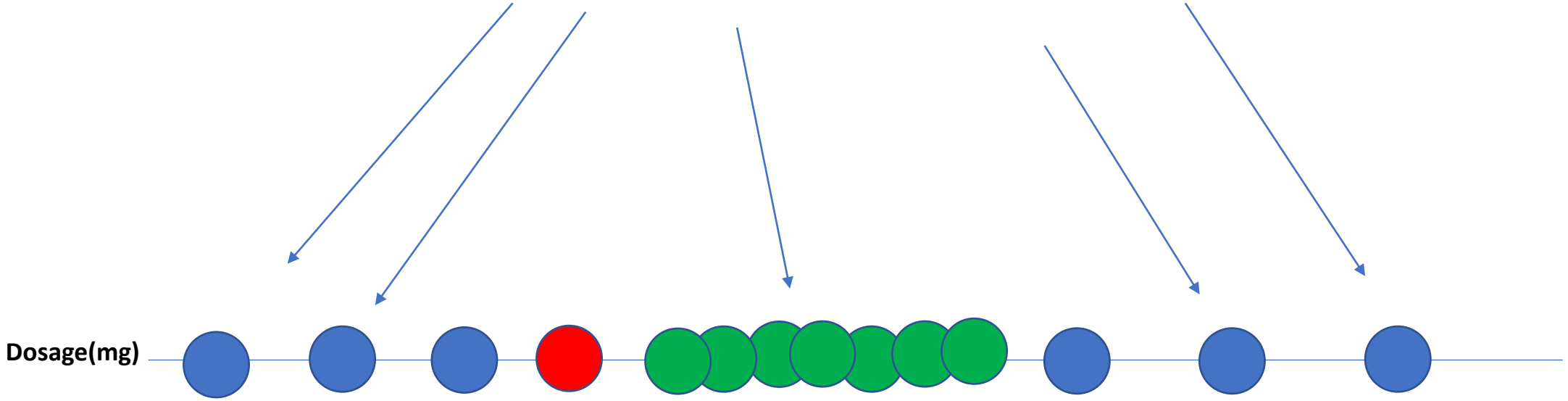


Support Vector Machine (SVM)



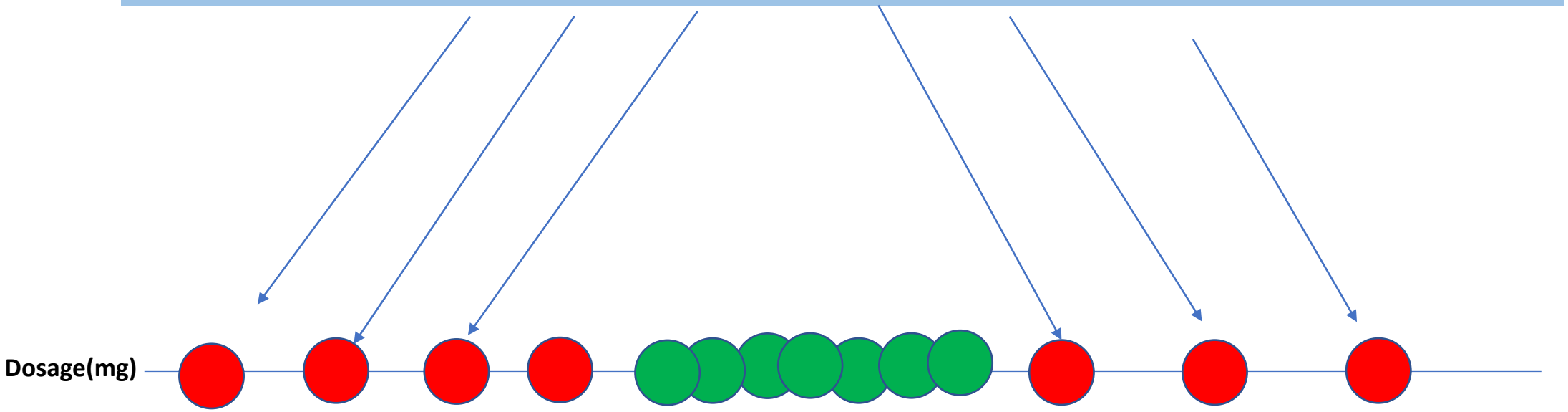
Support Vector Machine (SVM)

This our training data we are looking new drug dosages and red represent patients that were not curved and blue dots represents the patient were curved



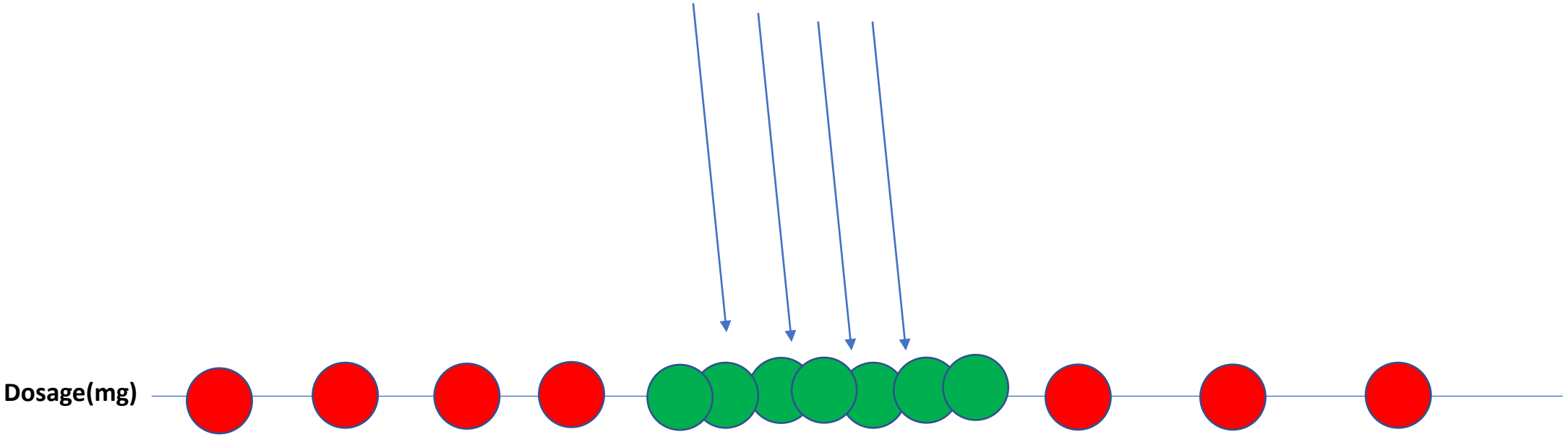
Support Vector Machine (SVM)

In other words, the drug does not work if the dosage is too small or too large



Support Vector Machine (SVM)

It only works when the dosage is just right

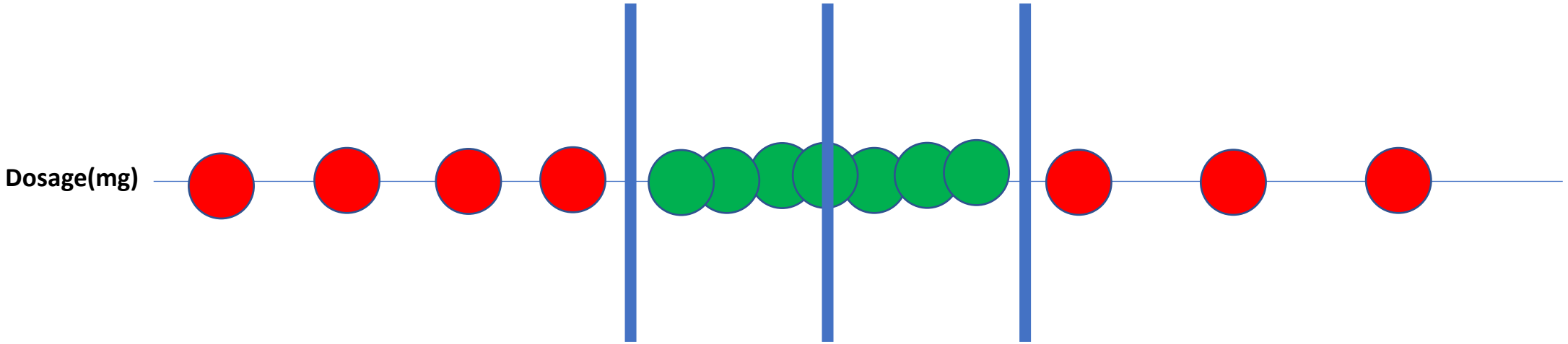


Support Vector Machine (SVM)

Now, no matter where we put the classifier, we will make a lot of misclassifications and so support vector classifiers are only semi cool since they don't perform well this type of data.

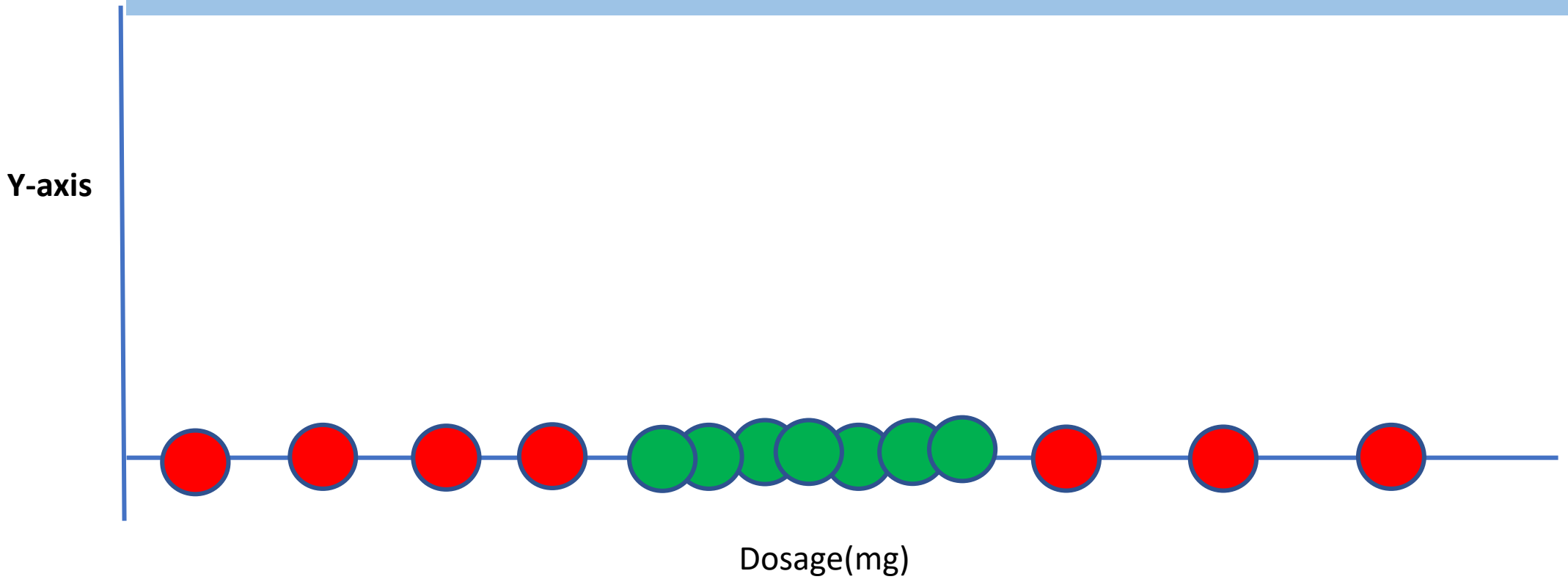
Can we do better than maximal margin classifiers and support vector classifiers?

Since maximal margin classifiers and support vector classifiers cant handle this data its high time we talked about?



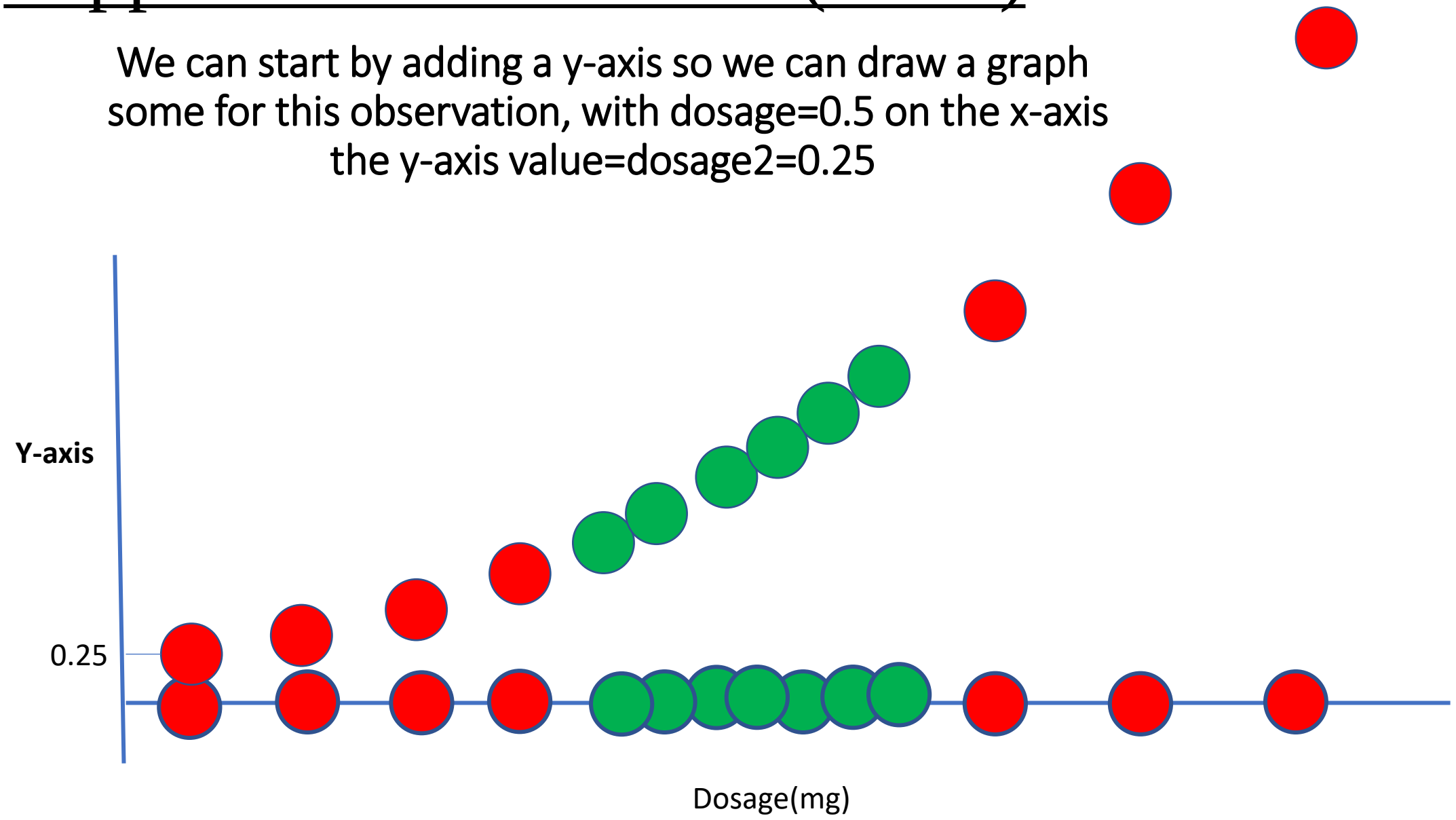
Support Vector Machine (SVM)

We can start by adding a y-axis so we can draw a graph and the x-axis coordinated in this graph will be the dosages that we have already observed and y-axis will be the square of the dosages



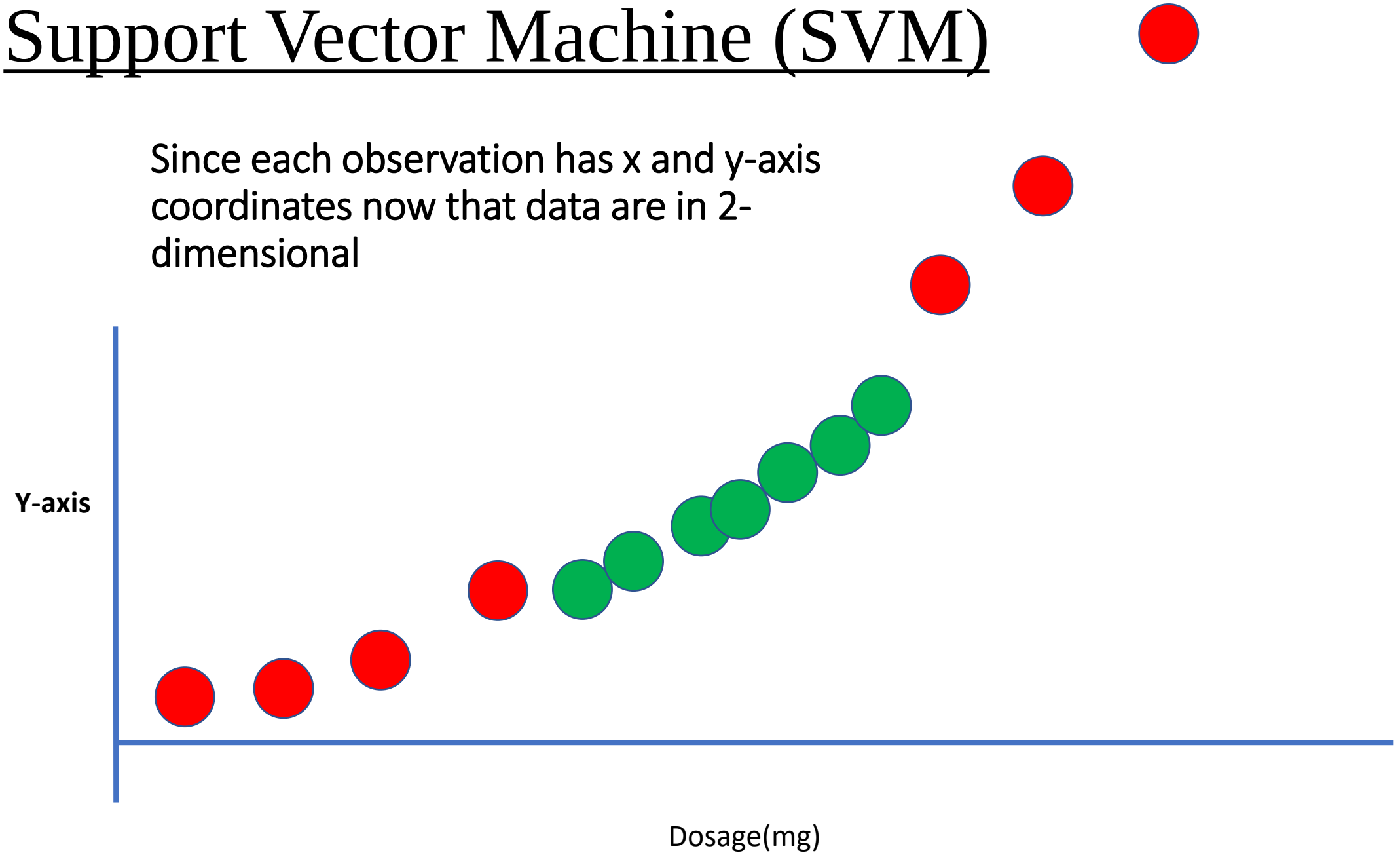
Support Vector Machine (SVM)

We can start by adding a y-axis so we can draw a graph some for this observation, with dosage=0.5 on the x-axis the y-axis value=dosage2=0.25



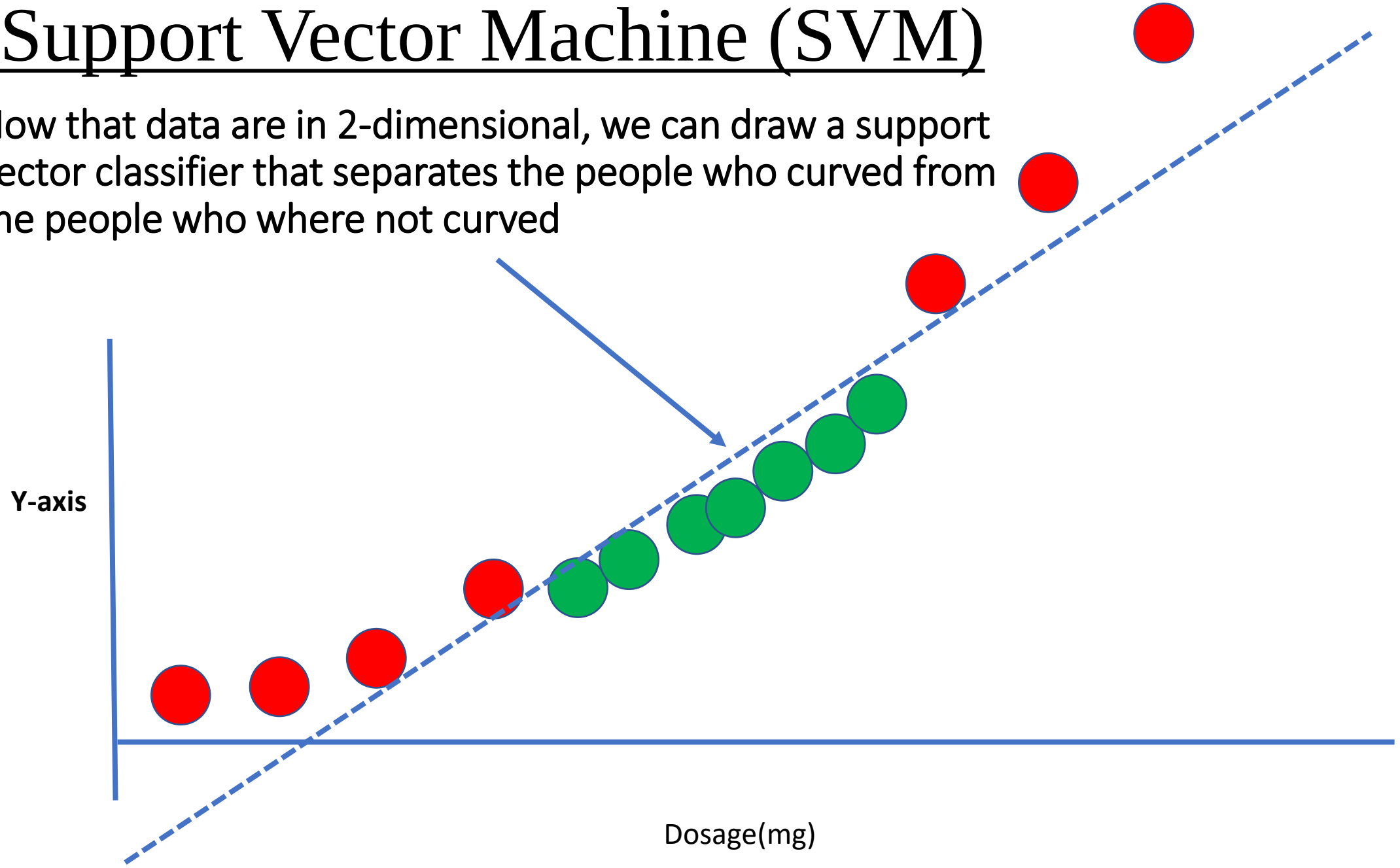
Support Vector Machine (SVM)

Since each observation has x and y-axis coordinates now that data are in 2-dimensional



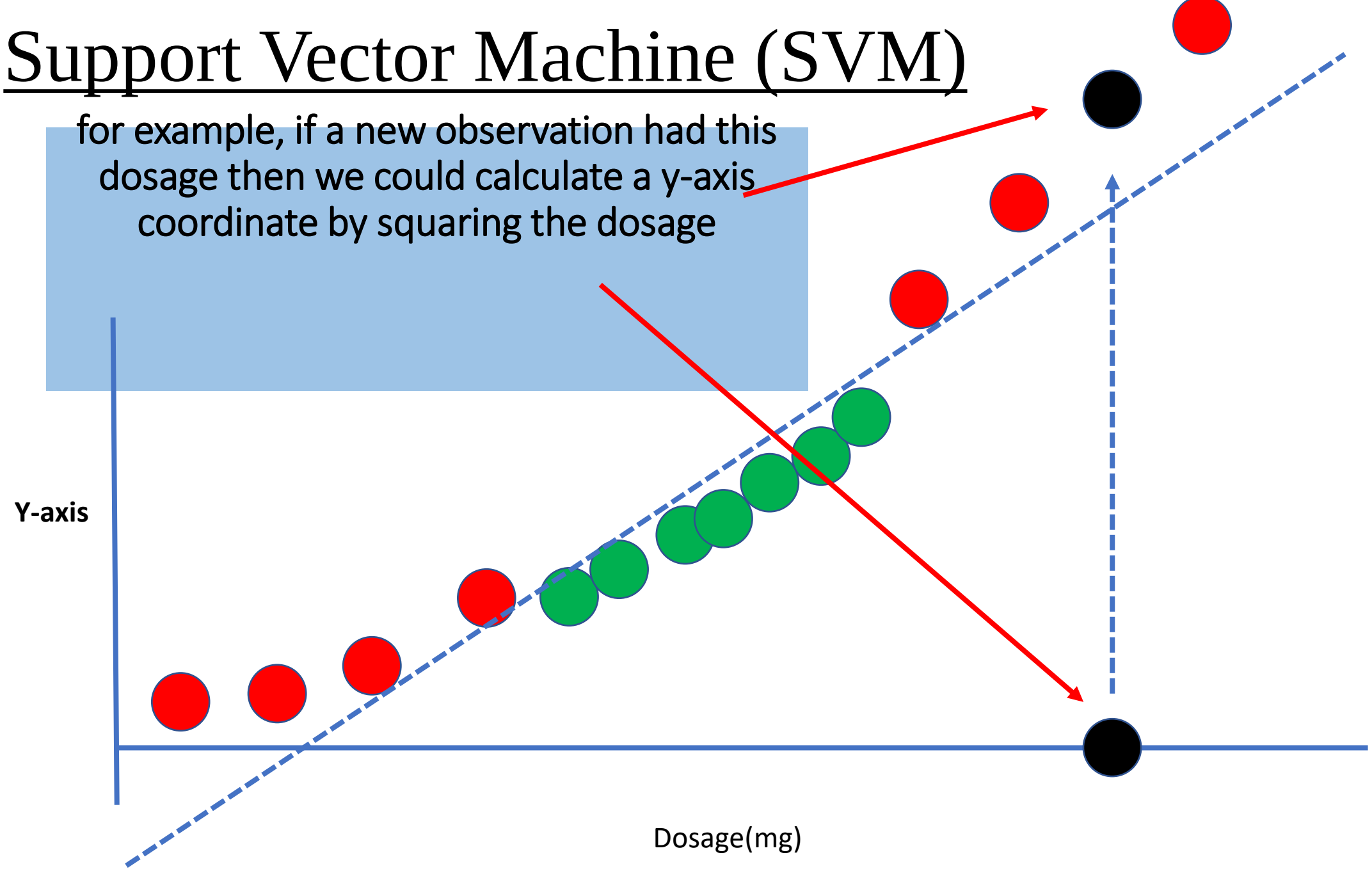
Support Vector Machine (SVM)

Now that data are in 2-dimensional, we can draw a support vector classifier that separates the people who curved from the people who where not curved



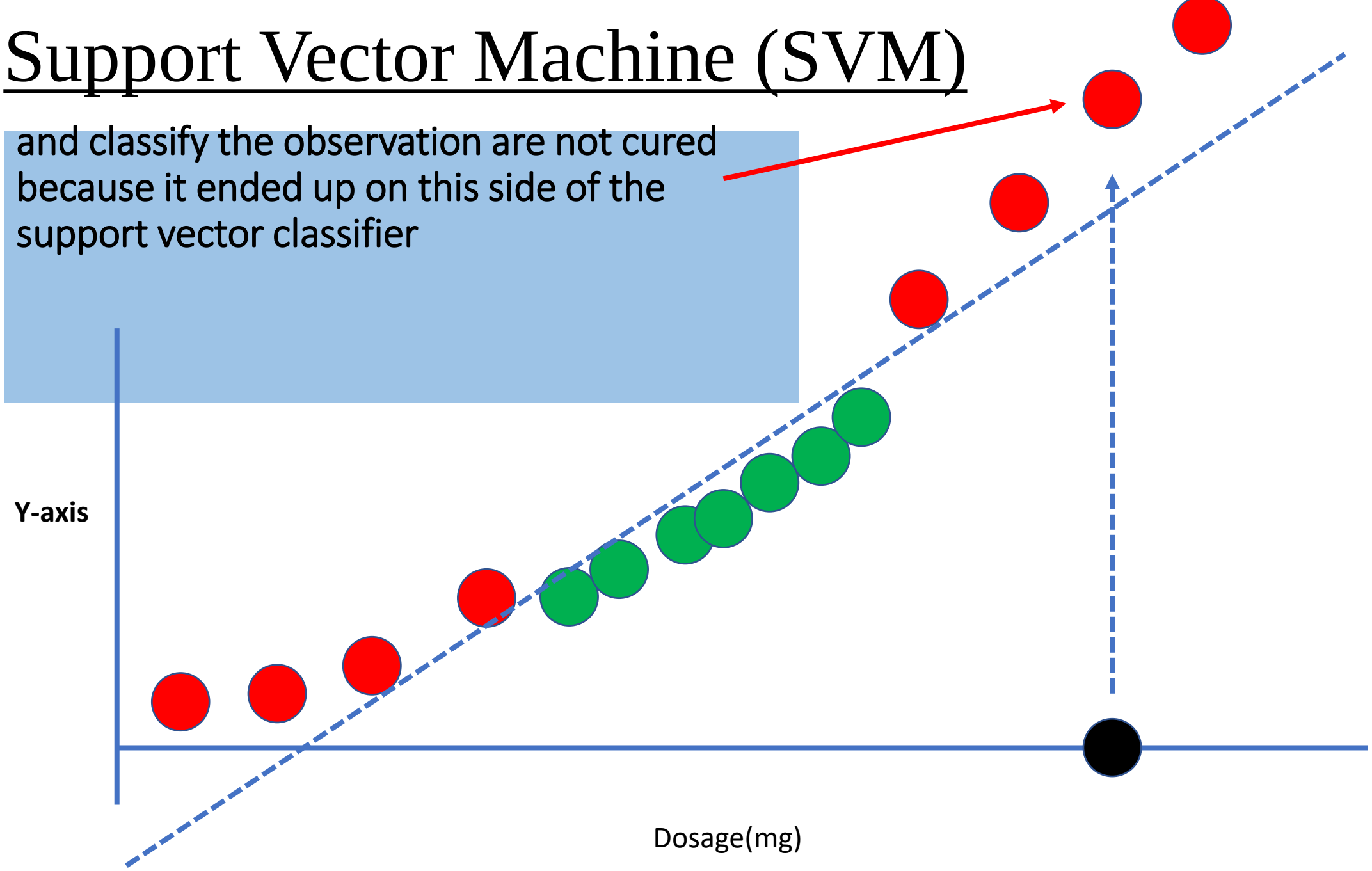
Support Vector Machine (SVM)

for example, if a new observation had this dosage then we could calculate a y-axis coordinate by squaring the dosage



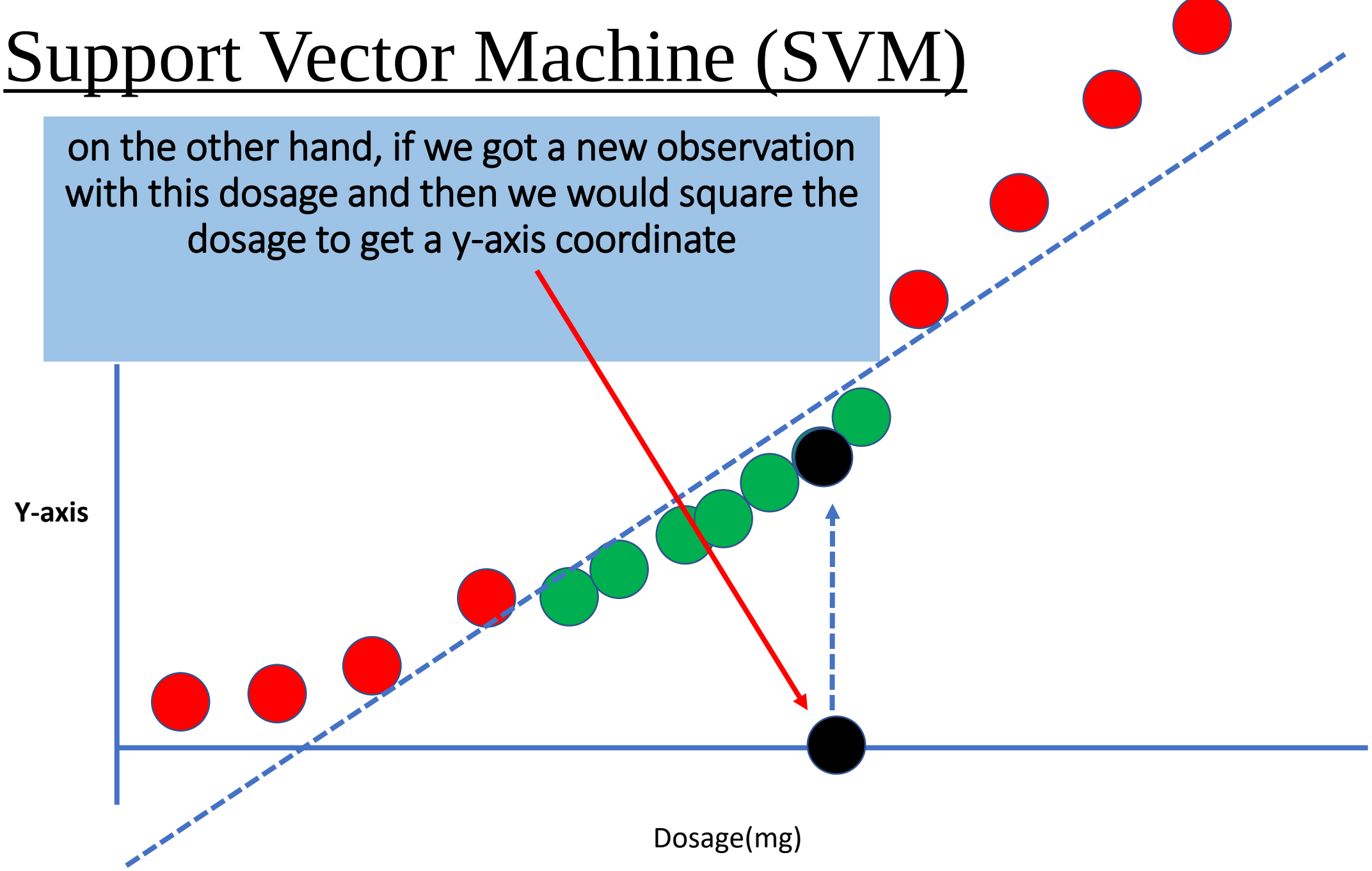
Support Vector Machine (SVM)

and classify the observation are not cured because it ended up on this side of the support vector classifier

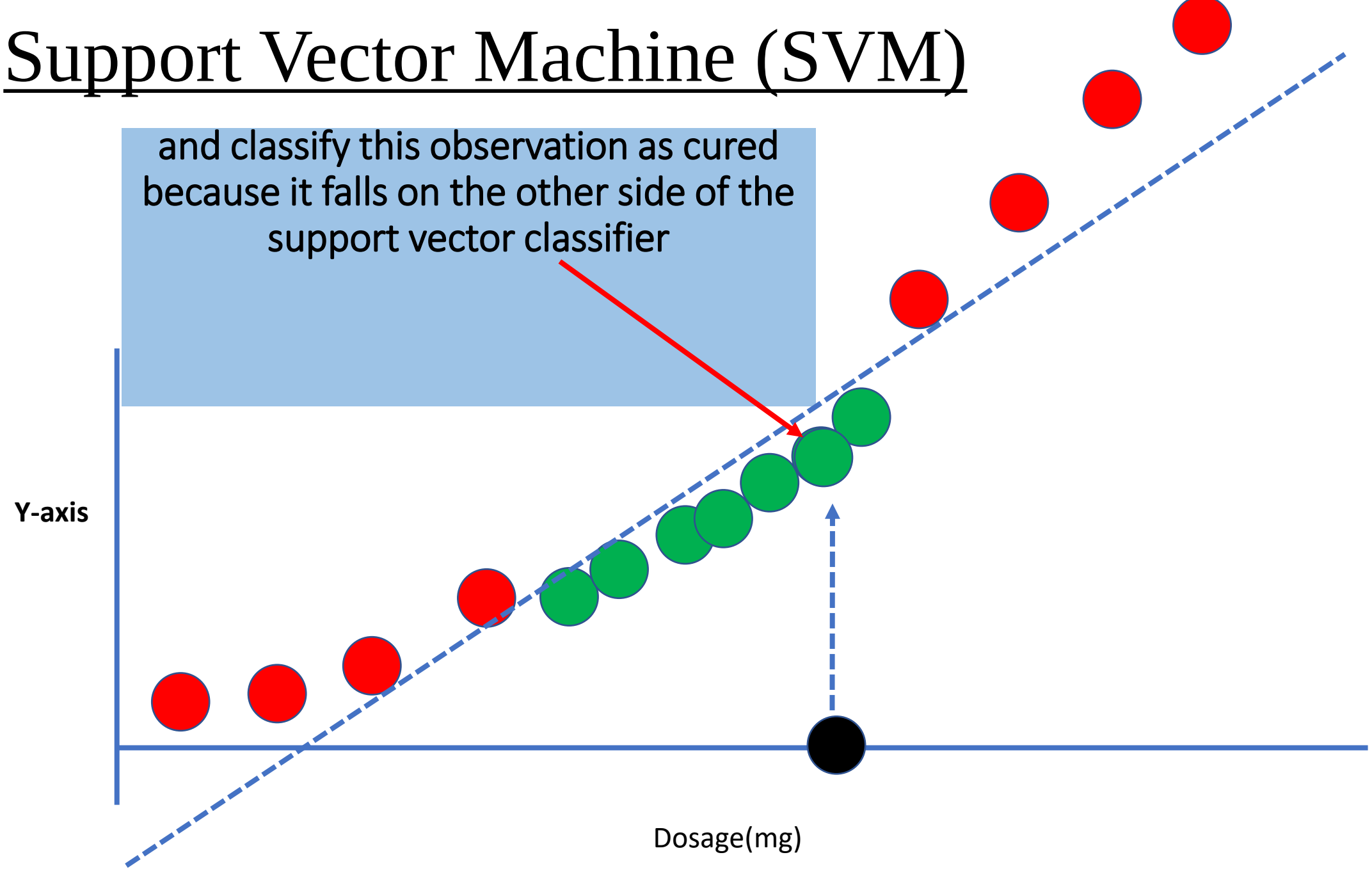


Support Vector Machine (SVM)

on the other hand, if we got a new observation with this dosage and then we would square the dosage to get a y-axis coordinate



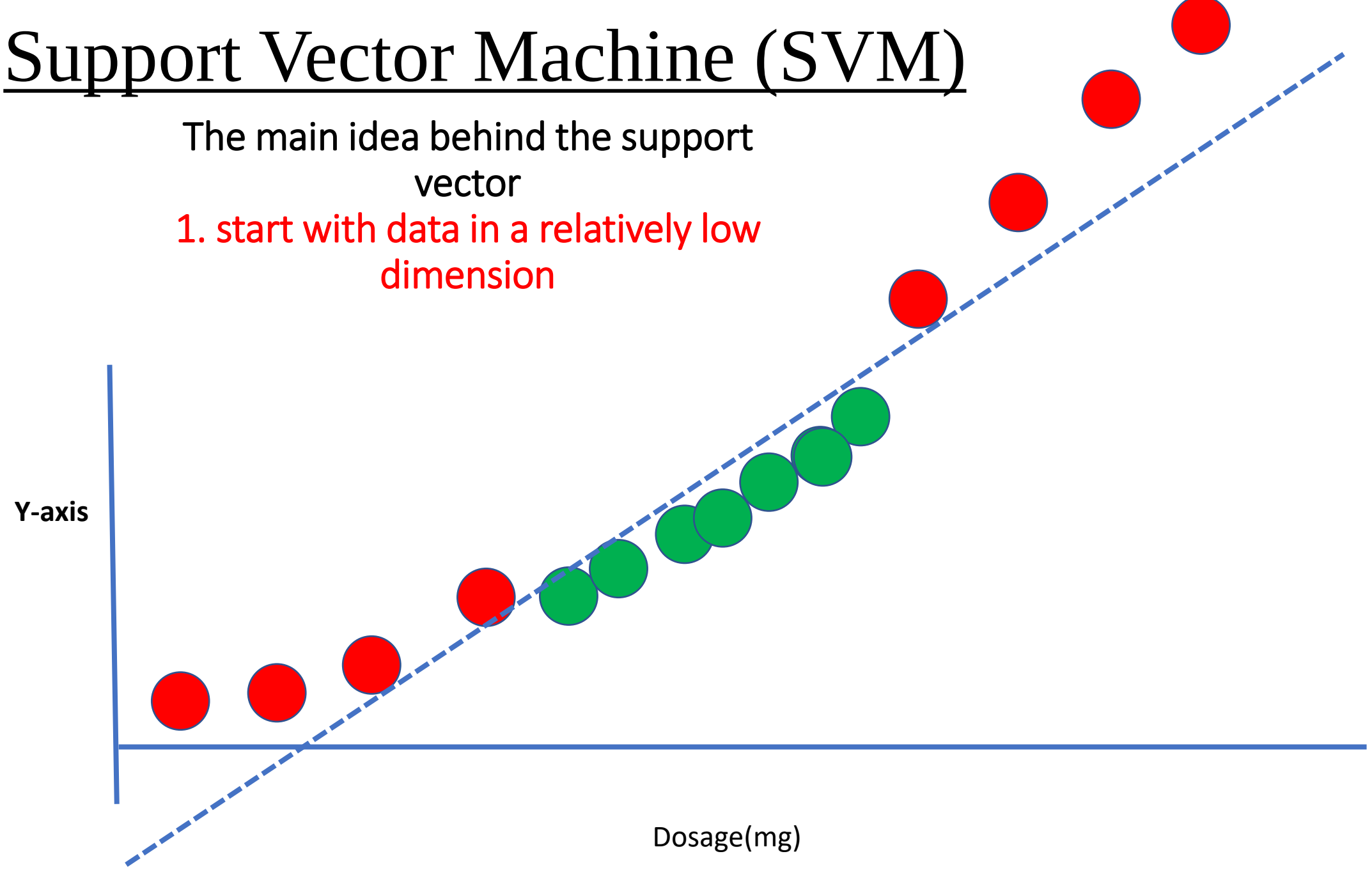
Support Vector Machine (SVM)



Support Vector Machine (SVM)

The main idea behind the support vector

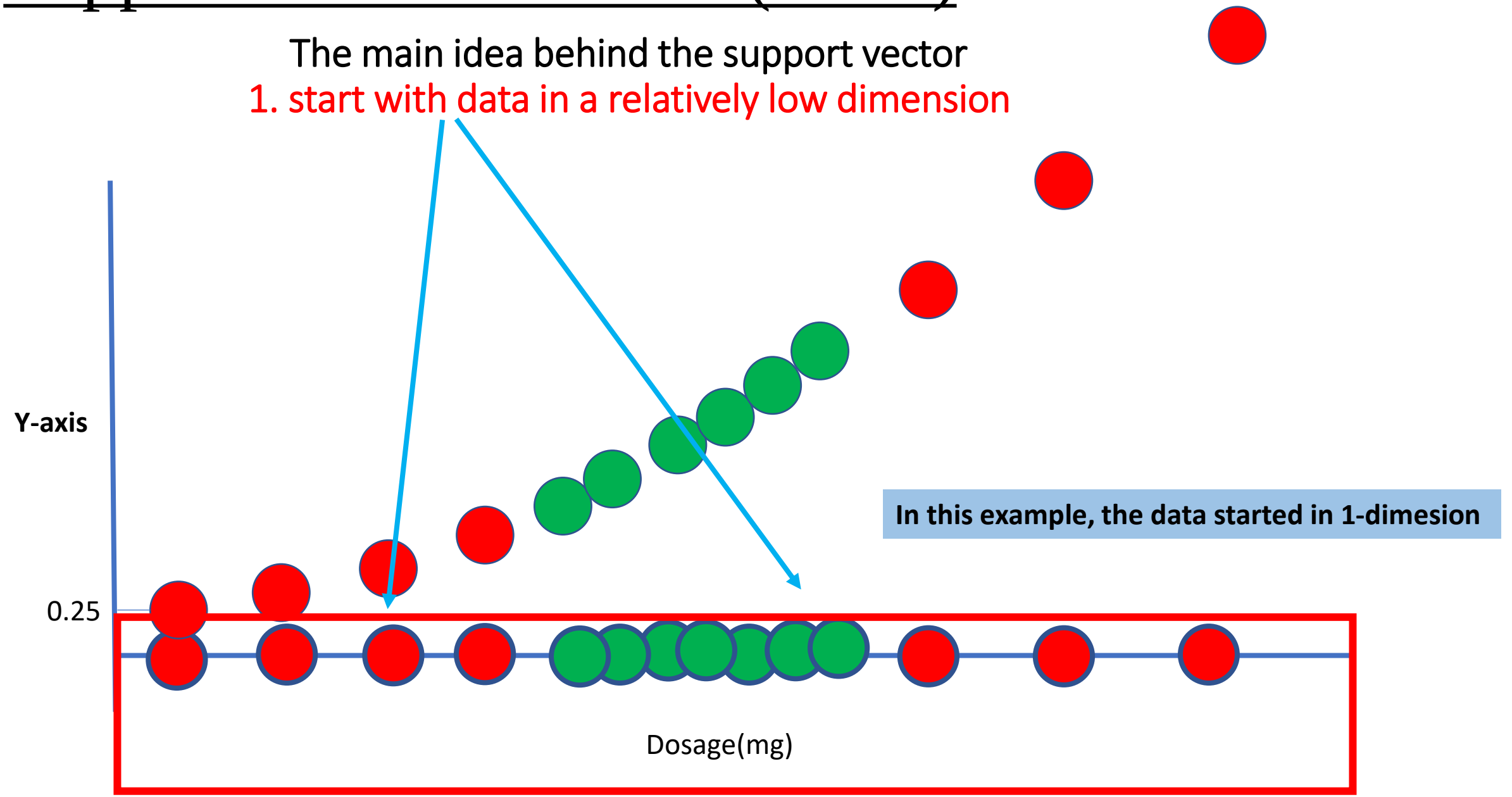
1. start with data in a relatively low dimension



Support Vector Machine (SVM)

The main idea behind the support vector

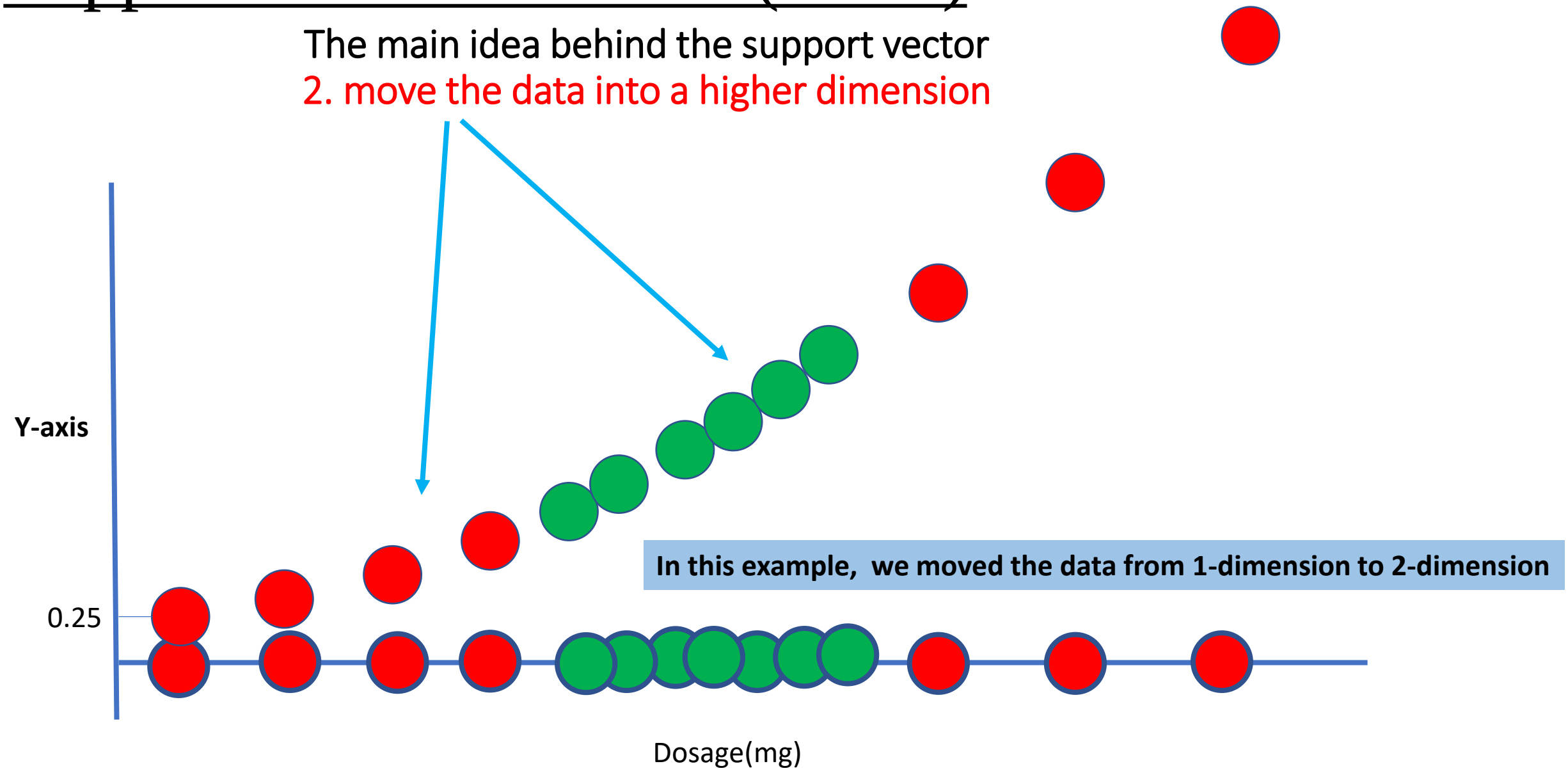
1. start with data in a relatively low dimension



Support Vector Machine (SVM)

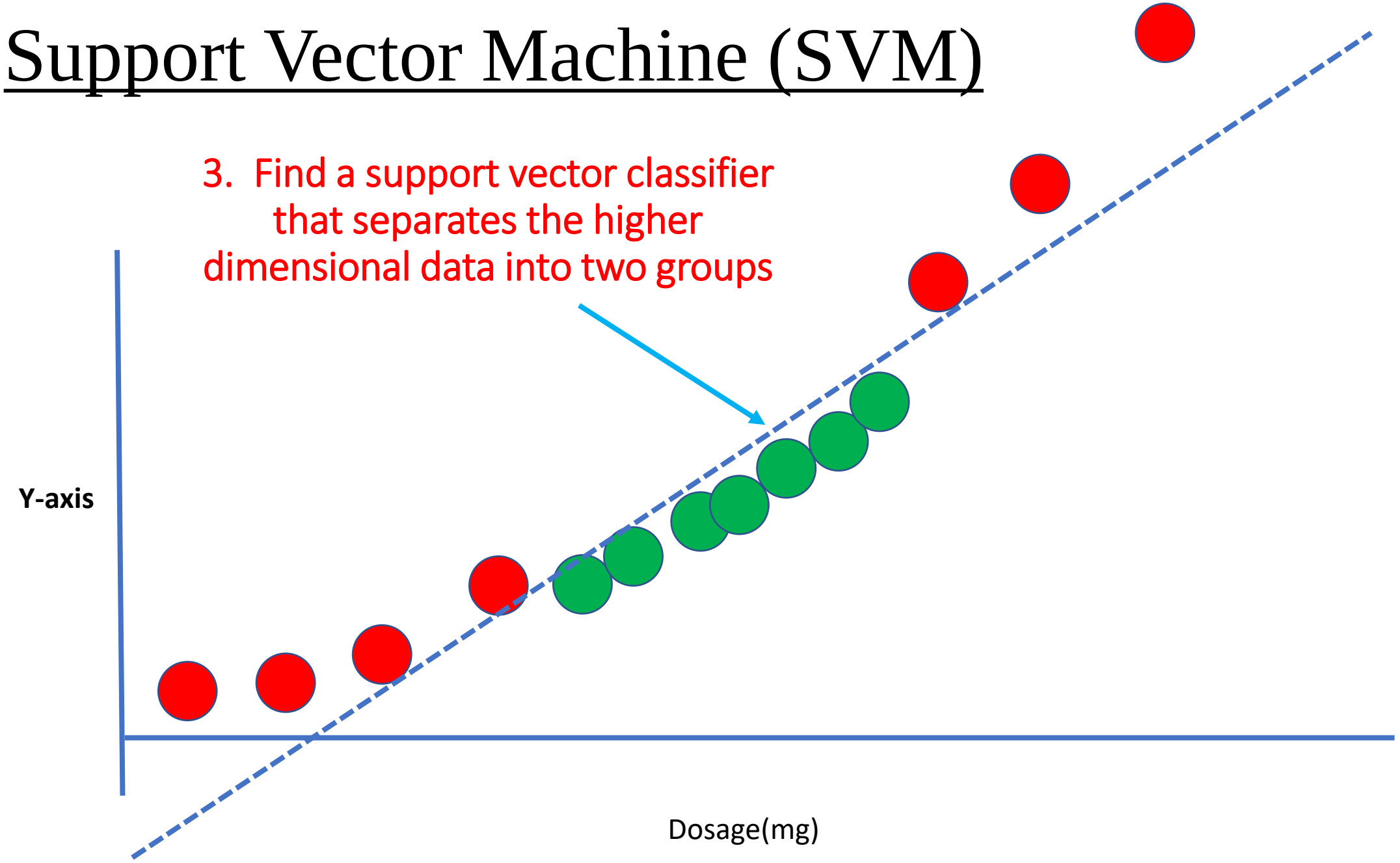
The main idea behind the support vector

2. move the data into a higher dimension



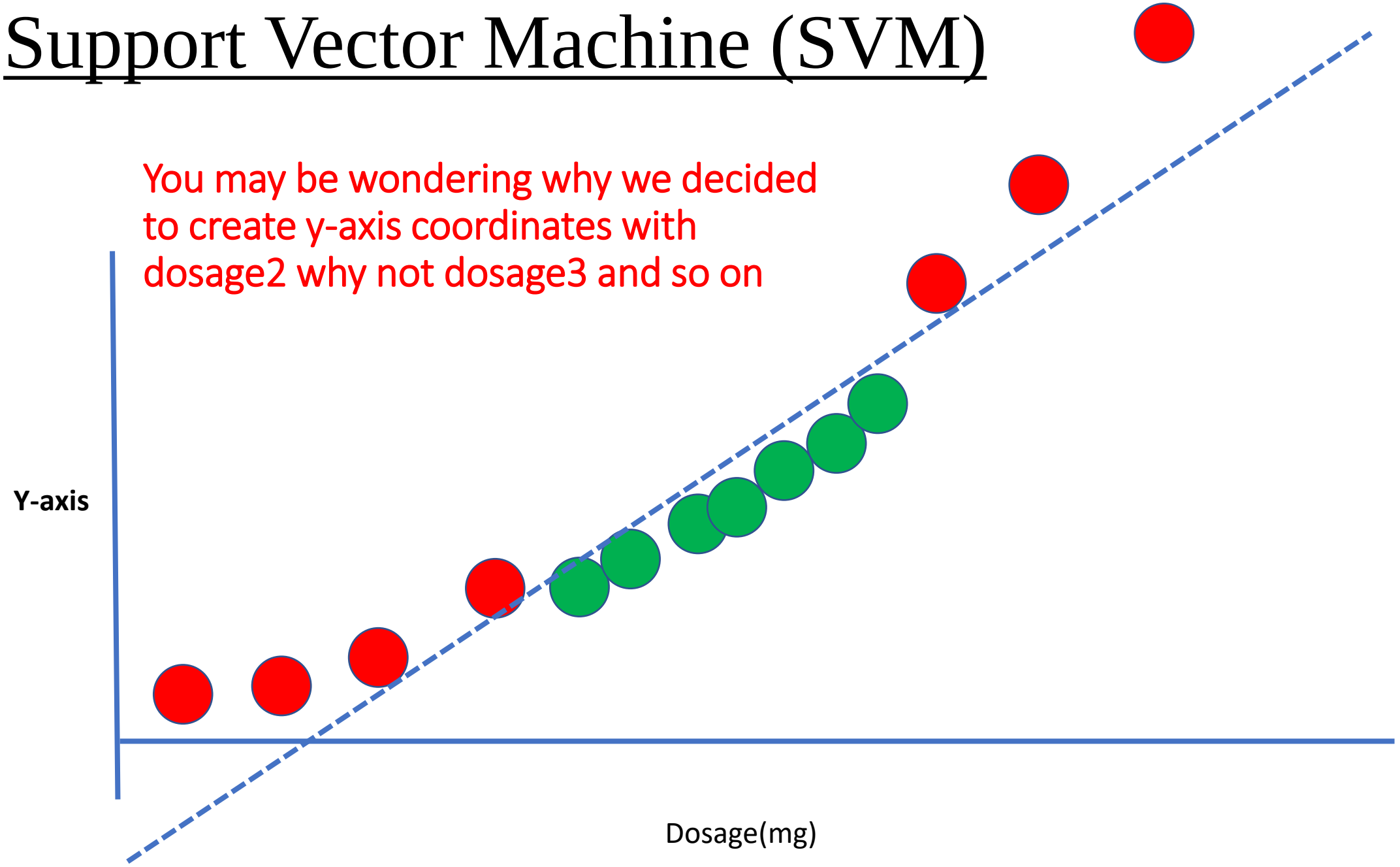
Support Vector Machine (SVM)

3. Find a support vector classifier that separates the higher dimensional data into two groups



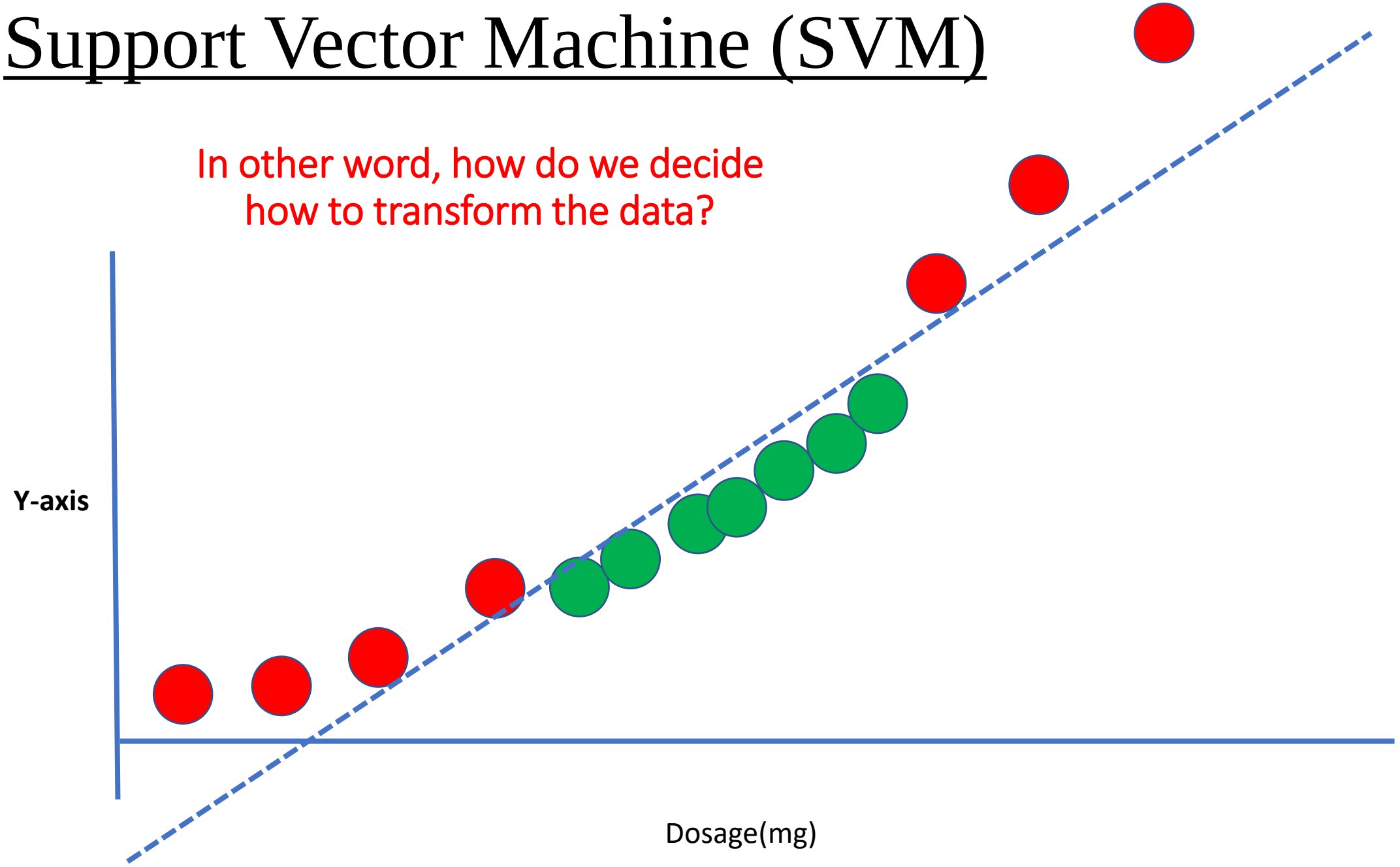
Support Vector Machine (SVM)

You may be wondering why we decided to create y-axis coordinates with dosage2 why not dosage3 and so on



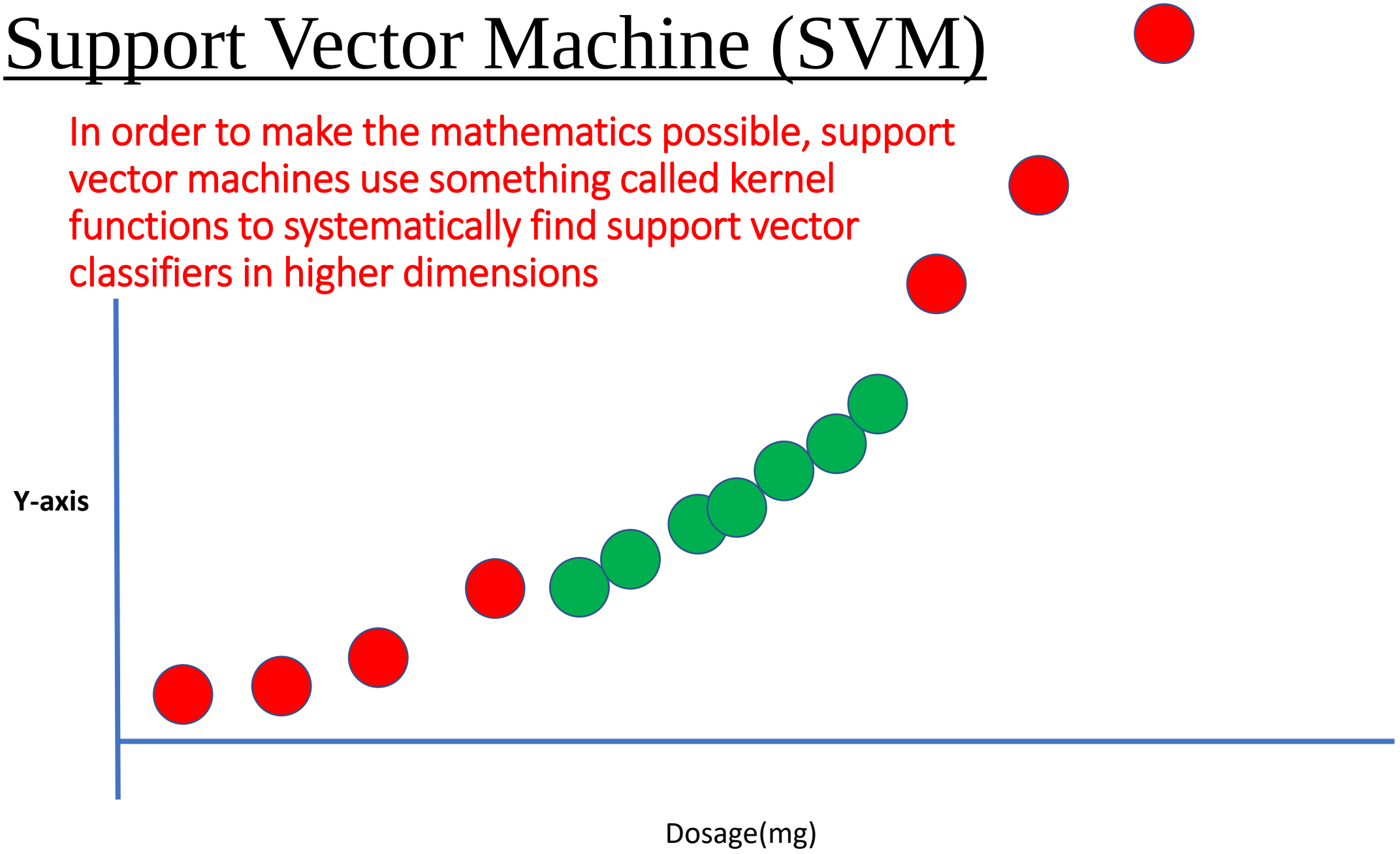
Support Vector Machine (SVM)

In other word, how do we decide how to transform the data?



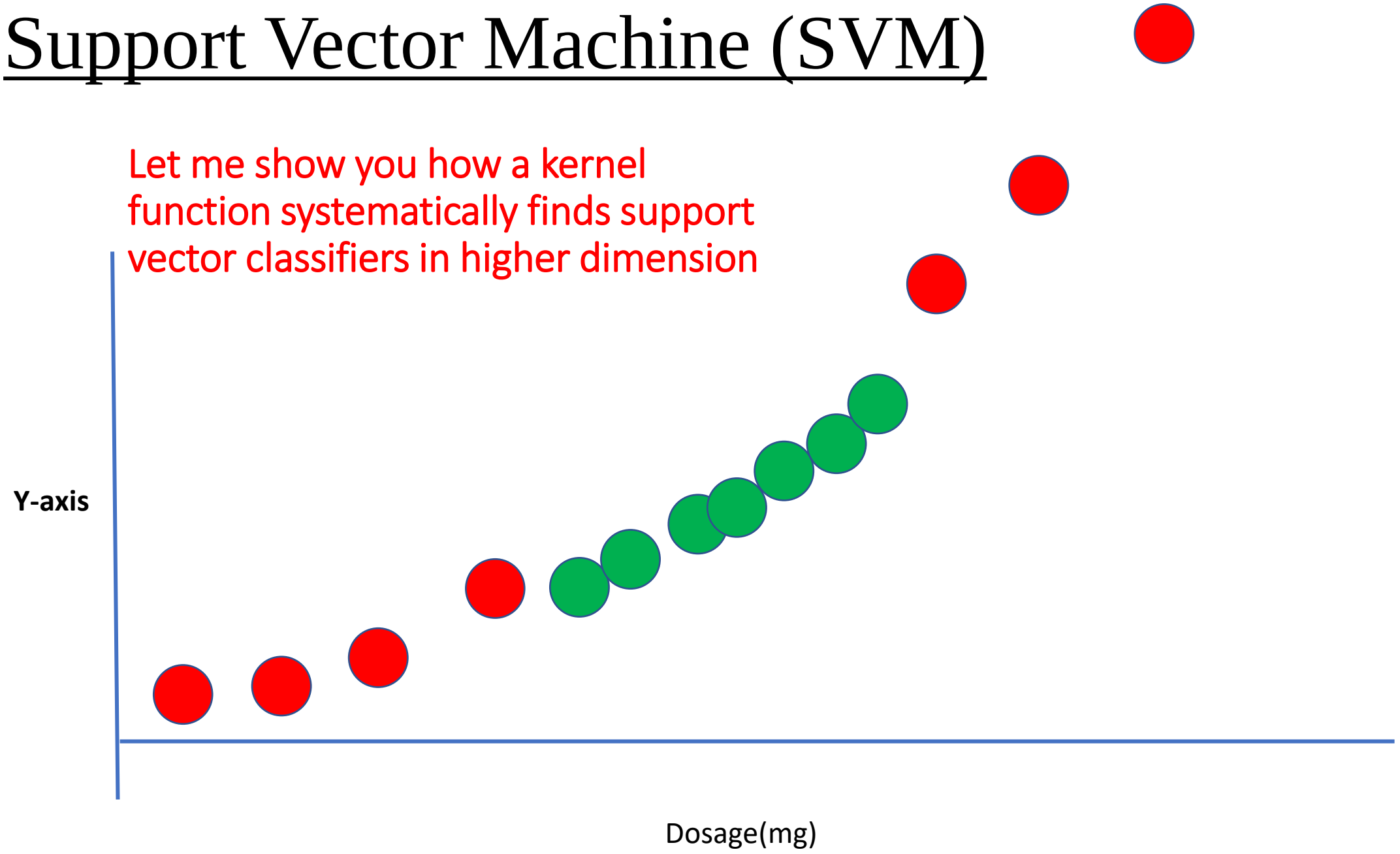
Support Vector Machine (SVM)

In order to make the mathematics possible, support vector machines use something called kernel functions to systematically find support vector classifiers in higher dimensions



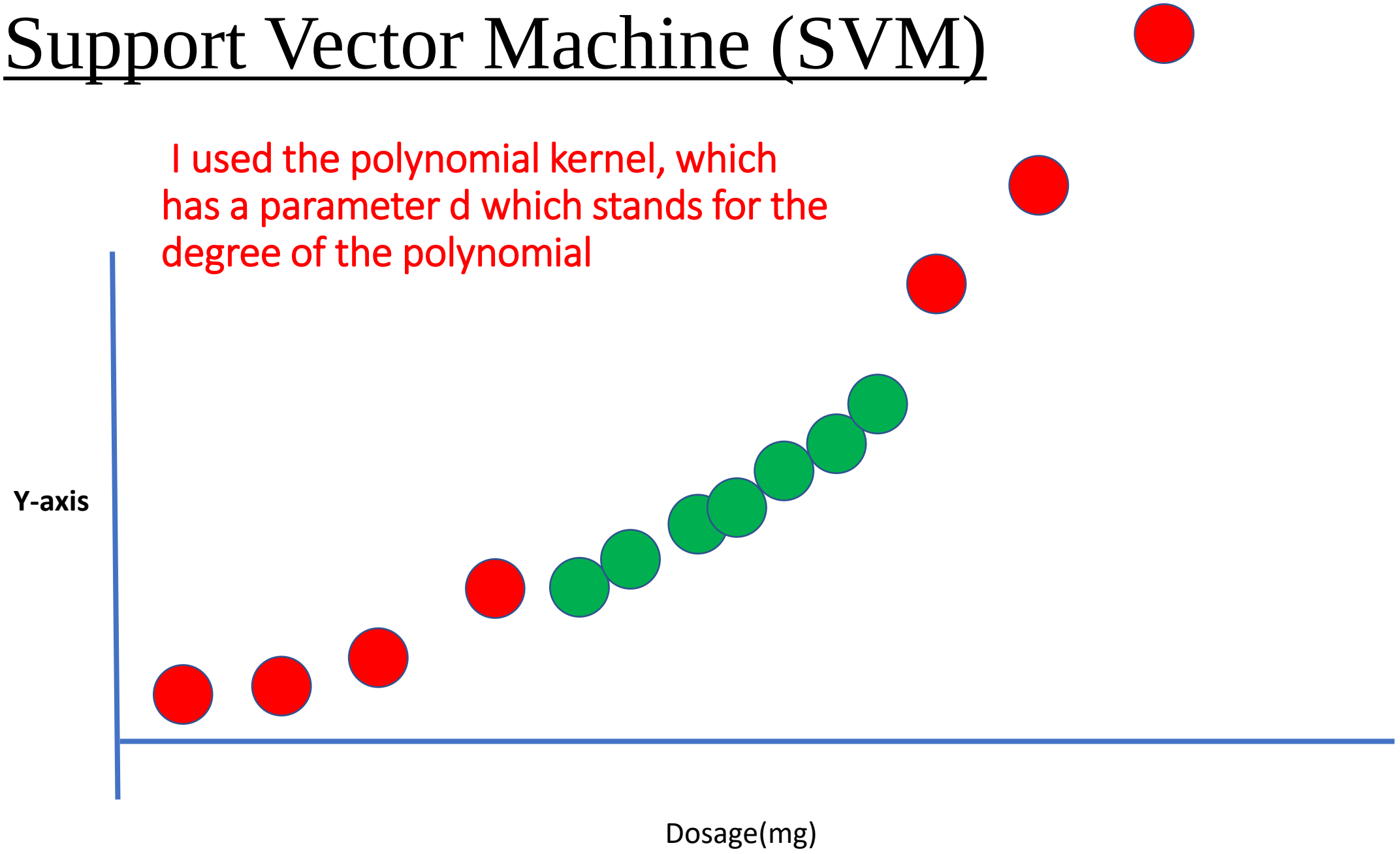
Support Vector Machine (SVM)

Let me show you how a kernel function systematically finds support vector classifiers in higher dimension



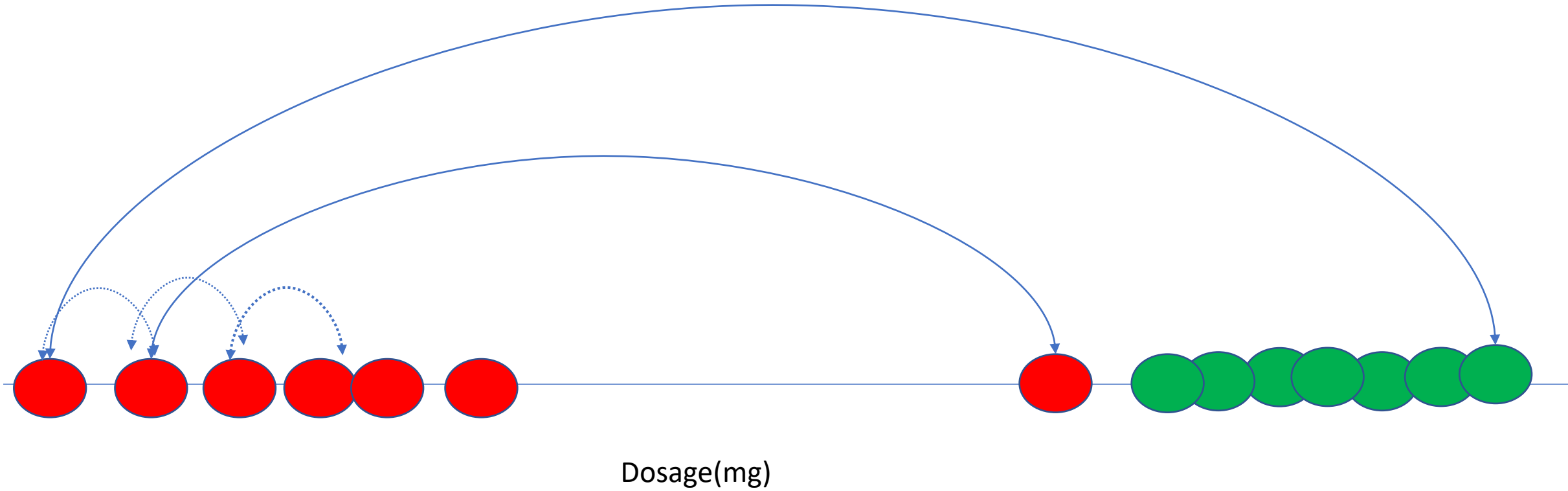
Support Vector Machine (SVM)

I used the polynomial kernel, which has a parameter d which stands for the degree of the polynomial



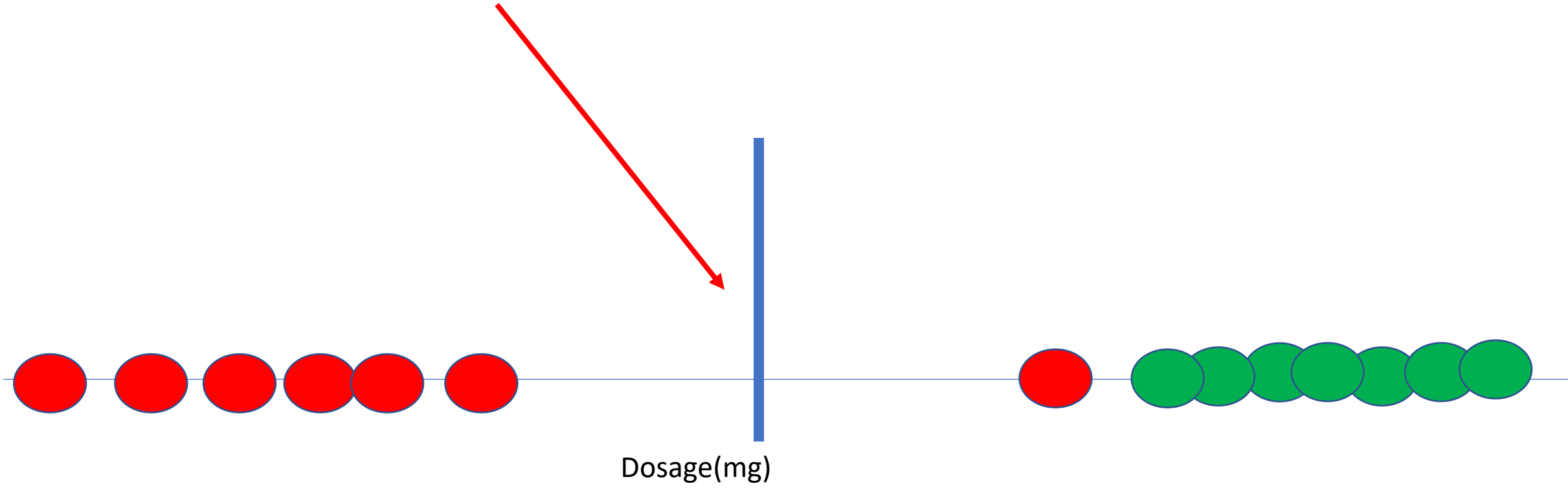
Support Vector Machine (SVM)

when d=1, the polynomial kernel computes the relationship between each pair of observations in 1-dimension



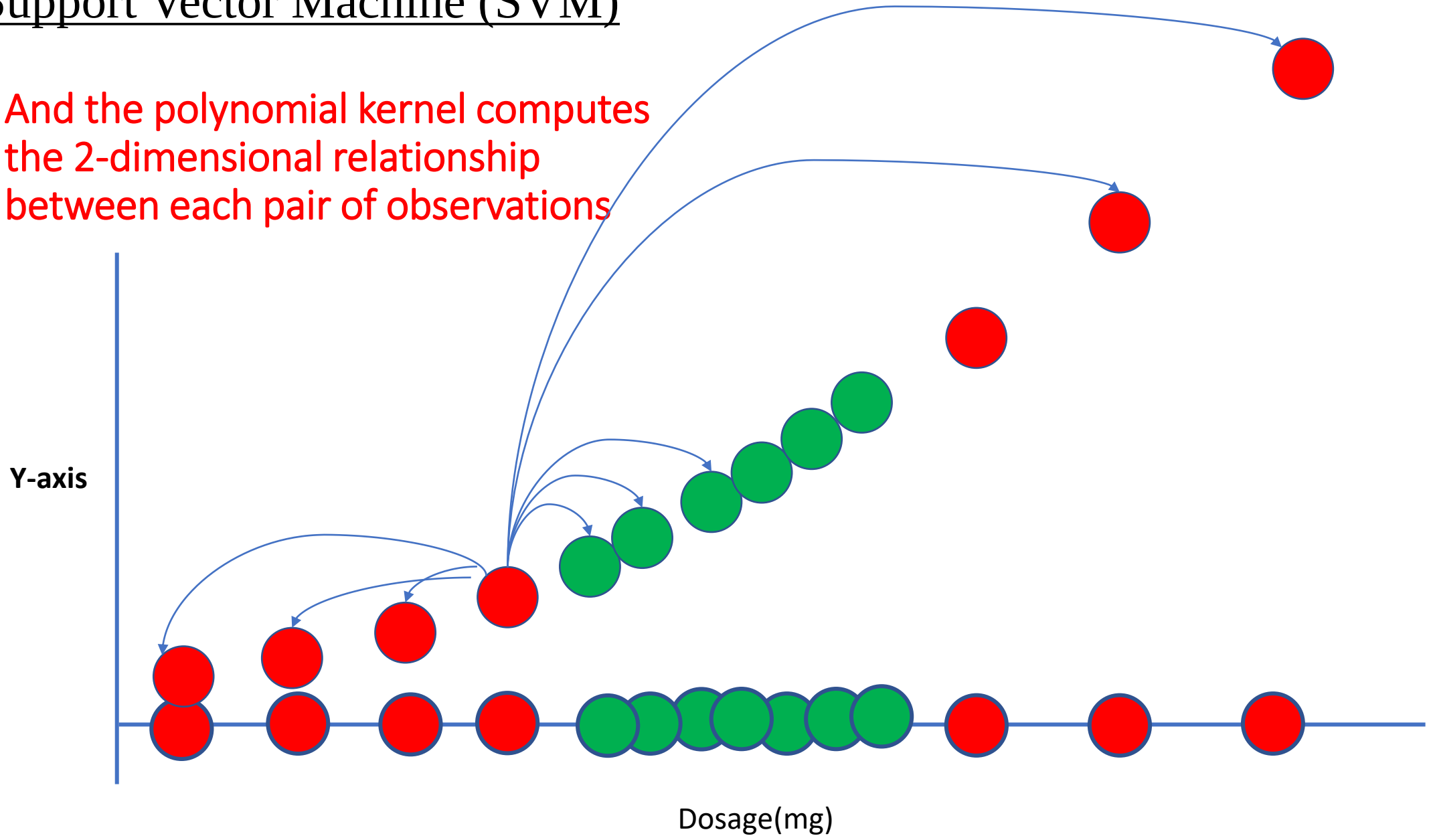
Support Vector Machine (SVM)

when d=1, the polynomial kernel computes the relationship between each pair of observations in 1-dimension and these relationship are used to find a support vector classifier



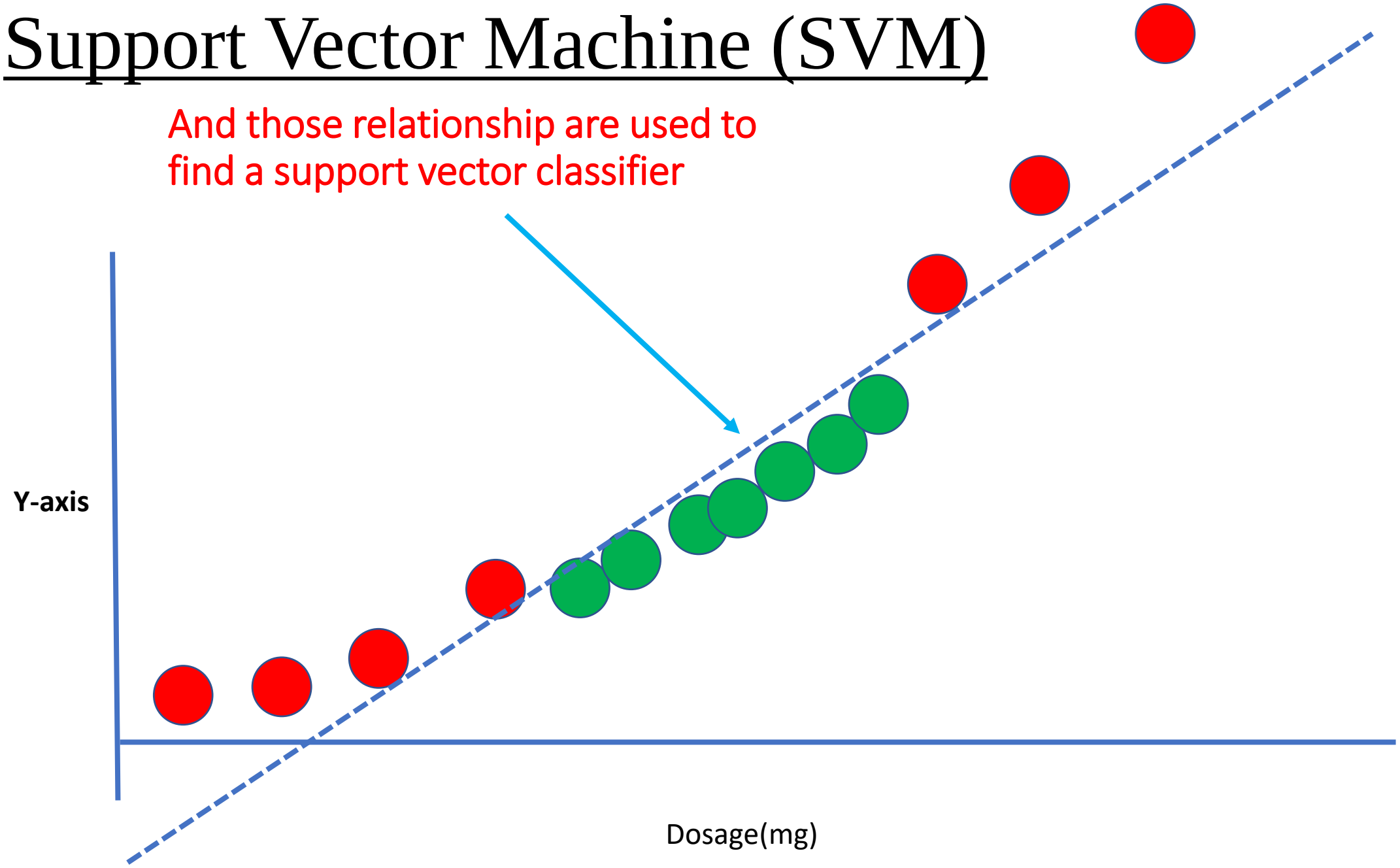
Support Vector Machine (SVM)

And the polynomial kernel computes the 2-dimensional relationship between each pair of observations



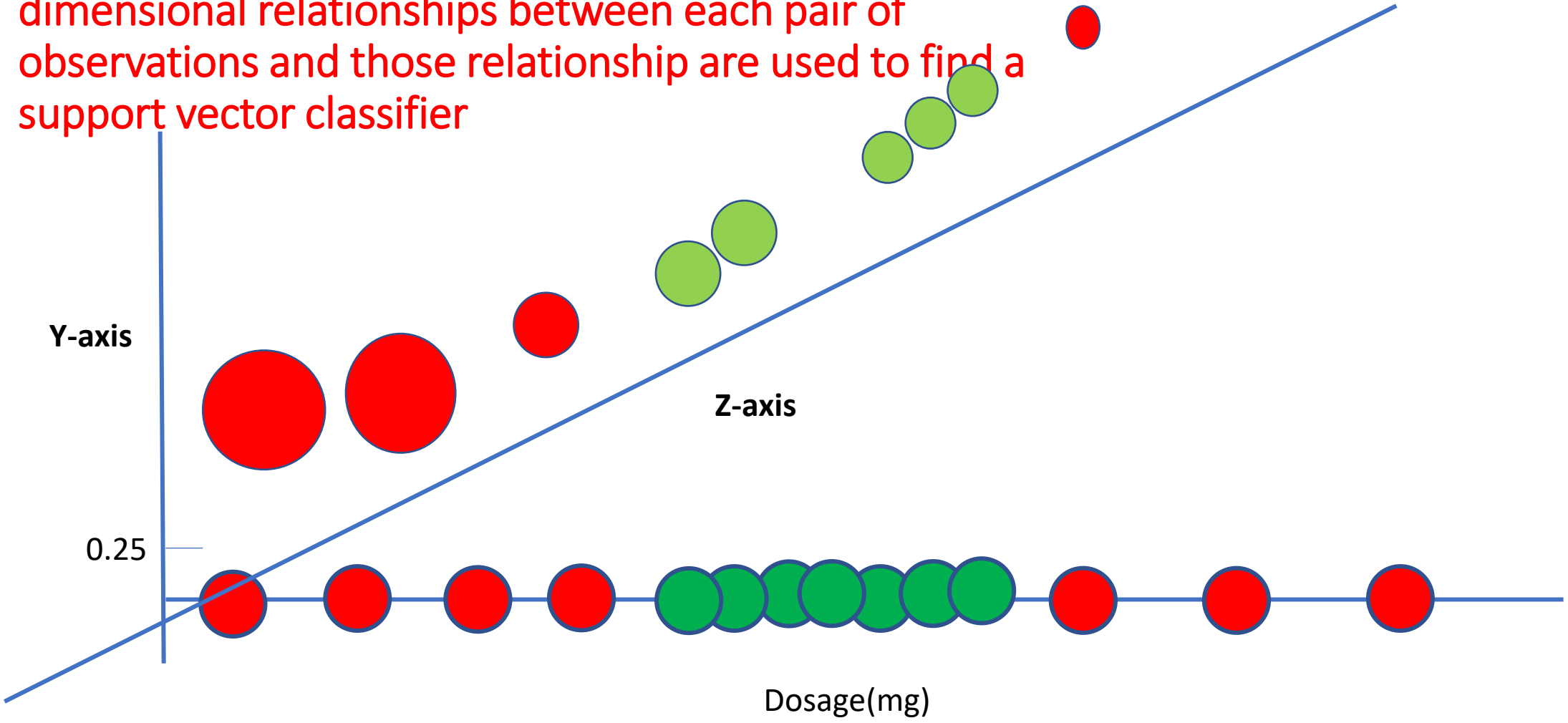
Support Vector Machine (SVM)

And those relationship are used to find a support vector classifier



Support Vector Machine (SVM)

And if we set d=3, then we would get a 3dr dimension based on dosgaes3 and polynomial kernel computes the 3-dimensional relationships between each pair of observations and those relationship are used to find a support vector classifier



Support Vector Machine (SVM)

when d=4 or more, then we get even more dimensions to find a support vector classifier.

In summary, the polynomial kernel systematically increases dimensions by setting d, the degree of the polynomial…. And the relationships between each pair of observations are used to find a support vector classifier.

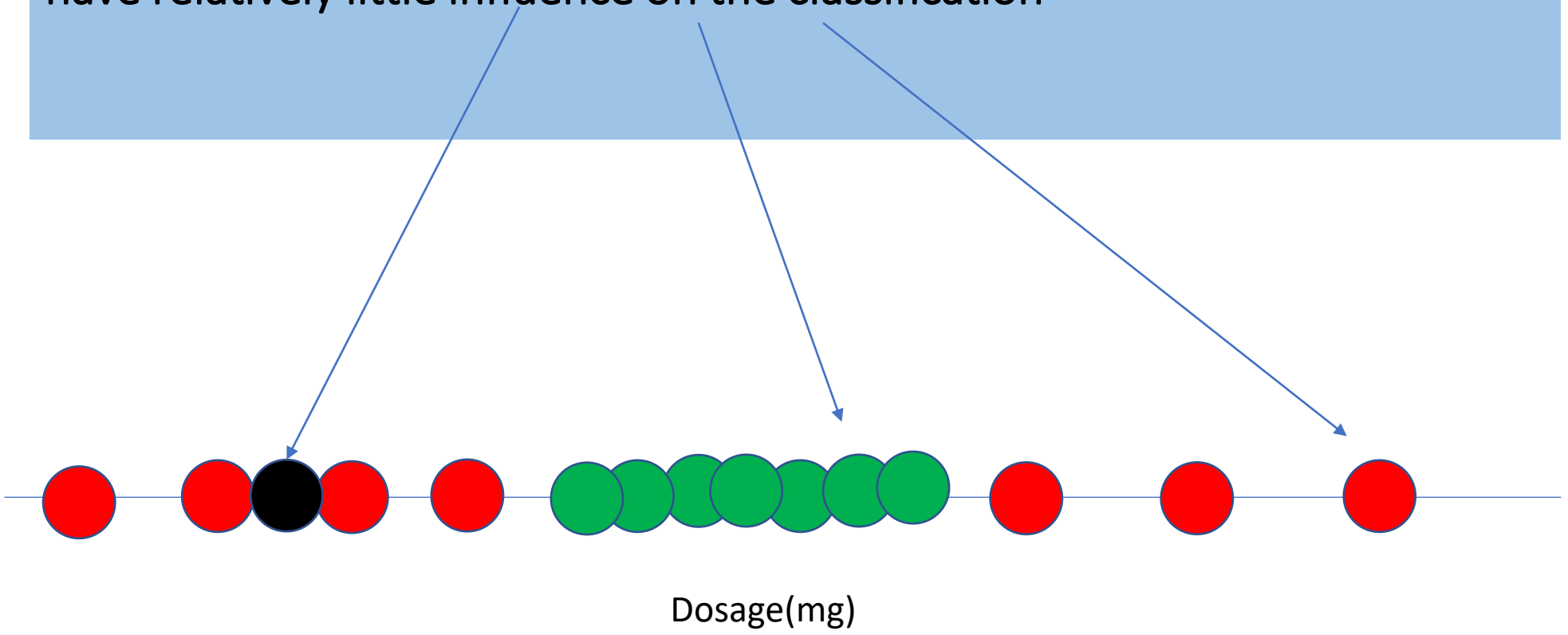
We can find a good value for d with cross validation

Support Vector Machine (SVM)

Another very commonly used kernel is the radial kernel, also known as the radial basis function (RBF) kernel and radial kernel finds support vector classifiers in infinite dimensions, so we cant use this example.

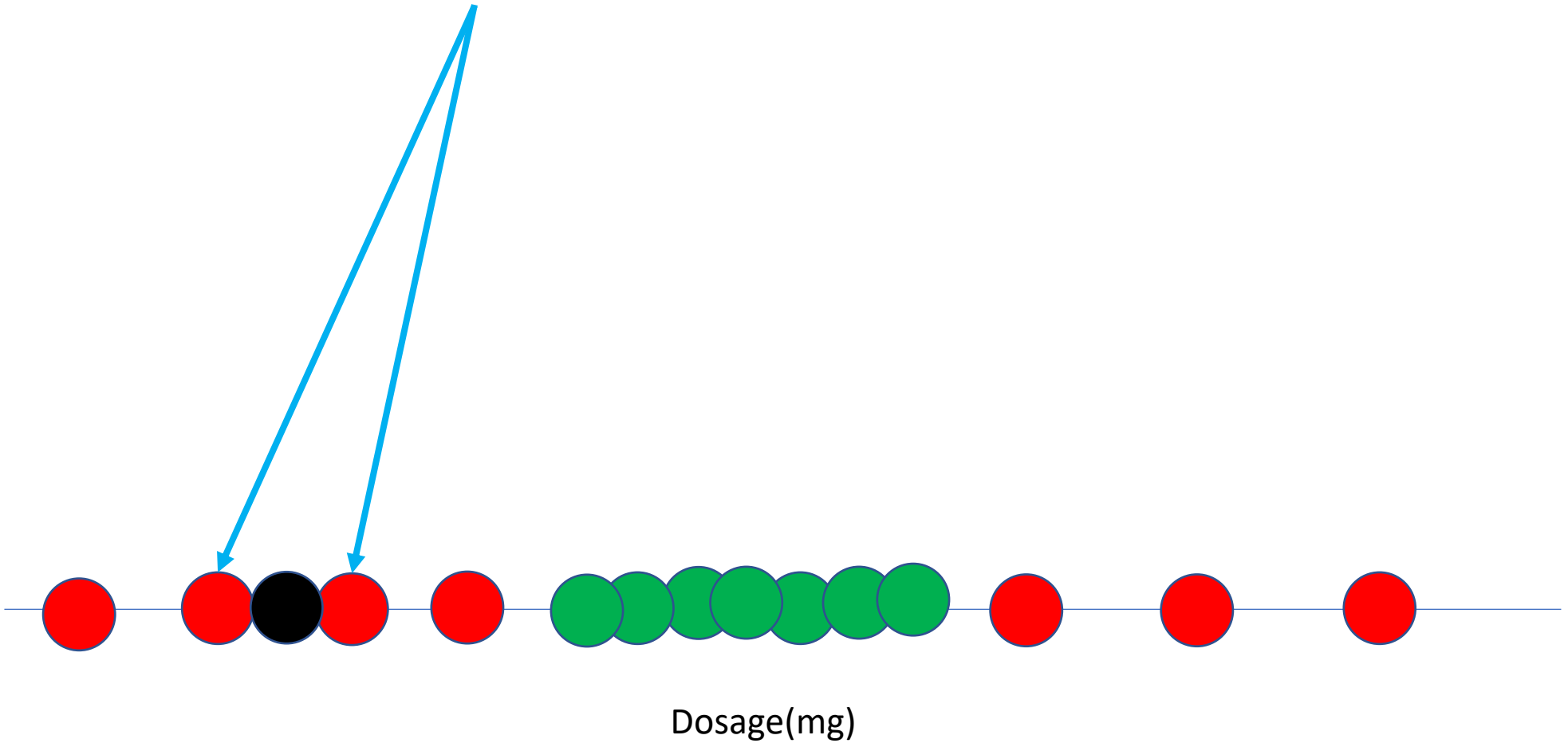
Support Vector Machine (SVM)

However , when using it on a new observation like this, the radial kernel behaves like a weighted nearest neighbor model
in other words, the closest observations have a lot of influence on how we classify the new observation and the observations that are further away have relatively little influence on the classification



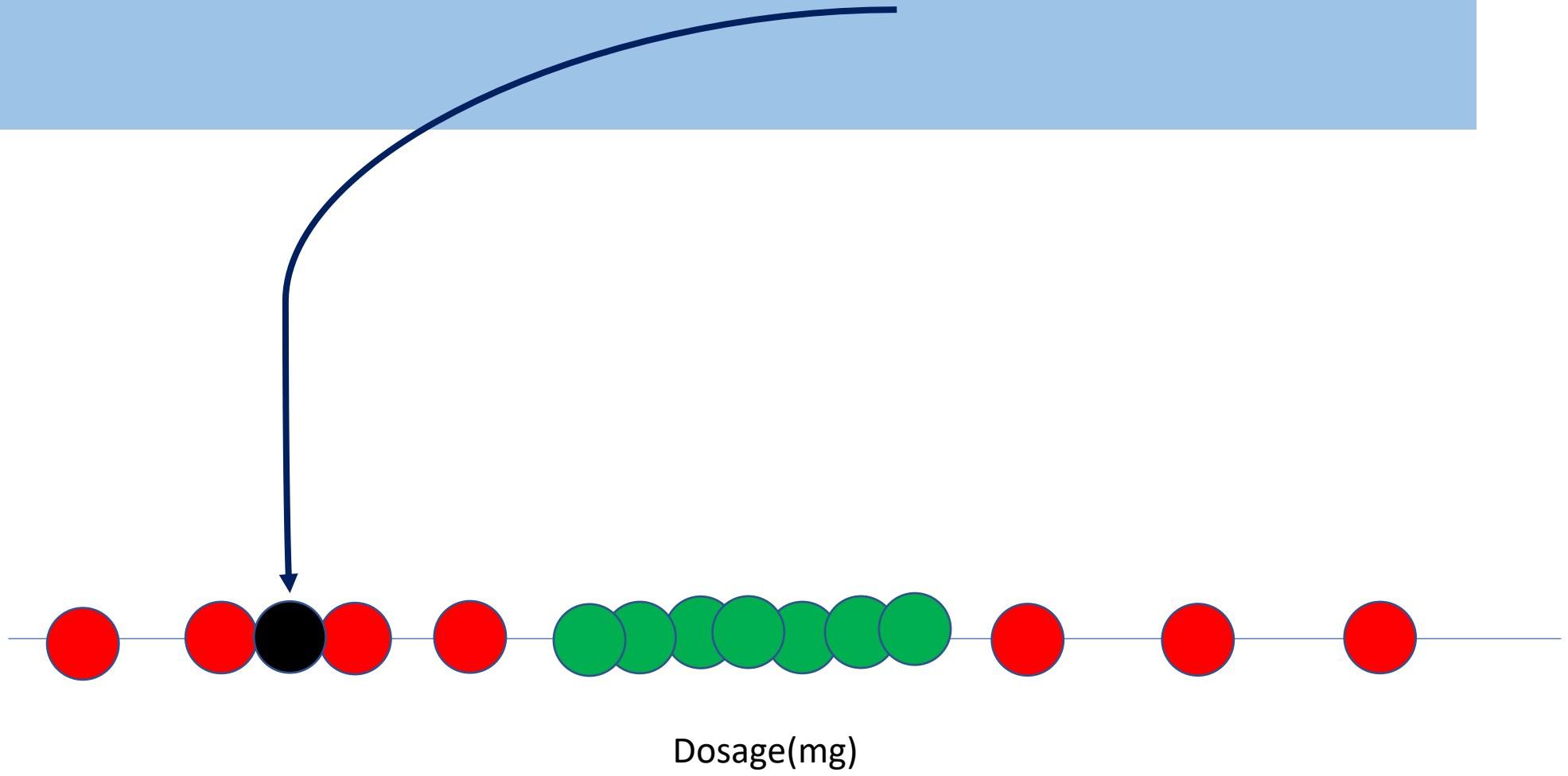
Support Vector Machine (SVM)

So, since these observations are the closest to the new observation…



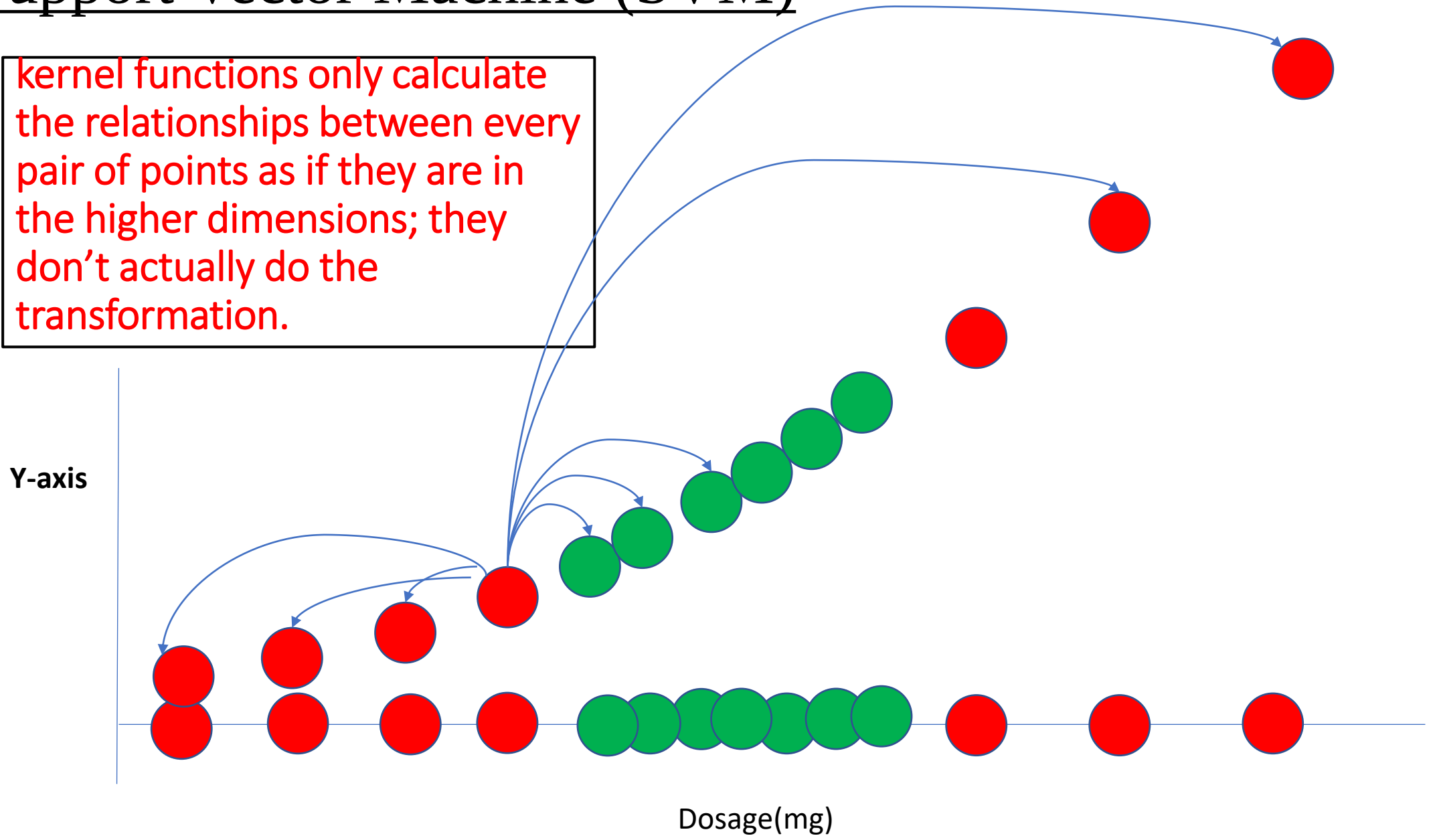
Support Vector Machine (SVM)

And radial kernel uses their classification for the new observation



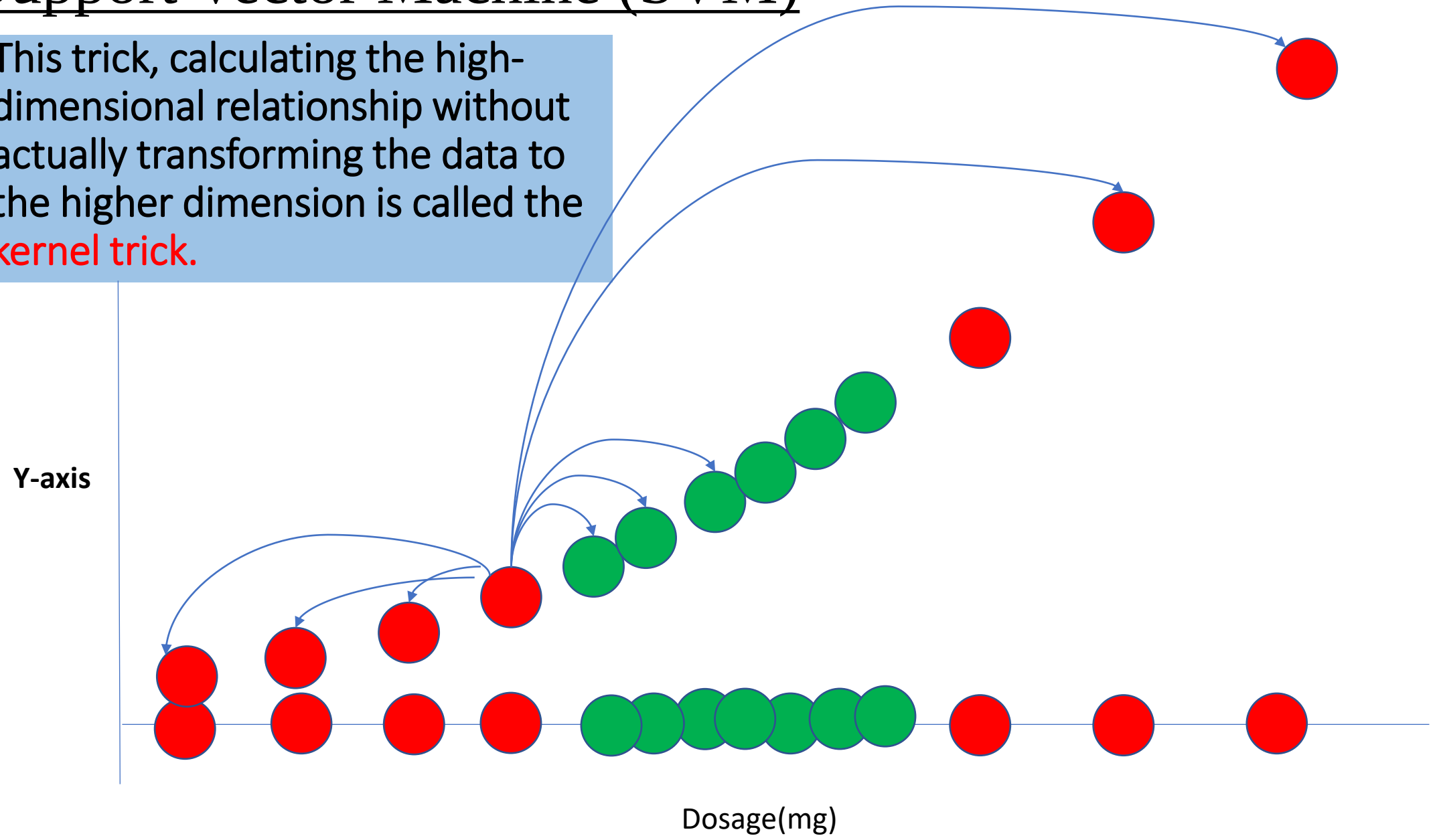
Support Vector Machine (SVM)

kernel functions only calculate the relationships between every pair of points as if they are in the higher dimensions; they don't actually do the transformation.



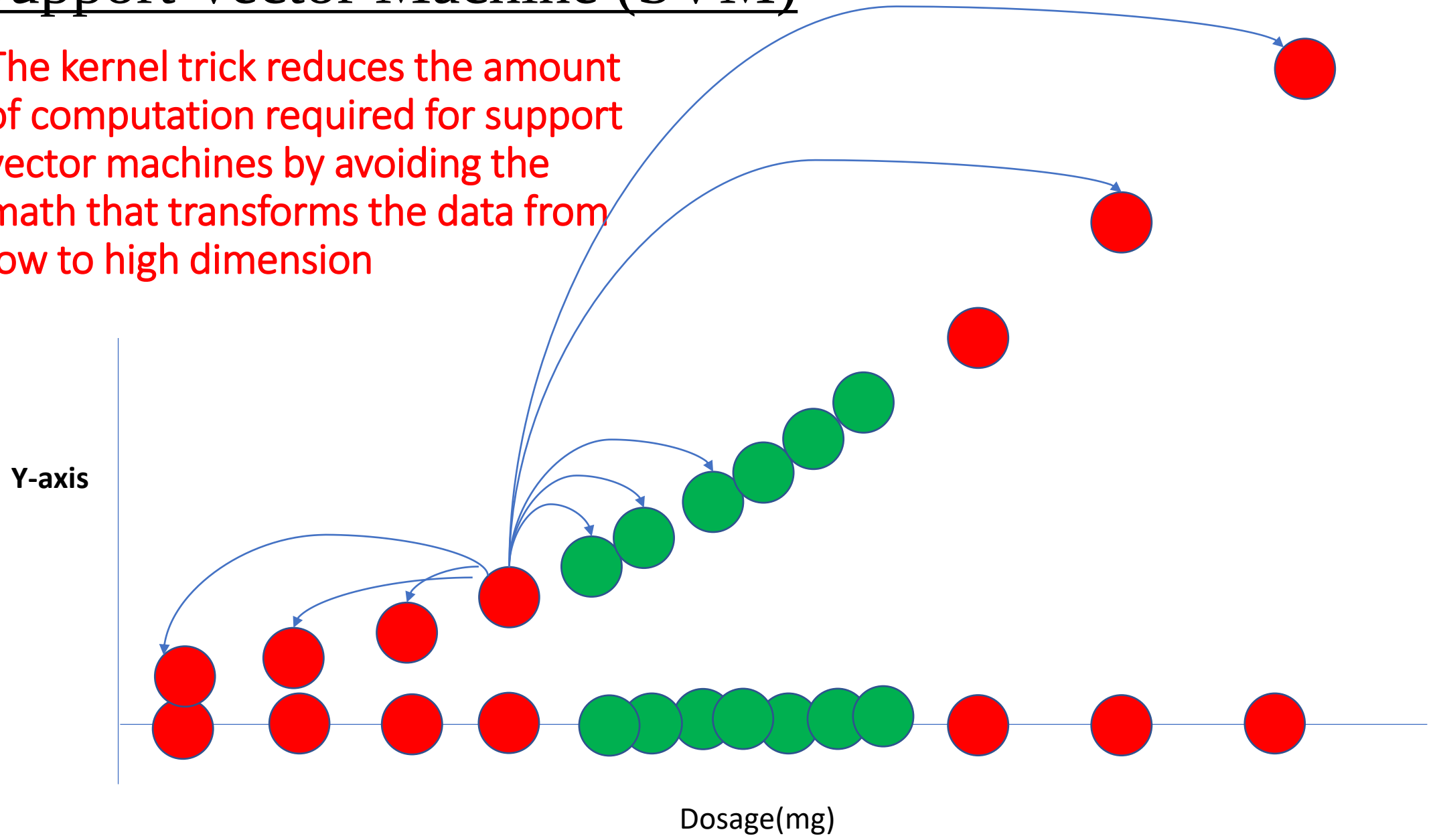
Support Vector Machine (SVM)

This trick, calculating the high-dimensional relationship without actually transforming the data to the higher dimension is called the kernel trick.



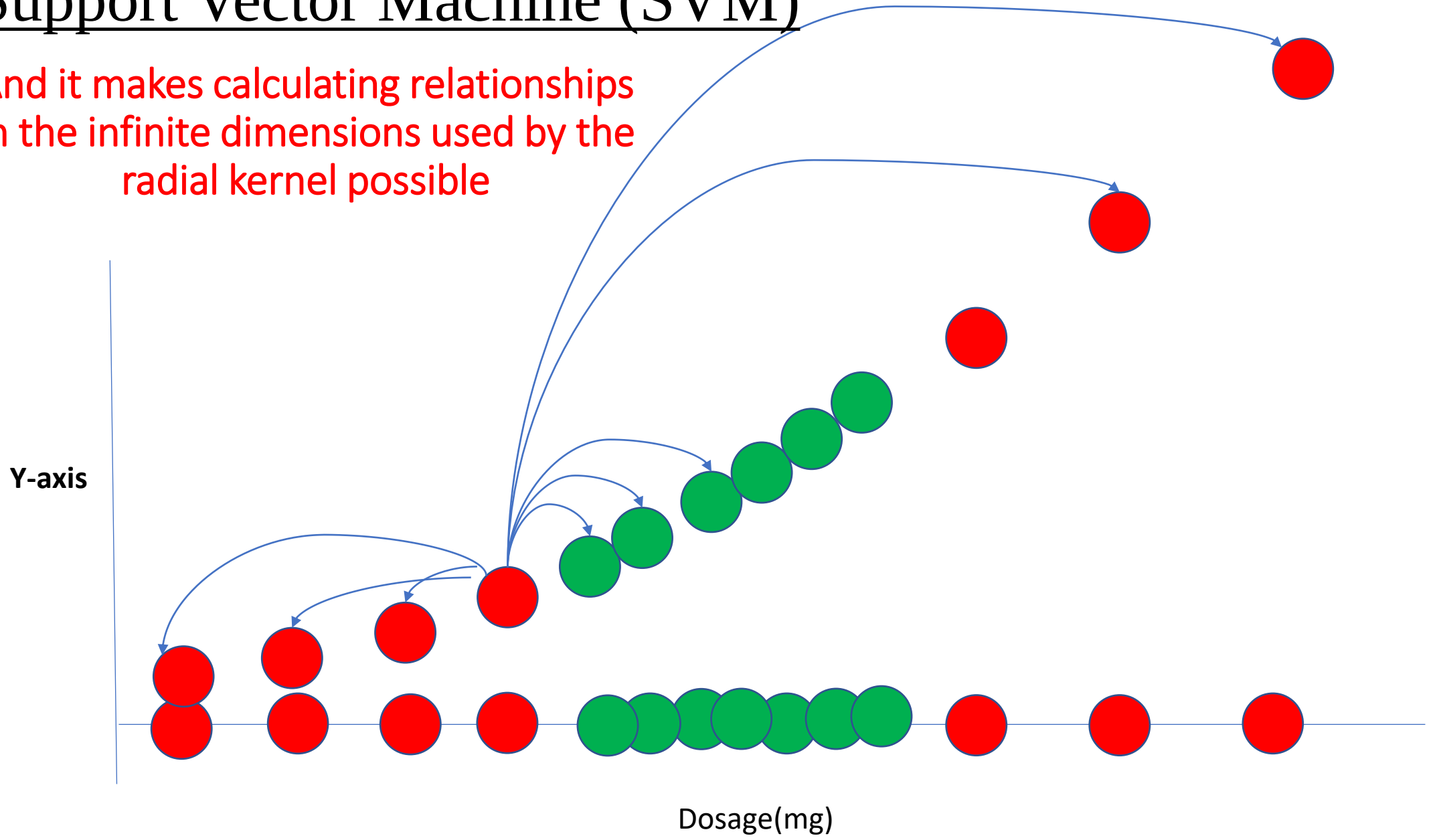
Support Vector Machine (SVM)

The kernel trick reduces the amount of computation required for support vector machines by avoiding the math that transforms the data from low to high dimension



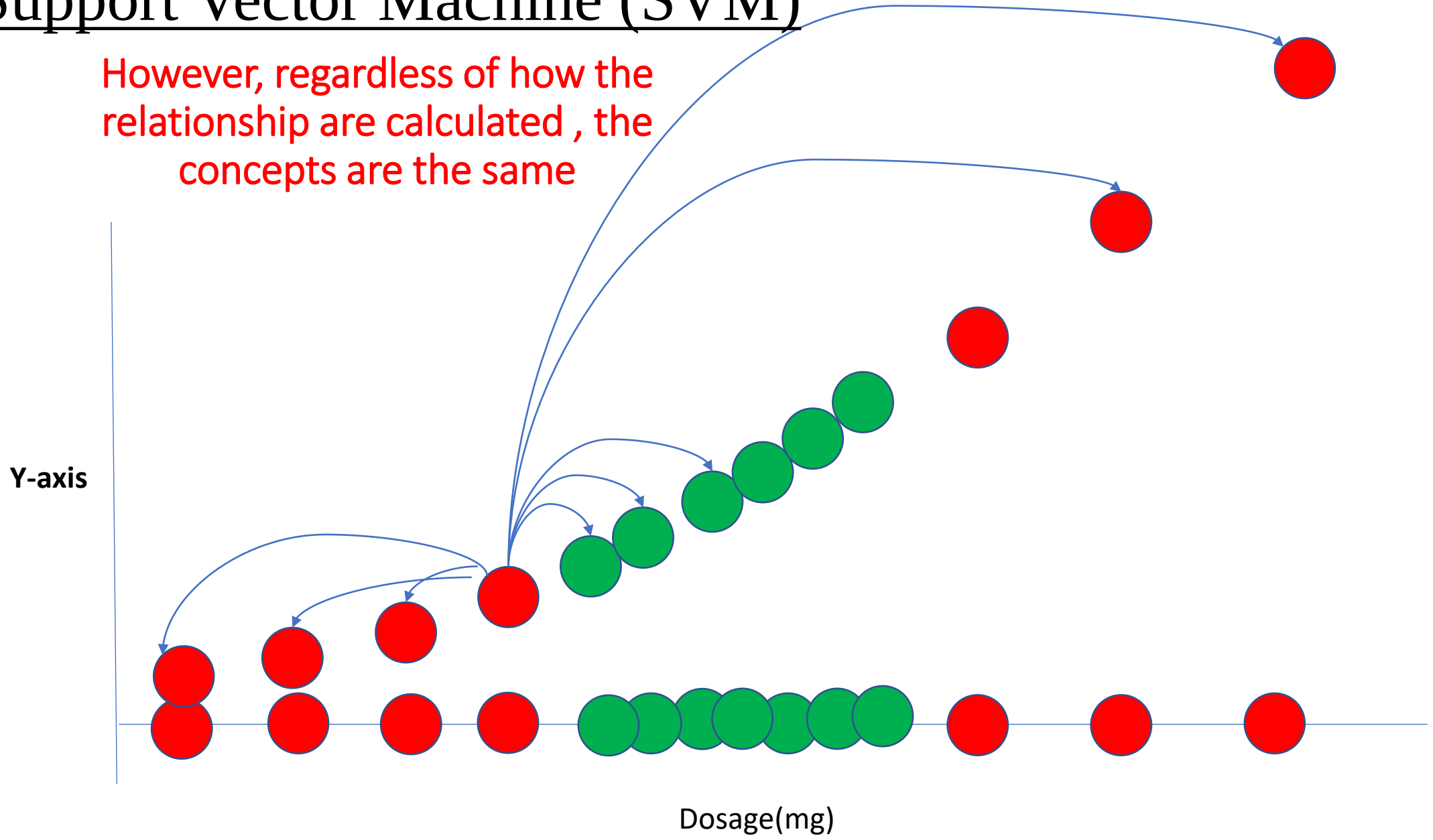
Support Vector Machine (SVM)

And it makes calculating relationships in the infinite dimensions used by the radial kernel possible



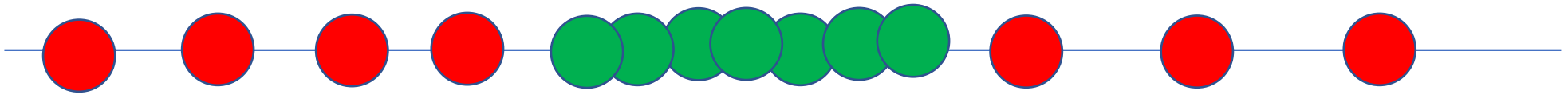
Support Vector Machine (SVM)

However, regardless of how the relationship are calculated , the concepts are the same



Support Vector Machine (SVM)

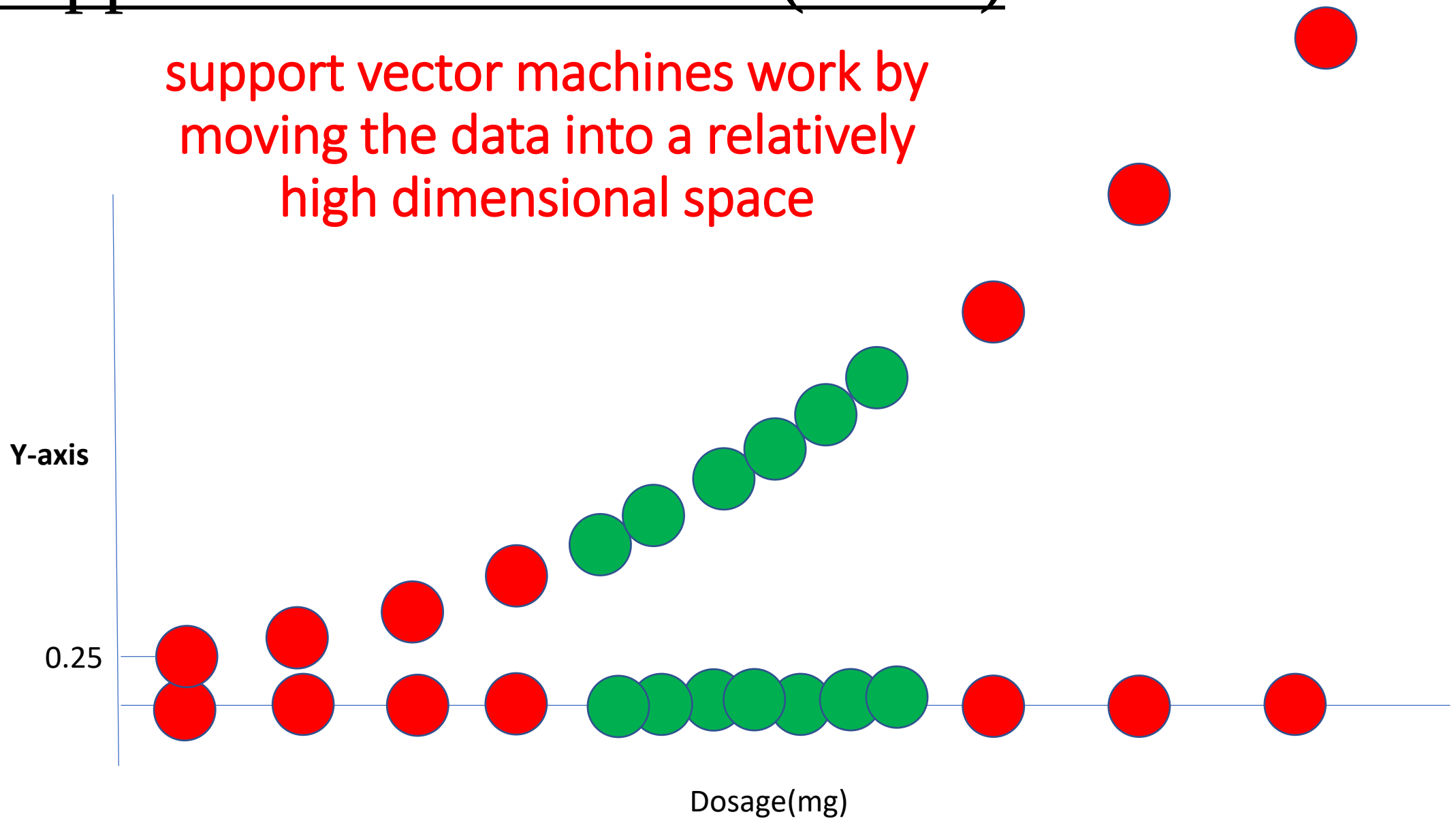
when we have 2 categories, but no obvious linear classifier that separates them in a nice way.



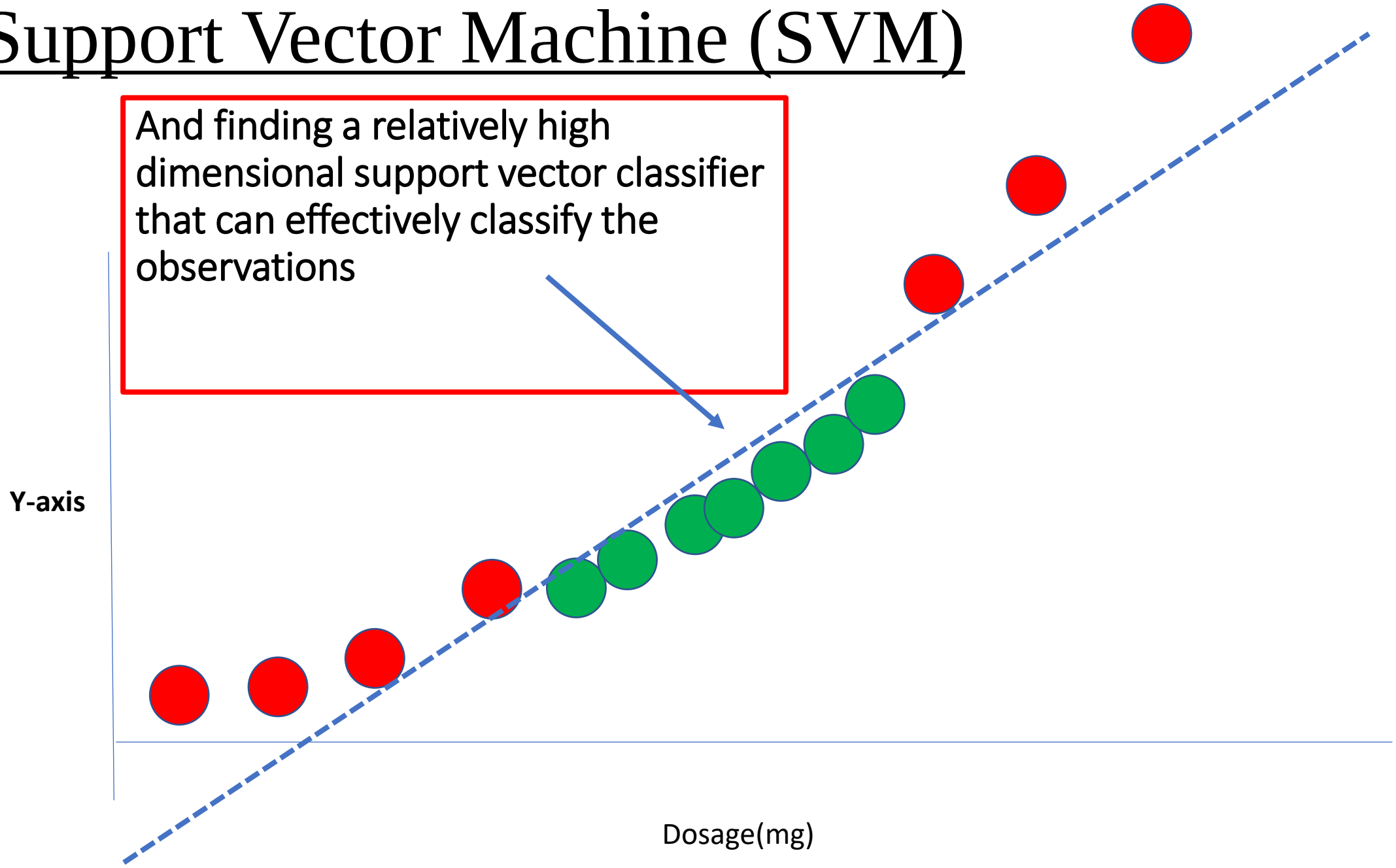
Dosage(mg)

Support Vector Machine (SVM)

support vector machines work by moving the data into a relatively high dimensional space

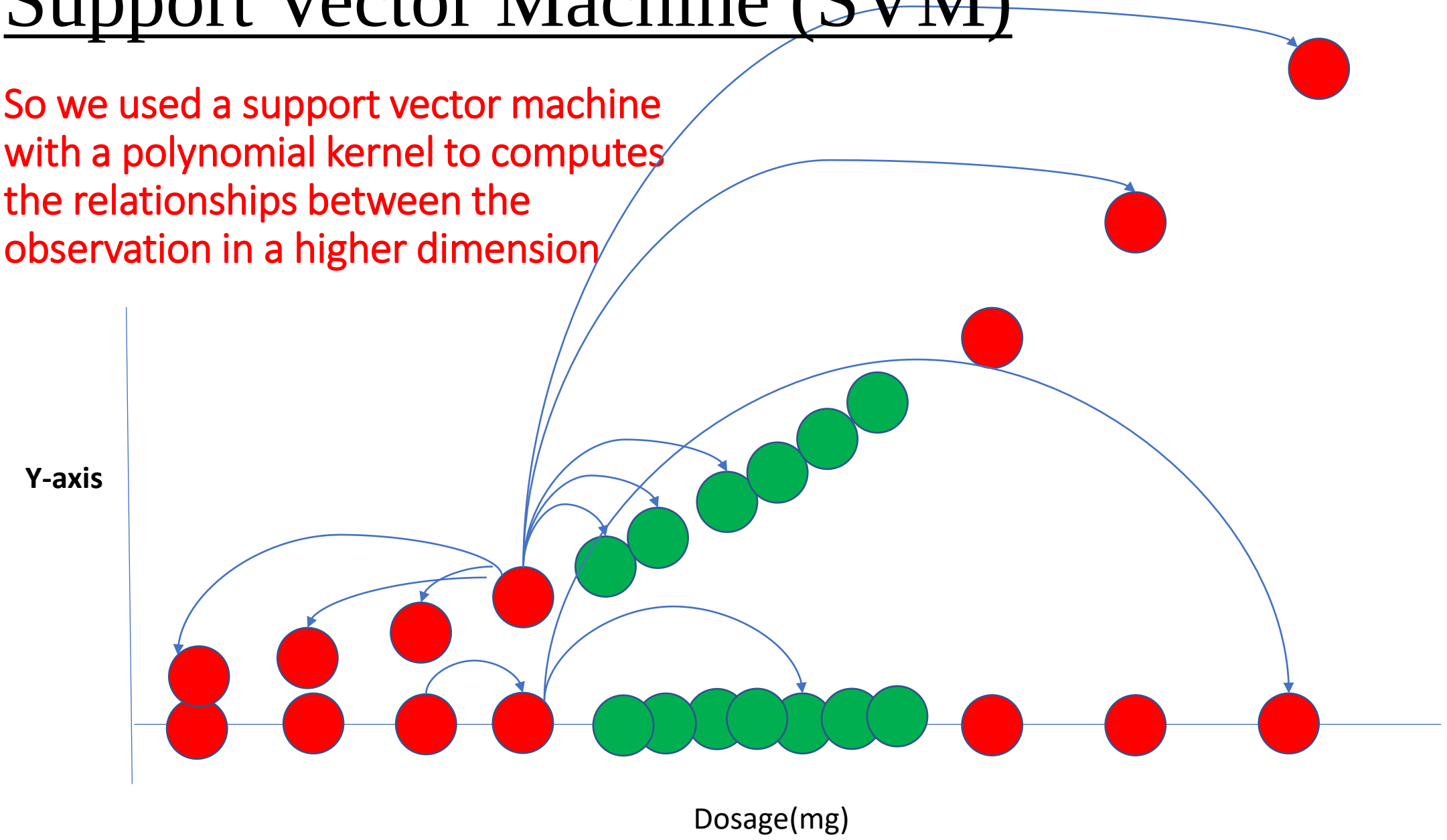


Support Vector Machine (SVM)



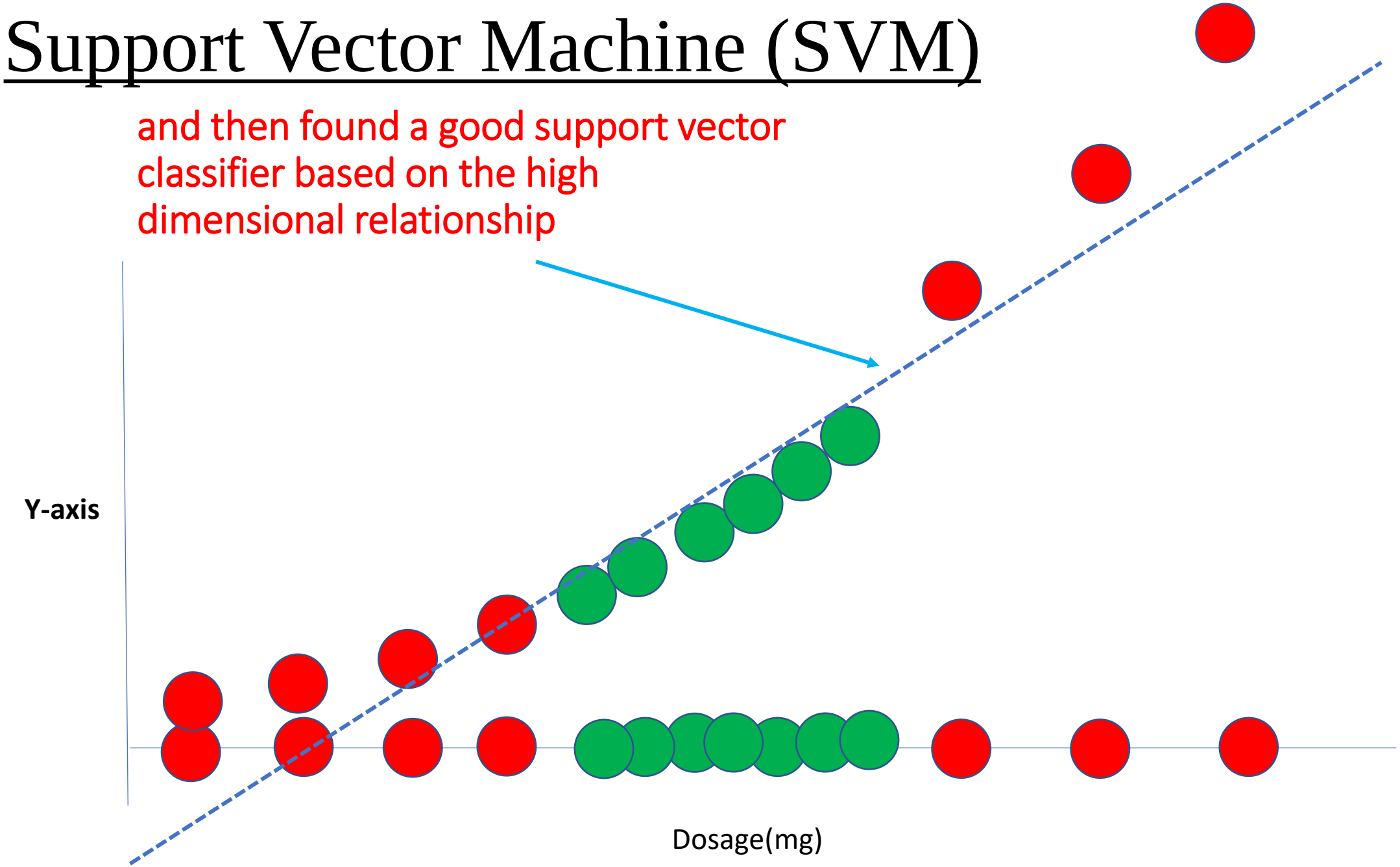
Support Vector Machine (SVM)

So we used a support vector machine with a polynomial kernel to computes the relationships between the observation in a higher dimension



Support Vector Machine (SVM)

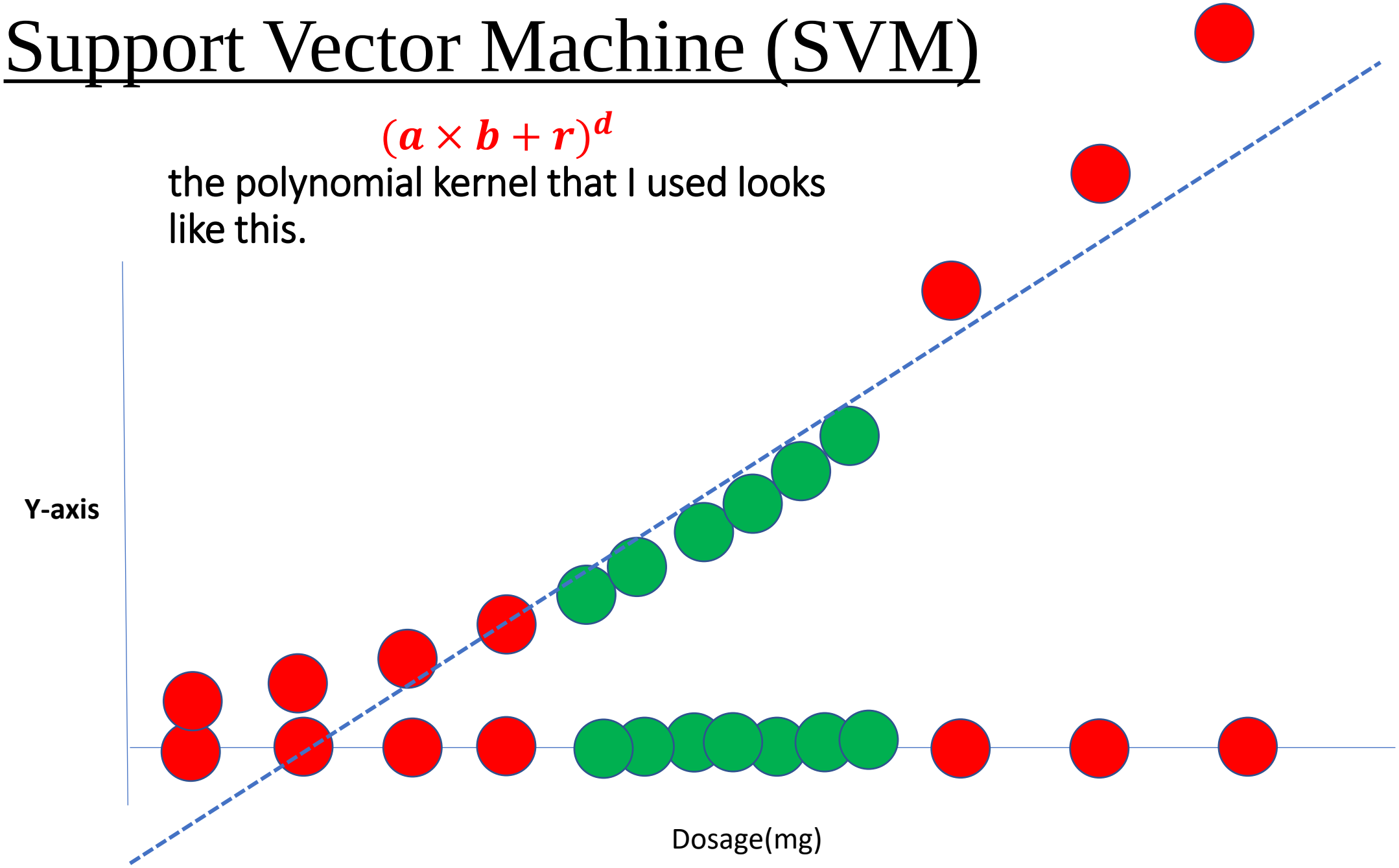
and then found a good support vector classifier based on the high dimensional relationship



Support Vector Machine (SVM)

$$(a \times b + r)^d$$

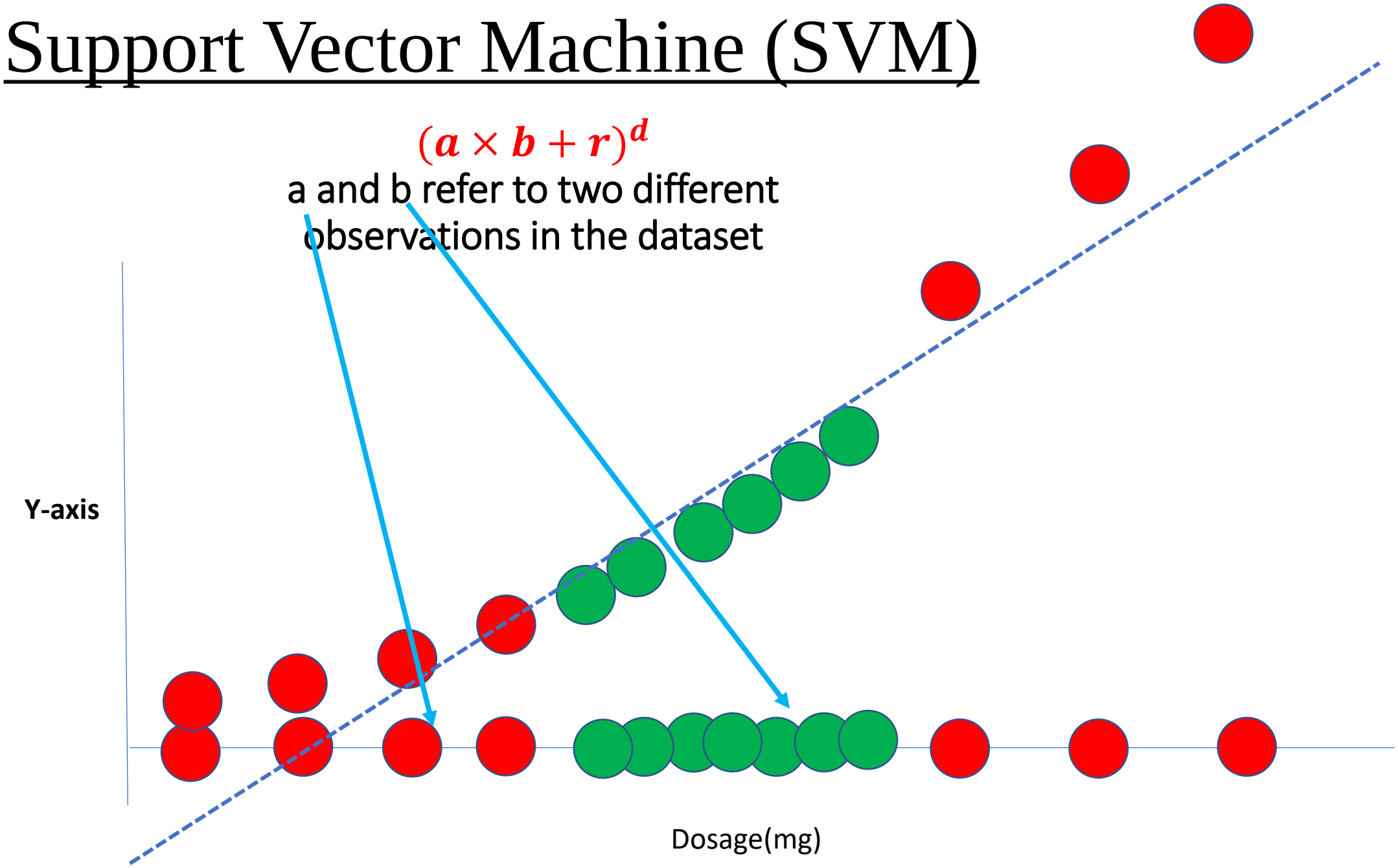
the polynomial kernel that I used looks like this.



Support Vector Machine (SVM)

$$(a \times b + r)^d$$

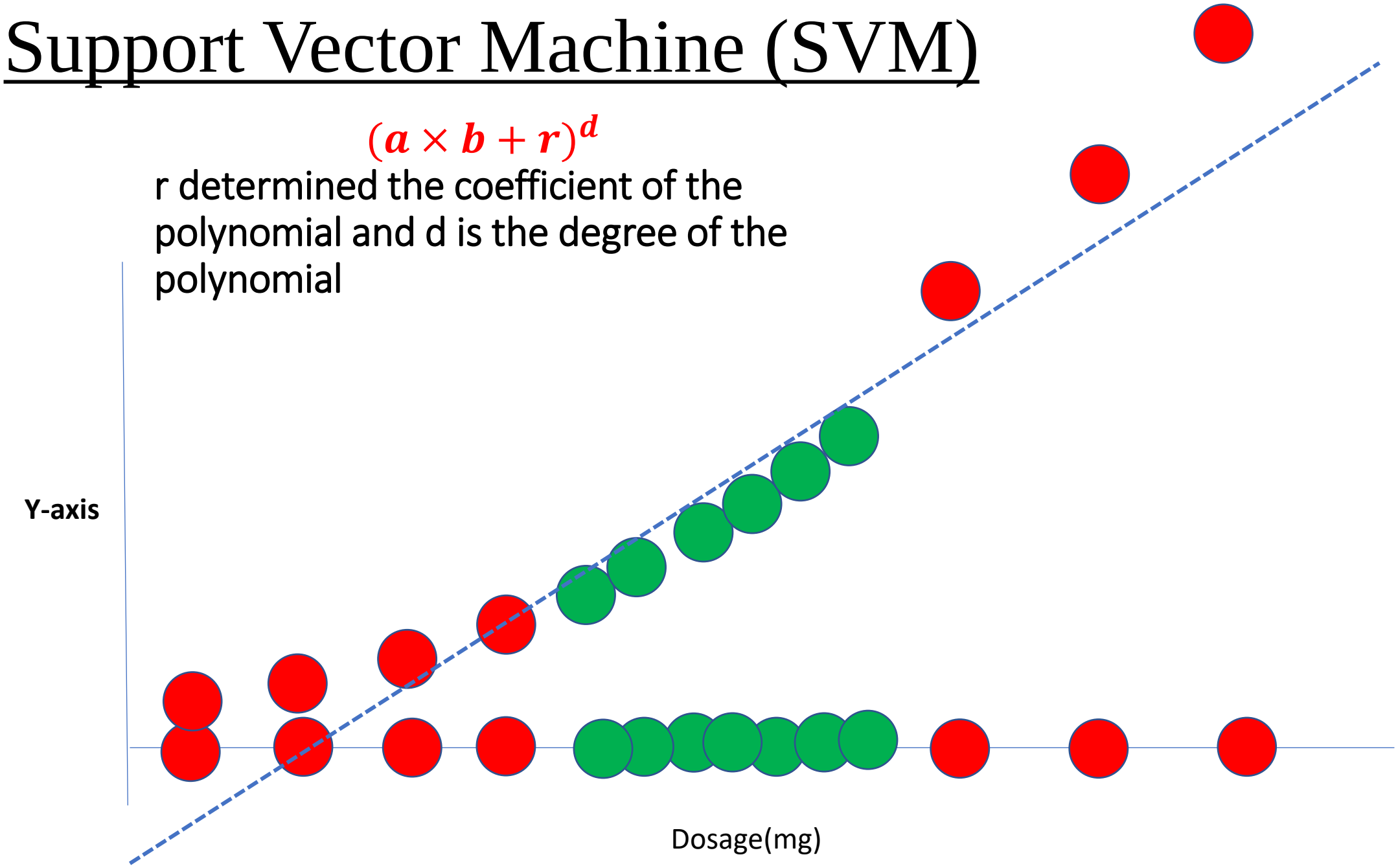
a and b refer to two different observations in the dataset



Support Vector Machine (SVM)

$$(a \times b + r)^d$$

r determined the coefficient of the polynomial and d is the degree of the polynomial

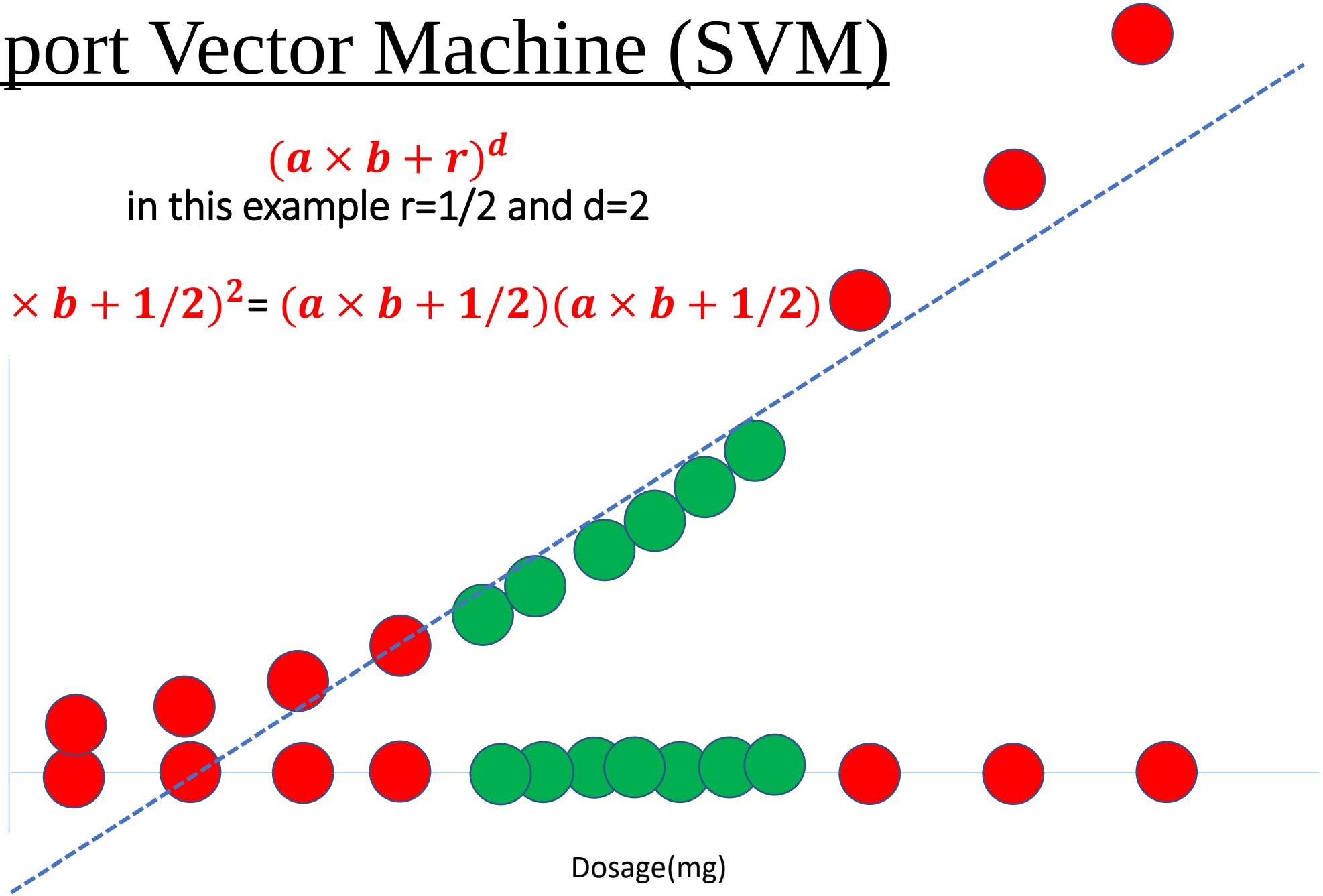


Support Vector Machine (SVM)

$$(a \times b + r)^d$$

in this example r=1/2 and d=2

$$(a \times b + 1/2)^2 = (a \times b + 1/2)(a \times b + 1/2)$$



Support Vector Machine (SVM)

$$(a \times b + r)^d$$

in this example r=1/2 and d=2

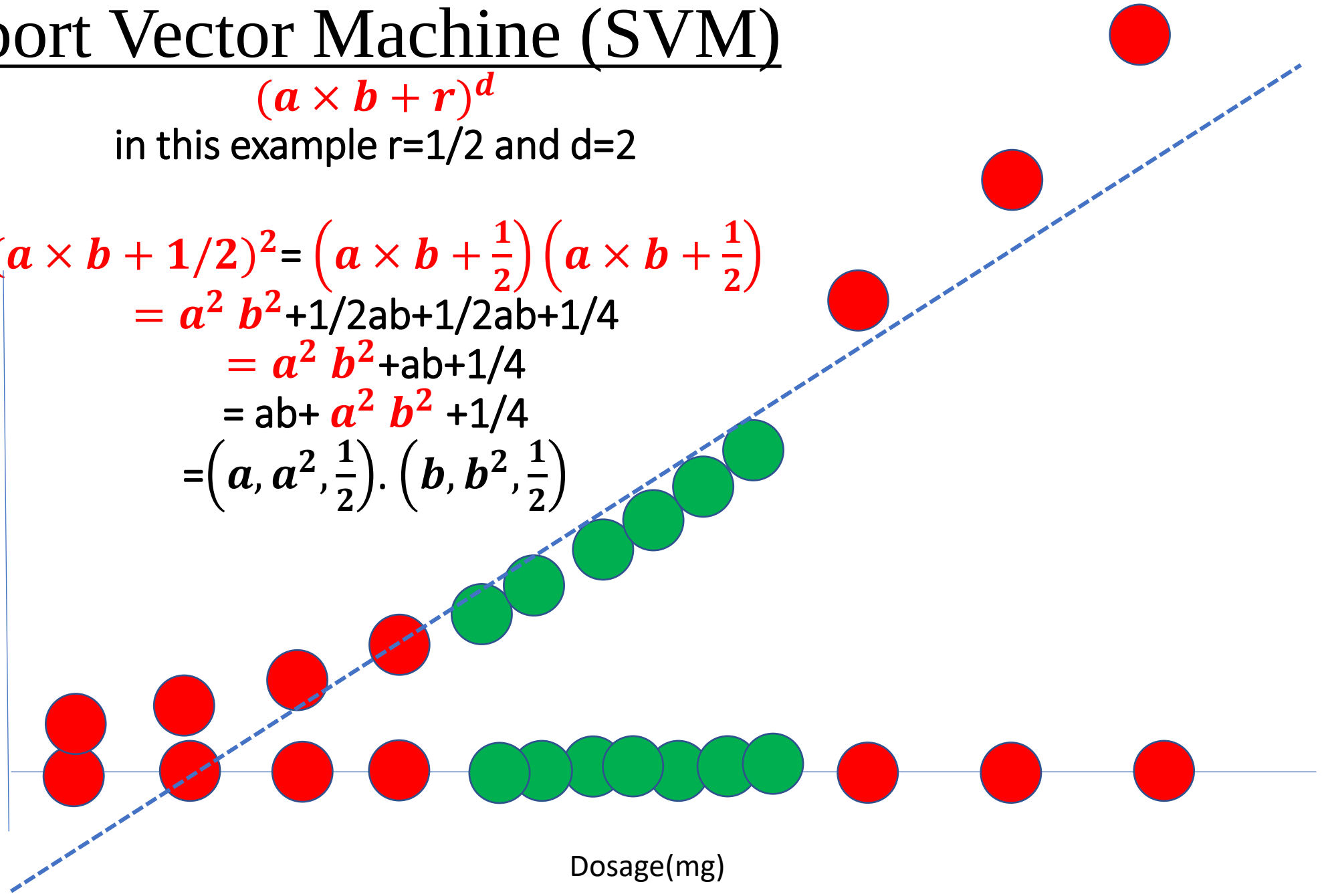
$$(a \times b + 1/2)^2 = \left(a \times b + \frac{1}{2}\right)\left(a \times b + \frac{1}{2}\right)$$

$$= a^2\, b^2 + 1/2ab + 1/2ab + 1/4$$

$$= a^2\, b^2 + ab + 1/4$$

$$= ab + a^2\, b^2 + 1/4$$

$$= \left(a, a^2, \frac{1}{2}\right).\left(b, b^2, \frac{1}{2}\right)$$



Support Vector Machine (SVM)

$(a \times b + 1/2)^2 = \left(a \times b + \frac{1}{2}\right)\left(a \times b + \frac{1}{2}\right)$

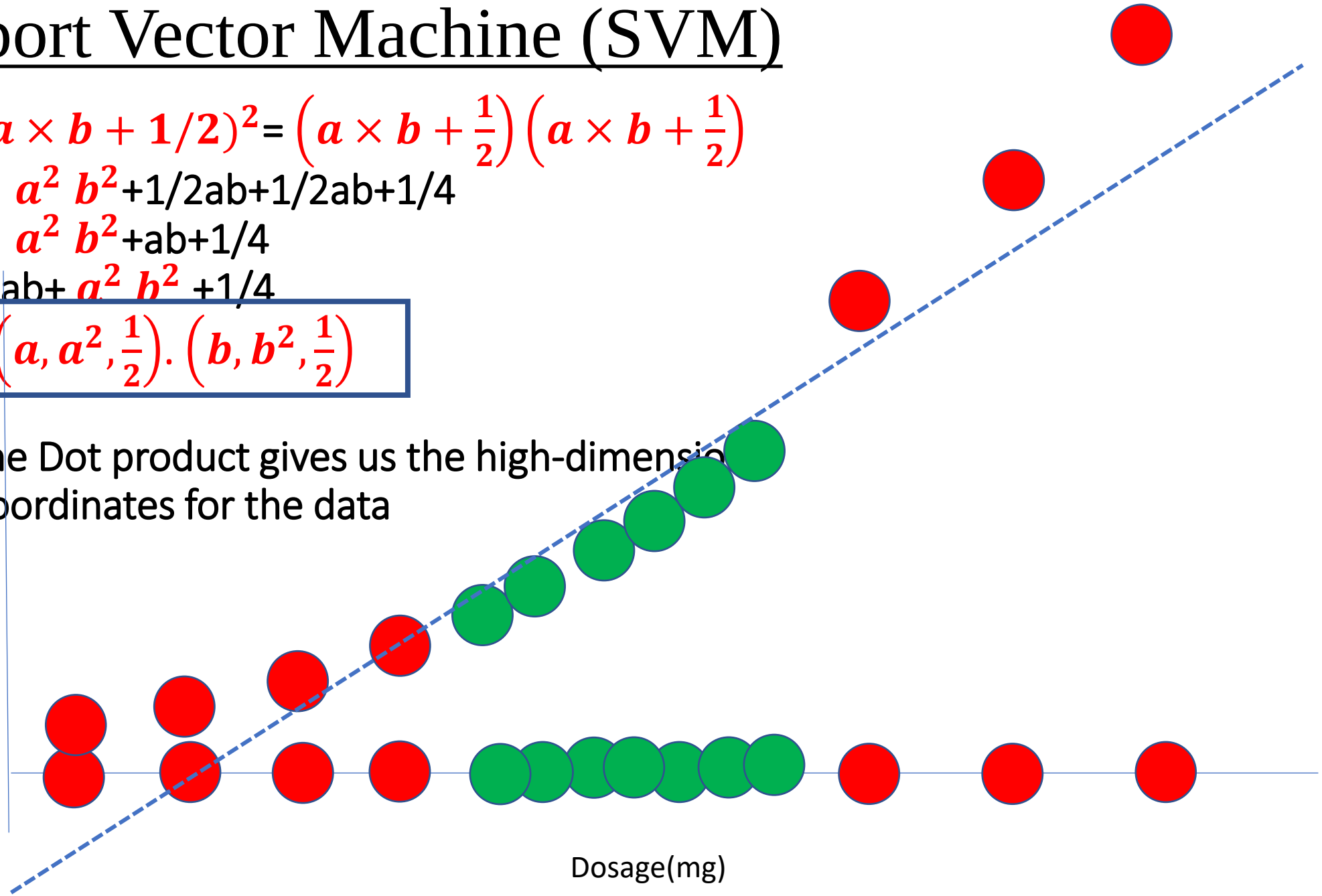
$= a^2\ b^2$+1/2ab+1/2ab+1/4

$= a^2\ b^2$+ab+1/4

= ab+ $a^2\ b^2$ +1/4

$= \left(a, a^2, \frac{1}{2}\right).\left(b, b^2, \frac{1}{2}\right)$

the Dot product gives us the high-dimensio
coordinates for the data



Support Vector Machine (SVM)

$(a \times b + 1/2)^2 = \left(a \times b + \frac{1}{2}\right)\left(a \times b + \frac{1}{2}\right)$

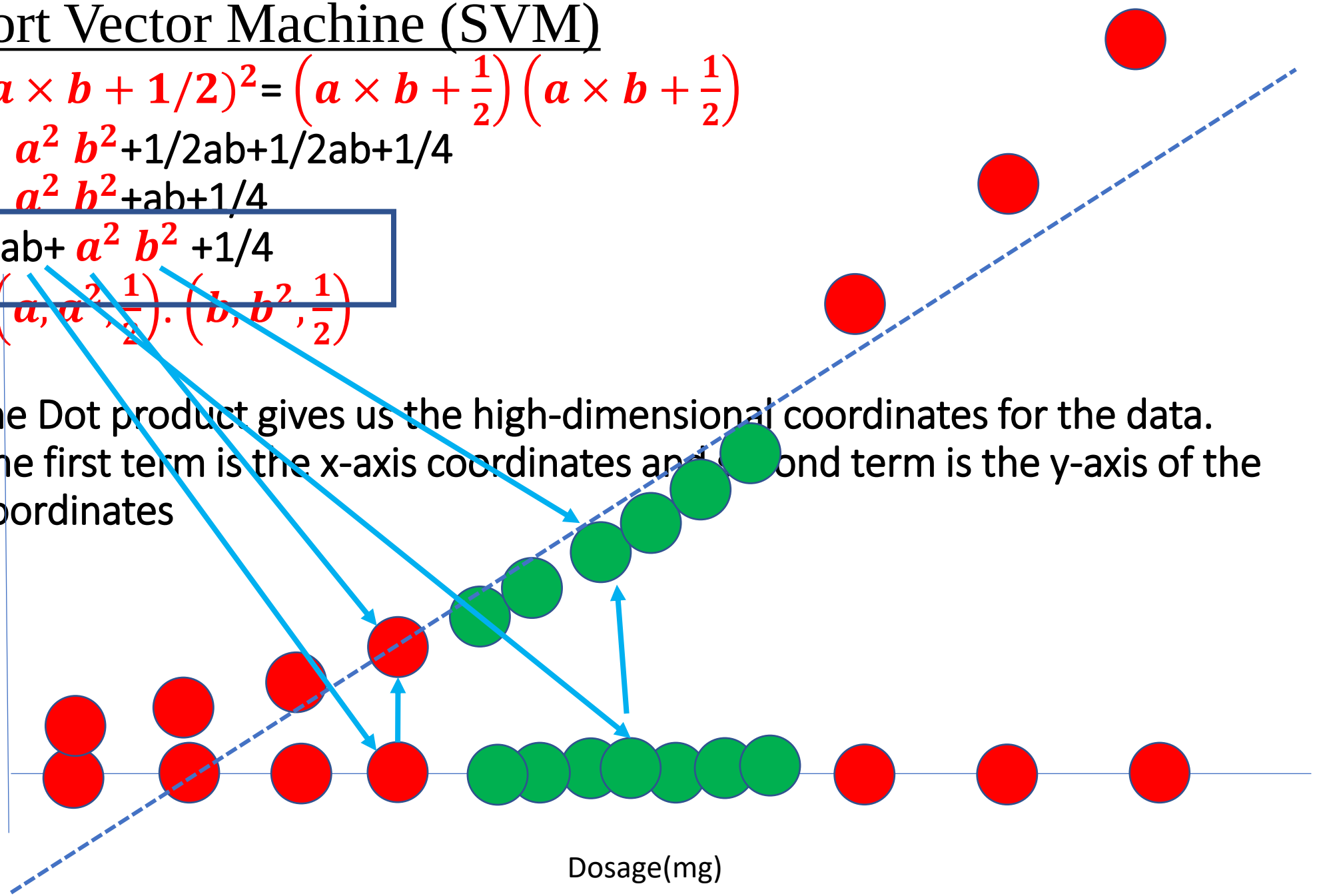
$= a^2\ b^2$+1/2ab+1/2ab+1/4

$= a^2\ b^2$+ab+1/4

= ab+ $a^2\ b^2$ +1/4

$= \left(a, a^2, \frac{1}{2}\right).\left(b, b^2, \frac{1}{2}\right)$

the Dot product gives us the high-dimensional coordinates for the data. The first term is the x-axis coordinates and second term is the y-axis of the coordinates



Support Vector Machine (SVM)

$(a \times b + 1/2)^2 = \left(a \times b + \frac{1}{2}\right)\left(a \times b + \frac{1}{2}\right)$

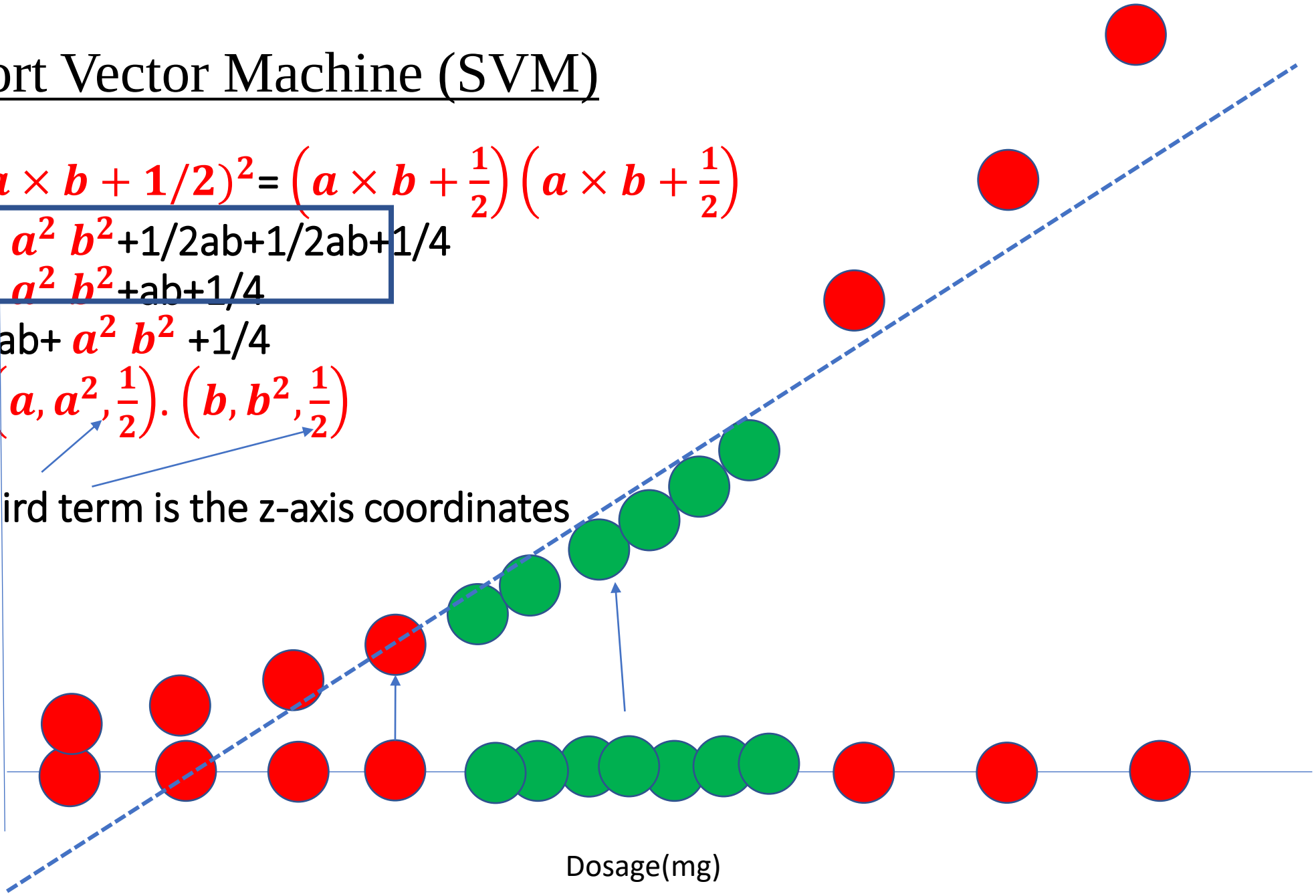
$= a^2\ b^2$+1/2ab+1/2ab+1/4

$= a^2\ b^2$+ab+1/4

= ab+ $a^2\ b^2$ +1/4

$= \left(a, a^2, \frac{1}{2}\right).\left(b, b^2, \frac{1}{2}\right)$

third term is the z-axis coordinates



Support Vector Machine (SVM)

$(a \times b + 1/2)^2 = \left(a \times b + \frac{1}{2}\right)\left(a \times b + \frac{1}{2}\right)$

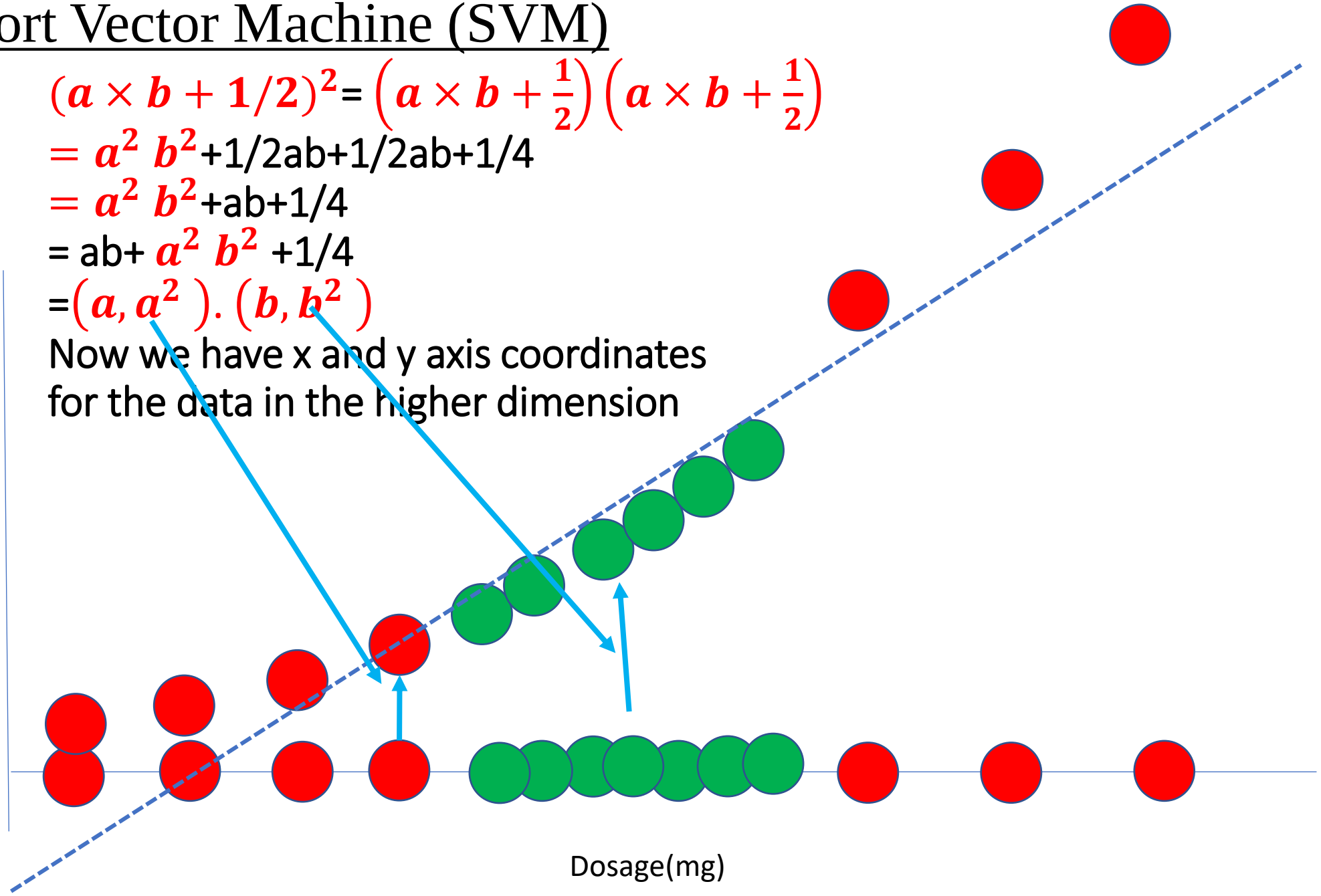
$= a^2\ b^2$+1/2ab+1/2ab+1/4

$= a^2\ b^2$+ab+1/4

= ab+ $a^2\ b^2$ +1/4

$=(a, a^2\ ).\ (b, b^2\ )$

Now we have x and y axis coordinates for the data in the higher dimension



Support Vector Machine (SVM)

$(a \times b + 1/2)^2 = \left(a \times b + \frac{1}{2}\right)\left(a \times b + \frac{1}{2}\right)$

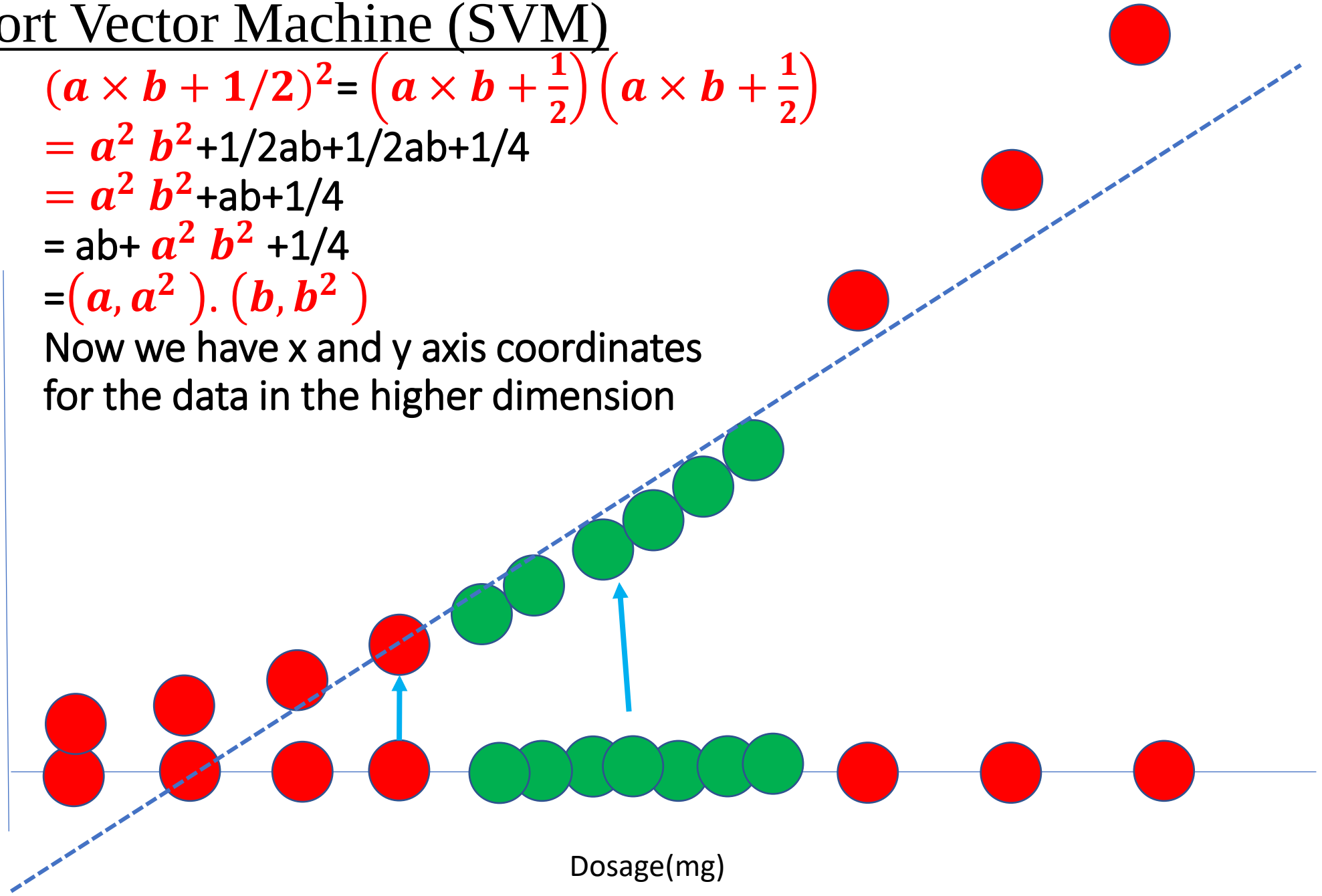
$= a^2\ b^2$+1/2ab+1/2ab+1/4

$= a^2\ b^2$+ab+1/4

= ab+ $a^2\ b^2$ +1/4

$=\left(a, a^2\right).\left(b, b^2\right)$

Now we have x and y axis coordinates for the data in the higher dimension



Support Vector Machine (SVM)

$(a \times b + r)^d$

r determined the r=1 and d=2

$(a \times b + 1)^2 = (a \times b + 1)(a \times b + 1)$

$= a^2\ b^2$+ab+ab+1

$= a^2\ b^2$+2ab+1

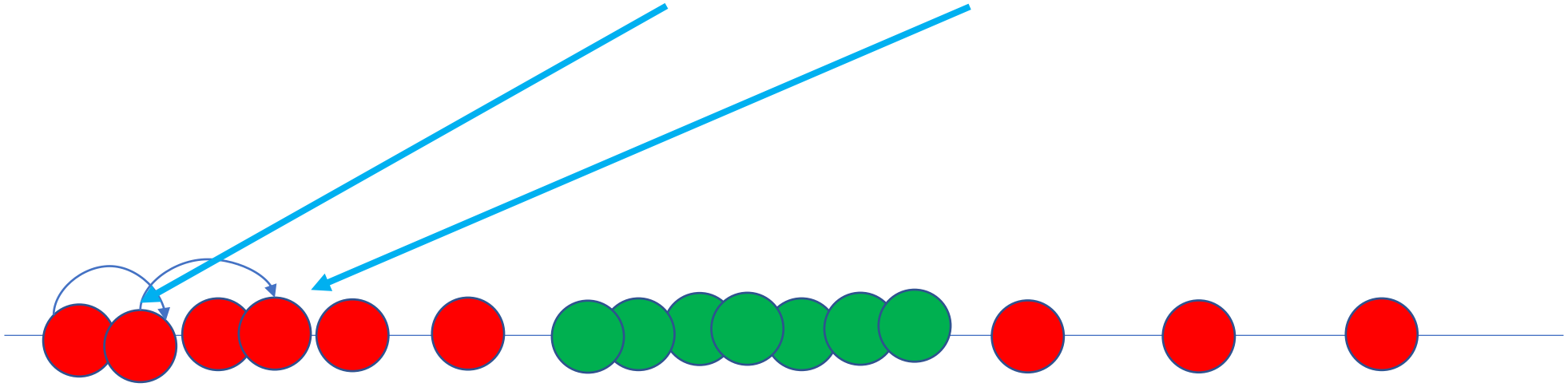
$= 2ab + a^2\ b^2$+1

$=(\sqrt{2}a, a^2, 1).(\sqrt{2}b, b^2, 1)$

this is dot product

the new axis coordinates are the square root of 2 times the original dosage values .

So we move the points on the x-axis over by a factor of the square root of 2

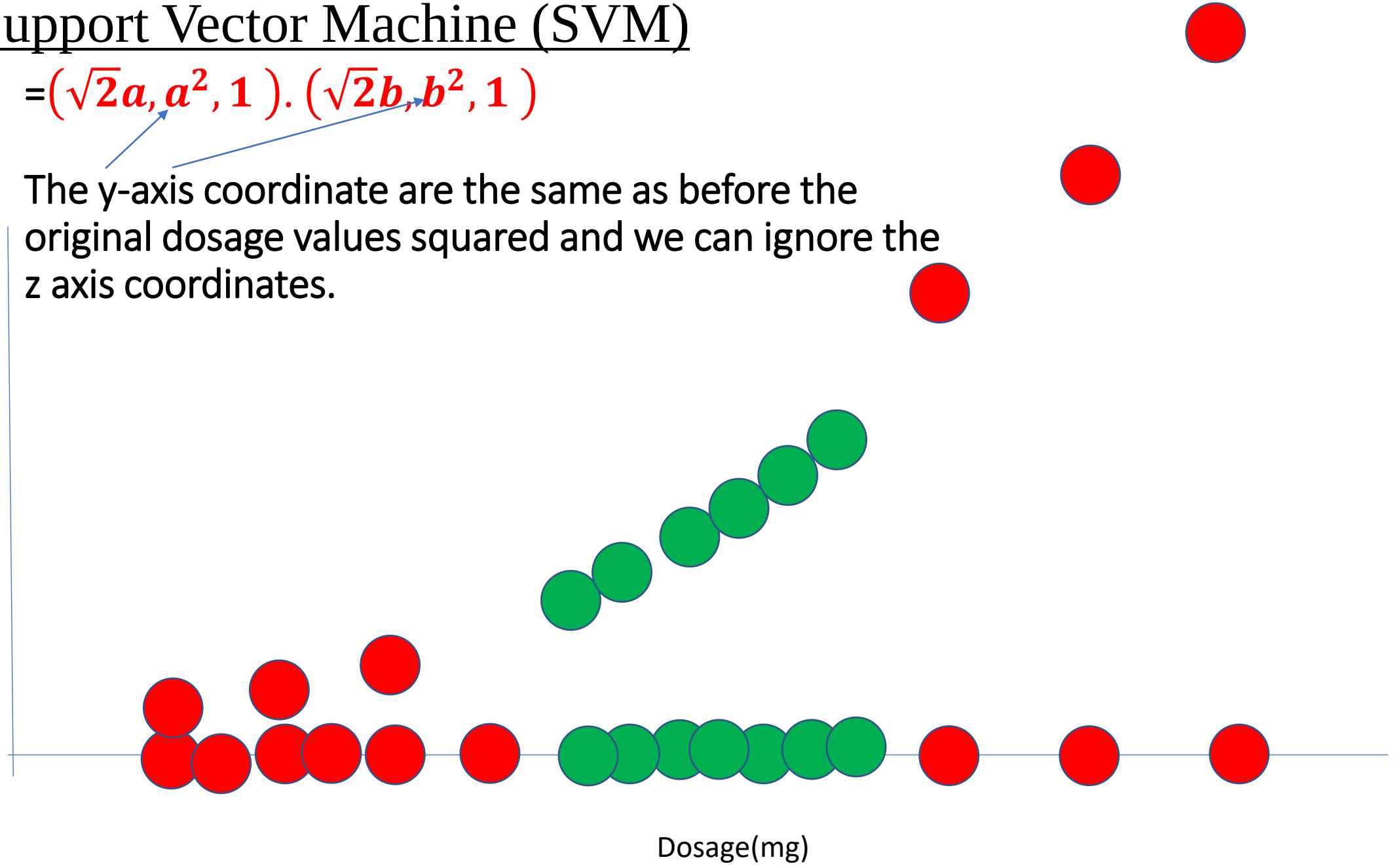


Dosage(mg)

Support Vector Machine (SVM)

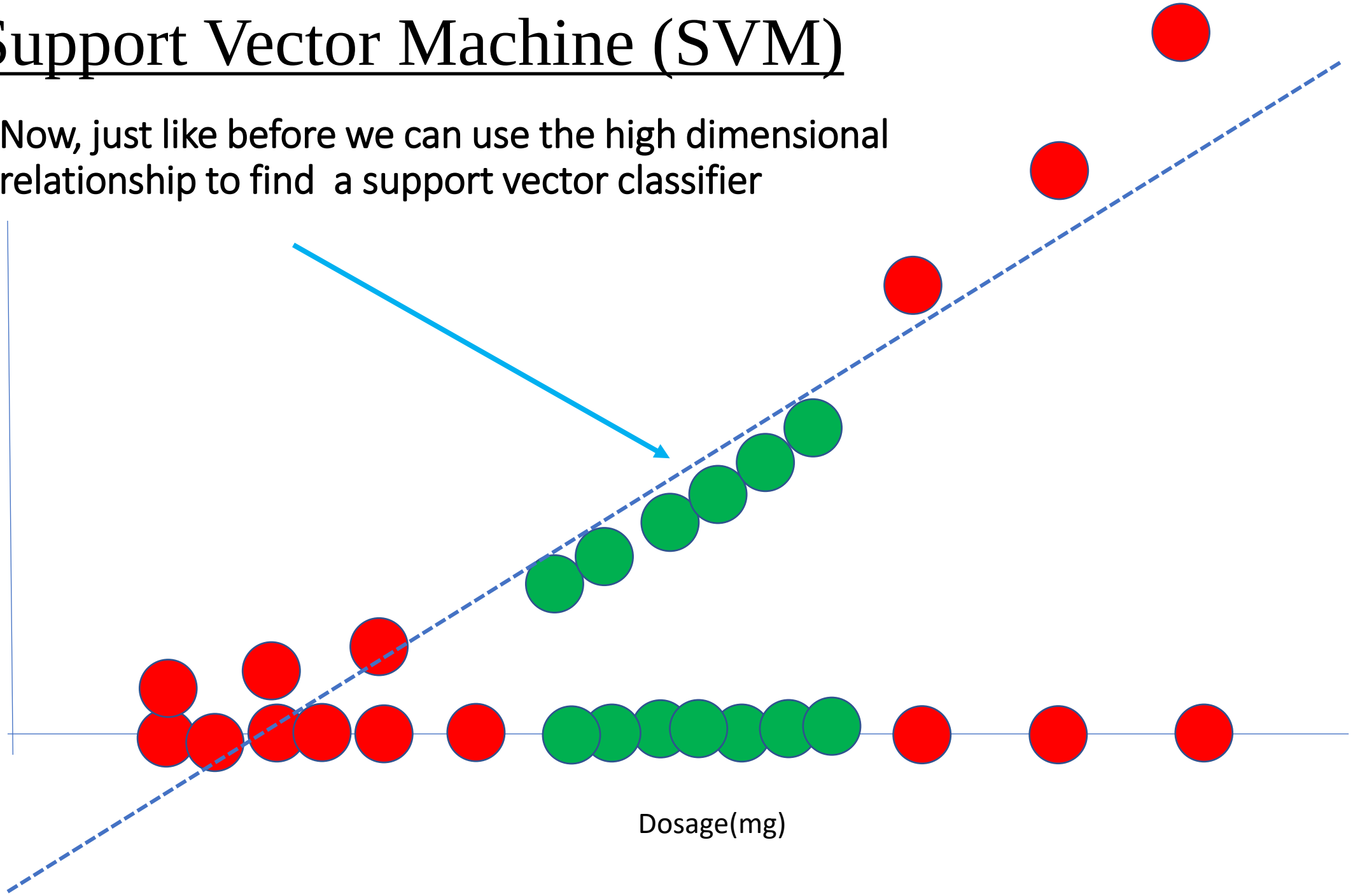
$=(\sqrt{2}a, a^2, 1).(\sqrt{2}b, b^2, 1)$

The y-axis coordinate are the same as before the original dosage values squared and we can ignore the z axis coordinates.



Support Vector Machine (SVM)

Now, just like before we can use the high dimensional relationship to find a support vector classifier

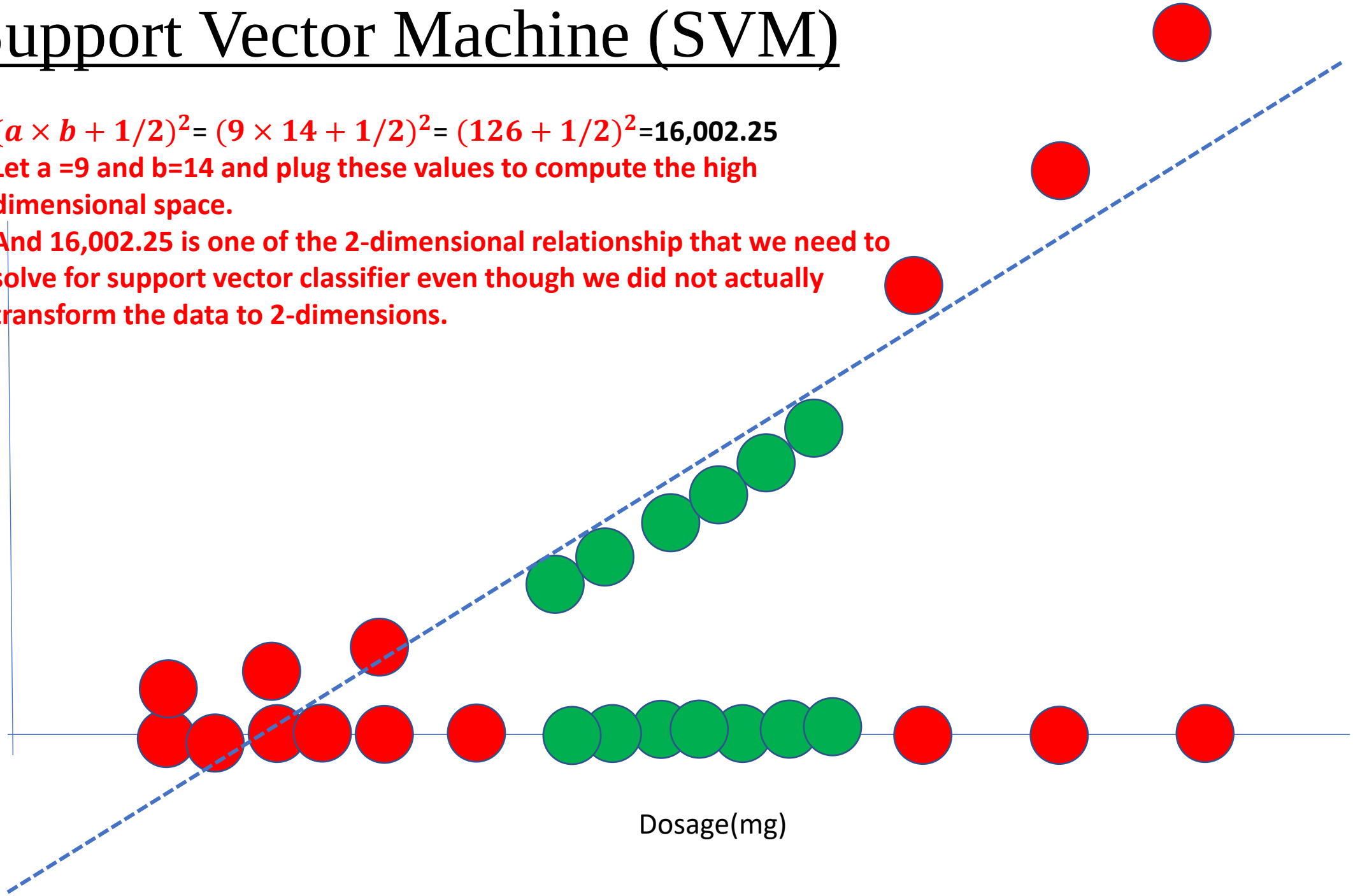


Support Vector Machine (SVM)

$(a \times b + 1/2)^2 = (9 \times 14 + 1/2)^2 = (126 + 1/2)^2$ **=16,002.25**

Let a =9 and b=14 and plug these values to compute the high dimensional space.

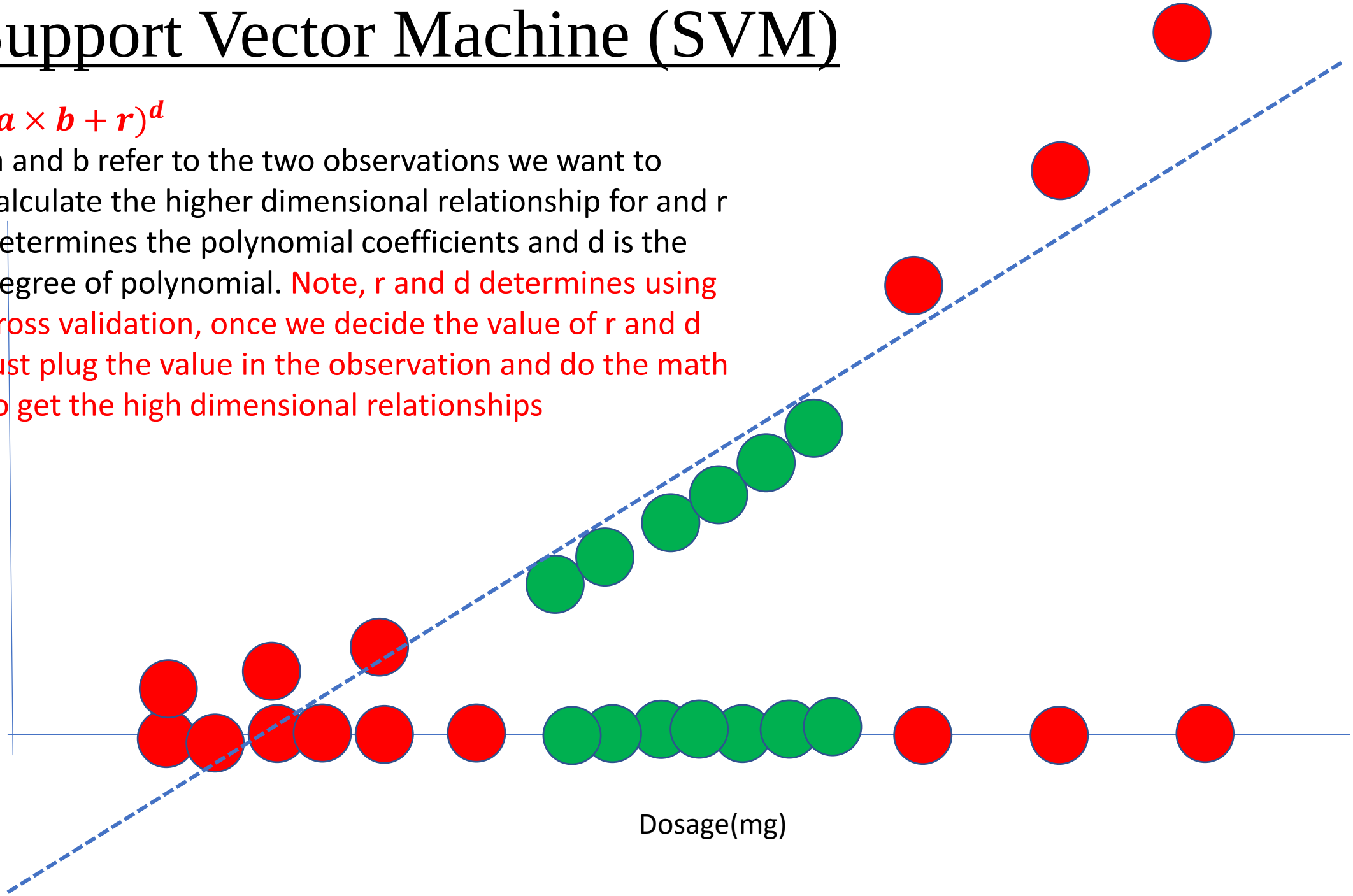
And 16,002.25 is one of the 2-dimensional relationship that we need to solve for support vector classifier even though we did not actually transform the data to 2-dimensions.



Support Vector Machine (SVM)

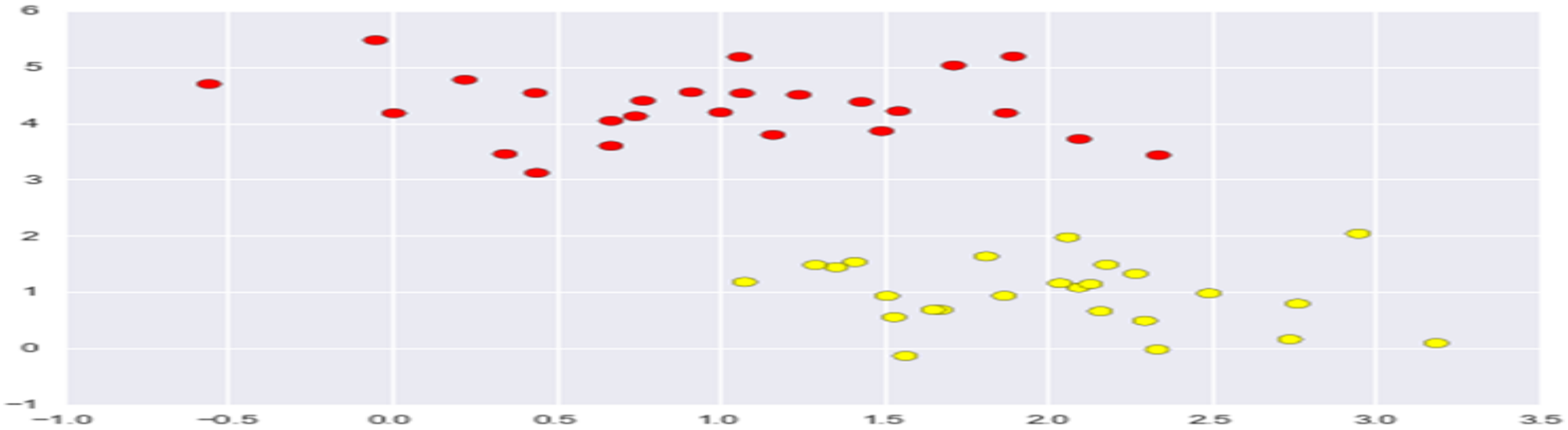
$(\boldsymbol{a} \times \boldsymbol{b} + \boldsymbol{r})^{\boldsymbol{d}}$

a and b refer to the two observations we want to calculate the higher dimensional relationship for and r determines the polynomial coefficients and d is the degree of polynomial. Note, r and d determines using cross validation, once we decide the value of r and d just plug the value in the observation and do the math to get the high dimensional relationships



SVM Example

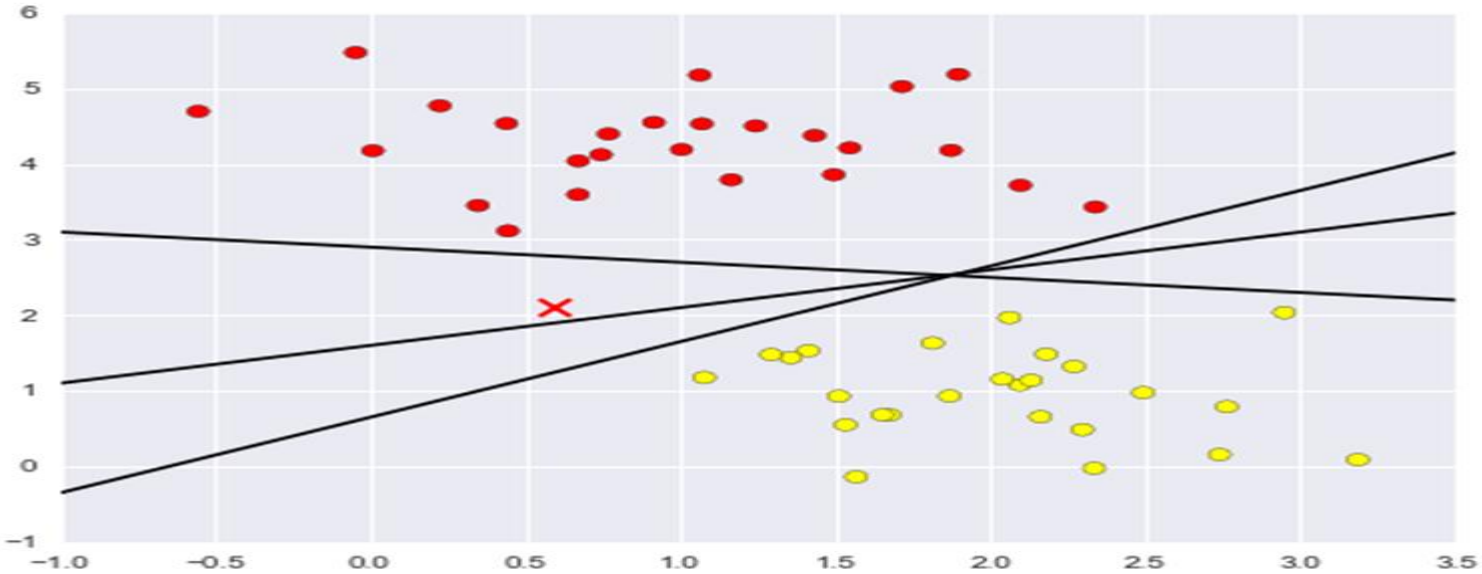
- Support vector machines (SVMs) are a particularly powerful and flexible class of supervised algorithms for both classification and regression.
- we simply find a line or curve (in two dimensions) or manifold (in multiple dimensions) that divides the classes from each other.
- As an example of this, consider the simple case of a classification task, in which the two classes of points are well separated.



SVM Example

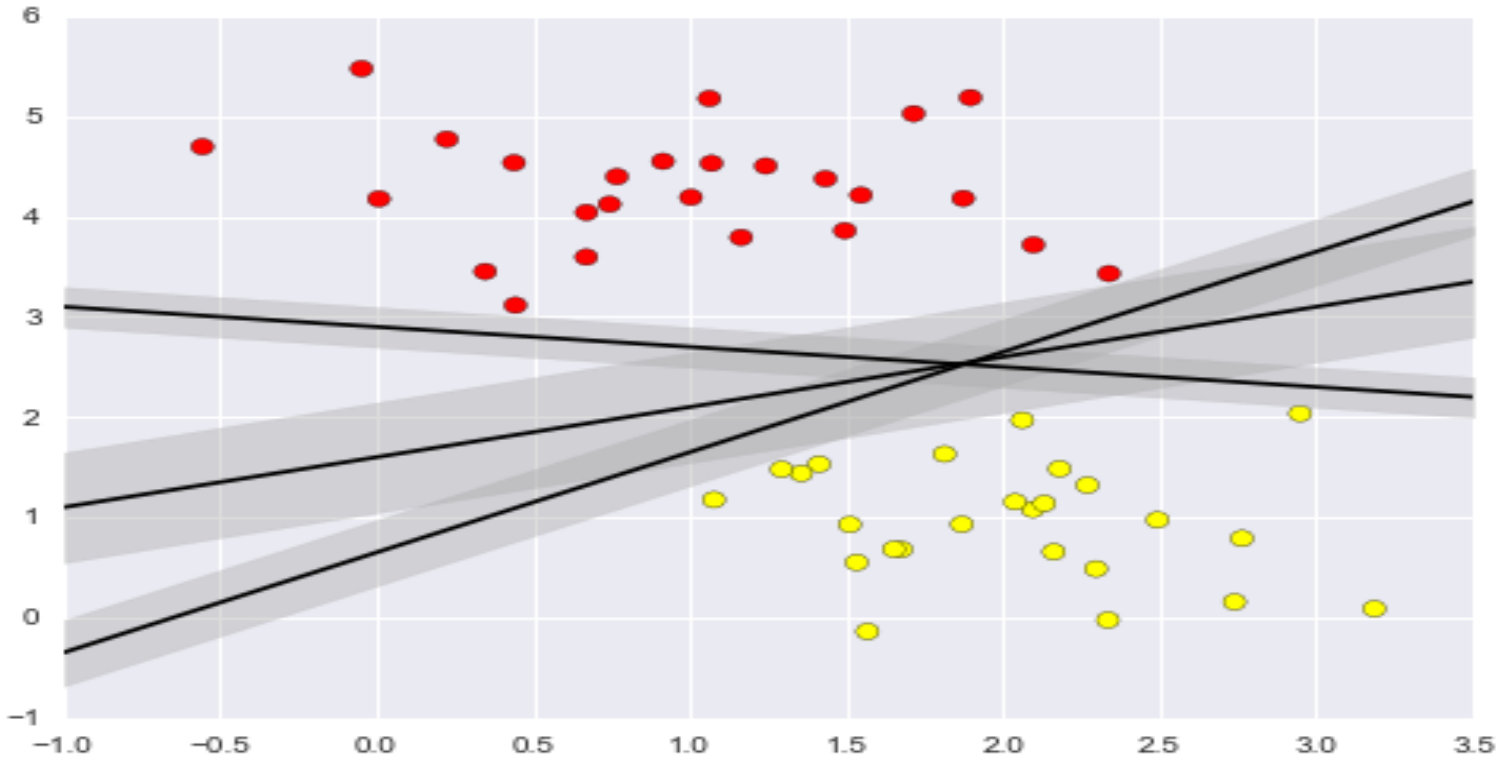
- A linear discriminative classifier would attempt to draw a straight line separating the two sets of data, and thereby create a model for classification. For two dimensional data like that shown in Figure,
- But immediately we see a problem: there is more than one possible dividing line that can perfectly discriminate between the two classes!

- These are three very different separators which, nevertheless, perfectly discriminate between these samples. Depending on which you choose, a new data point (e.g., the one marked by the "X" in this plot) will be assigned a different label!



SVM Example

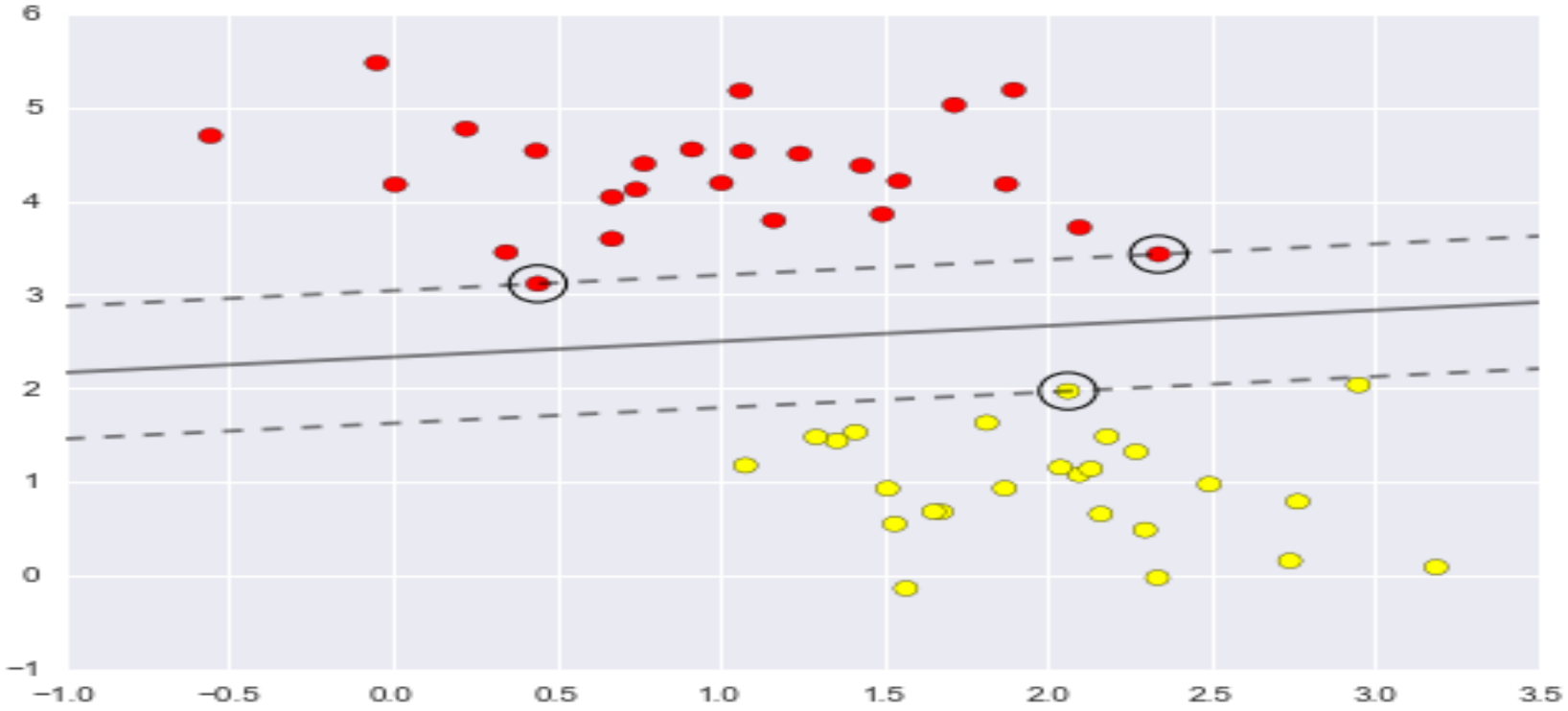
- **Maximizing the Margin**
- Support vector machines offer one way to improve on this.
- The intuition is this: rather than simply drawing a zero-width line between the classes, we can draw around each line a margin of some width, up to the nearest point.
- Here is an example of how this might look:



SVM Example

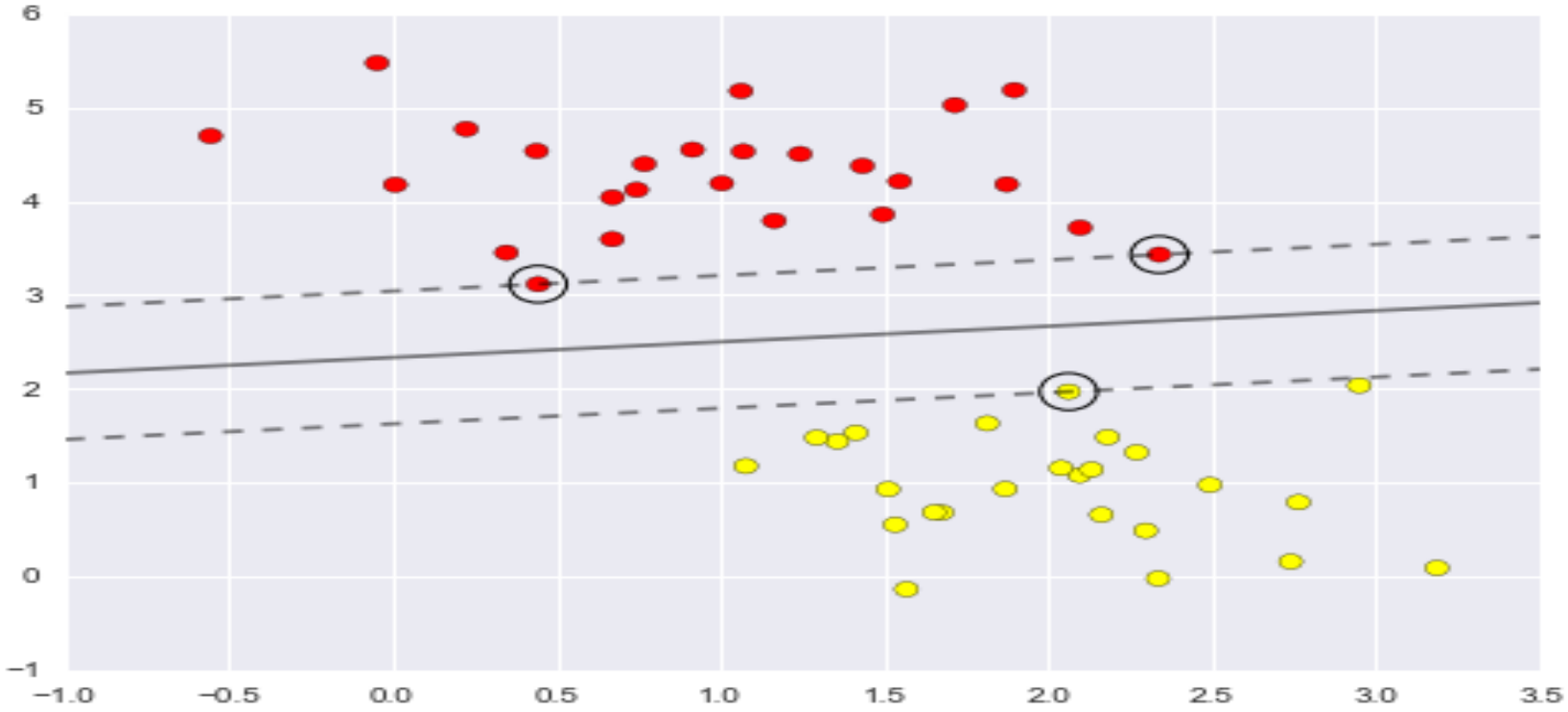
- **Fitting a support vector machine**
- For the time being, we will use a linear kernel and set the C parameter to a very large number (we'll discuss the meaning of these in more depth momentarily).

- To better visualize what's happening here, let's create a quick convenience function that will plot SVM decision boundaries for us.



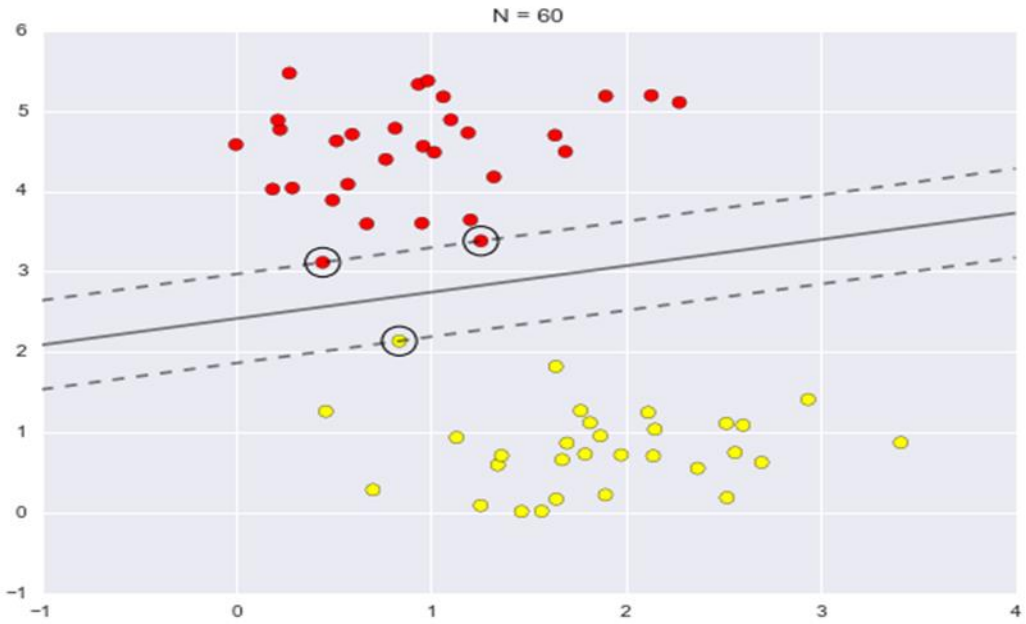
SVM Example

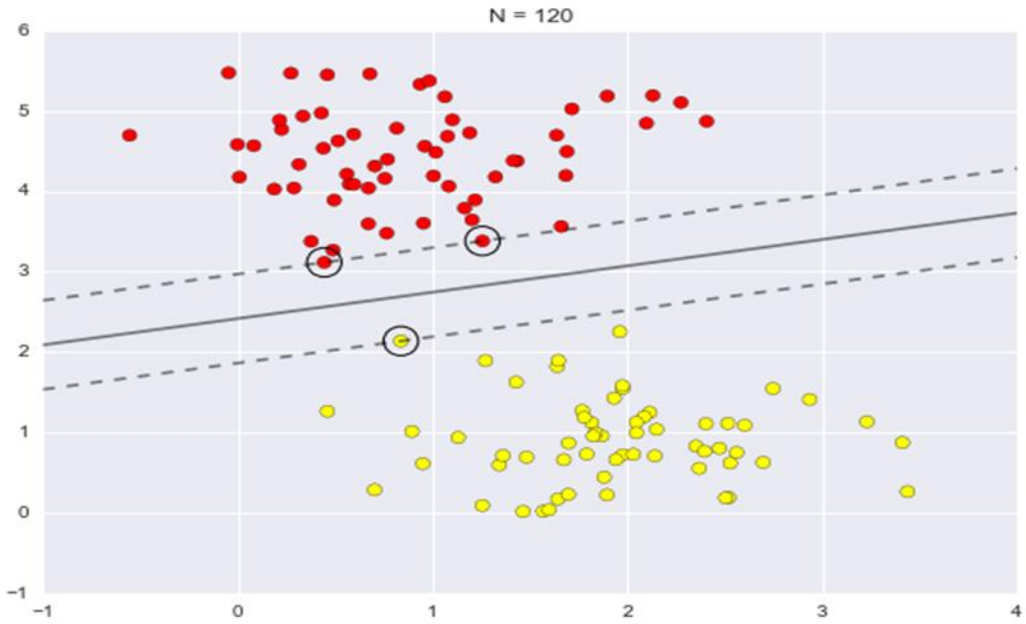
- **Fitting a support vector machine**
- This is the dividing line that maximizes the margin between the two sets of points.
- Notice that a few of the training points just touch the margin:
  - they are indicated by the black circles in this figure.
- These points are the pivotal elements of this fit, and are known as the support vectors, and give the algorithm its name.



SVM Example

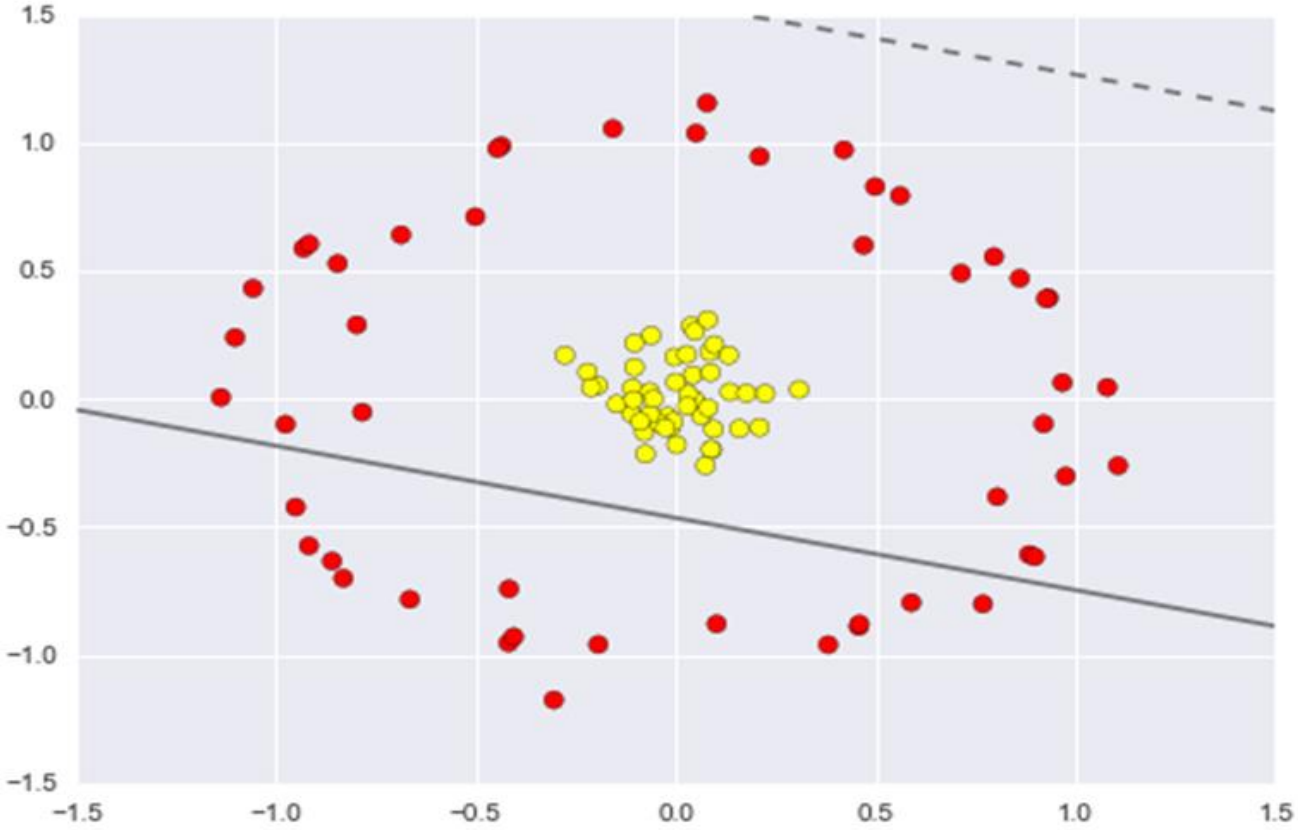
- A key to this classifier's success is that for the fit, only the position of the support vectors matter; any points further from the margin which are on the correct side do not modify the fit!
- Technically, this is because these points do not contribute to the loss function used to fit the model, so their position and number do not matter so long as they do not cross the margin.
- We can see this, for example, if we plot the model learned from the first 60 points and first 120 points of this dataset:





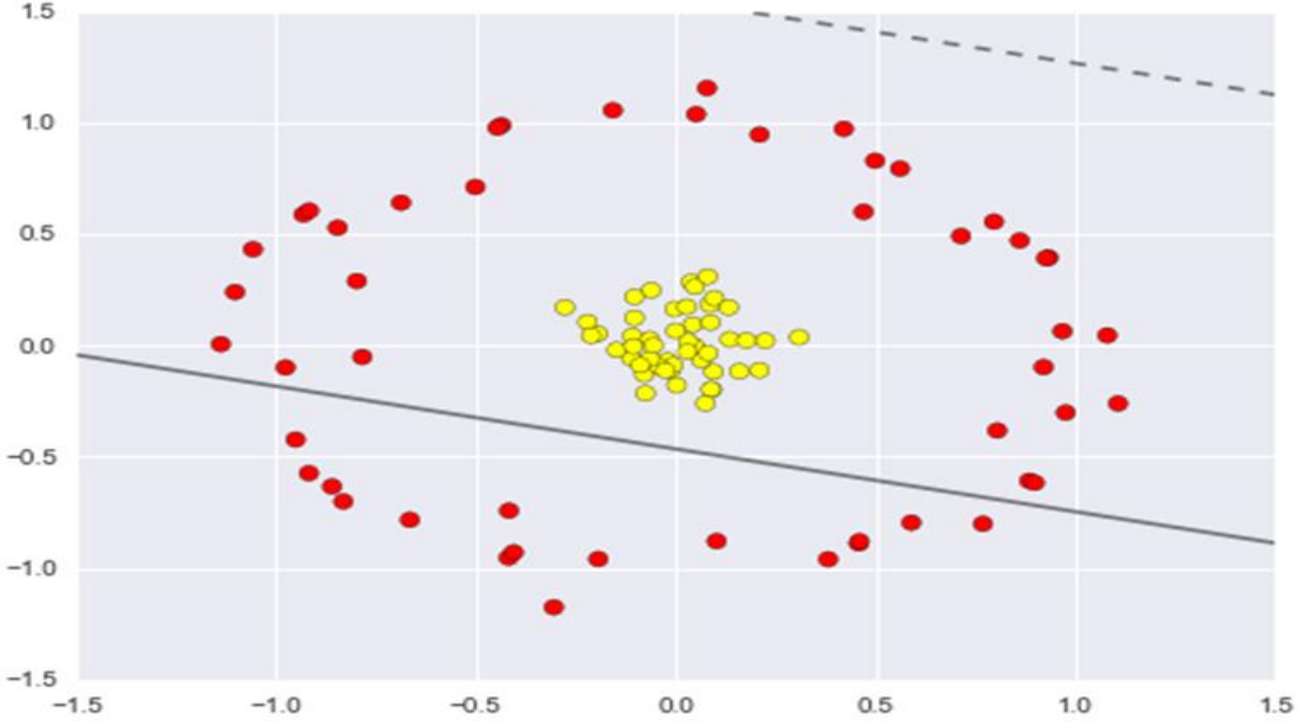
SVM Kernel

- **Beyond linear boundaries: Kernel SVM**
- SVM becomes extremely powerful is when it is combined with kernels.
- There we projected our data into higher-dimensional space defined by polynomials and Gaussian basis functions, and thereby were able to fit for nonlinear relationships with a linear classifier.



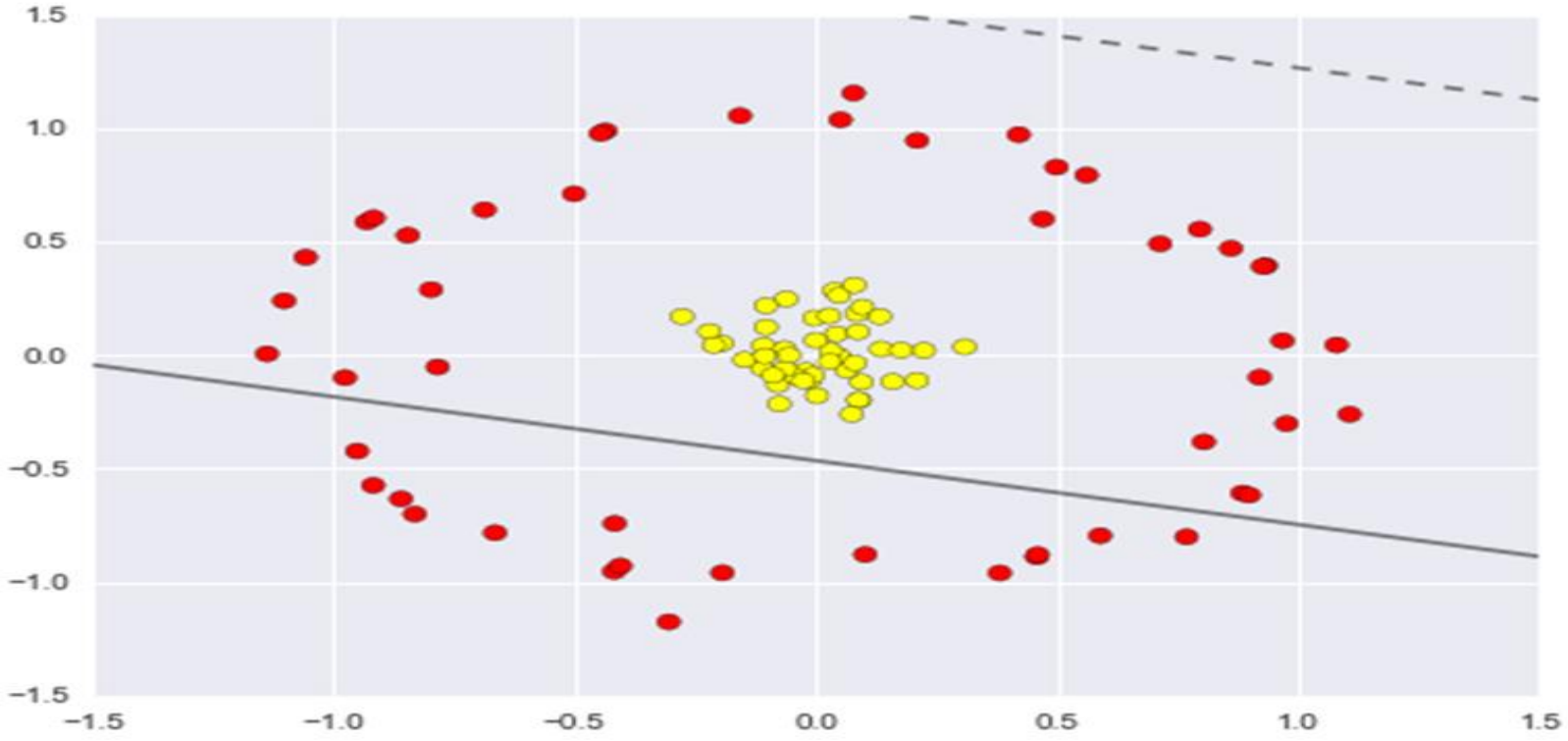
SVM Kernel

- **Beyond linear boundaries: Kernel SVM**
- SVM becomes extremely powerful is when it is combined with kernels.
- There we projected our data into higher-dimensional space defined by polynomials and Gaussian basis functions, and thereby were able to fit for nonlinear relationships with a linear classifier.
- To motivate the need for kernels, let's look at some data that is not linearly separable:



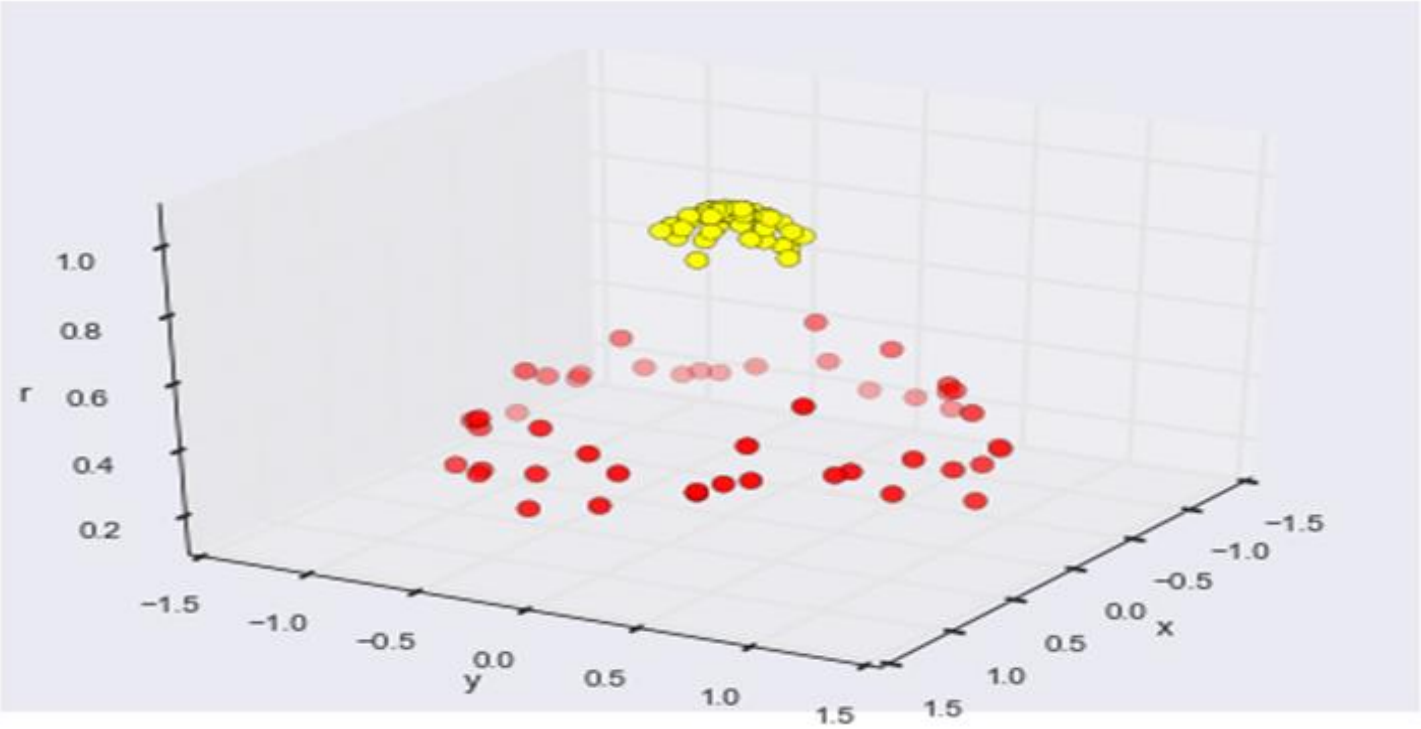
SVM Kernel

- **Beyond linear boundaries: Kernel SVM**
- It is clear that no linear discrimination will ever be able to separate this data.
- Think about how we might project the data into a higher dimension such that a linear separator would be sufficient.



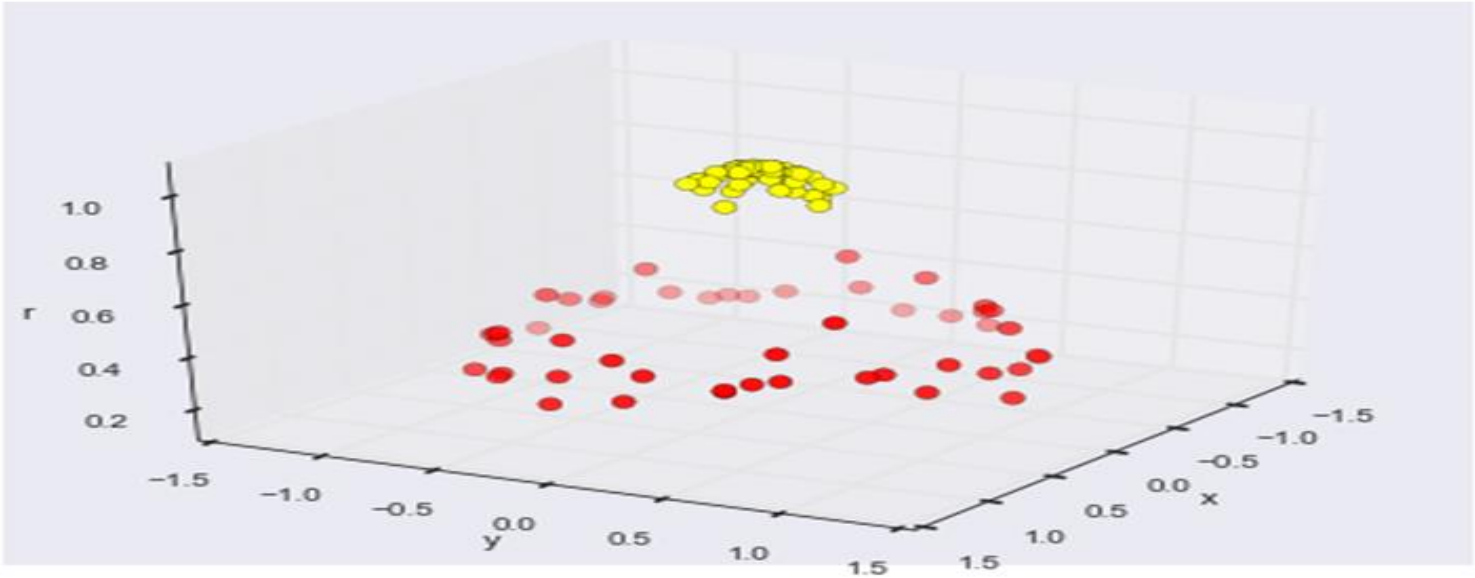
SVM Kernel

- **Beyond linear boundaries: Kernel SVM**
- We can visualize this extra data dimension using a three-dimensional plot.
- We can see that with this additional dimension, the data becomes trivially linearly separable, by drawing a separating plane at, say, r=0.7.



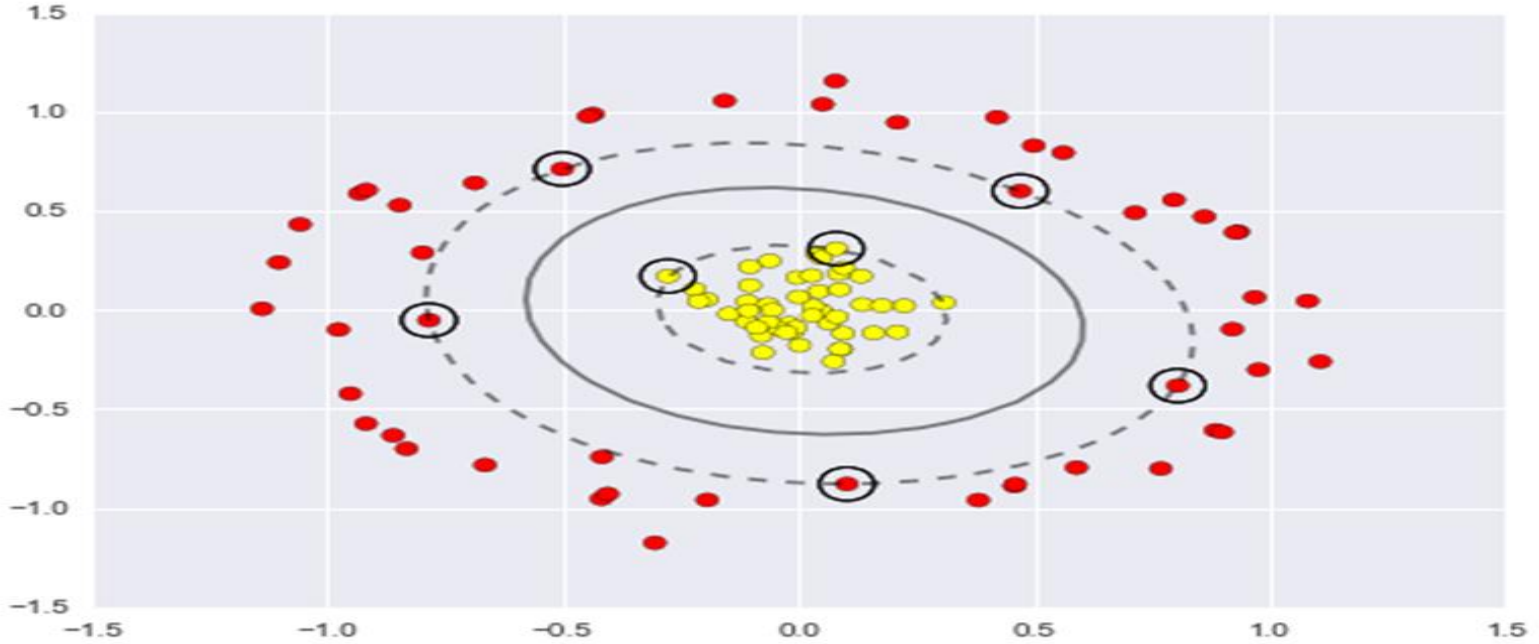
SVM Kernel

- **Beyond linear boundaries: Kernel SVM**
- Here we had to choose and carefully tune our projection: if we had not centered our radial basis function in the right location, we would not have seen such clean, linearly separable results.
- In general, the need to make such a choice is a problem: we would like to somehow automatically find the best basis functions to use.
- One strategy is to compute a basis function centered at every point in the dataset. **This type of basis function transformation is known as a kernel transformation, as it is based on a similarity relationship (or kernel) between each pair of points.**



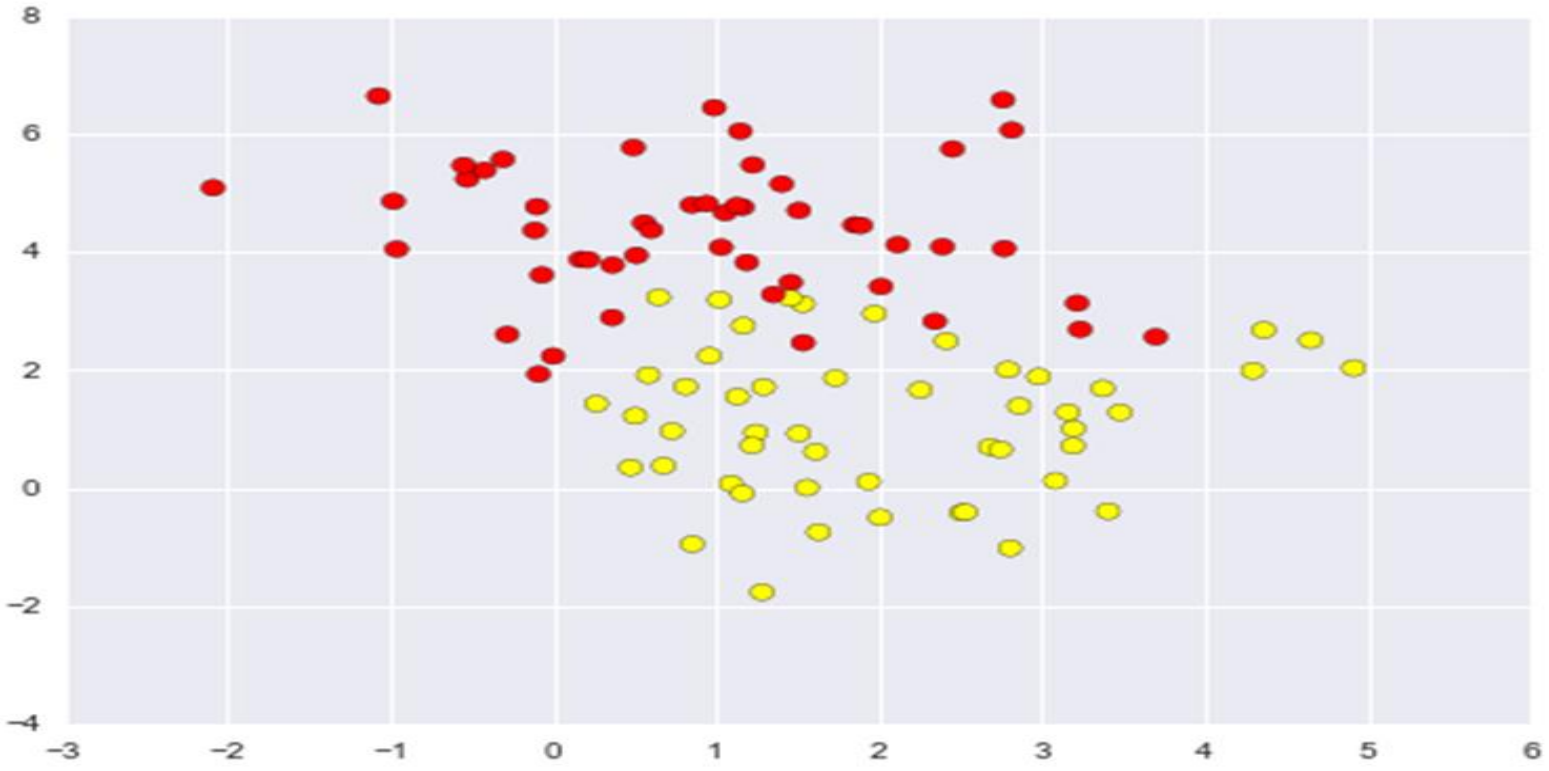
SVM Kernel

- **Beyond linear boundaries: Kernel SVM**
- A potential problem with this strategy—**projecting N points into N dimensions**—is that it might become very computationally intensive as **N grows large.**
- However, kernel trick, a fit on kernel-transformed data can be done implicitly—that is, without ever building the full N-dimensional representation of the kernel projection!
- We can apply **kernelized SVM** simply by changing **our linear kernel to an RBF (radial basis function) kernel, using the kernel model hyperparameter and** and is one of the reasons the method is so powerful.



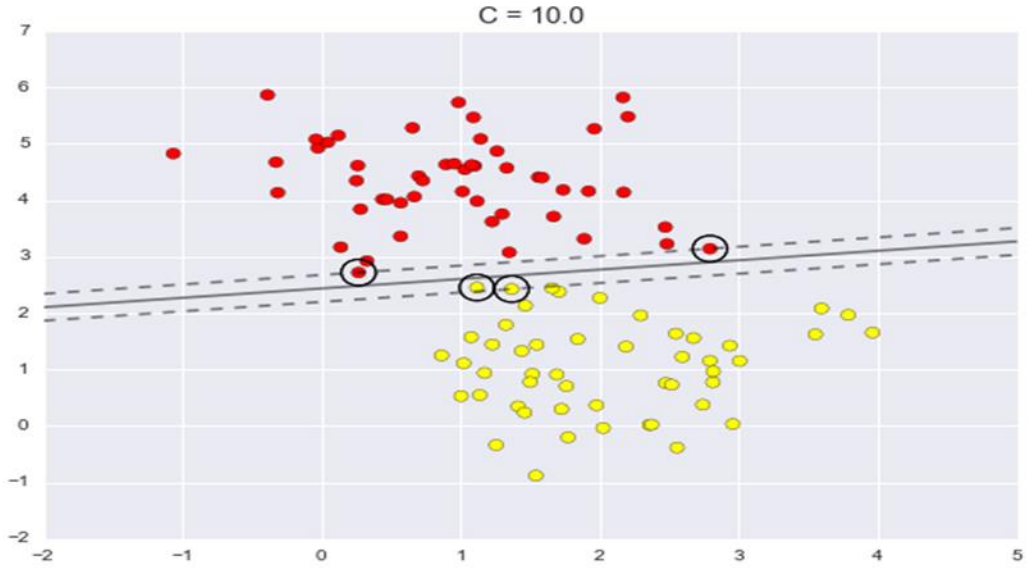
SVM Kernel

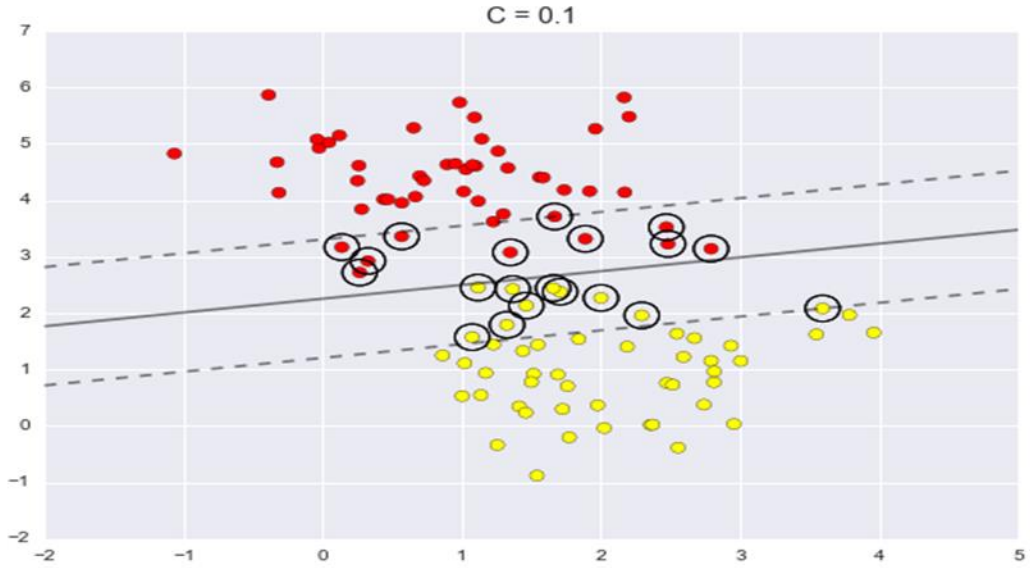
- **Tuning the SVM: Softening Margins**
- Our discussion thus far has centered around very clean datasets, in which a perfect decision boundary exists. But what if your data has some amount of overlap? For example, you may have data like this:



SVM Kernel

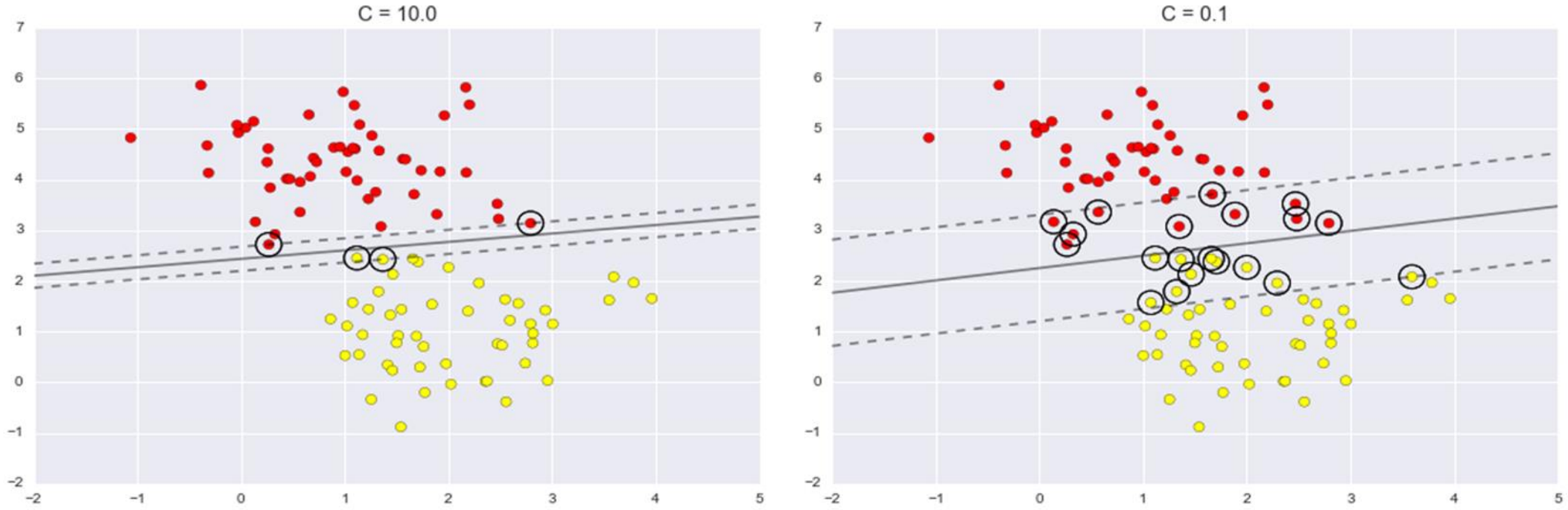
- **Tuning the SVM: Softening Margins**
- To handle this case, the SVM implementation has a bit of a factor which **"softens" the margin:** that is, it allows some of the points to creep into the margin if that allows a better fit.
- The hardness of the margin is controlled by a tuning parameter, most often known as C. For very large C, the margin is hard, and points cannot lie in it. For smaller C, the margin is softer, and can grow to surround points.
- The plot shown below gives a visual picture of how a changing C parameter affects the final fit, via the softening of the margin:





SVM Kernel?

- **Tuning the SVM: Softening Margins**
- The optimal value of the C parameter will depend on your dataset, and should be tuned using cross-validation.
- Finally, we can use a **grid search cross-validation** to explore combinations of parameters. Here we will adjust **C** (**which controls the margin hardness**) and **gamma** (**which controls the size of the radial basis function kernel**), and determine the best model:



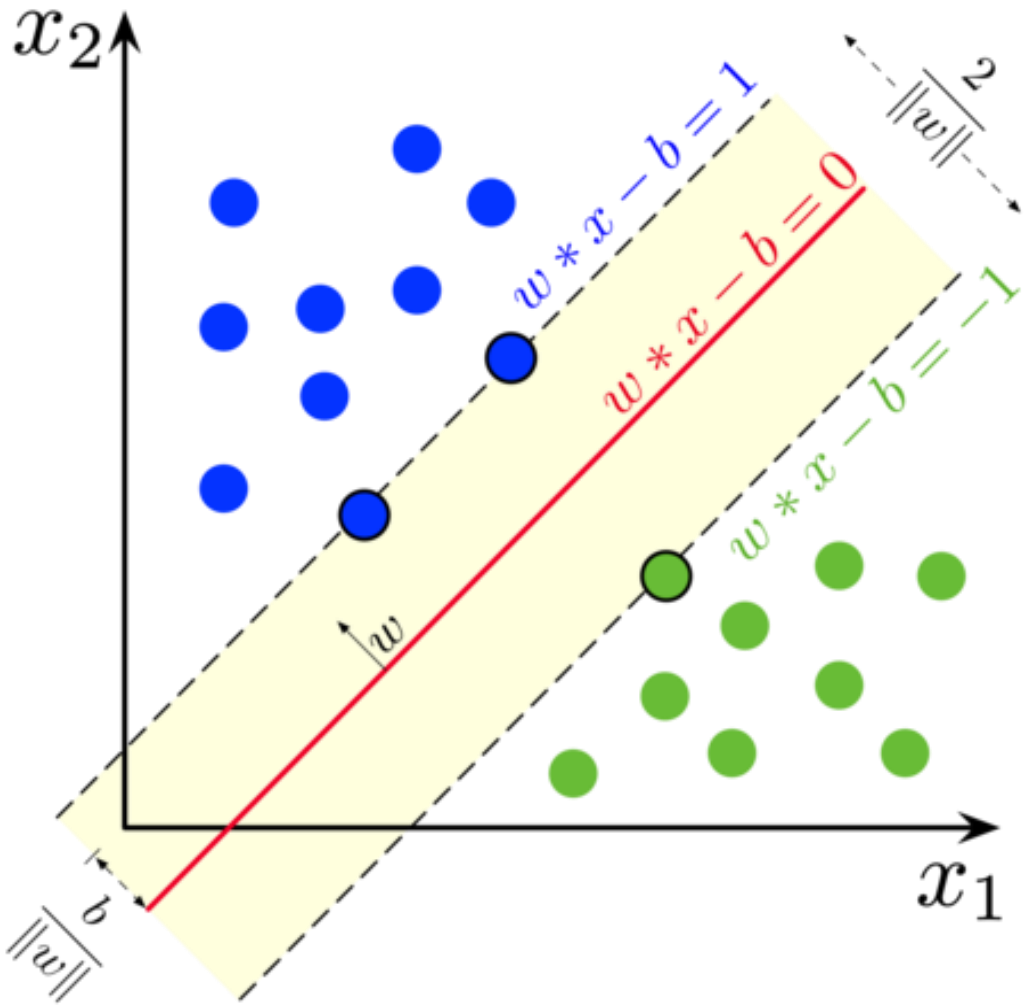
SVM Kernel

- **We have seen here a brief intuitive introduction to the principals behind support vector machines. These methods are a powerful classification method for a number of reasons:**
- Their dependence on relatively few support vectors means that they are very compact models, and take up very little memory.
- Once the model is trained, the prediction phase is very fast.
- Because they are affected only by points near the margin, they work well with **high-dimensional data**—even data with more dimensions than samples, which is a challenging regime for other algorithms.
- Their integration with kernel methods makes them very versatile, able to adapt to many types of data.

SVM Kernel

- **However, SVMs have several disadvantages as well**
- The scaling with the number of samples (N) is **{O}[N^3] at worst, or {O}[N^2]** for efficient implementations. For large numbers of training samples, this computational cost can be prohibitive.
- The results are strongly dependent on a suitable choice for the softening parameter C. This must be carefully chosen via cross-validation, which can be expensive as datasets grow in size.
- The results do not have a direct probabilistic interpretation. This can be estimated via an internal cross-validation **(see the probability parameter of SVC (support vector classifier)),** but this extra estimation is costly.
- SVMs once other **simpler, faster, and less tuning-intensive methods**. Nevertheless, if you have the CPU cycles to commit to training and cross-validating an SVM on your data, the method can lead to excellent results.

SVM classifier Mathematical Formulation



Linear Model

$$w.x - b = 0$$

$$w.x_i - b \geq 1 \; if \; y_i\text{=}1$$

$$w.x_i - b \leq -1 \; if \; y_i\text{=-}1$$

$$y_i(w.x_i - b) \geq 1$$

SVM classifier Mathematical Formulation

Linear Model

$$w.x_i - b \geq 1 \; if \; y_i\text{=}1$$

$$w.x_i - b \leq -1 \; if \; y_i\text{=-}1$$

$$y_i(w.x_i - b) \geq 1$$

Cost function

Hinge Loss

$$l = \max(0, 1 - y_i(w.x_i - b))$$

$$l = \begin{cases} 0 & if \; y.f(x) \geq 1 \\ 1 - y.f(x) & otherwise \end{cases}$$

SVM classifier Mathematical Formulation

Add Regularization

$$J = \lambda\|w\|^2 + 1/2 \sum_{i=1}^{n} \max(0,1 - y_i(w.x_i - b))$$

$if\ y.f(x) \geq 1:$

$J_i = \lambda\|w\|^2$

Else:

$J_i = \lambda\|w\|^2 + 1 - y_i(w.x_i - b))$

SVM classifier Mathematical Formulation

Gradients:

$if\ yi.f(x) \geq 1:$

$$\frac{dJ_i}{dw_k} = 2\lambda w_k$$

$$\frac{dJ_i}{db} = 0$$

Else:

$$\frac{dJ_i}{dw_k} = 2\lambda w_k - y_i.x_i$$

$$\frac{dJ_i}{db} = y_i$$

Update rule: for each training sample x_i:

W=$w - \alpha dw$

b=b- αdb

SVM function from scratch

```
import numpy as np
class SVM:

    def __init__(self, learning_rate=0.001, lambda_param=0.01, n_iters=1000):
        self.lr = learning_rate
        self.lambda_param = lambda_param
        self.n_iters = n_iters
        self.w = None
        self.b = None


    def fit(self, X, y):
        n_samples, n_features = X.shape

        y_ = np.where(y <= 0, -1, 1)

        self.w = np.zeros(n_features)
        self.b = 0

        for _ in range(self.n_iters):
            for idx, x_i in enumerate(X):
                condition = y_[idx] * (np.dot(x_i, self.w) - self.b) >= 1
                if condition:
                    self.w -= self.lr * (2 * self.lambda_param * self.w)
                else:
                    self.w -= self.lr * (2 * self.lambda_param * self.w - np.dot(x_i, y_[idx]))
                    self.b -= self.lr * y_[idx]
```

```
def predict(self, X):
        approx = np.dot(X, self.w) - self.b
        return np.sign(approx)
```

SVM function from scratch

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets

from svm import SVM

X, y =  datasets.make_blobs(n_samples=50,
n_features=2, centers=2, cluster_std=1.05,
random_state=40)
y = np.where(y == 0, -1, 1)

clf = SVM()
clf.fit(X, y)
#predictions = clf.predict(X)

print(clf.w, clf.b)
```

```
def visualize_svm():
    def get_hyperplane_value(x, w, b, offset):
        return (-w[0] * x + b + offset) / w[1]

    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)
    plt.scatter(X[:,0], X[:,1], marker='o',c=y)

    x0_1 = np.amin(X[:,0])
    x0_2 = np.amax(X[:,0])

    x1_1 = get_hyperplane_value(x0_1, clf.w, clf.b, 0)
    x1_2 = get_hyperplane_value(x0_2, clf.w, clf.b, 0)

    x1_1_m = get_hyperplane_value(x0_1, clf.w, clf.b, -1)
    x1_2_m = get_hyperplane_value(x0_2, clf.w, clf.b, -1)

    x1_1_p = get_hyperplane_value(x0_1, clf.w, clf.b, 1)
    x1_2_p = get_hyperplane_value(x0_2, clf.w, clf.b, 1)

    ax.plot([x0_1, x0_2],[x1_1, x1_2], 'y--')
    ax.plot([x0_1, x0_2],[x1_1_m, x1_2_m], 'k')
    ax.plot([x0_1, x0_2],[x1_1_p, x1_2_p], 'k')

    x1_min = np.amin(X[:,1])
    x1_max = np.amax(X[:,1])
    ax.set_ylim([x1_min-3,x1_max+3])

    plt.show()

visualize_svm()
```


Q&A