Software

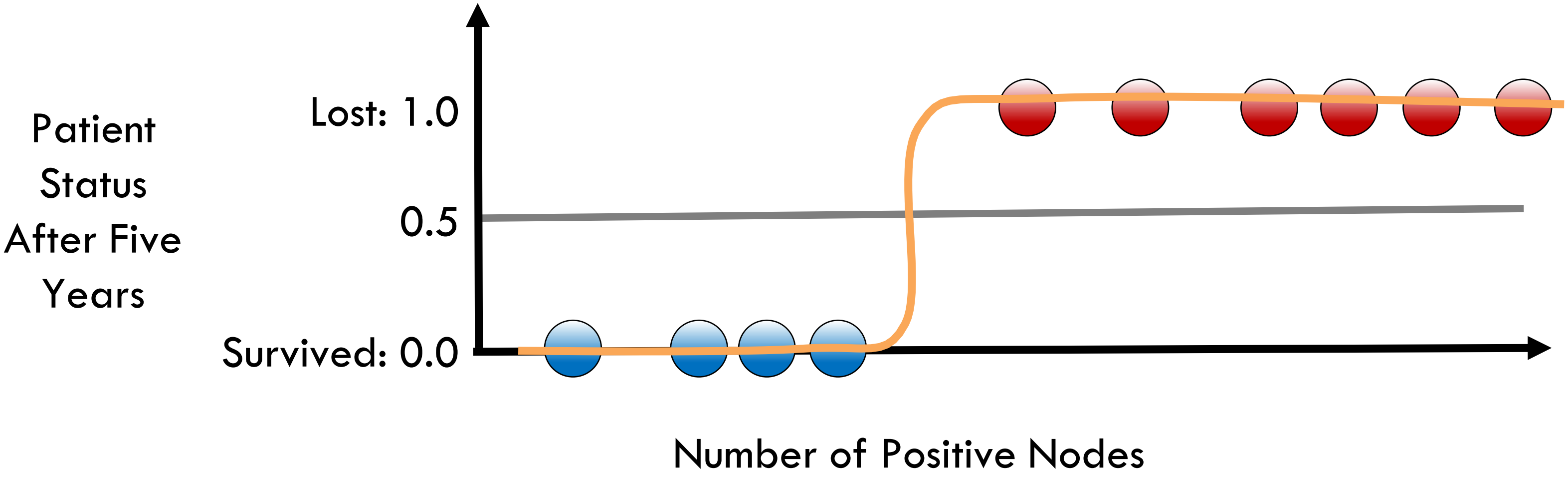
Support Vector Machines

Legal Notices and Disclaimers

Learning Objectives

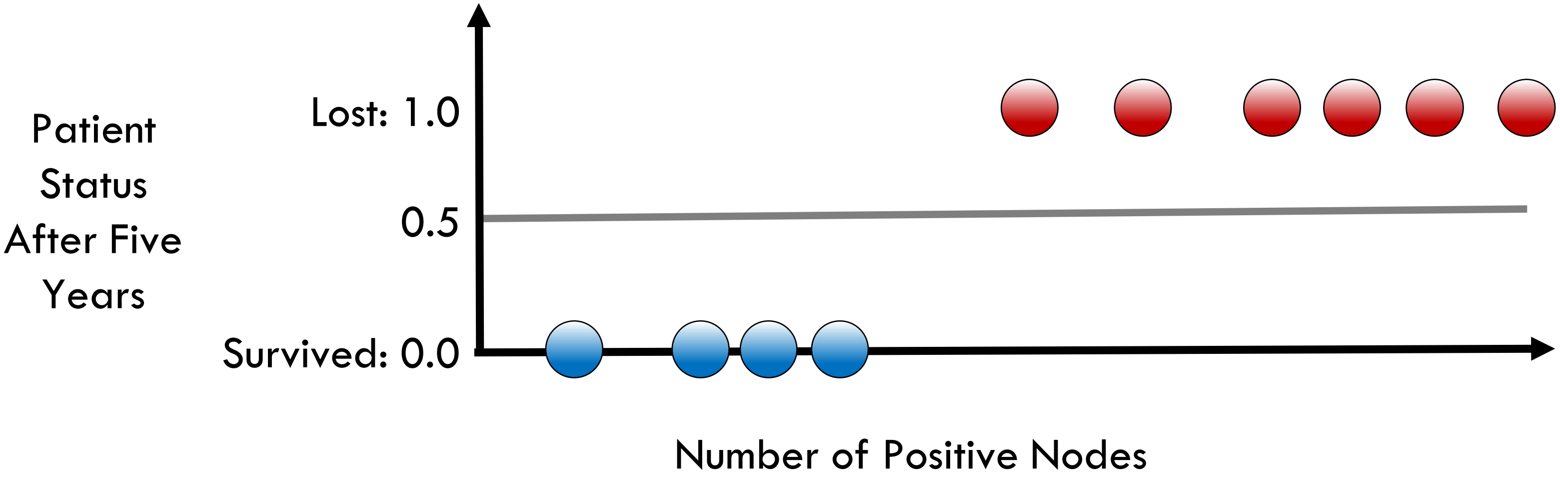
- Apply support vector machines (SVMs)—a popular algorithm used for classification problems
- Recognize SVM similarity to logistic regression
- Compute the cost function of SVMs
- Apply regularization in SVMs and some tips to obtain non-linear classifications with SVMs
- Apply Intel® Extension for Scikit-learn* to leverage underlying compute capabilities of hardware

Relationship to Logistic Regression

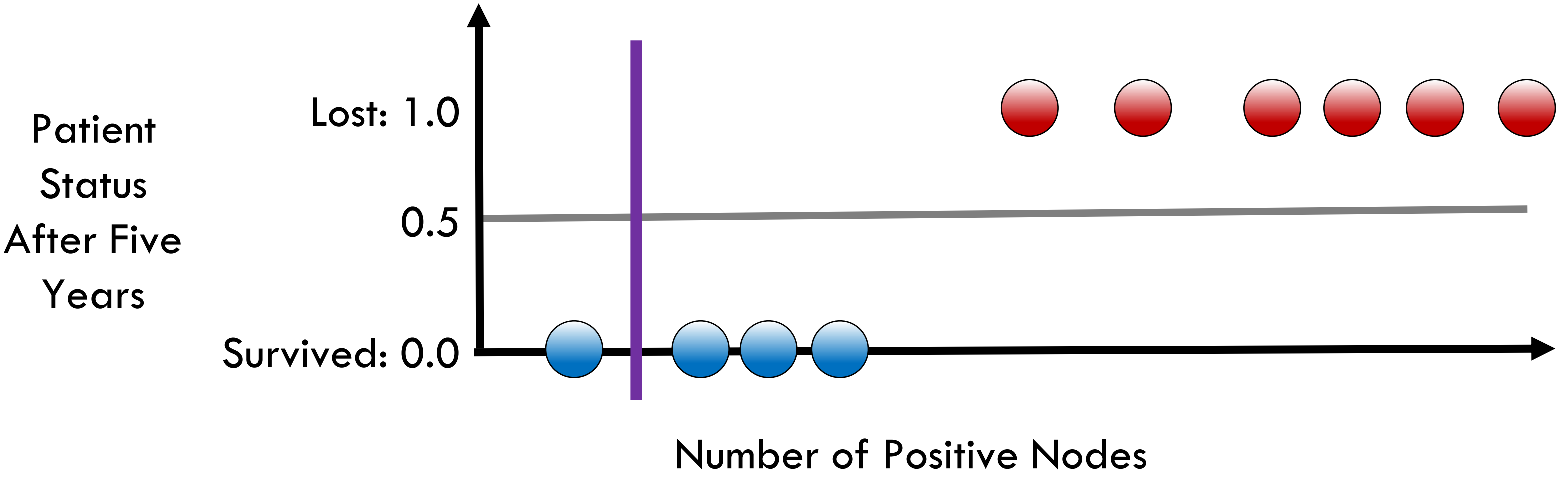


$$y_\beta(x) = \frac{1}{1+e^{-(\beta_0 + \beta_1 x + \varepsilon\ )}}$$

Support Vector Machines (SVM)

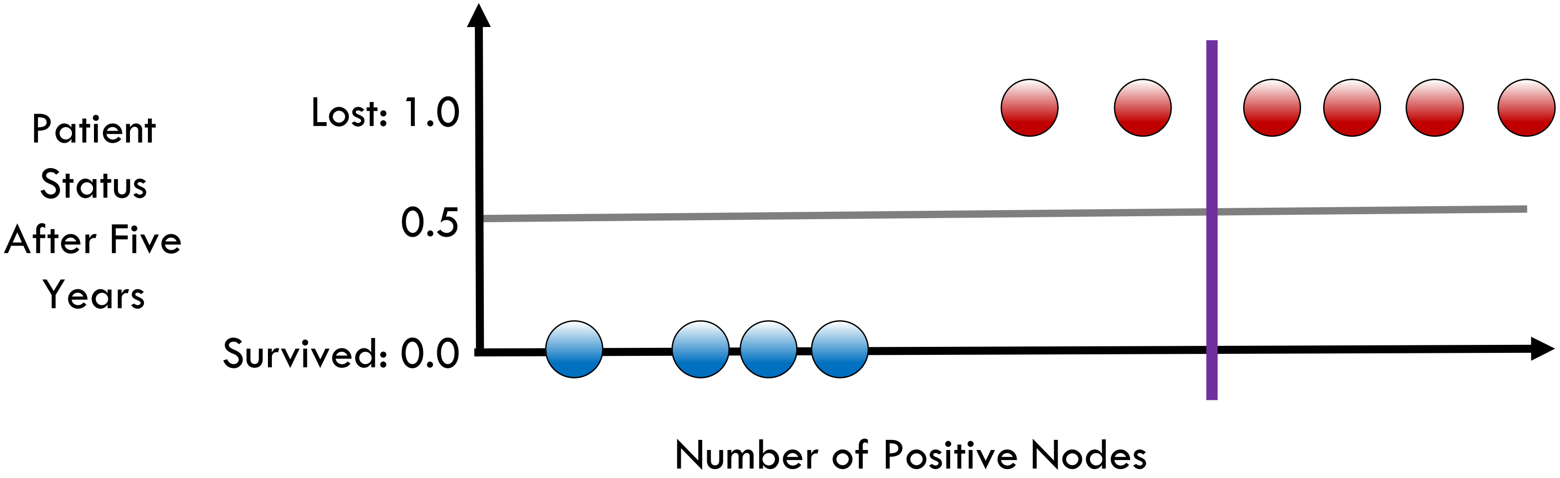


Support Vector Machines (SVM)



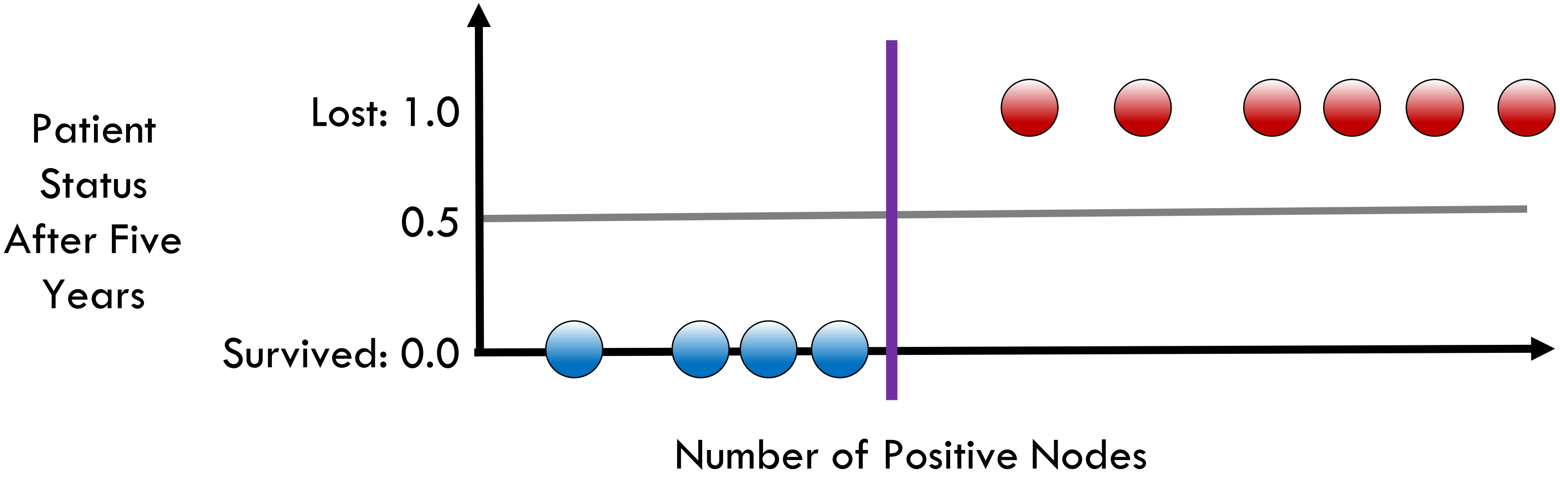
Three misclassifications

Support Vector Machines (SVM)



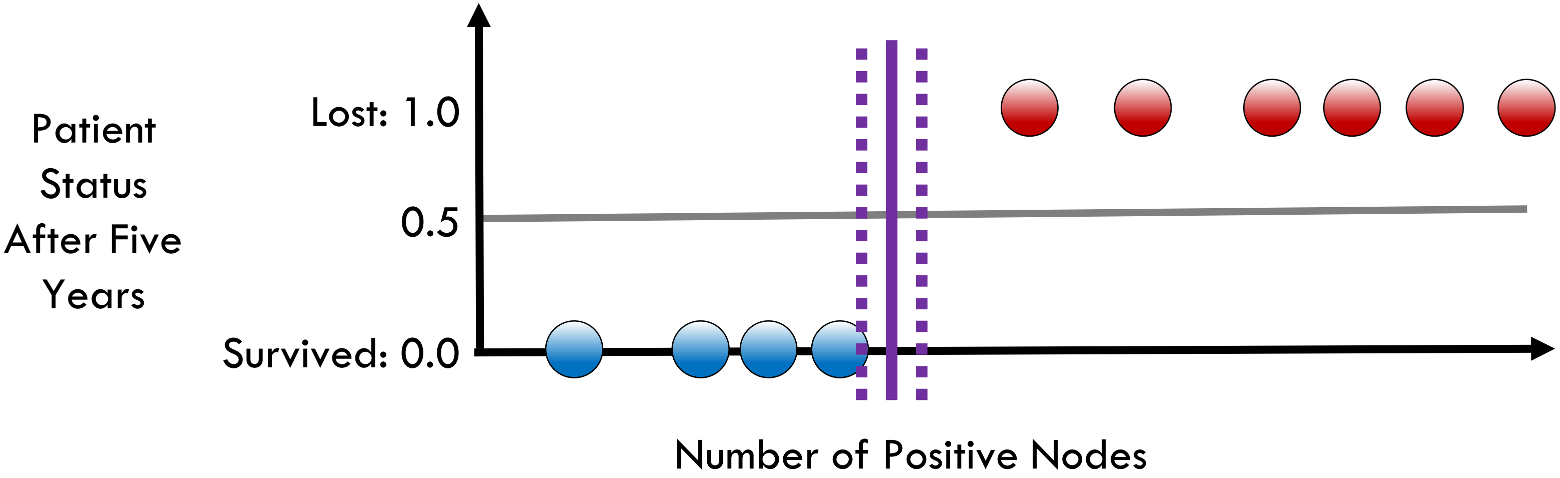
Two misclassifications

Support Vector Machines (SVM)



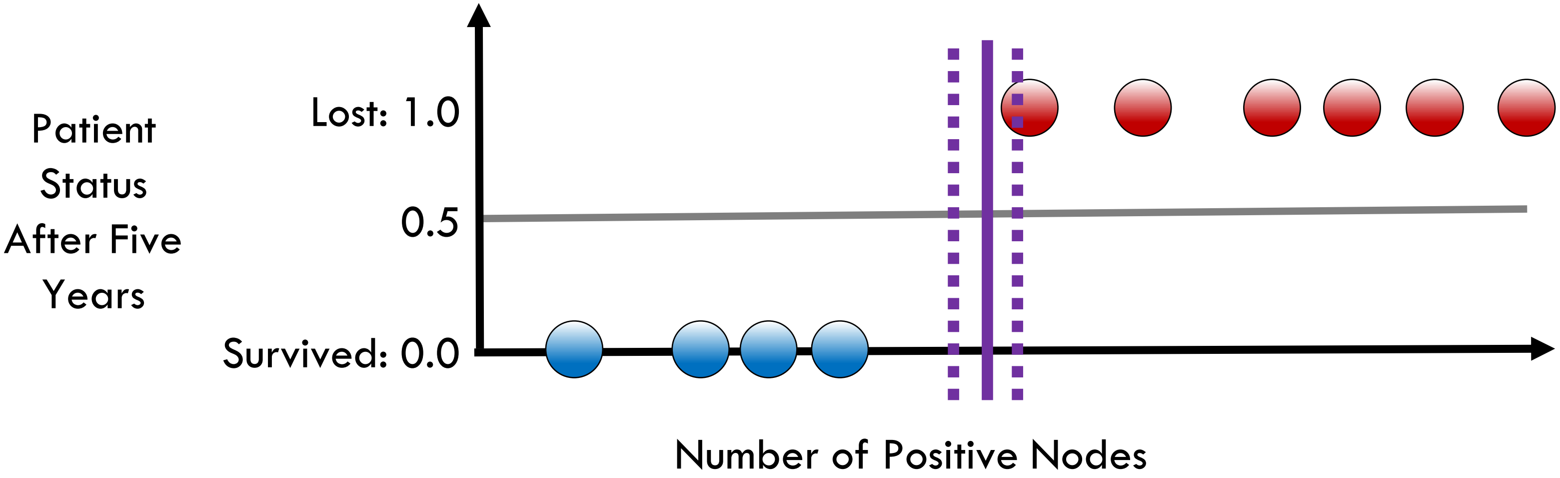
No misclassifications

Support Vector Machines (SVM)



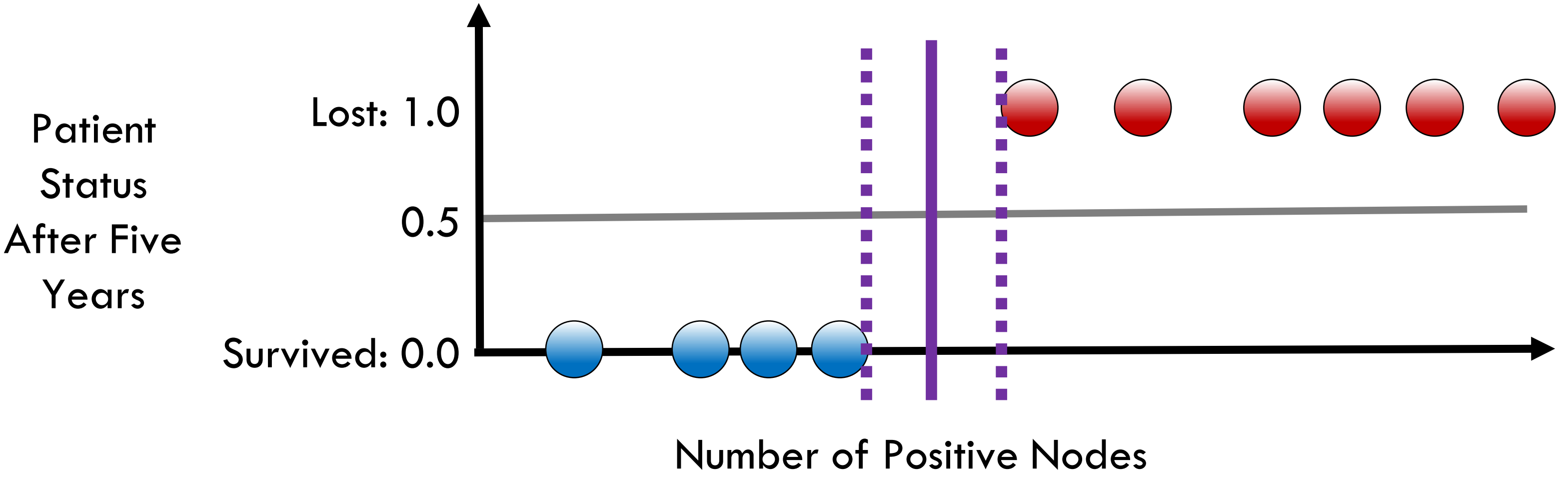
No misclassifications—but is this the best position?

Support Vector Machines (SVM)



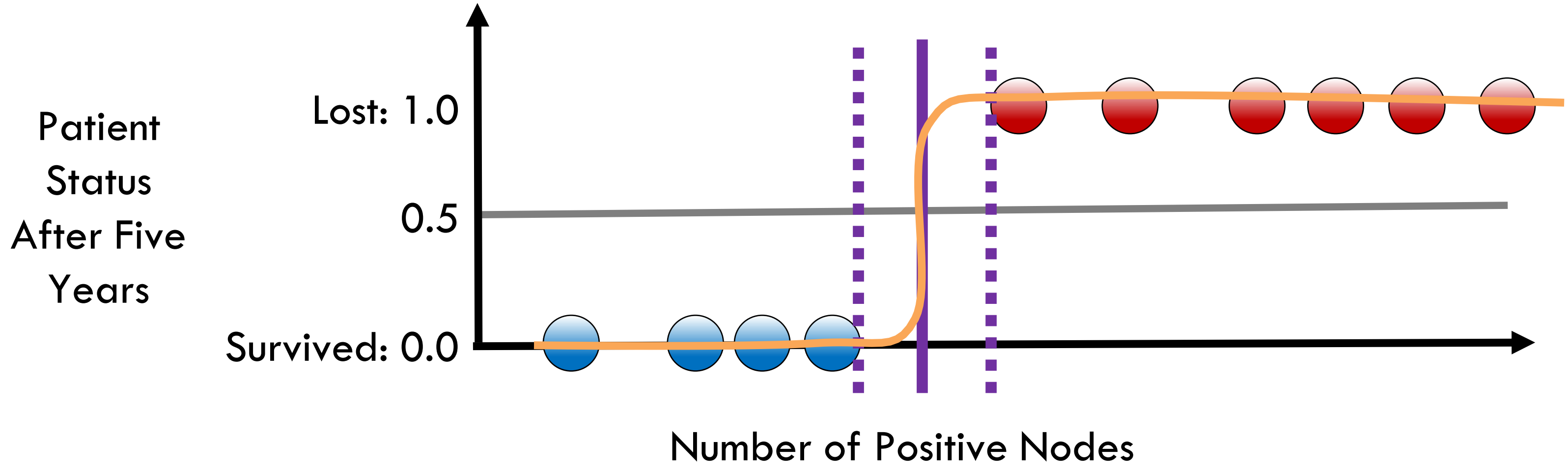
No misclassifications—but is this the best position?

Support Vector Machines (SVM)



Maximize the region between classes

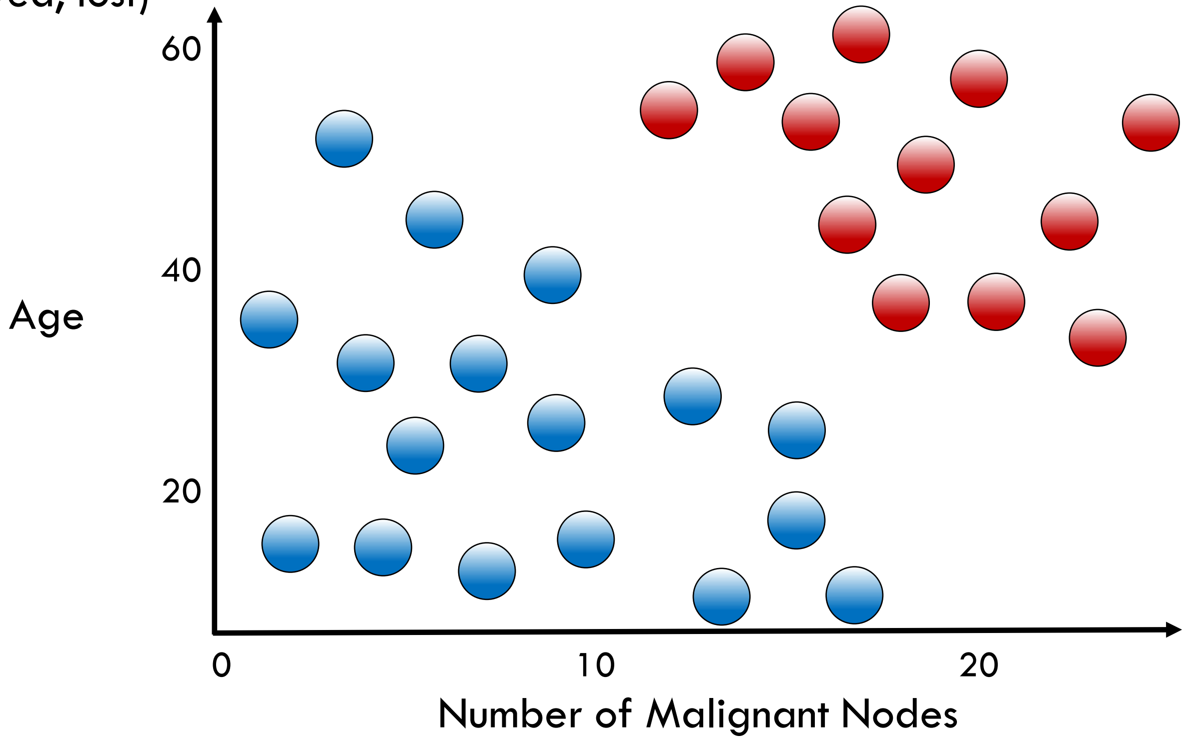
Similarity Between Logistic Regression and SVM



Classification with SVMs

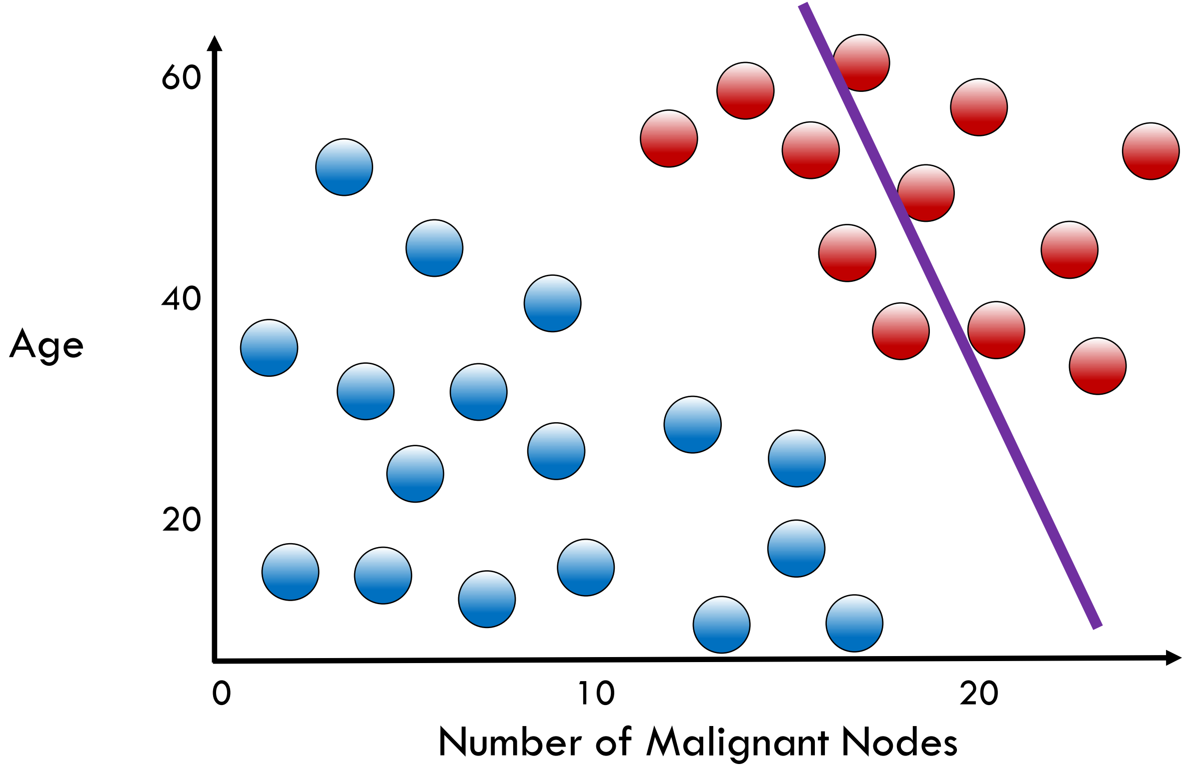
Two features (nodes, age)

Two labels (survived, lost)



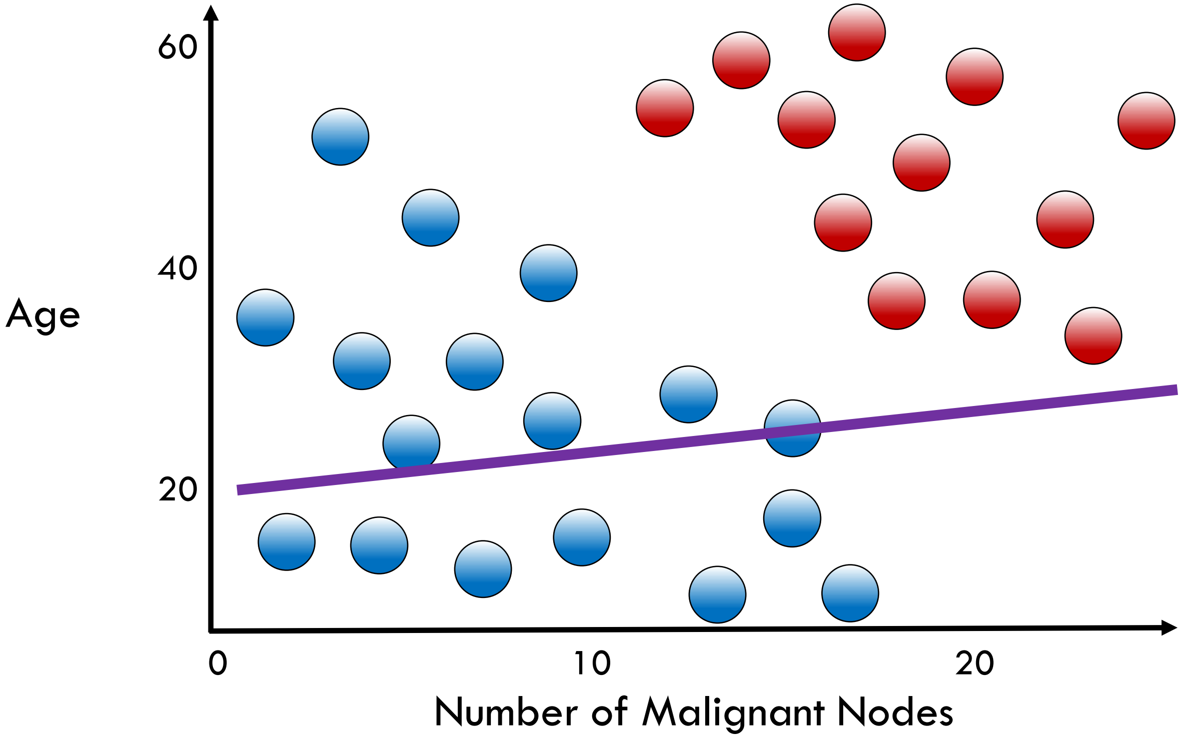
Classification with SVMs

Find the line that best separates classes



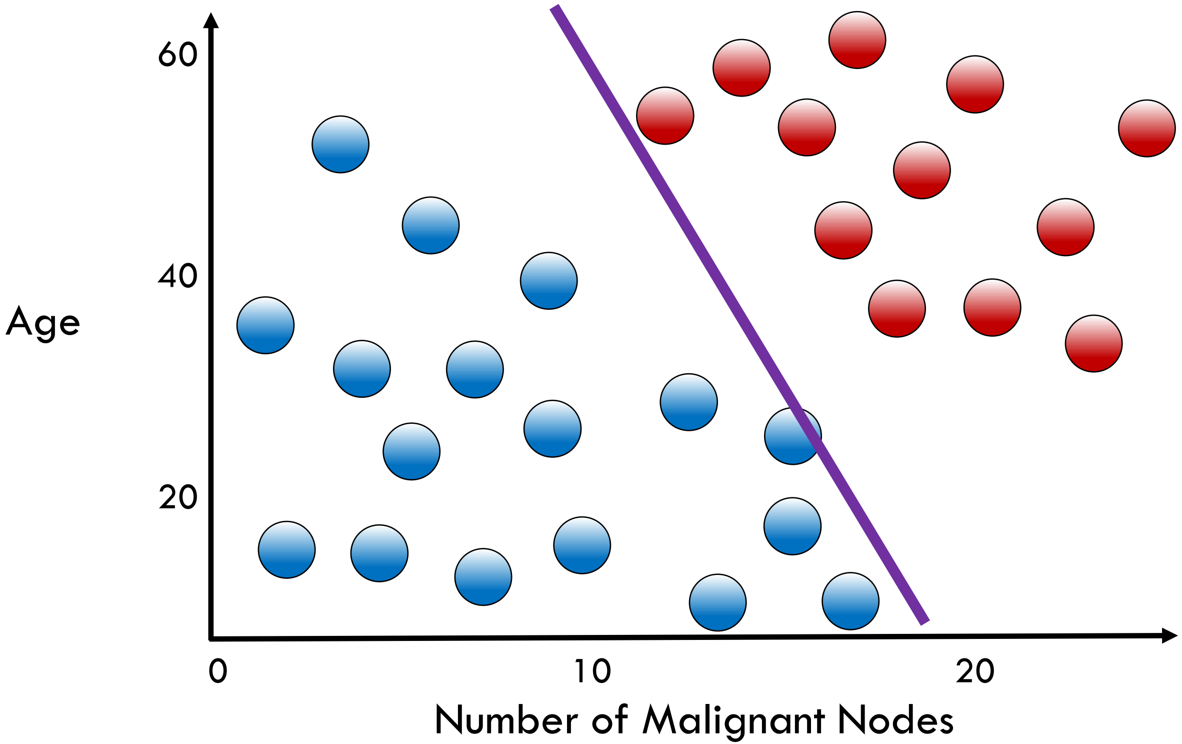
Classification with SVMs

Find the line that best separates classes



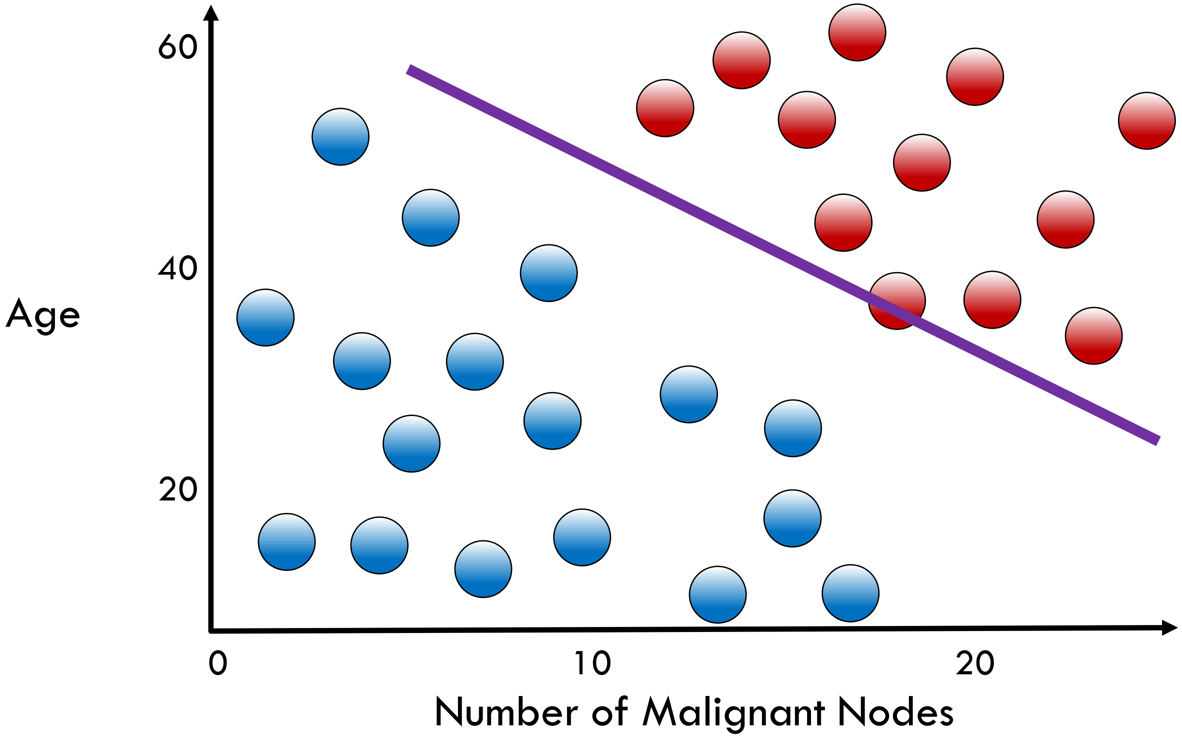
Classification with SVMs

Find the line that best separates classes



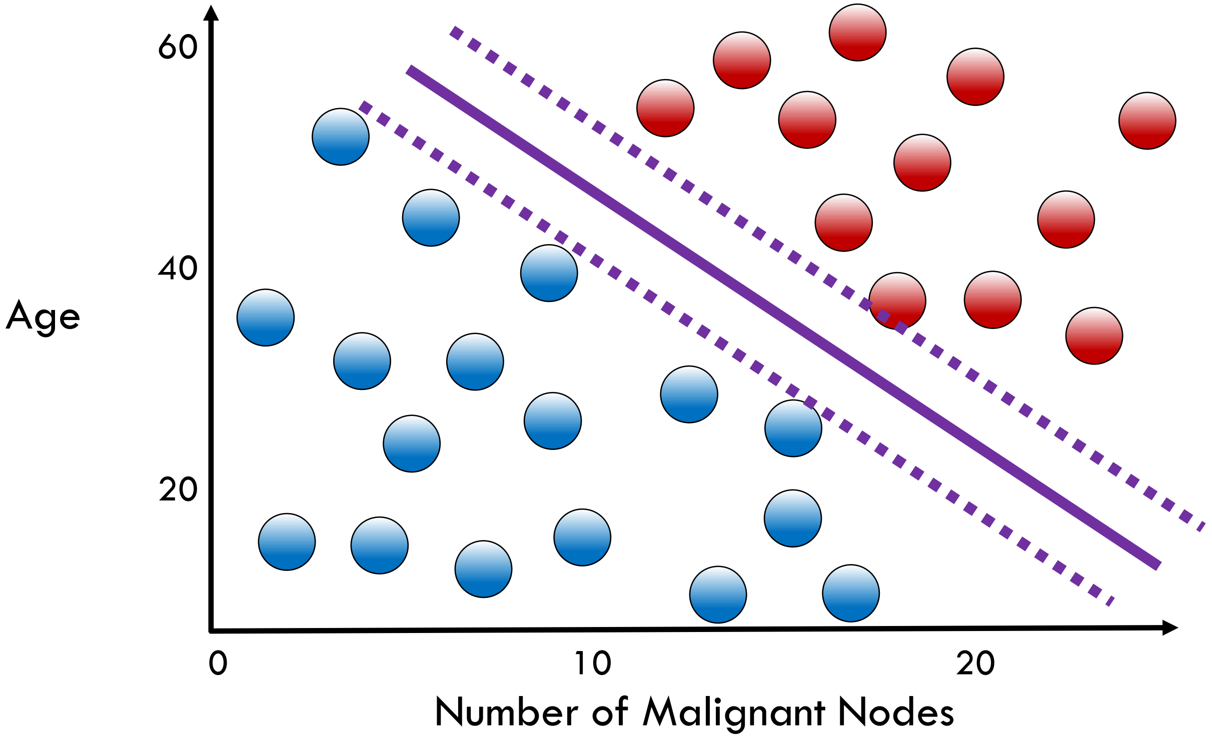
Classification with SVMs

Find the line that best separates classes

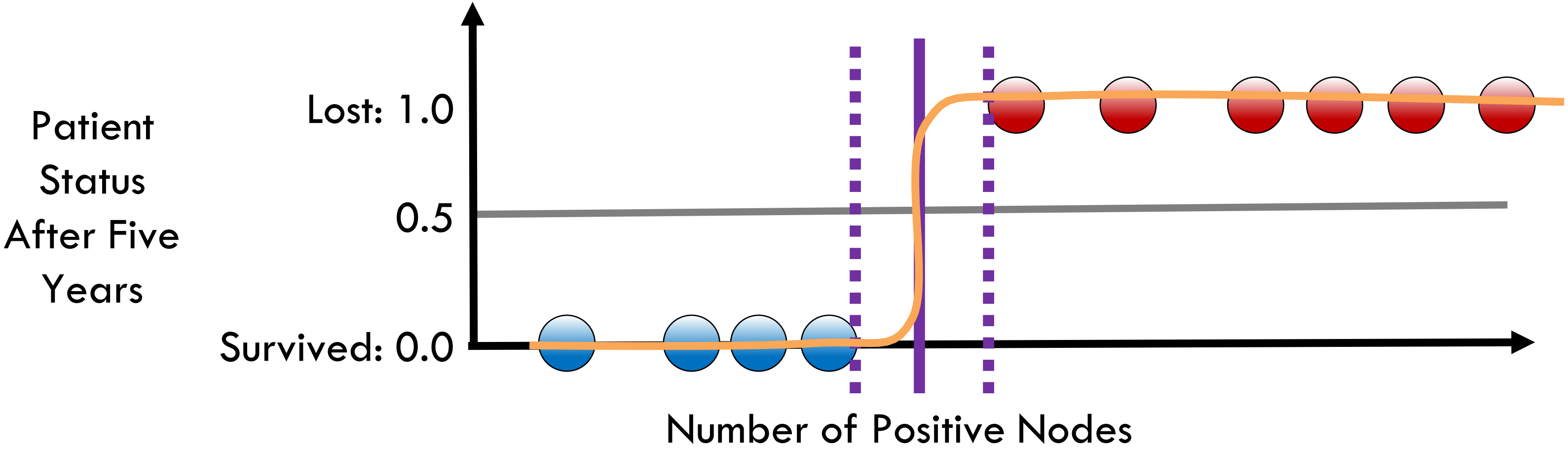


Classification with SVMs

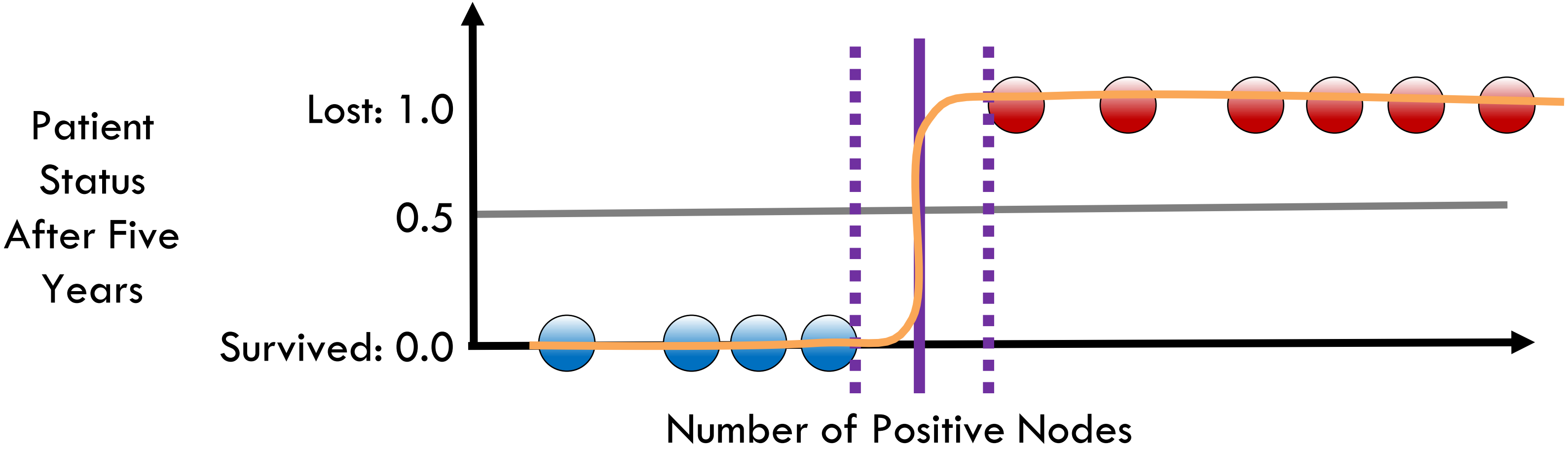
And include the largest boundary possible

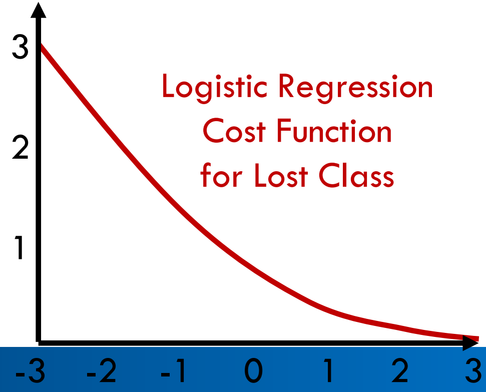


Logistic Regression vs SVM Cost Functions

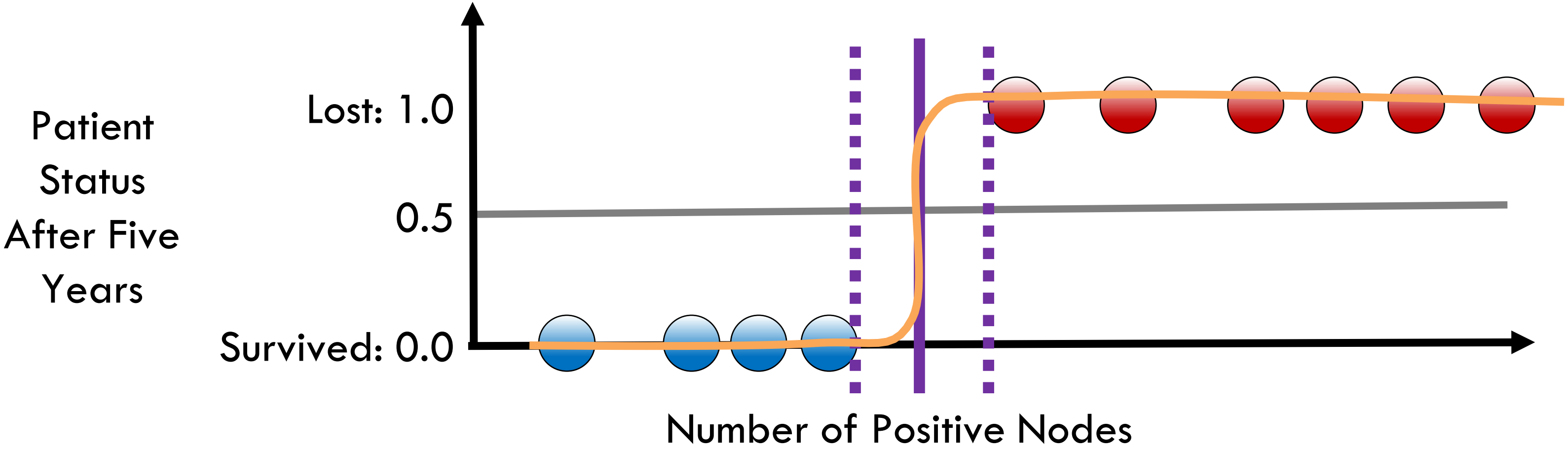


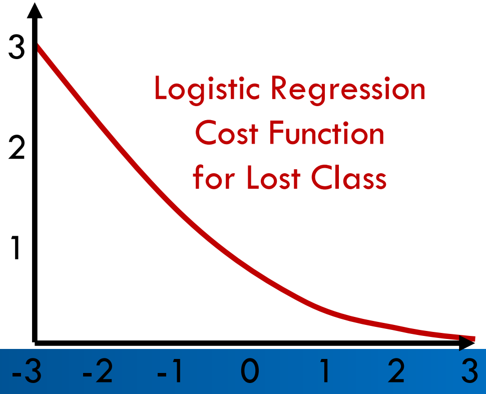
Logistic Regression vs SVM Cost Functions

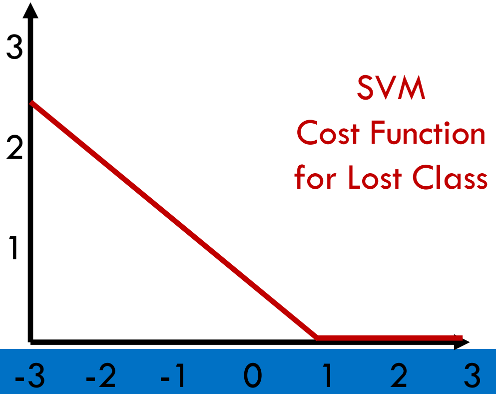




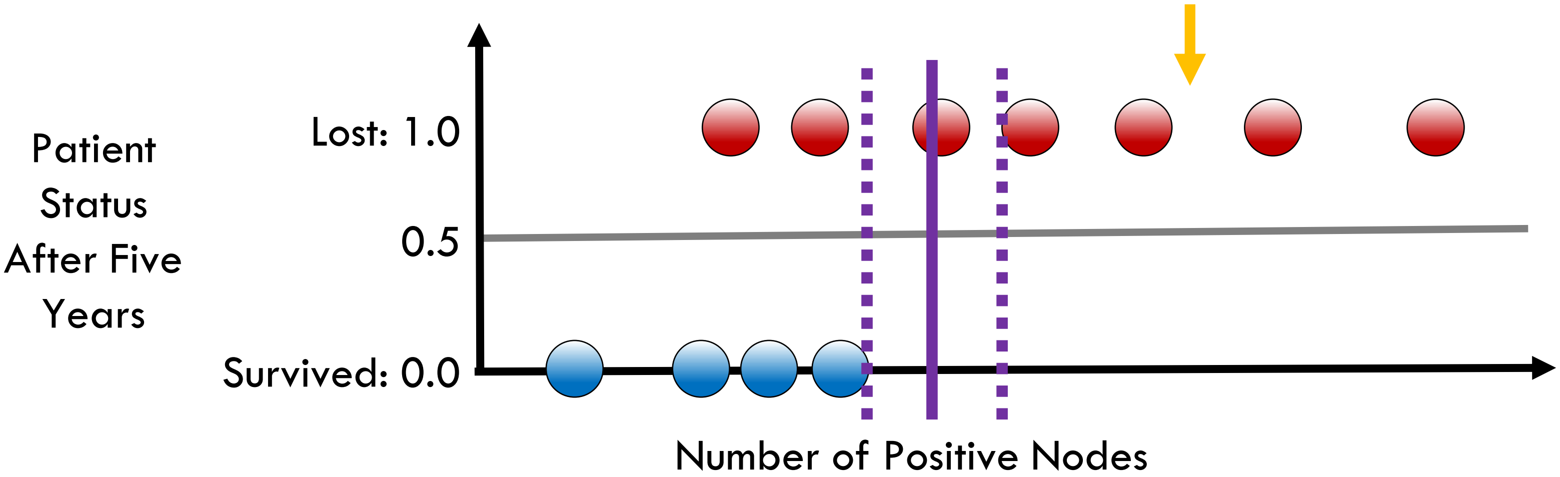
Logistic Regression vs SVM Cost Functions

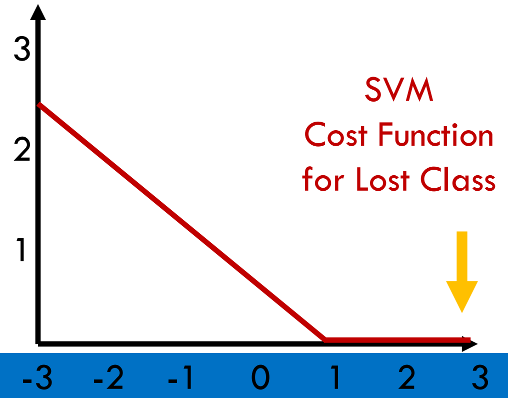




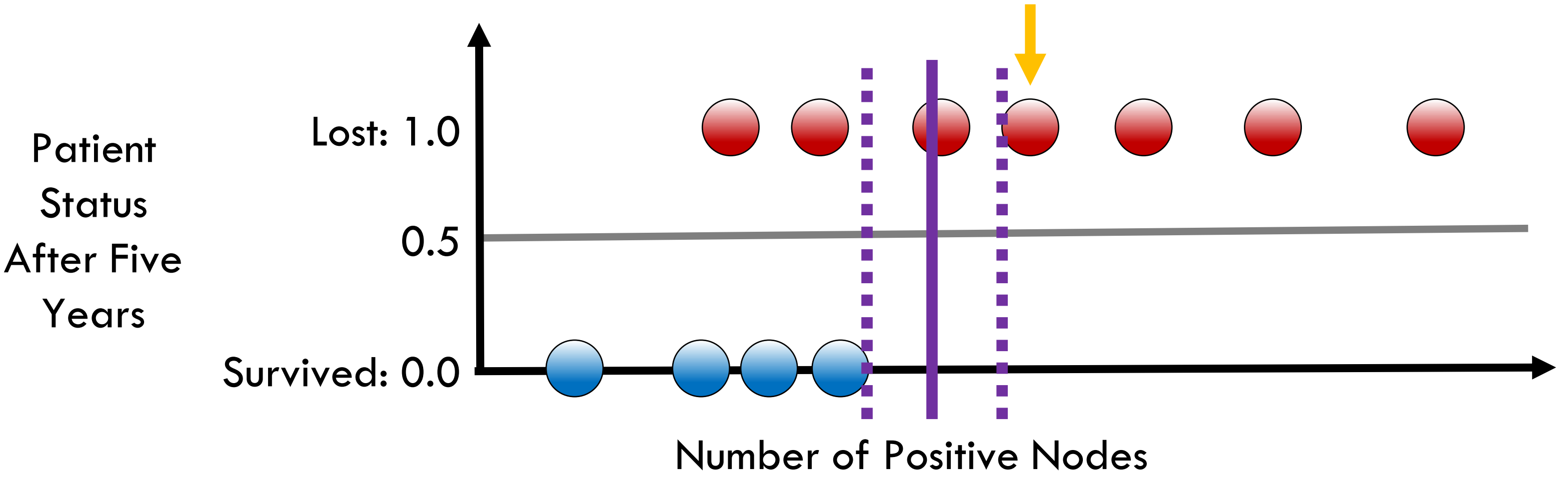


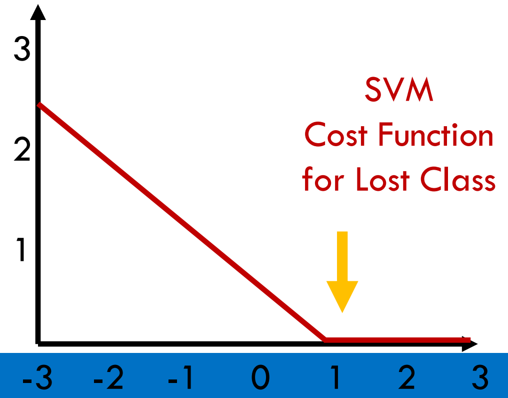
The SVM Cost Function



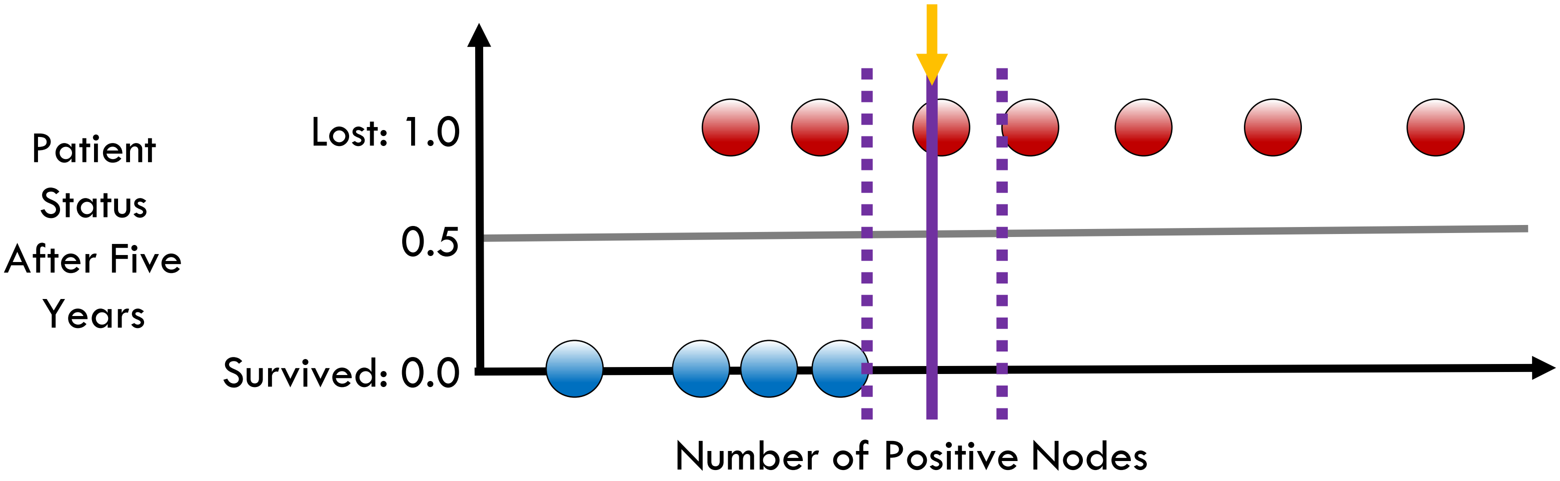


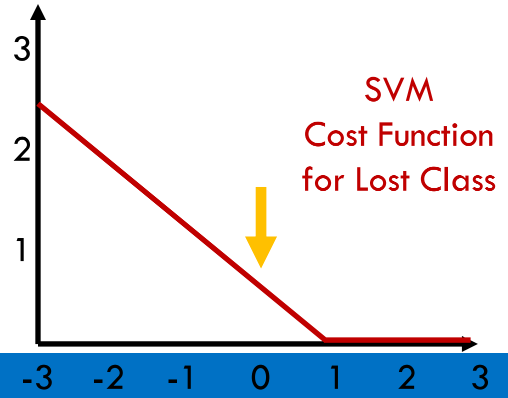
The SVM Cost Function



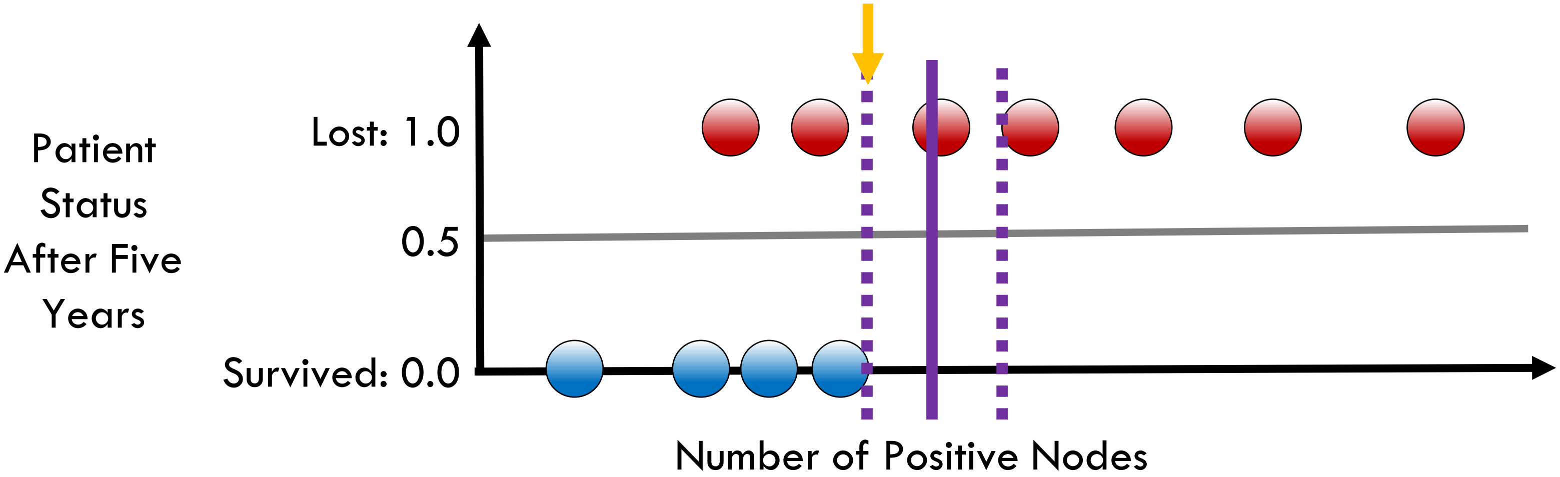


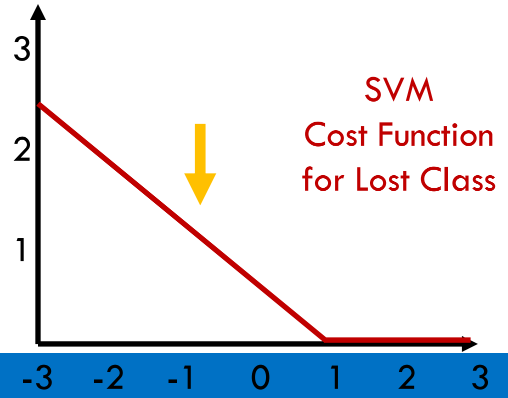
The SVM Cost Function



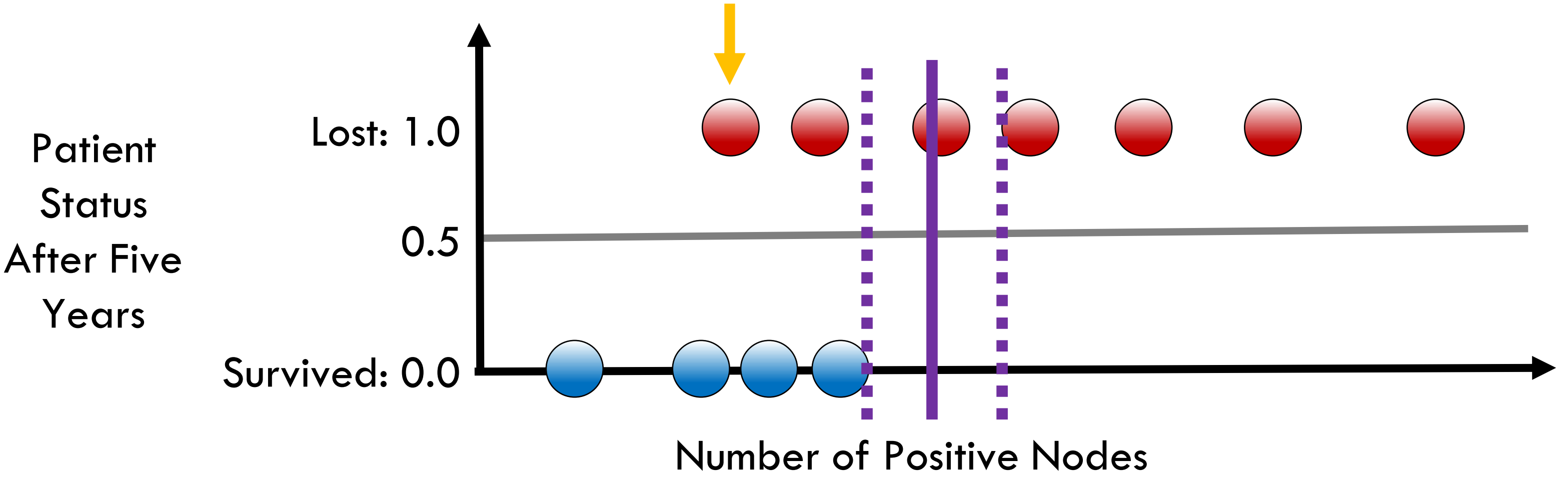


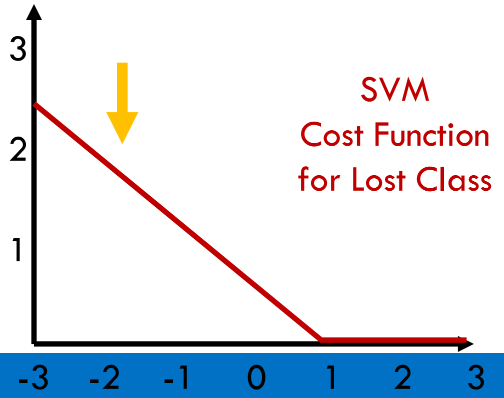
The SVM Cost Function



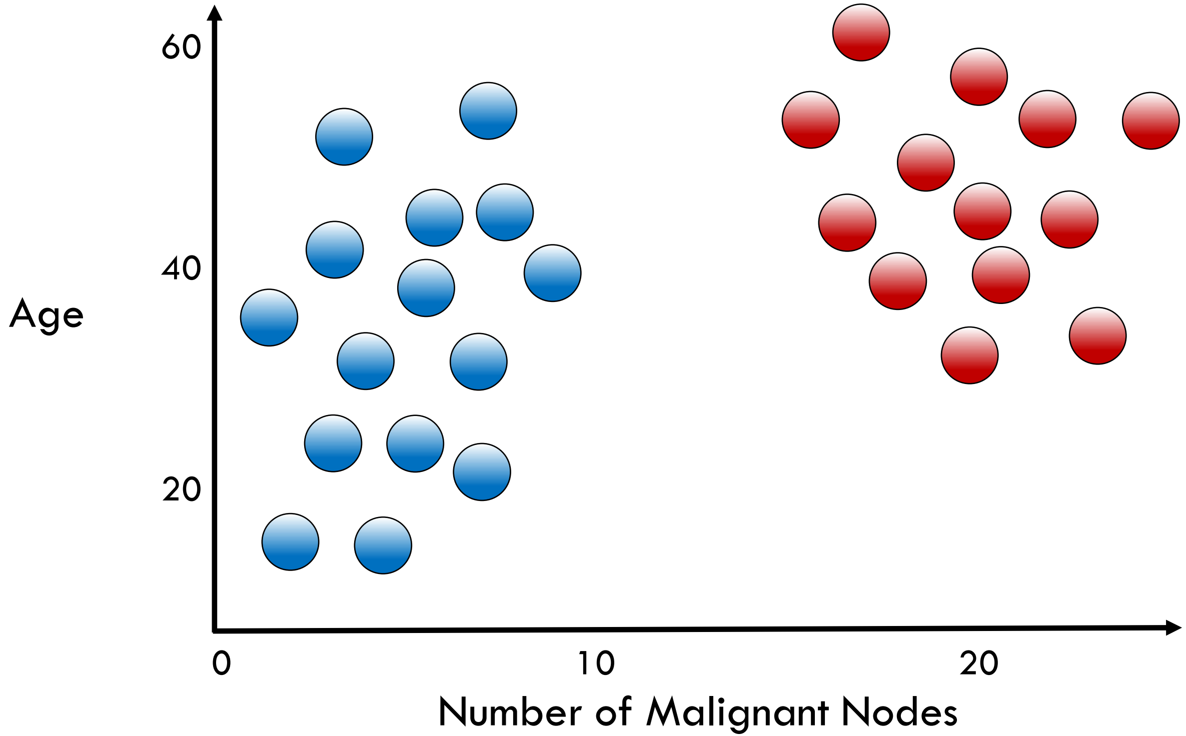


The SVM Cost Function

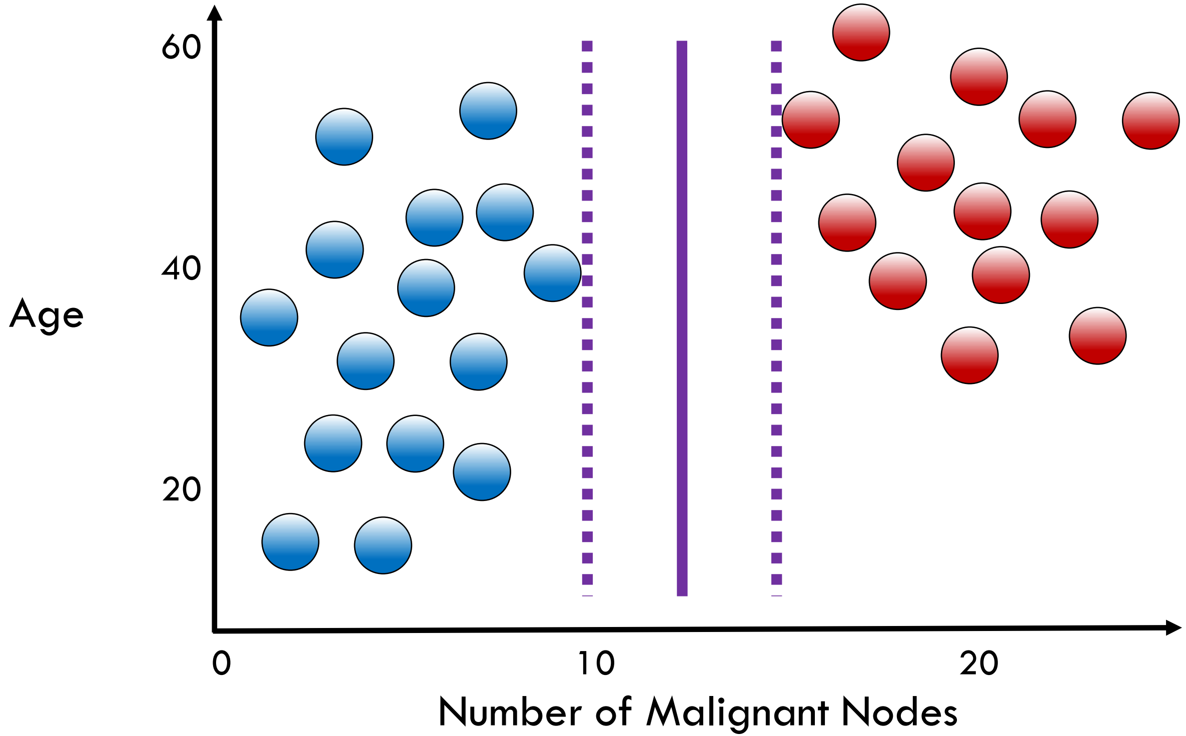




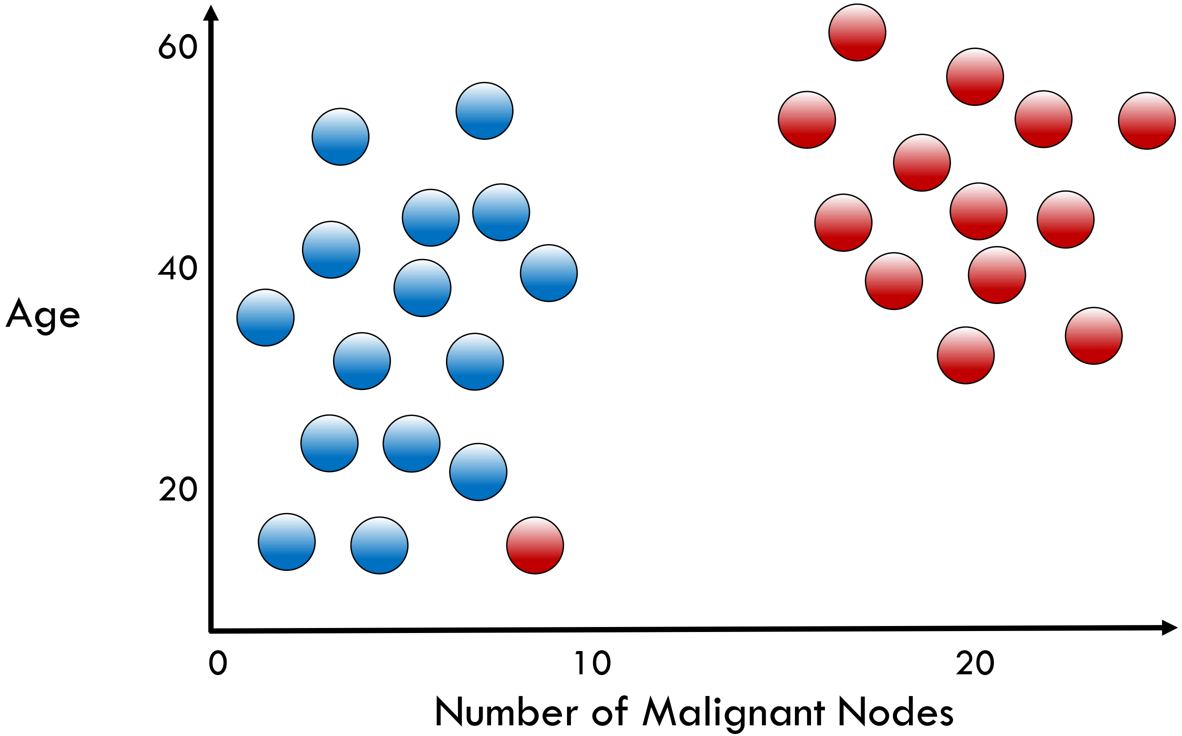
Outlier Sensitivity in SVMs



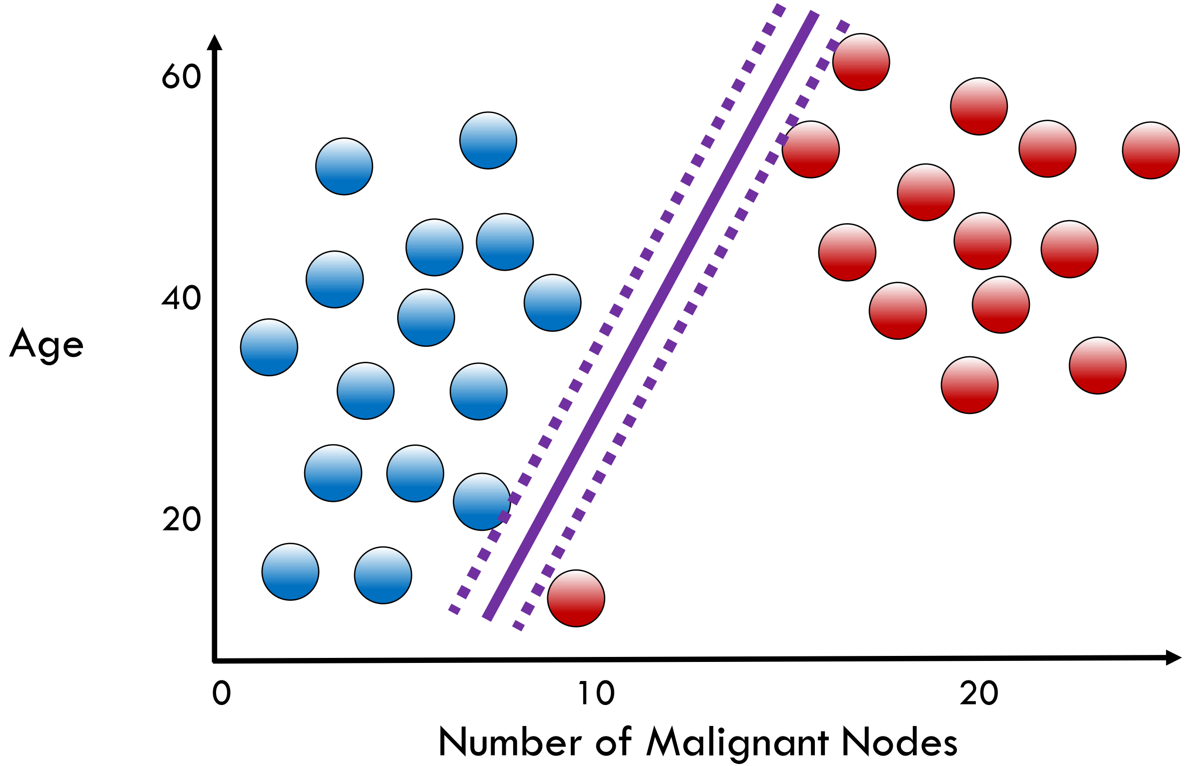
Outlier Sensitivity in SVMs



Outlier Sensitivity in SVMs

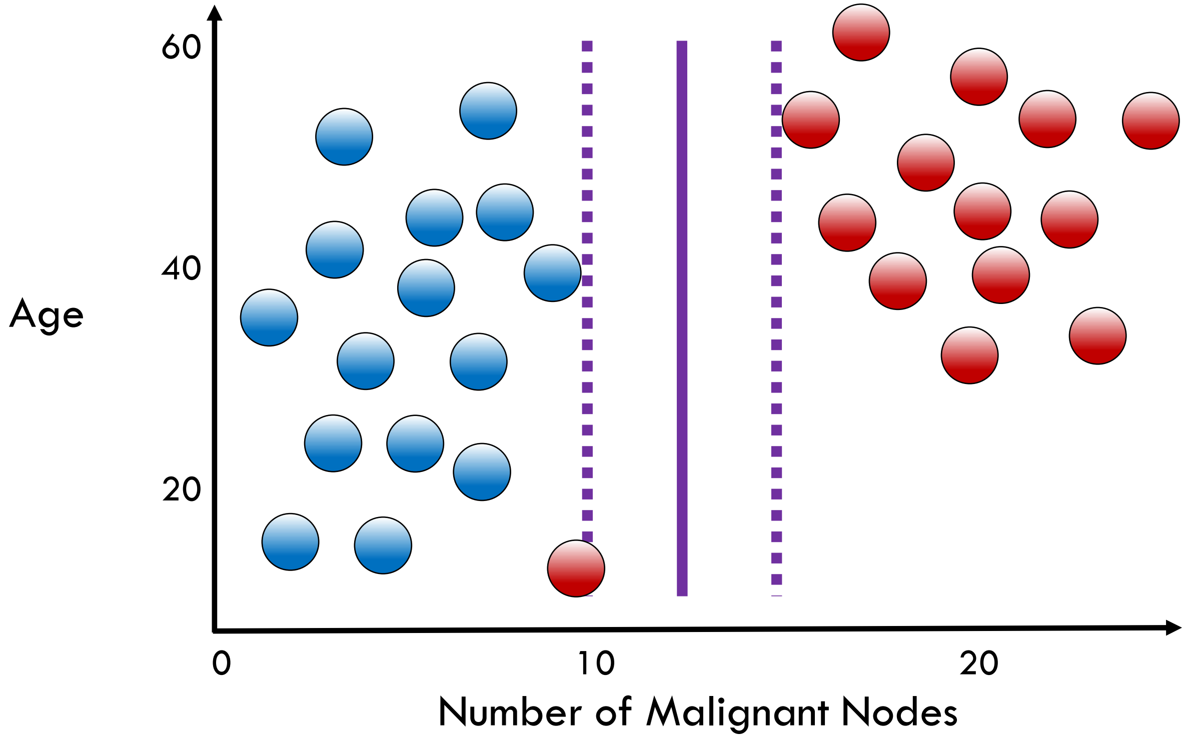


Outlier Sensitivity in SVMs



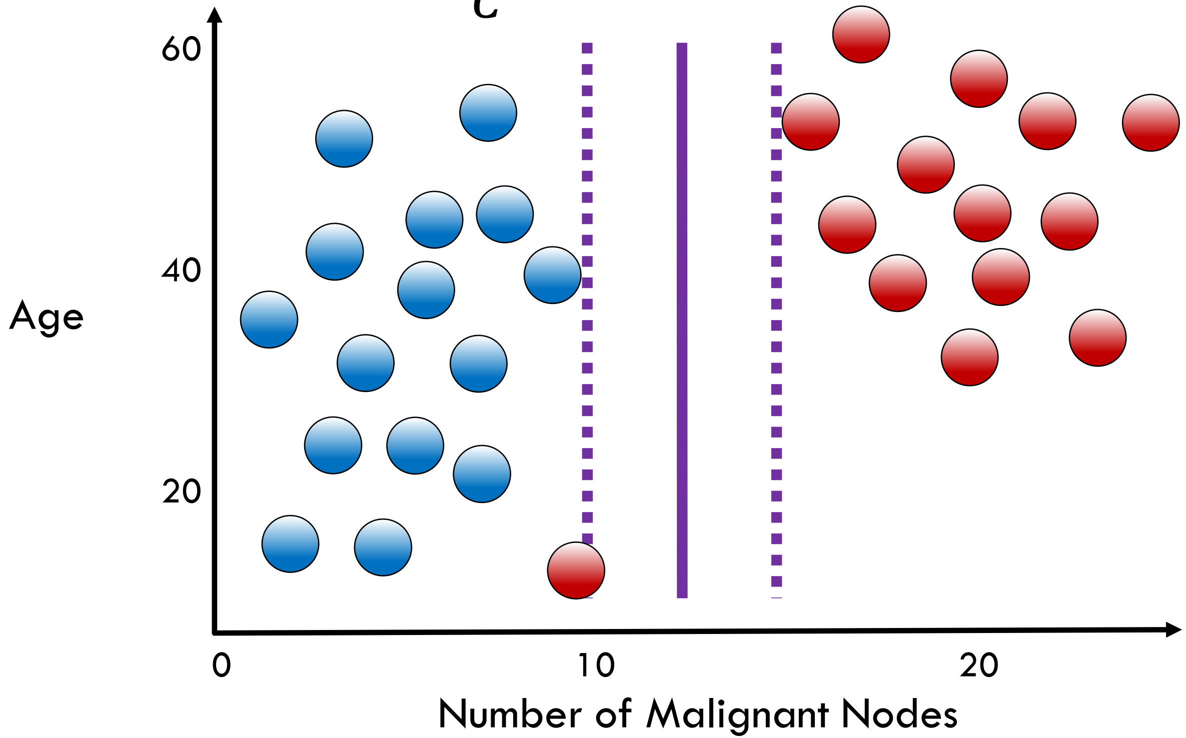
Outlier Sensitivity in SVMs

This is probably still the correct boundary



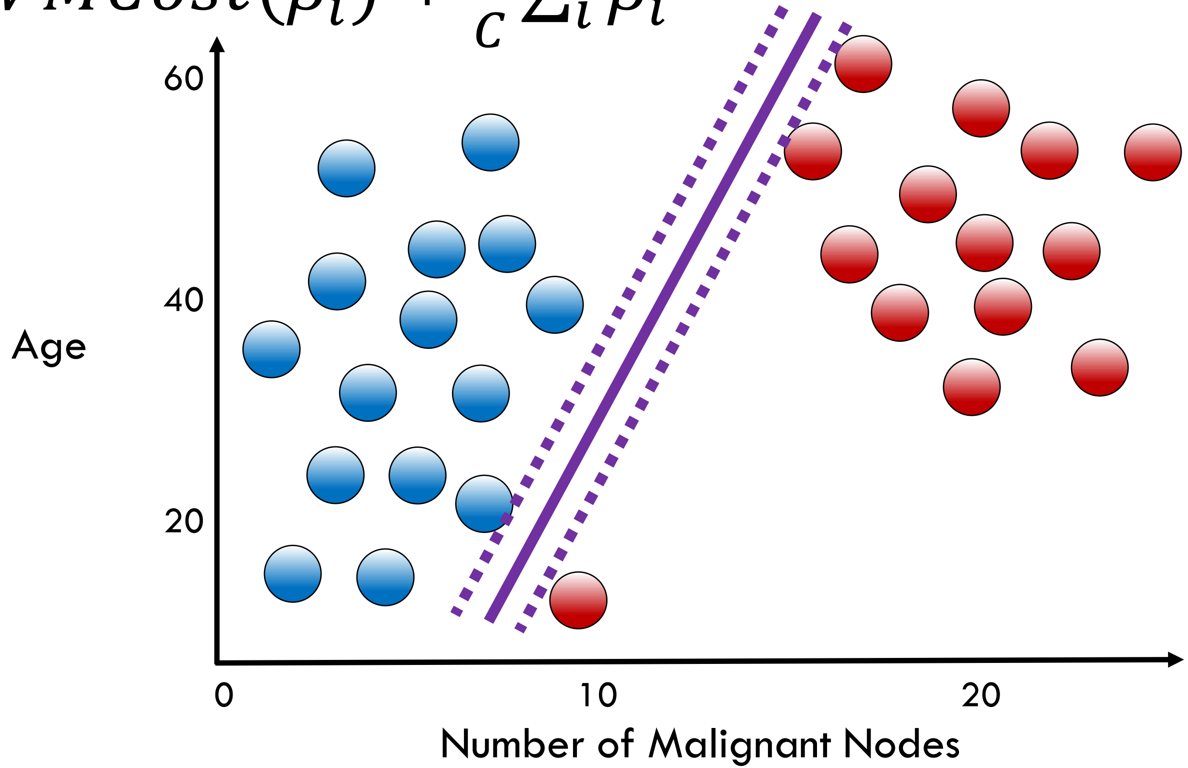
Regularization in SVMs

$$J(\beta_i) = SVMCost(\beta_i) + \frac{1}{C}\sum_i \beta_i$$



Regularization in SVMs

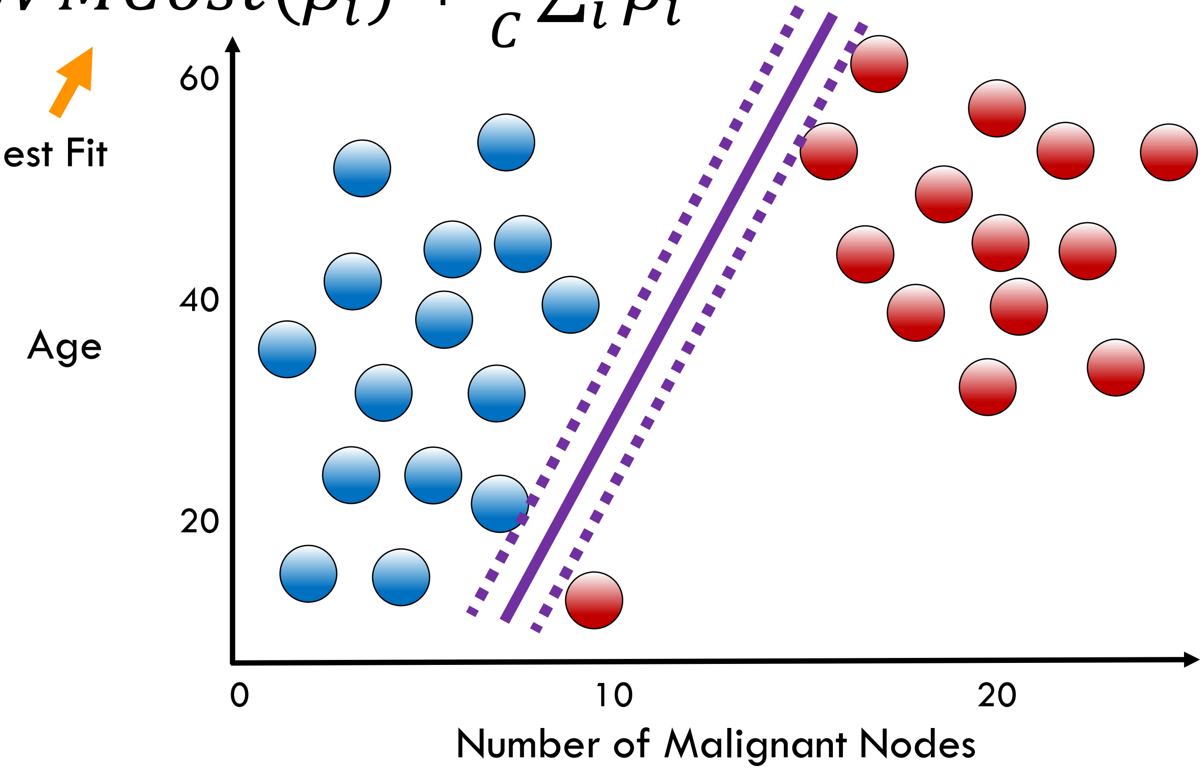
$$J(\beta_i) \ = SVMCost(\beta_i) + \frac{1}{C}\sum_i \beta_i$$



Regularization in SVMs

$$J(\beta_i) = SVMCost(\beta_i) + \frac{1}{C}\sum_i \beta_i$$

Best Fit

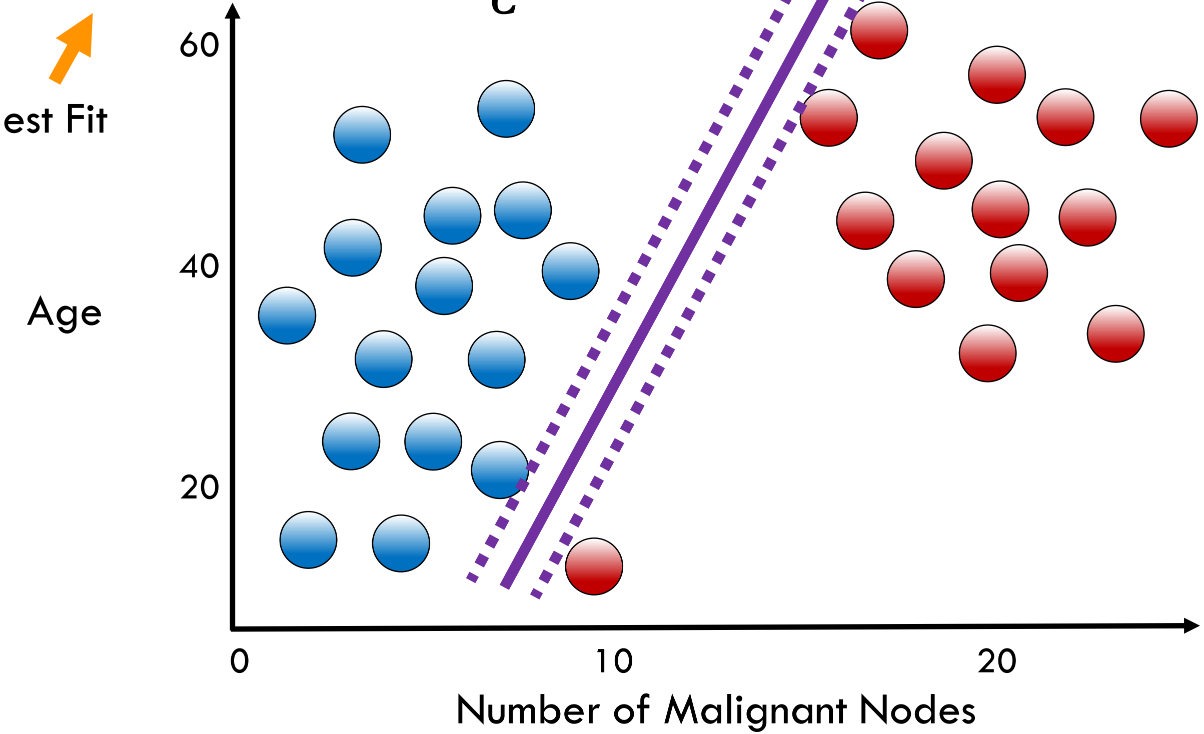


Regularization in SVMs

$$J(\beta_i) = SVMCost(\beta_i) + \frac{1}{C}\sum_i \beta_i$$

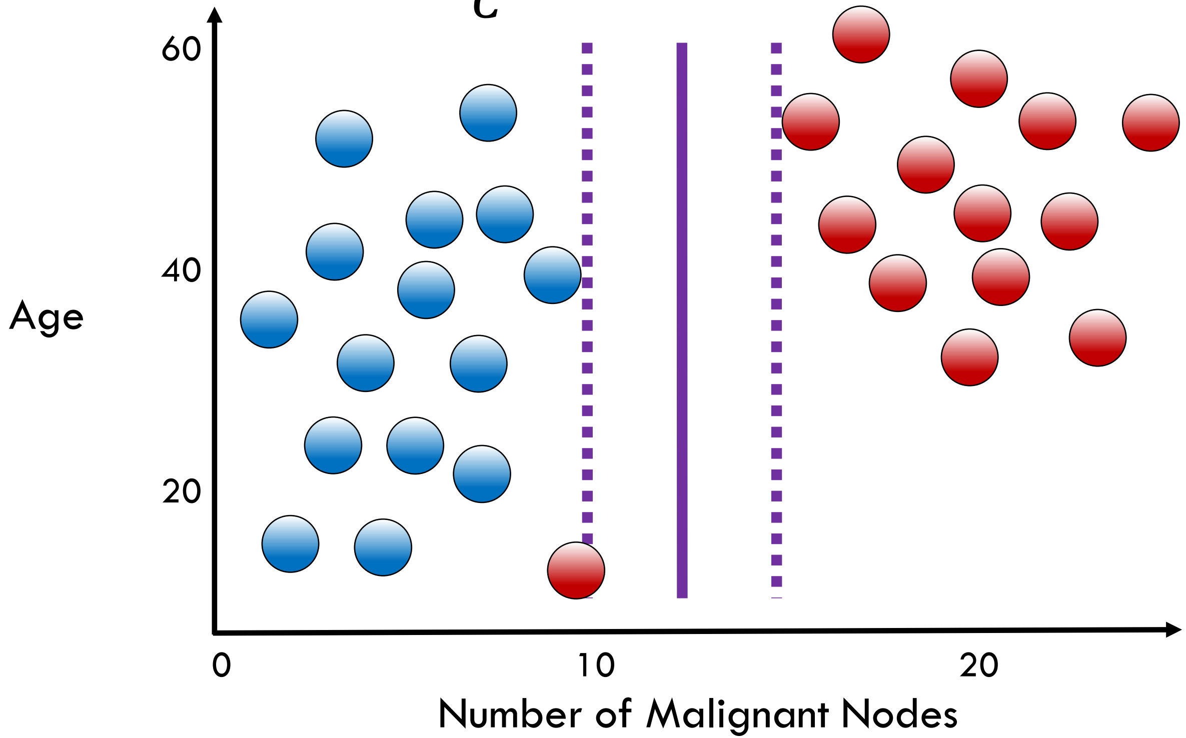
Large

Best Fit



Regularization in SVMs

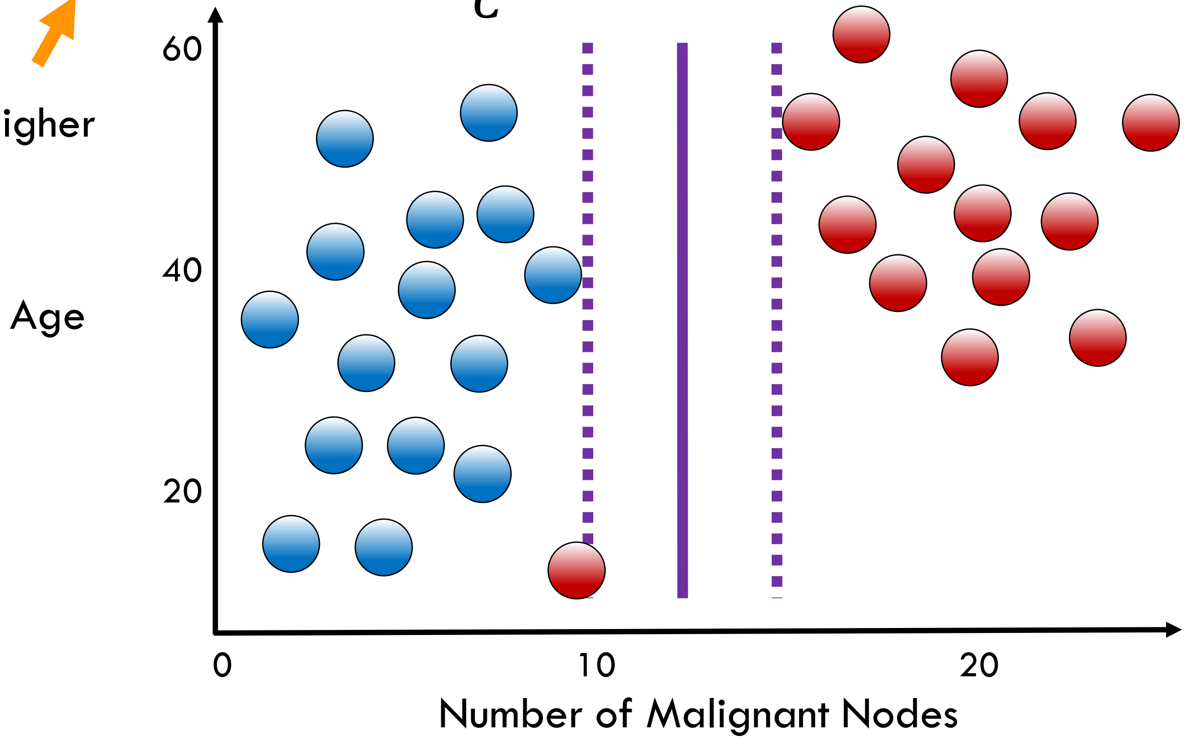
$$J(\beta_i) \;= SVMCost(\beta_i) + \frac{1}{C}\sum_i \beta_i$$



Regularization in SVMs

$$J(\beta_i) \;= SVMCost(\beta_i) + \frac{1}{C}\sum_i \beta_i$$

Slightly Higher

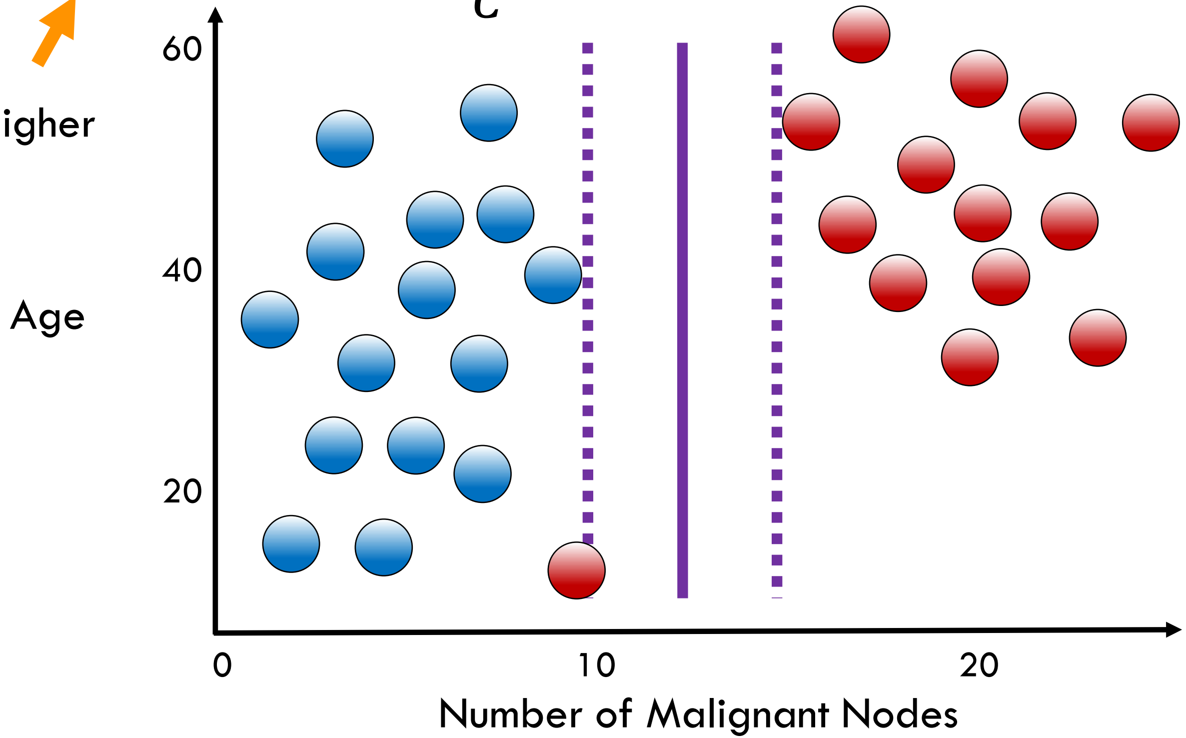


Regularization in SVMs

$$J(\beta_i) = SVMCost(\beta_i) + \frac{1}{C}\sum_i \beta_i$$

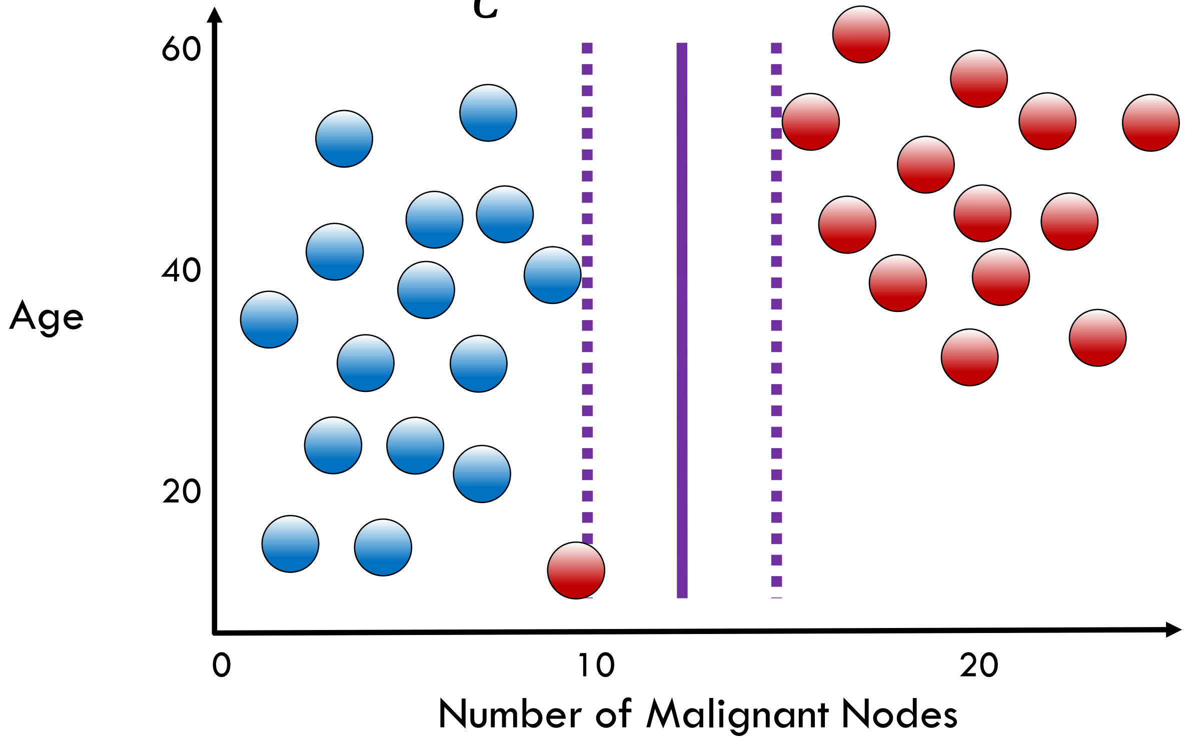
Much Smaller

Slightly Higher



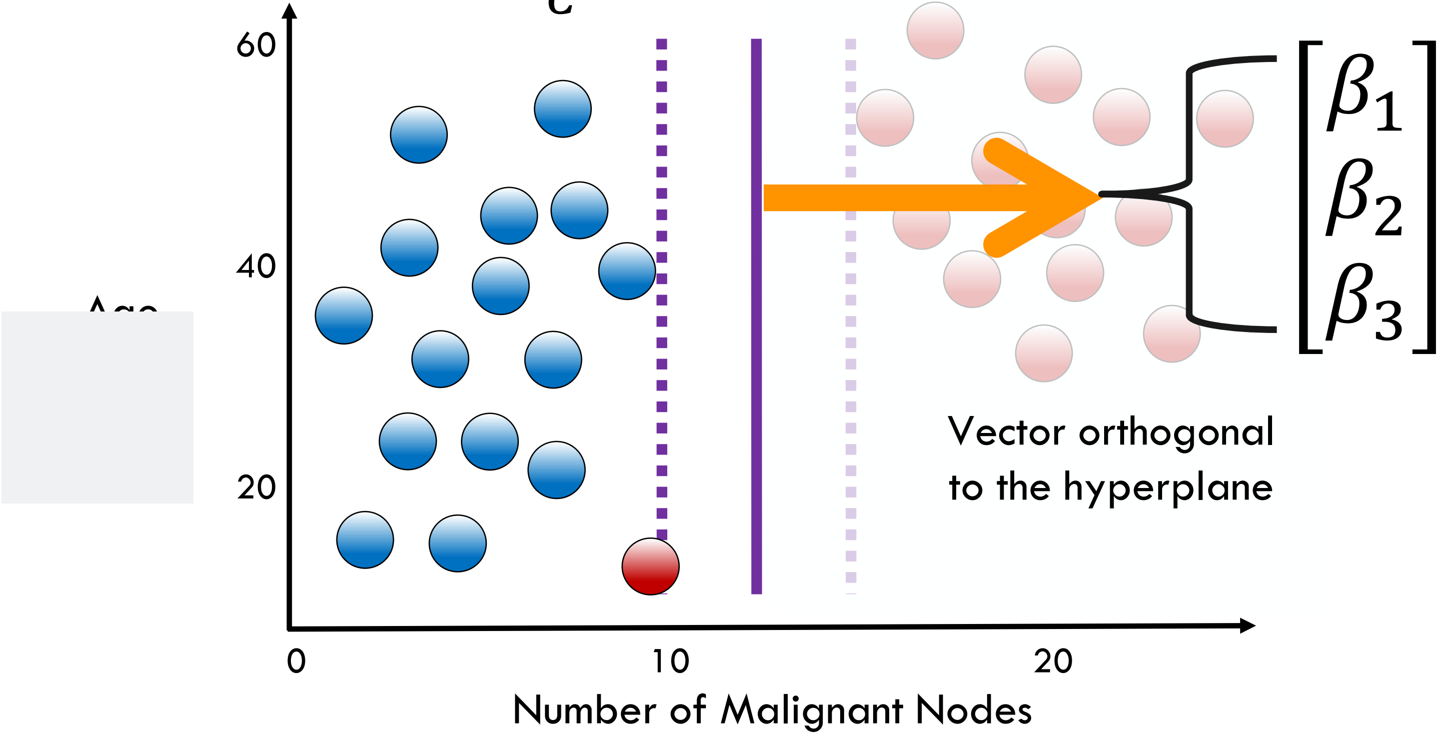
Interpretation of SVM Coefficients

$$J(\beta_i) = SVMCost(\beta_i) + \frac{1}{C}\sum_i \beta_i$$



Interpretation of SVM Coefficients

$$J(\beta_i) = SVMCost(\beta_i) + \frac{1}{C}\sum_i \beta_i$$



Linear SVM: The Syntax

Import the class containing the classification method

```
from sklearn.svm import LinearSVC
```

To use the Intel® Extension for Scikit-learn* variant of this algorithm:

- **Install Intel® oneAPI AI Analytics Toolkit (AI Kit)**
- **Add the following two lines of code after the above code:**

```
import patch_sklearn
patch_sklearn()
```

Linear SVM: The Syntax

Import the class containing the classification method

```
from sklearn.svm import LinearSVC
```

Linear SVM: The Syntax

Import the class containing the classification method

```
from sklearn.svm import LinearSVC
```

Create an instance of the class

```
LinSVC = LinearSVC(penalty='l2', C=10.0)
```

Linear SVM: The Syntax

Import the class containing the classification method

```
from sklearn.svm import LinearSVC
```

Create an instance of the class

```
LinSVC = LinearSVC(penalty='l2', C=10.0)
```

regularization parameters

Linear SVM: The Syntax

Import the class containing the classification method

```
from sklearn.svm import LinearSVC
```

Create an instance of the class

```
LinSVC = LinearSVC(penalty='l2', C=10.0)
```

Fit the instance on the data and then predict the expected value

```
LinSVC = LinSVC.fit(X_train, y_train)

y_predict = LinSVC.predict(X_test)
```

Linear SVM: The Syntax

Import the class containing the classification method

```
from sklearn.svm import LinearSVC
```

Create an instance of the class

```
LinSVC = LinearSVC(penalty='l2', C=10.0)
```

Fit the instance on the data and then predict the expected value

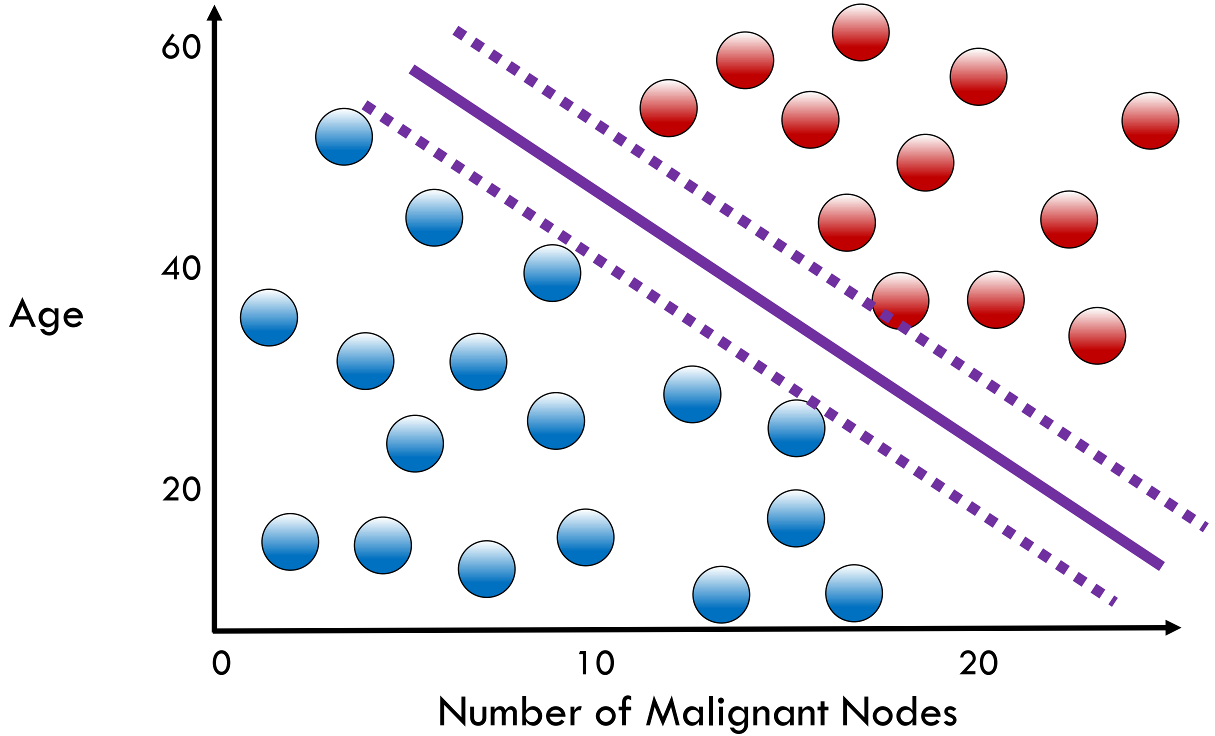
```
LinSVC = LinSVC.fit(X_train, y_train)
y_predict = LinSVC.predict(X_test)
```

Tune regularization parameters with cross-validation.

Software

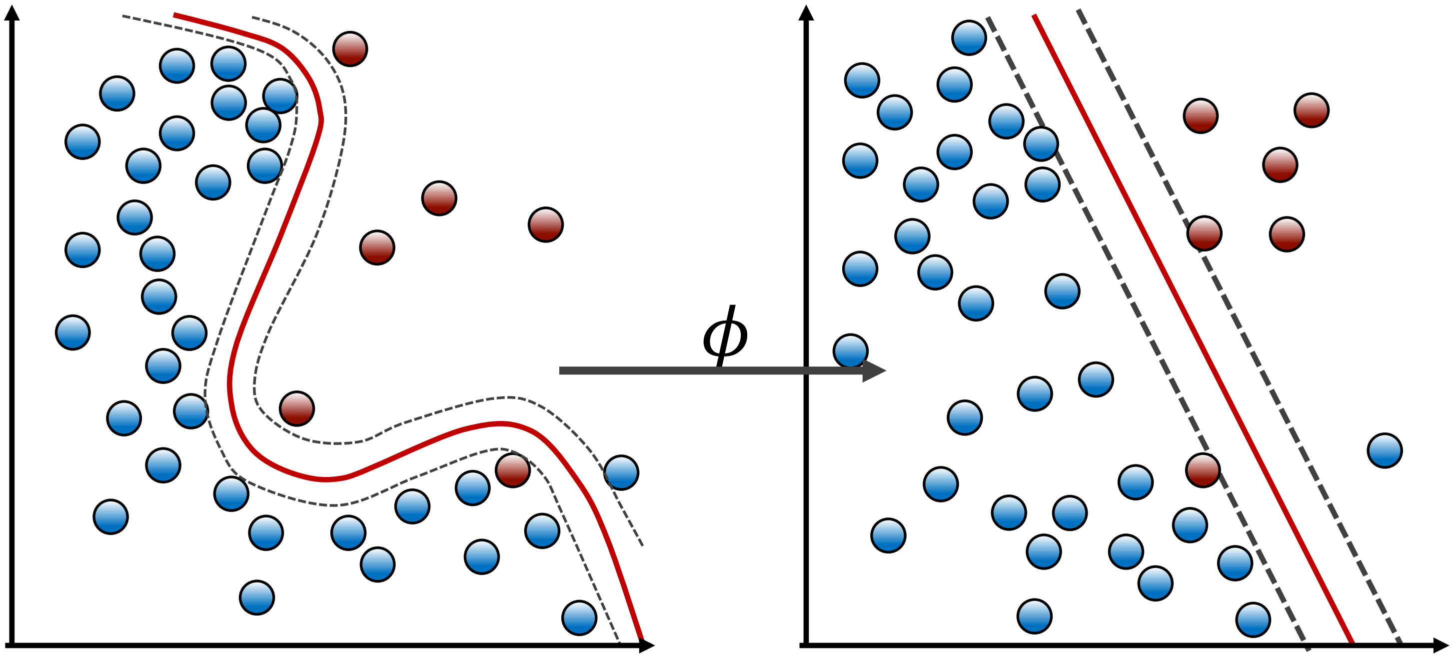
Kernels

Classification with SVMs



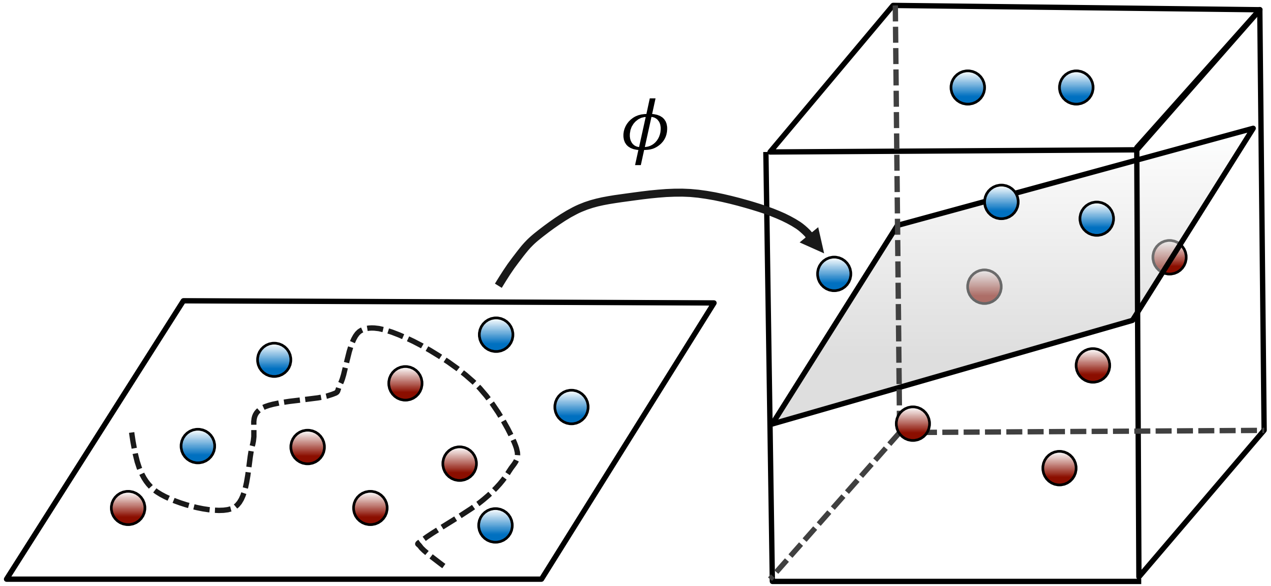
Non-Linear Decision Boundaries with SVM

Non-linear data can be made linear
with higher dimensionality



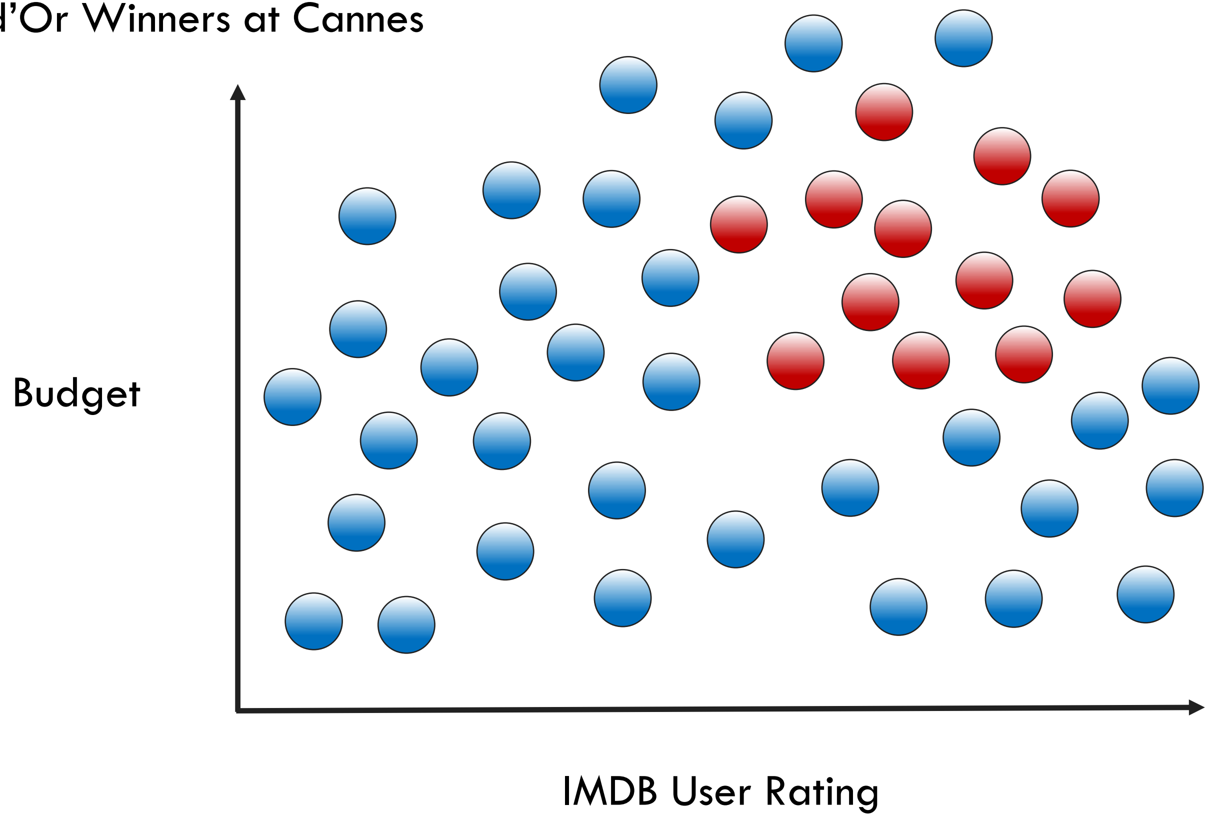
The Kernel Trick

Transform data so it is linearly separable



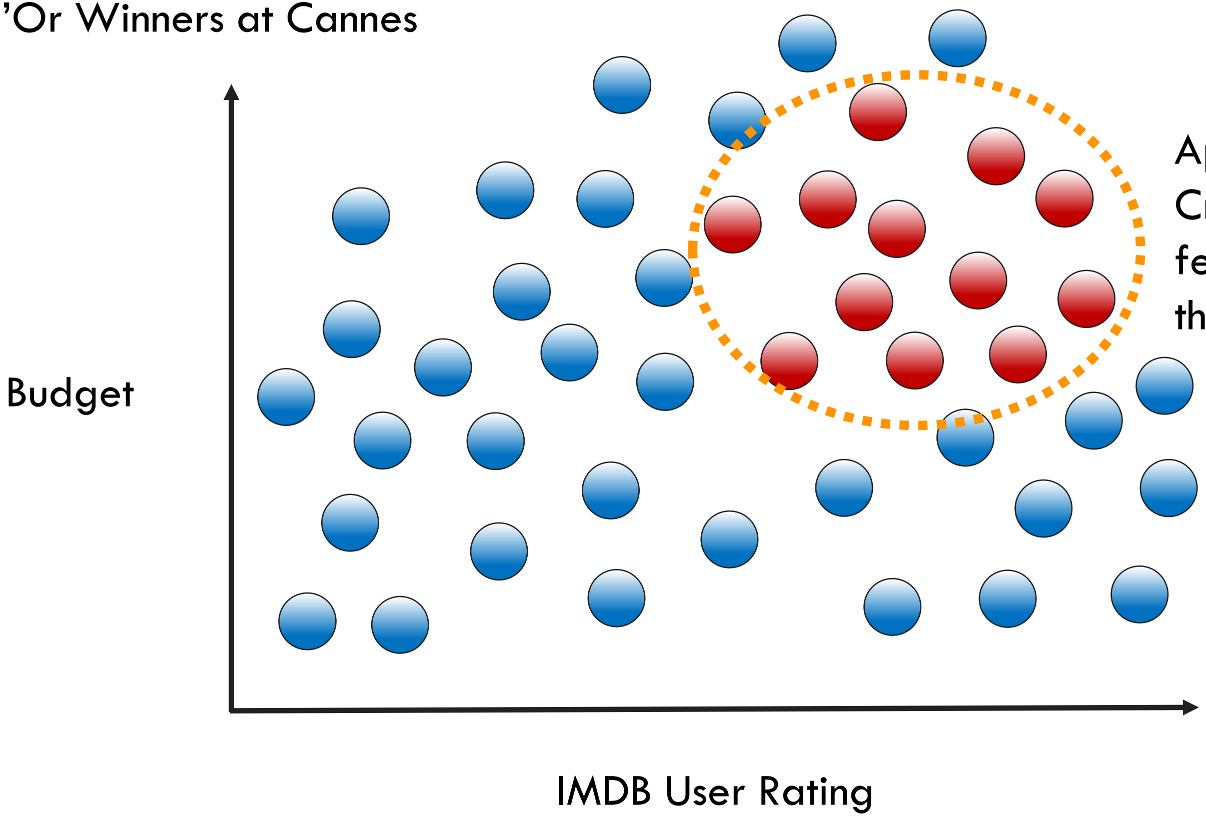
SVM Gaussian Kernel

Palme d'Or Winners at Cannes



SVM Gaussian Kernel

Palme d'Or Winners at Cannes

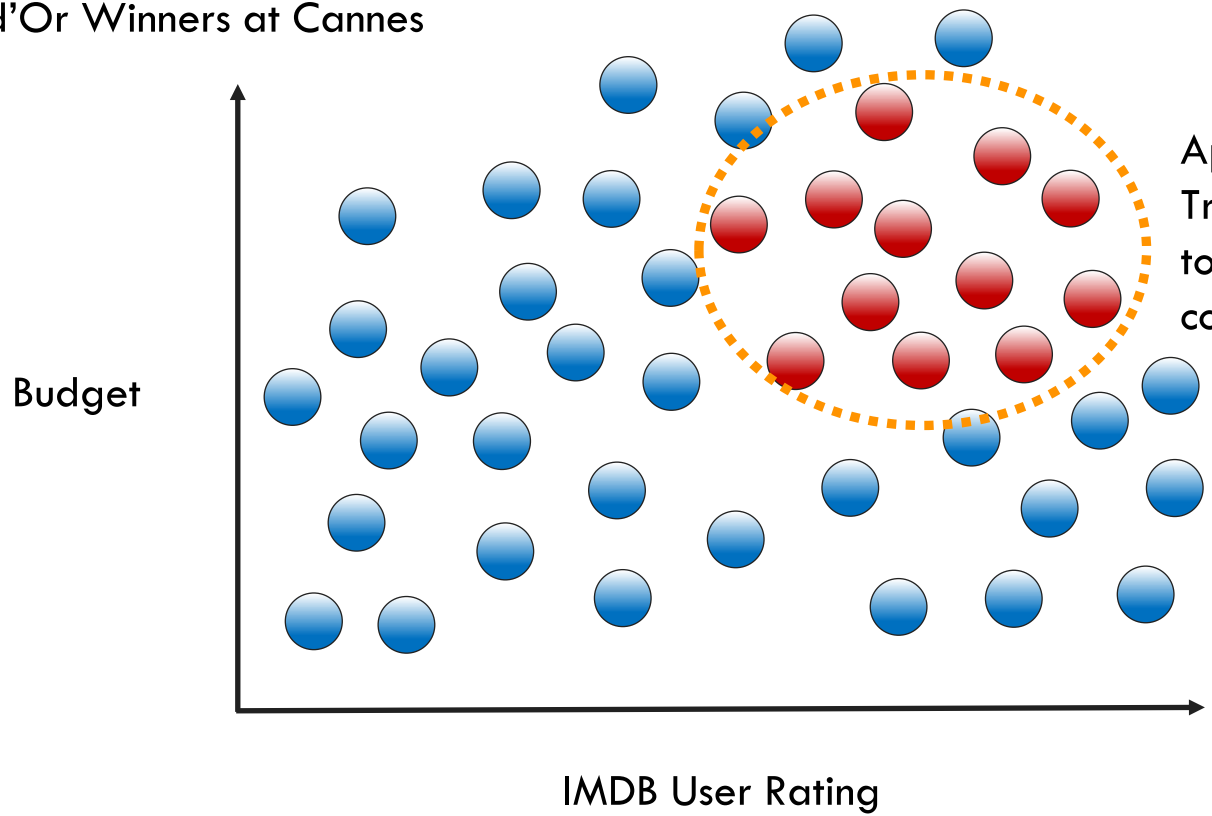


Approach 1:
Create higher order features to transform the data.

Budget^2 +
Rating^2 +
Budget * Rating +
…

SVM Gaussian Kernel

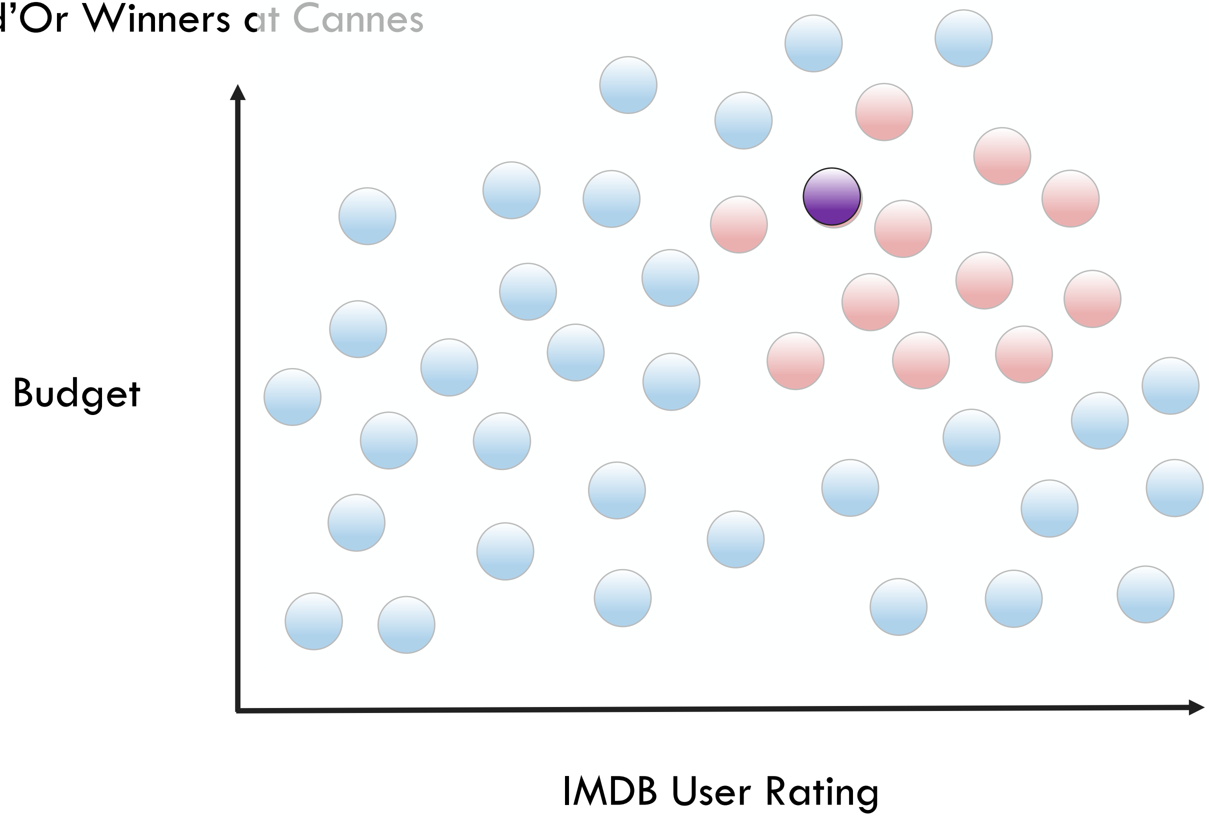
Palme d'Or Winners at Cannes



Approach 2:
Transform the space to a different coordinate system.

SVM Gaussian Kernel

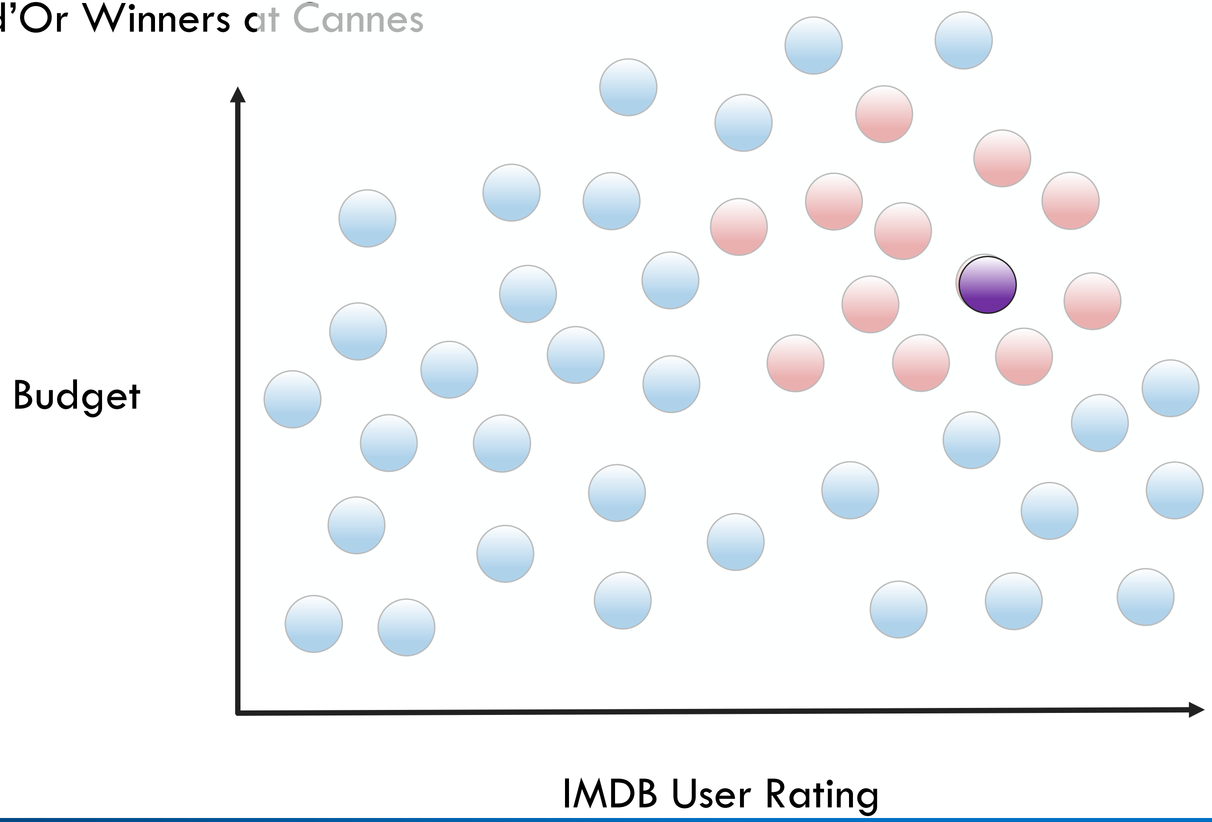
Palme d'Or Winners at Cannes



Define Feature 1: Similarity to "Pulp Fiction"

SVM Gaussian Kernel

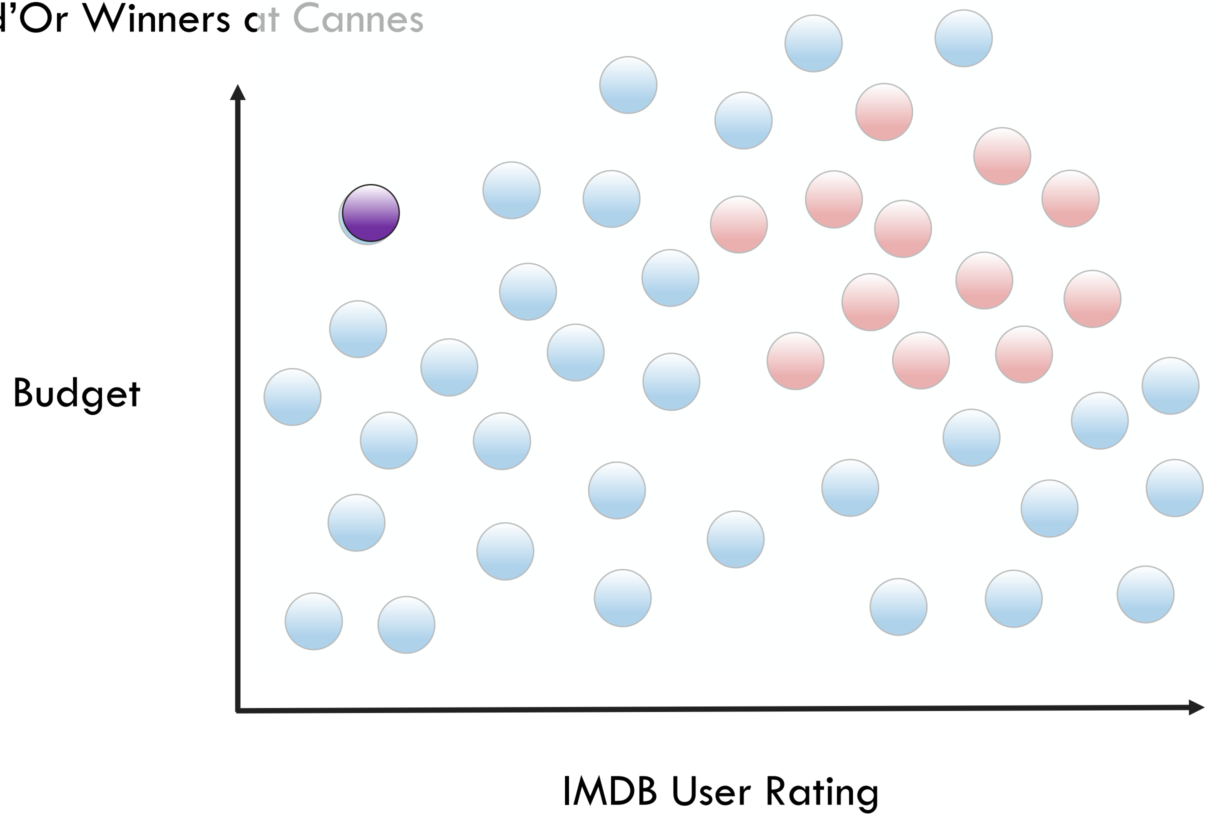
Palme d'Or Winners at Cannes



Define Feature 2: Similarity to "Black Swan"

SVM Gaussian Kernel

Palme d'Or Winners at Cannes



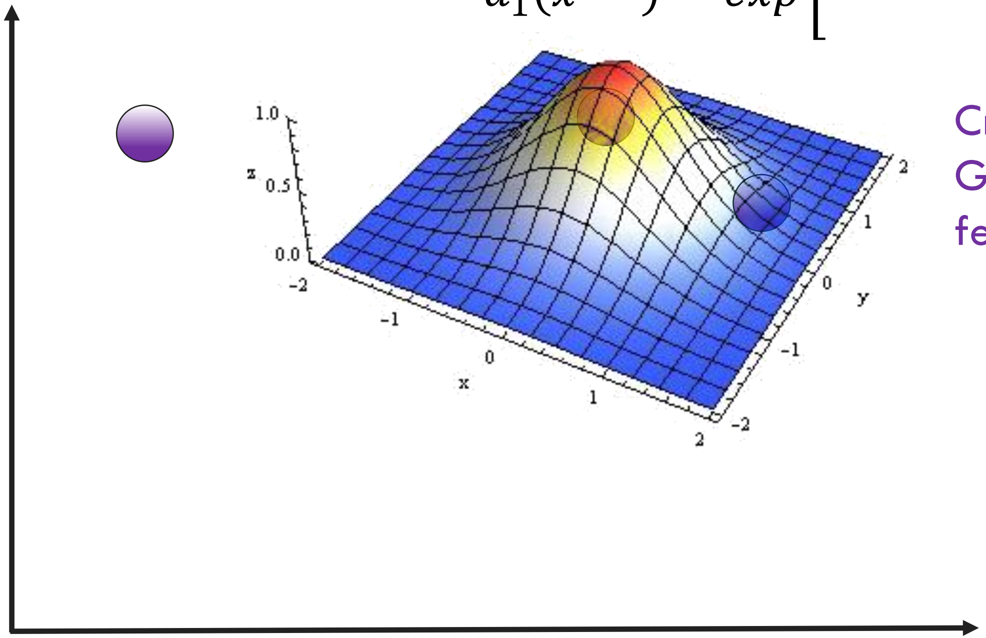
Define Feature 3: Similarity to "Transformers"

SVM Gaussian Kernel

Palme d'Or Winners at Cannes

$$a_1(x^{obs}) = exp\left[\frac{-\sum(x_i^{obs} - x_i^{Pulp\ Fiction})^2}{2\sigma^2}\right]$$

Budget



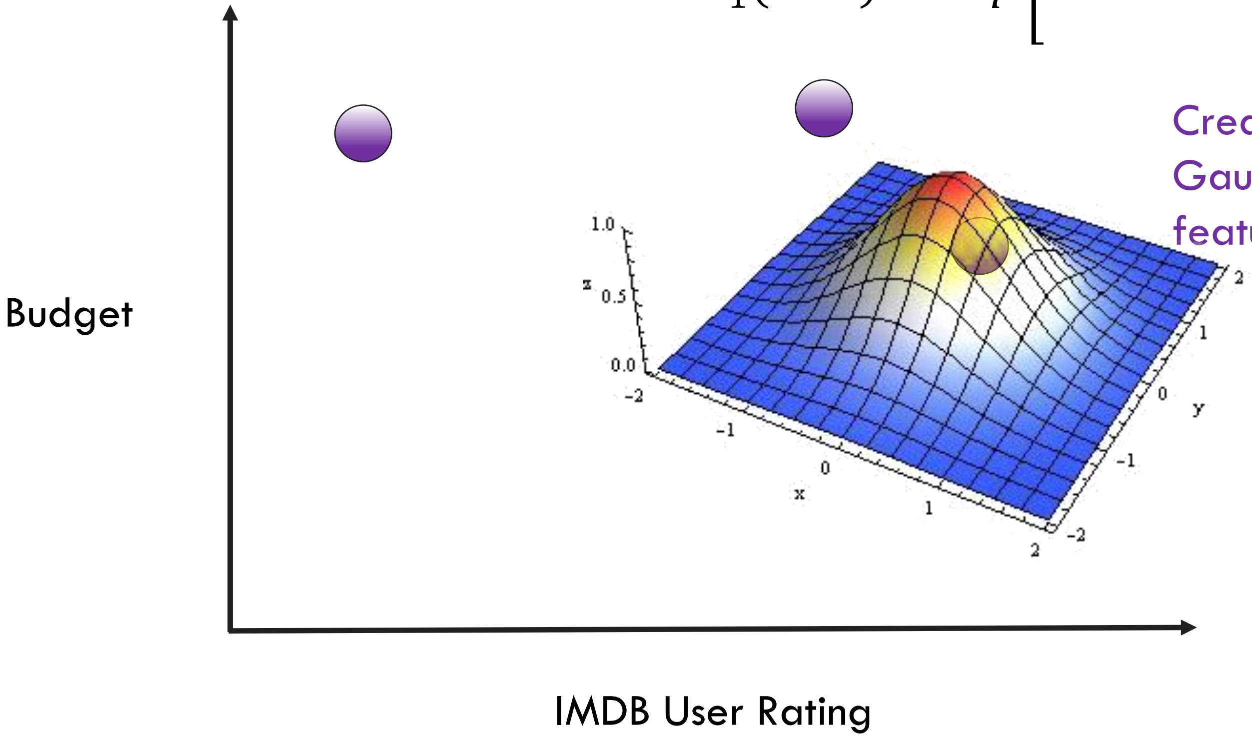
IMDB User Rating

Create a Gaussian function at feature 1

SVM Gaussian Kernel

Palme d'Or Winners at Cannes

$$a_1(x^{obs}) = exp\left[\frac{-\sum(x_i^{obs} - x_i^{Black\ Swan})^2}{2\sigma^2}\right]$$

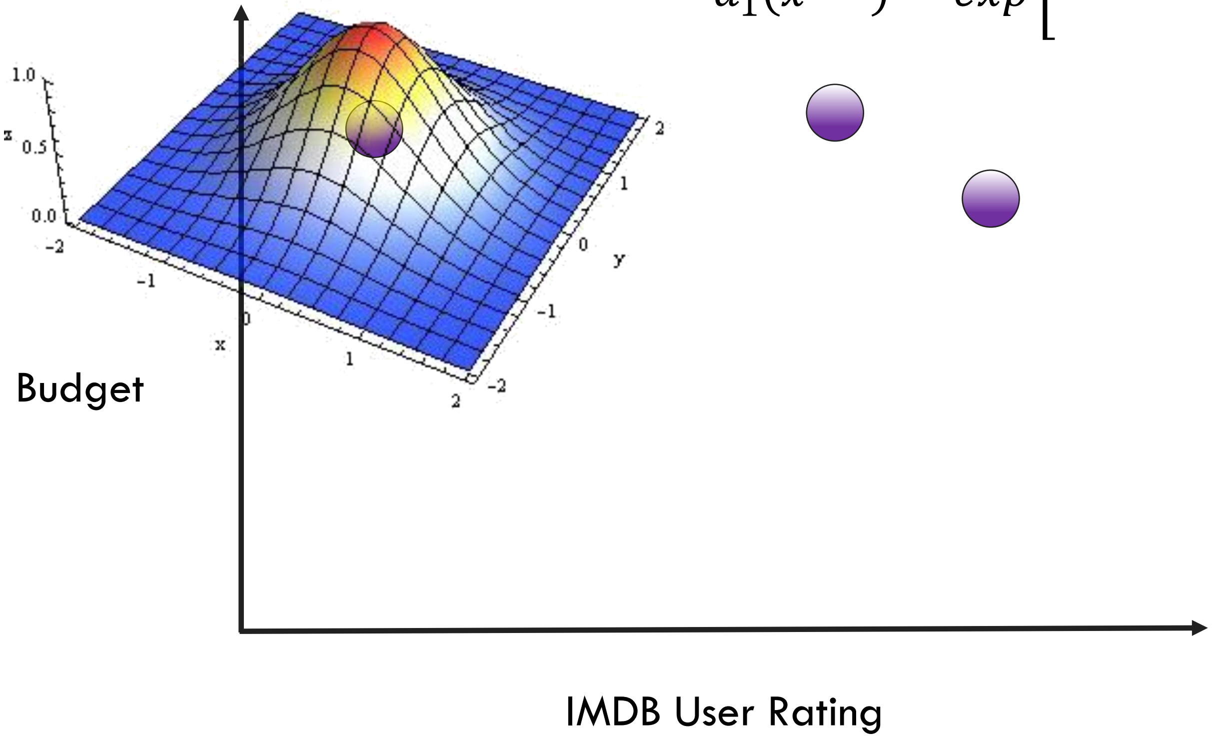


Create a Gaussian function at feature 2

SVM Gaussian Kernel

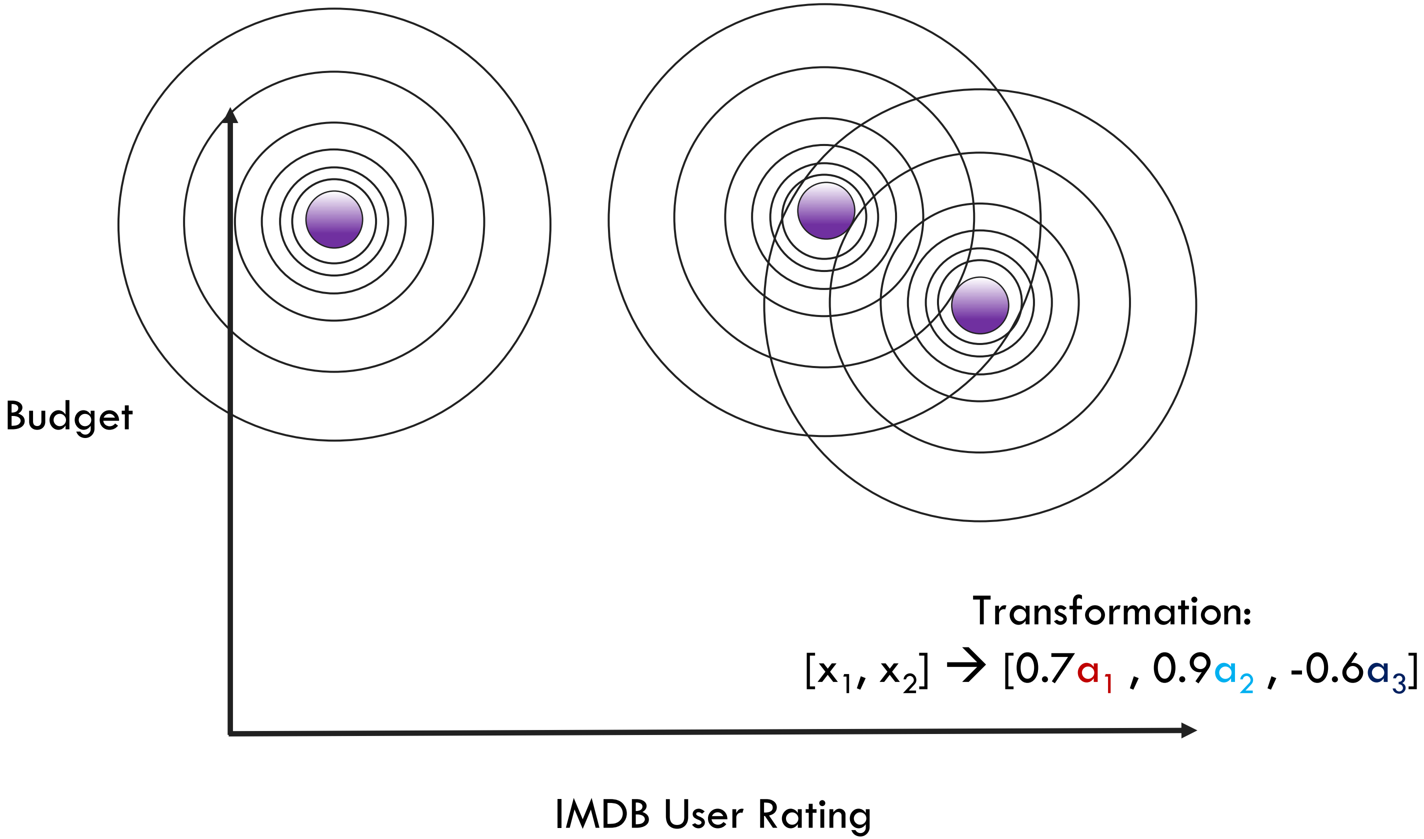
Palme d'Or Winners at Cannes

$$a_1(x^{obs}) = exp\left[\frac{-\sum(x_i^{obs} - x_i^{Transformers})^2}{2\sigma^2}\right]$$

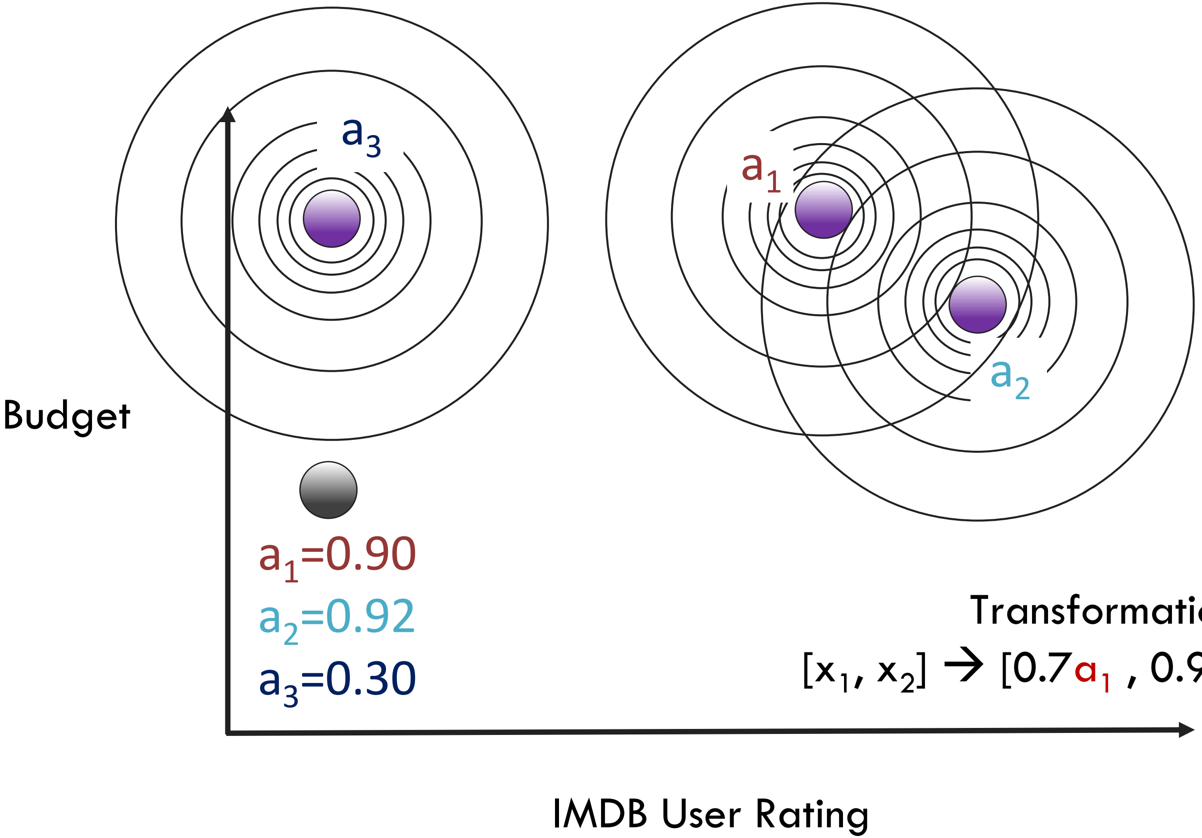


Create a Gaussian function at feature 3

SVM Gaussian Kernel



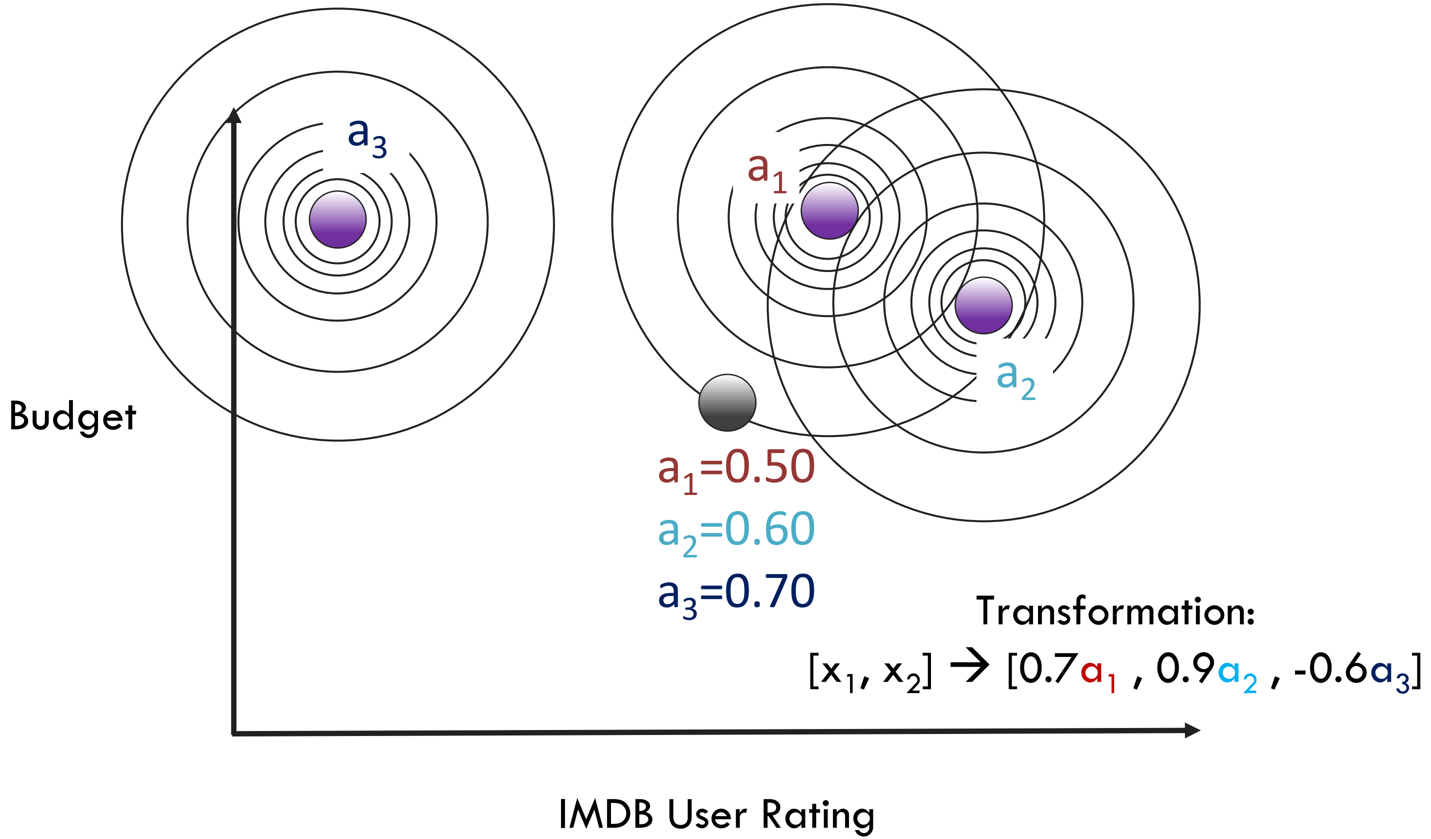
SVM Gaussian Kernel



Transformation:

$[x_1, x_2] \rightarrow [0.7a_1 \ , 0.9a_2 \ , -0.6a_3]$

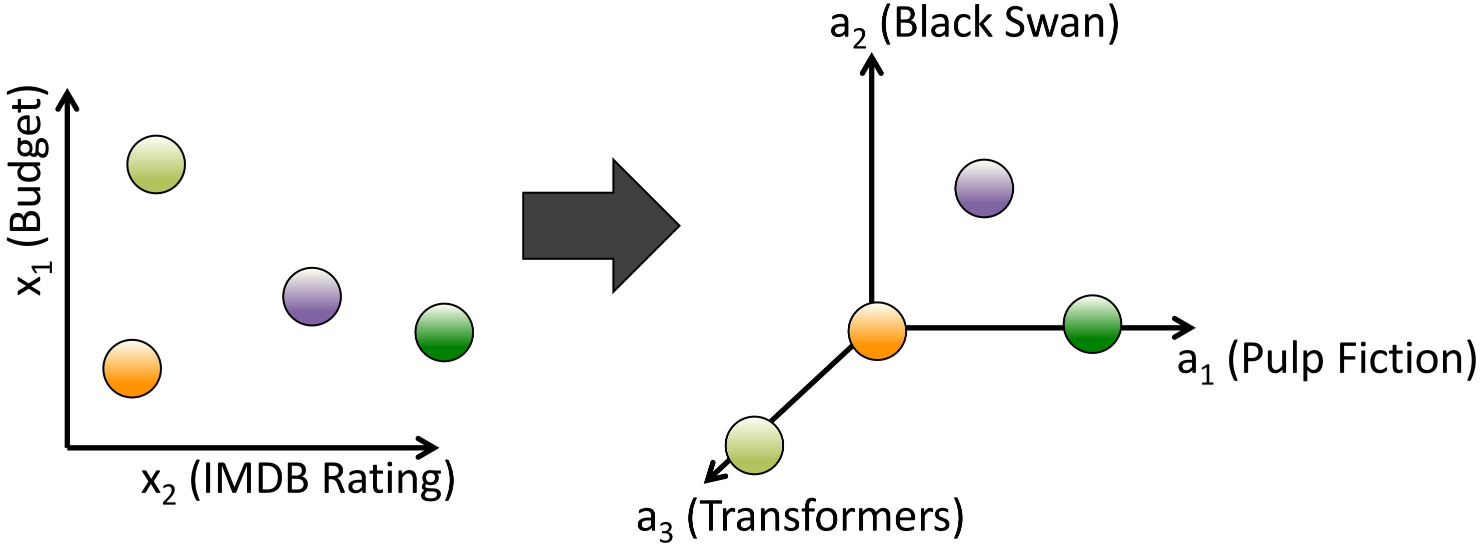
SVM Gaussian Kernel



SVM Gaussian Kernel

Transformation:

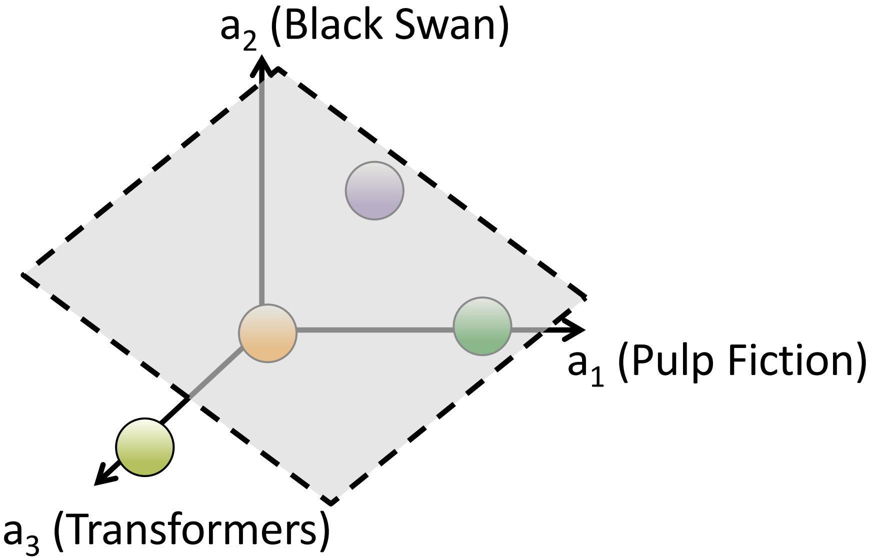
$[x_1, x_2] \rightarrow [0.7a_1 , 0.9a_2 , -0.6a_3]$



Classification in the New Space

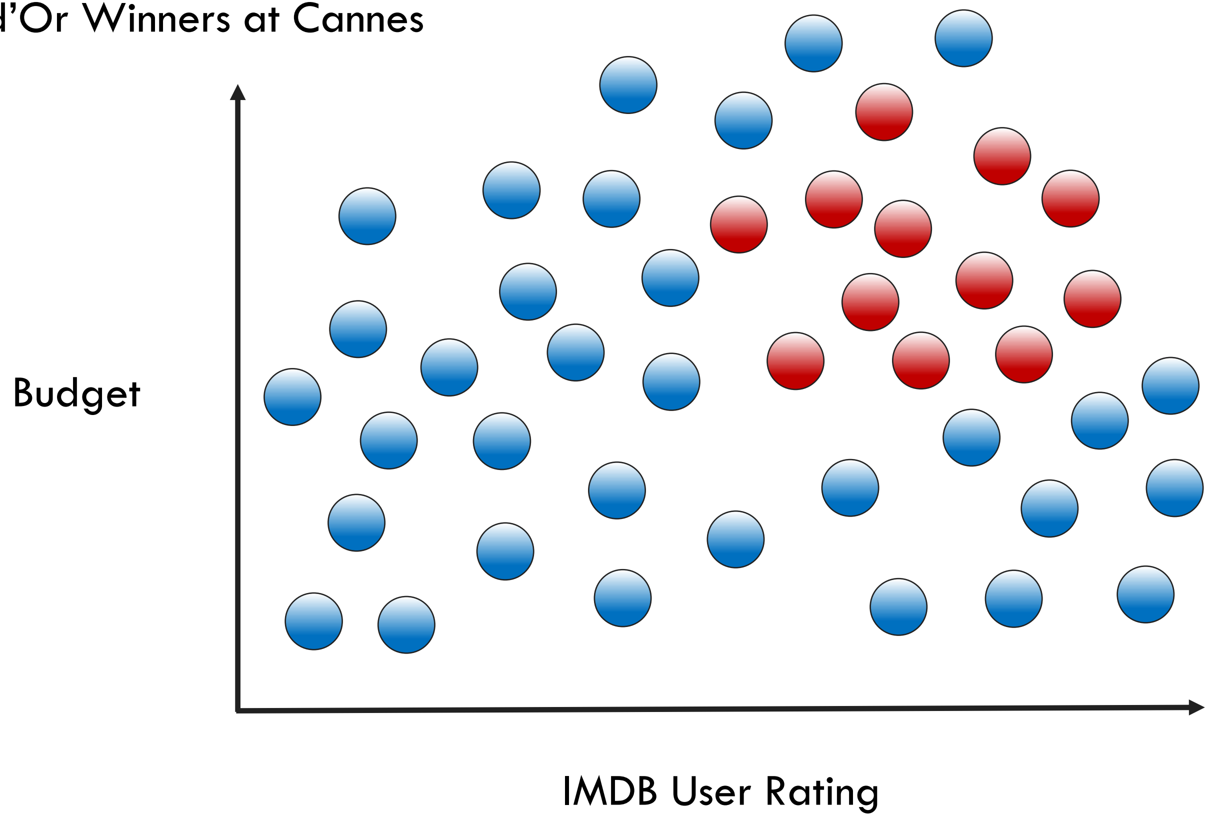
Transformation:

$[x_1, x_2] \rightarrow [0.7a_1 , 0.9a_2 , -0.6a_3]$



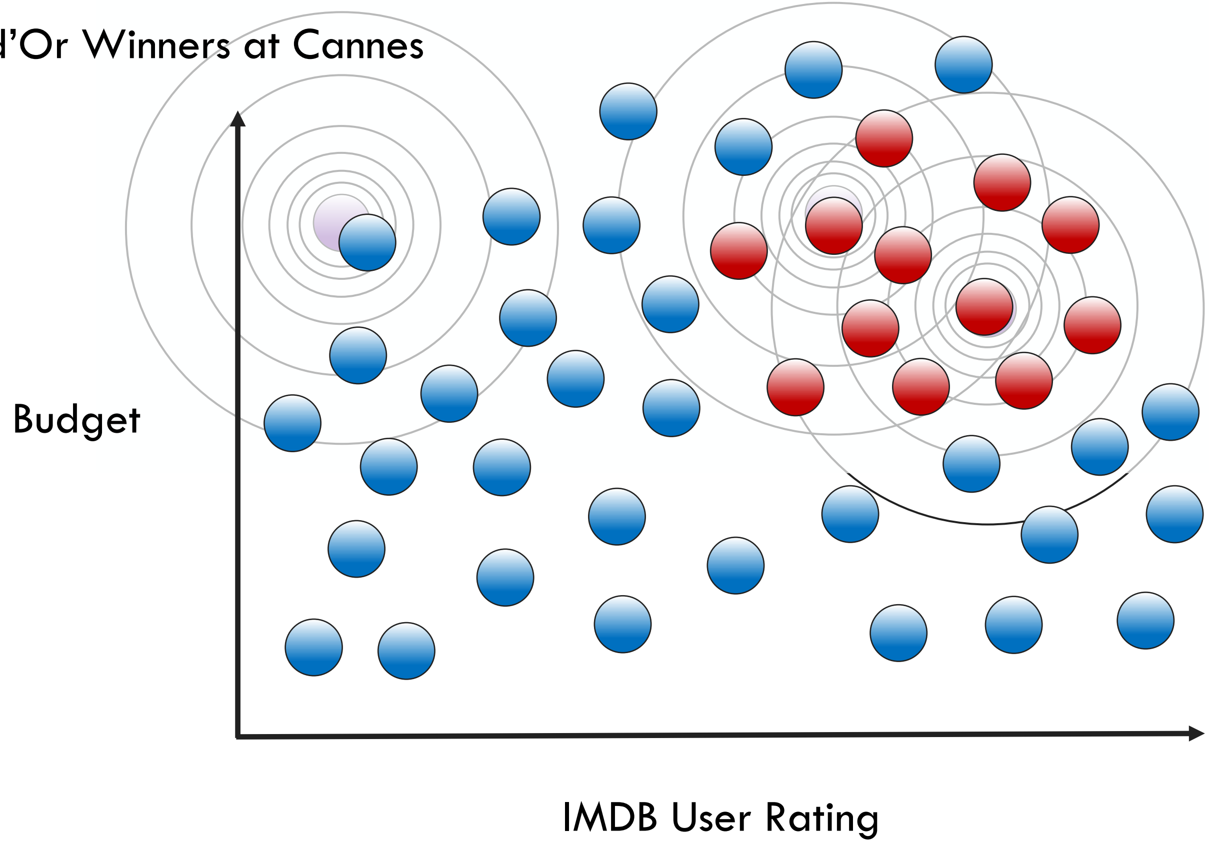
SVM Gaussian Kernel

Palme d'Or Winners at Cannes



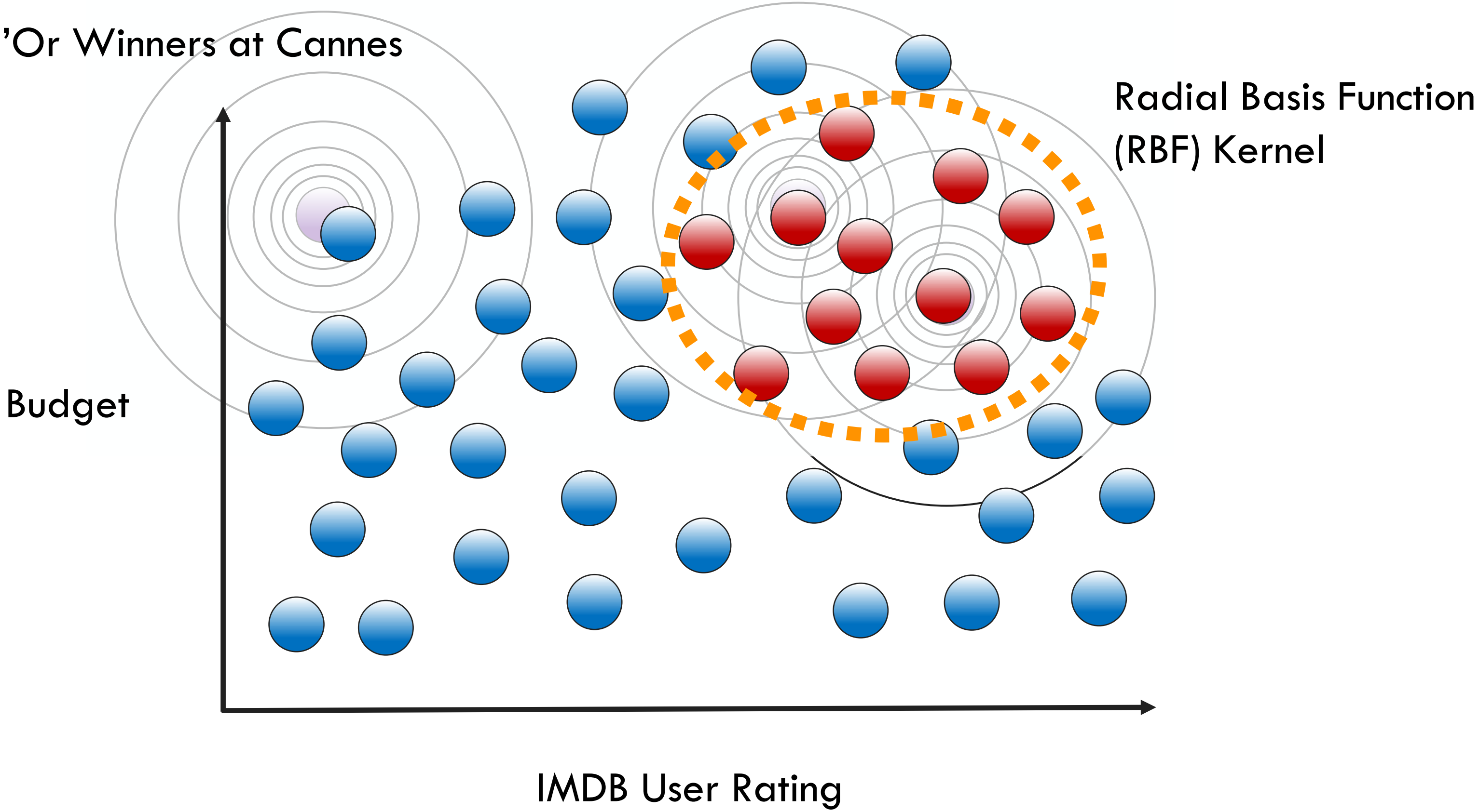
SVM Gaussian Kernel

Palme d'Or Winners at Cannes



SVM Gaussian Kernel

Palme d'Or Winners at Cannes



SVMs with Kernels: The Syntax

Import the class containing the classification method

```
from sklearn.svm import SVC
```

SVMs with Kernels: The Syntax

Import the class containing the classification method

```
from sklearn.svm import SVC
```

Create an instance of the class

```
rbfSVC = SVC(kernel='rbf', gamma=1.0, C=10.0)
```

SVMs with Kernels: The Syntax

Import the class containing the classification method

```
from sklearn.svm import SVC
```

Create an instance of the class

```
rbfSVC = SVC(kernel='rbf', gamma=1.0, C=10.0)
```

set kernel and associated coefficient (gamma)

SVMs with Kernels: The Syntax

Import the class containing the classification method

```
from sklearn.svm import SVC
```

Create an instance of the class

```
rbfSVC = SVC(kernel='rbf', gamma=1.0, C=10.0)
```

"C" is penalty associated with the error term

SVMs with Kernels: The Syntax

Import the class containing the classification method

```
from sklearn.svm import SVC
```

Create an instance of the class

```
rbfSVC = SVC(kernel='rbf', gamma=1.0, C=10.0)
```

Fit the instance on the data and then predict the expected value

```
rbfSVC = rbfSVC.fit(X_train, y_train)
y_predict = rbfSVC.predict(X_test)
```

SVMs with Kernels: The Syntax

Import the class containing the classification method

```
from sklearn.svm import SVC
```

Create an instance of the class

```
rbfSVC = SVC(kernel='rbf', gamma=1.0, C=10.0)
```

Fit the instance on the data and then predict the expected value

```
rbfSVC = rbfSVC.fit(X_train, y_train)
y_predict = rbfSVC.predict(X_test)
```

Tune kernel and associated parameters with cross-validation.

Feature Overload

Problem

SVMs with RBF Kernels are very slow to train with lots of features or data

Feature Overload

Problem

SVMs with RBF Kernels are very slow to train with lots of features or data

Solution

Construct approximate kernel map with SGD using Nystroem or RBF sampler

Feature Overload

Problem

SVMs with RBF Kernels are very slow to train with lots of features or data

Solution

Construct approximate kernel map with SGD using Nystroem or RBF sampler.

Fit a linear classifier.

Faster Kernel Transformations: The Syntax

Import the class containing the classification method

```
from sklearn.kernel_approximation import Nystroem
```

Create an instance of the class

```
nystroemSVC = Nystroem(kernel='rbf', gamma=1.0,
                       n_components=100)
```

Fit the instance on the data and transform

```
X_train = nystroemSVC.fit_transform(X_train)
X_test = nystroemSVC.transform(X_test)
```

Tune kernel parameters and components with cross-validation.

Faster Kernel Transformations: The Syntax

Import the class containing the classification method

```
from sklearn.kernel_approximation import Nystroem
```

Create an instance of the class

```
nystroemSVC = Nystroem(kernel='rbf', gamma=1.0,
                       n_components=100)
```

multiple non-linear kernels can be used

Fit the instance on the data and transform

```
X_train = nystroemSVC.fit_transform(X_train)
X_test = nystroemSVC.transform(X_test)
```

Tune kernel parameters and components with cross-validation.

Faster Kernel Transformations: The Syntax

Import the class containing the classification method

```
from sklearn.kernel_approximation import Nystroem
```

Create an instance of the class

```
nystroemSVC = Nystroem(kernel='rbf', gamma=1.0,
              n_components=100)
```

kernel and gamma are identical to SVC

Fit the instance on the data and transform

```
X_train = nystroemSVC.fit_transform(X_train)
X_test = nystroemSVC.transform(X_test)
```

Tune kernel parameters and components with cross-validation.

Faster Kernel Transformations: The Syntax

Import the class containing the classification method

```
from sklearn.kernel_approximation import Nystroem
```

Create an instance of the class

```
nystroemSVC = Nystroem(kernel='rbf', gamma=1.0,
                       n_components=100)
```

n_components is number of samples

Fit the instance on the data and transform

```
X_train = nystroemSVC.fit_transform(X_train)
X_test = nystroemSVC.transform(X_test)
```

Tune kernel parameters and components with cross-validation.

Faster Kernel Transformations: The Syntax

Import the class containing the classification method

```
from sklearn.kernel_approximation import RBFsampler
```

Create an instance of the class

```
rbfSample = RBFsampler(gamma=1.0,
                        n_components=100)
```

Fit the instance on the data and transform

```
X_train = rbfSample.fit_transform(X_train)
X_test = rbfSample.transform(X_test)
```

Tune kernel parameters and components with cross-validation.

Faster Kernel Transformations: The Syntax

Import the class containing the classification method

```
from sklearn.kernel_approximation import RBFsampler
```

Create an instance of the class

```
rbfSample = RBFsampler(gamma=1.0,
                       n_components=100)
```

RBF is only kernel that can be used

Fit the instance on the data and transform

```
X_train = rbfSample.fit_transform(X_train)
X_test = rbfSample.transform(X_test)
```

Tune kernel parameters and components with cross-validation.

Faster Kernel Transformations: The Syntax

Import the class containing the classification method

```
from sklearn.kernel_approximation import RBFsampler
```

Create an instance of the class

```
rbfSample = RBFsampler(gamma=1.0,
                       n_components=100)
```

parameter names are identical to previous

Fit the instance on the data and transform

```
X_train = rbfSample.fit_transform(X_train)
X_test = rbfSample.transform(X_test)
```

Tune kernel parameters and components with cross-validation.

When to Use Logistic Regression vs SVC

Features	Data	Model Choice
Many (~10K Features)	Small (1K rows)	Simple, Logistic or LinearSVC
Few (<100 Features)	Medium (~10k rows)	SVC with RBF
Few (<100 Features)	Many (>100K Points)	Add features, Logistic, LinearSVC or Kernel Approx.

Software