3D geoinformation
Department of Urbanism
Faculty of Architecture and the Built Environment
Delft University of Technology

Lab Session
Using SVM with Scikit Learn

Shenglan Du

scikit-learn

Machine Learning in Python

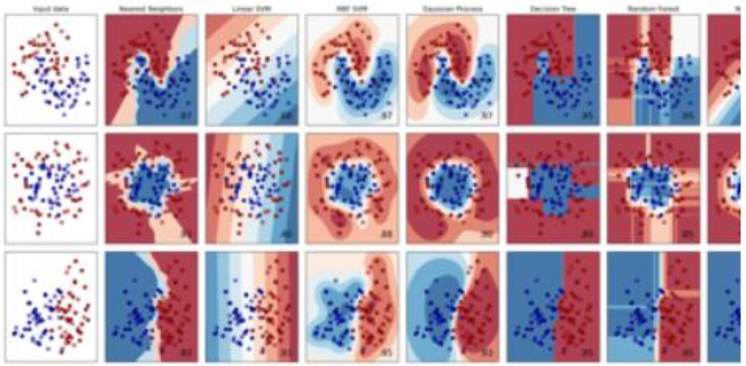
Getting Started | Release Highlights for 1.0 | GitHub

- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying which category an object belongs to.

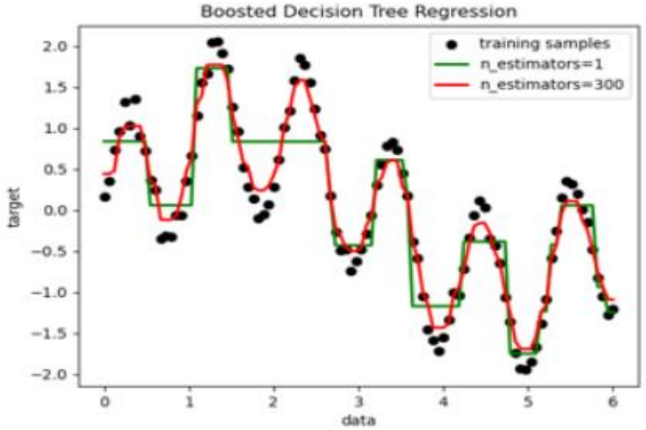
Applications: Spam detection, image recognition.
Algorithms: SVM, nearest neighbors, random forest, and more...



Regression

Predicting a continuous-valued attribute associated with an object.

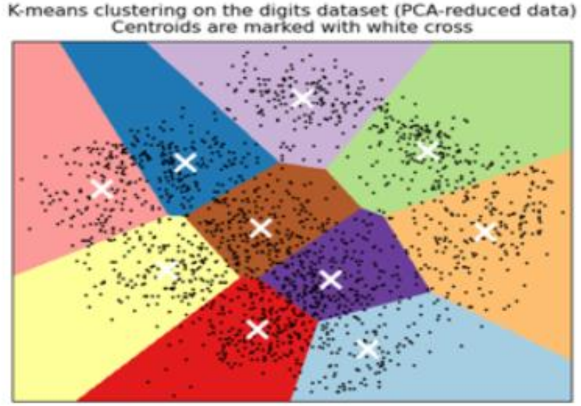
Applications: Drug response, Stock prices.
Algorithms: SVR, nearest neighbors, random forest, and more...



Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, and more...



https://scikit-learn.org/stable/

Scikit-Learn

- A free machine learning library in Python, featuring:
  - Classification
  - Regression
  - Clustering

- Supports algorithms such as SVM, Random forest, K-means, etc.

- Online documentation: https://scikit-learn.org/stable/

Scikit-Learn: Getting Started

- Install using either pip or conda

```
$ pip install -U scikit-learn
```

- In case you would like to check your installation

```
$ python -m pip show scikit-learn  # to see which version and where scikit-learn is installed
$ python -m pip freeze  # to see all packages installed in the active virtualenv
$ python -c "import sklearn; sklearn.show_versions()"
```

Scikit-Learn: SVM

- 3 versions of SVM classifiers are provided
  - SVC: commonly used in practice
  - NuSVC: similar to SVC, has slightly different yet equivalent mathematical formulations and parameter set
  - LinearSVC: faster implementation of SVM, but can only adopt linear kernels

A Complete SVC Classifier

sklearn.svm.SVC ¶

class `sklearn.svm.SVC`(**, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None*) [source]

- Documentation:
  https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC
- User guide:
  https://scikit-learn.org/stable/modules/svm.html#svm-classification

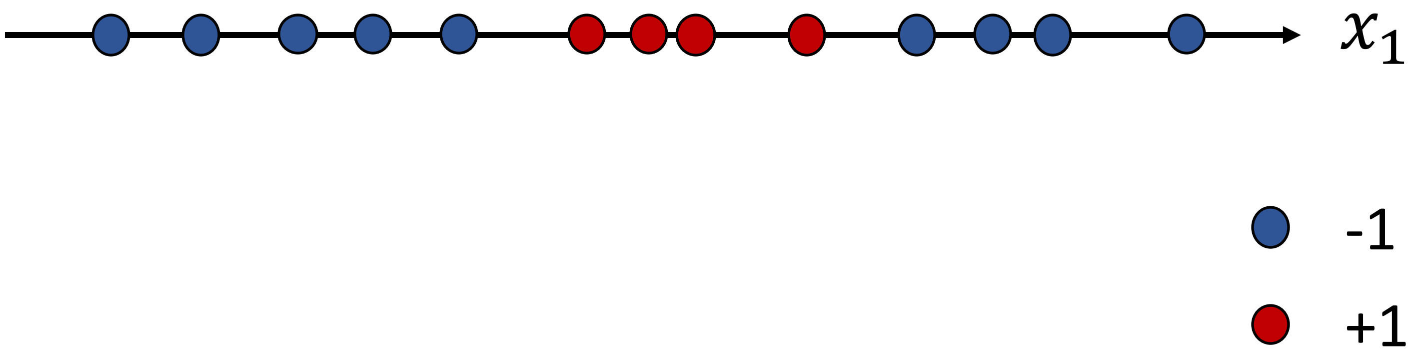
Hyperparameters and Arguments

- *C*: the regularization hyperparameter
- *kernel*: a trick you can use to transform input features
- *class_weight*: specify the weight per class
- *max_iter*: hard limit on iterations within solver, or -1 for no limit.
- *decision_function_shape*:
    - ‘ovr’: one to rest, default
    - ‘ovo’: one to one

Support Vector Machine using Kernels

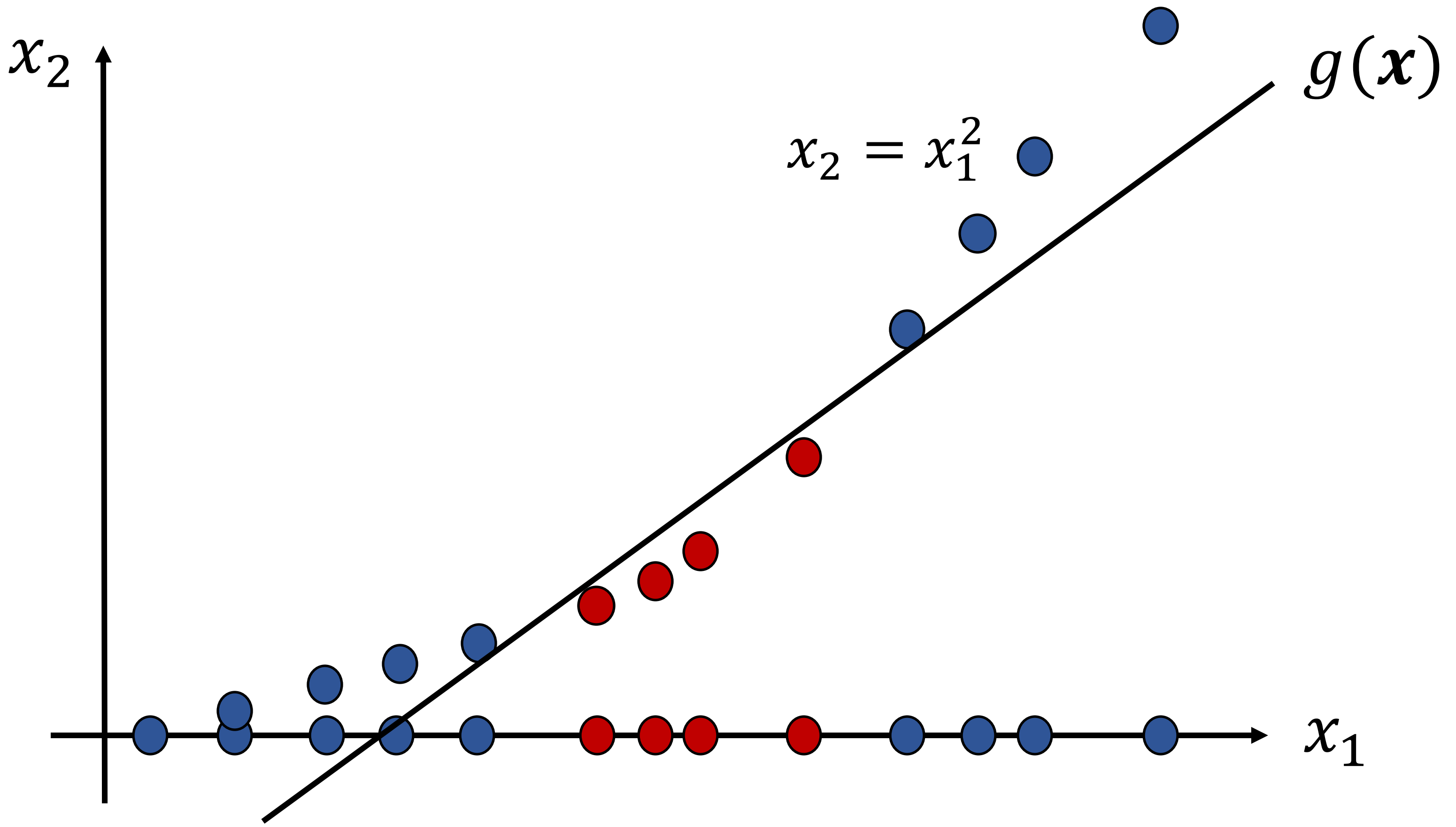
Why Kernel SVM?

- Classification on a 1D feature space



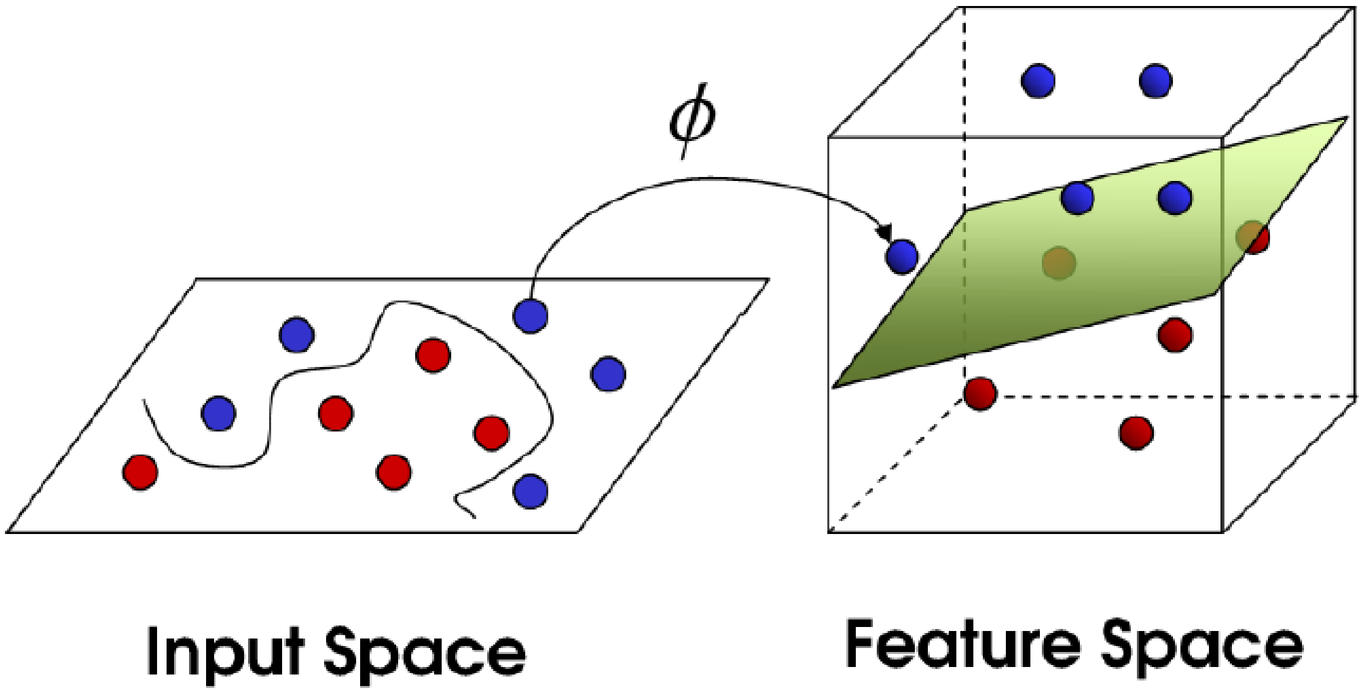
Why Kernel SVM?

- Transforming features to higher dimensions to fit g(x)



Why Kernel SVM?

- Classification on a high-dimensional feature space



Adapted from WI4635, TUDelft

Kernel SVM: How?

- Recall the SVM solution:

$$\boldsymbol{w} = \sum_{i=1}^{n} \lambda_i y_i \boldsymbol{x_i}$$

- Bring this solution back to the model:

$$f(\boldsymbol{x}) = \boldsymbol{w}^{\boldsymbol{T}} \boldsymbol{x} + b = \sum_{i=1}^{n} \lambda_i y_i \boldsymbol{x_i}^{\boldsymbol{T}} \boldsymbol{x} + b$$

Kernel SVM: How?

- After applying a feature transformation function $\Phi(\boldsymbol{x})$

$$f(\Phi(\boldsymbol{x})) = \sum_{i=1}^{n} \lambda_i y_i \Phi(\boldsymbol{x_i})^{\boldsymbol{T}} \Phi(\boldsymbol{x}) + b$$

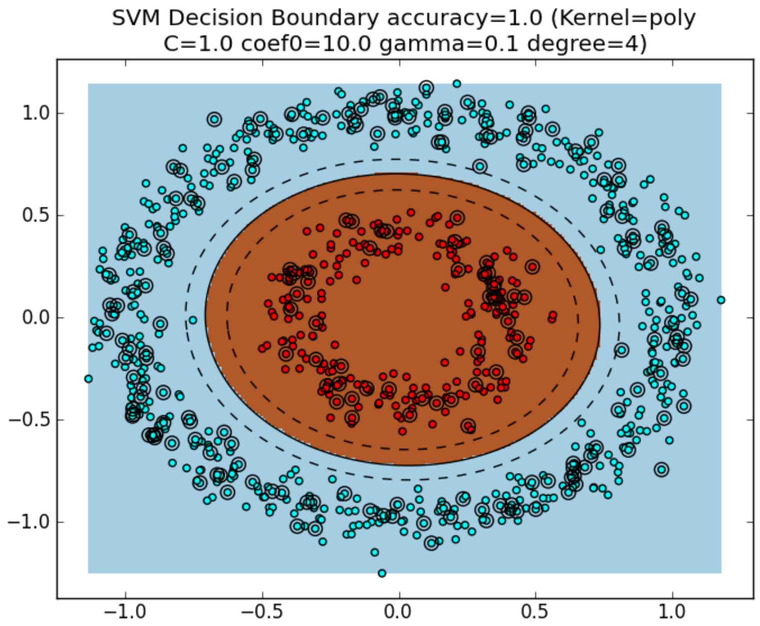
Kernel, also can be written as $K(\boldsymbol{x_i}, \boldsymbol{x})$

Feature Transformation

- Apply a function, i.e., $\Phi(\boldsymbol{x})$, that transforms the raw feature vectors to a set new feature vectors
- Main goal: to enhance the representation capability
- Widely used in classical machine learning models
- Deep learning has strong power to automatically transform features into very high dimensions

SVM Kernels

- Scikit Learn provides various options for choosing kernels
  - Polynomial kernel, i.e., ‘poly’
  - Gaussian kernel, i.e., ‘rbf’



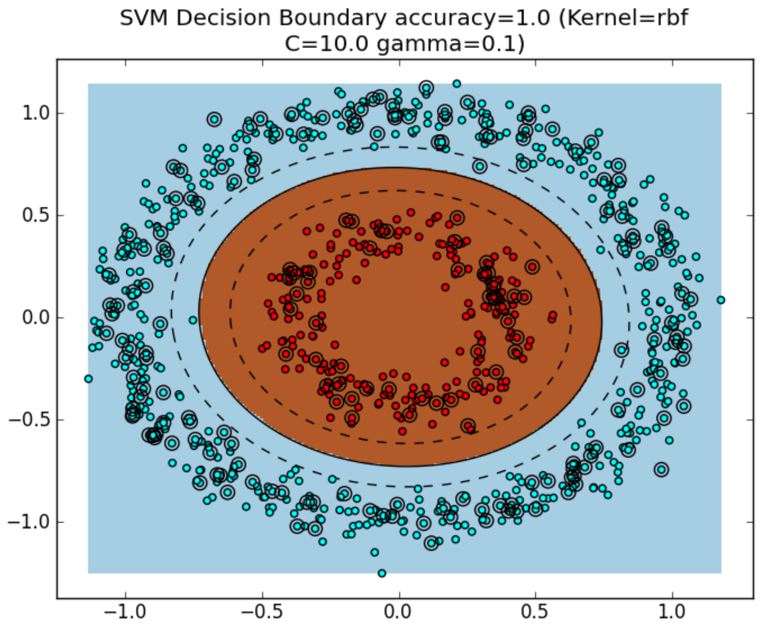


Image source: https://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html

SVM Kernels

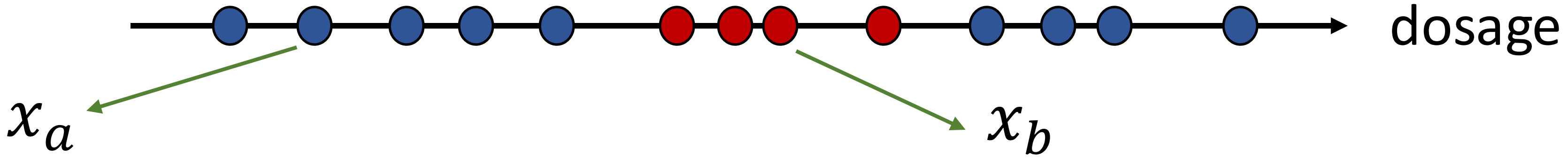
$$f(\Phi(\boldsymbol{x})) = \sum_{i=1}^{n} \lambda_i y_i \Phi(\boldsymbol{x_i})^{\boldsymbol{T}} \Phi(\boldsymbol{x}) + b$$

- We don't conduct feature transformation, i.e., x to $\Phi(\boldsymbol{x})$
- Instead, we apply kernel trick to obtain the dot product of the transformed features in high dimensional space

Polynomial Kernel (Optional)

A polynomial kernel in 1D dimension:

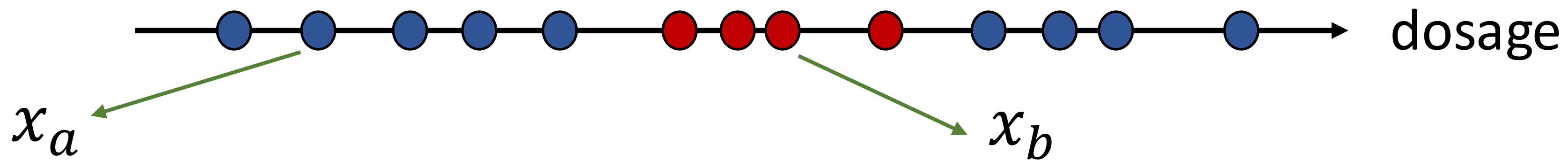
$$K(x_a, x_b) = (x_a x_b + \frac{1}{2})^2$$



Polynomial Kernel (Optional)

$$K(x_a, x_b) = (x_a x_b + \frac{1}{2})^2 = x_a x_b + {x_a}^2 {x_b}^2 + \frac{1}{4}$$

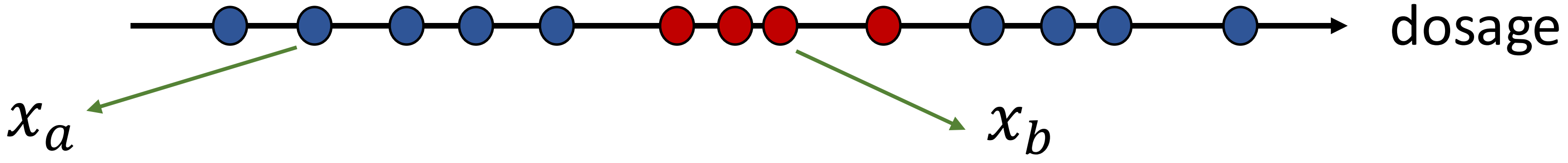
$$= \left\{x_a, {x_a}^2, \frac{1}{2}\right\}^T \left\{x_b, {x_b}^2, \frac{1}{2}\right\}$$



Polynomial Kernel (Optional)

- A general polynomial kernel in abstract high dimension:

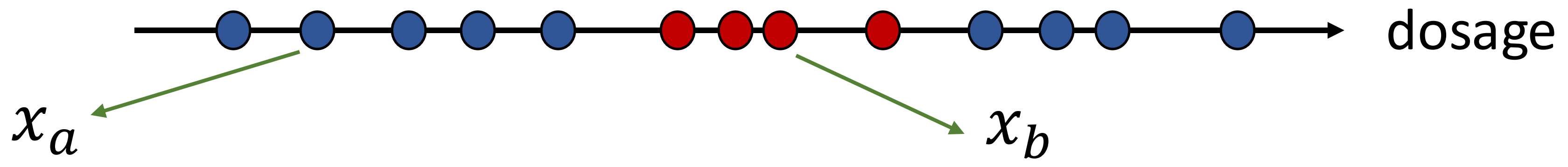
$$K(\boldsymbol{x_a}, \boldsymbol{x_b}) = (\boldsymbol{x_a}^T \boldsymbol{x_b} + r)^d$$



RBF Kernel (Optional)

- An RBF kernel in 1D dimension:

$$K(x_a, x_b) = e^{-\frac{1}{2}(x_a - x_b)^2}$$



RBF Kernel (Optional)

- RBF naturally contains a polynomial kernel in infinite space

$$K(x_a, x_b) = e^{-\frac{1}{2}(x_a - x_b)^2} = e^{-\frac{1}{2}({x_a}^2 + {x_b}^2) + x_a x_b}$$

$$= e^{-\frac{1}{2}({x_a}^2 + {x_b}^2)}\ e^{x_a x_b}$$

RBF Kernel (Optional)

- RBF naturally contains a polynomial kernel in infinite space

$$e^{x} = 1 + x + \frac{x^{2}}{2!} + \frac{x^{3}}{3!} + \cdots + \frac{x^{\infty}}{\infty!}$$

$$e^{x_a x_b} = 1 + x_a x_b + \frac{(x_a x_b)^{2}}{2!} + \frac{(x_a x_b)^{3}}{3!} + \cdots + \frac{(x_a x_b)^{\infty}}{\infty!}$$

$$= \left\{1, x_a, \frac{{x_a}^{2}}{2!}, \frac{{x_a}^{3}}{3!}, \dots, \frac{{x_a}^{\infty}}{\infty!}\right\}^{T} \left\{1, x_b, \frac{{x_b}^{2}}{2!}, \frac{{x_b}^{3}}{3!}, \dots, \frac{{x_b}^{\infty}}{\infty!}\right\}$$

RBF Kernel (Optional)

- RBF naturally contains a polynomial kernel in infinite space

$$K(x_a, x_b) = e^{-\frac{1}{2}({x_a}^2 + {x_b}^2)}\, e^{x_a x_b}$$

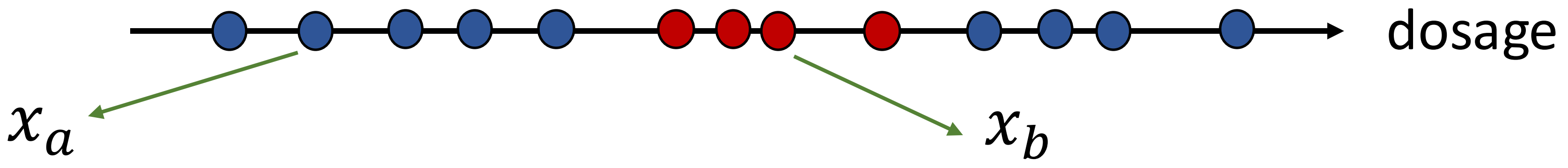
$$= e^{-\frac{1}{2}({x_a}^2 + {x_b}^2)} \left\{1, x_a, \frac{{x_a}^2}{2!}, \frac{{x_a}^3}{3!}, \dots, \frac{{x_a}^\infty}{\infty!}\right\}^T \left\{1, x_b, \frac{{x_b}^2}{2!}, \frac{{x_b}^3}{3!}, \dots, \frac{{x_b}^\infty}{\infty!}\right\}$$

RBF Kernel (Optional)

- A general RBF kernel in multi-feature dimension:

$$K(\boldsymbol{x_a}, \boldsymbol{x_b}) = e^{-\frac{1}{\sigma^2}(\|\boldsymbol{x_a} - \boldsymbol{x_b}\|)^2}$$

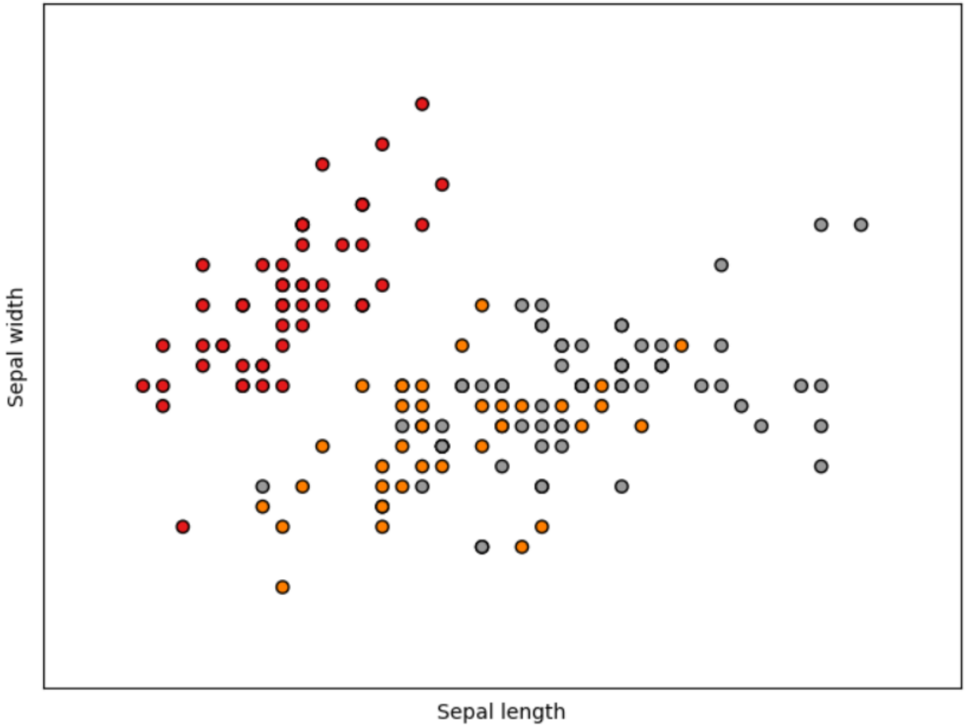
- It measures the influence one sample has over another sample



Using SVC to perform Multi-Class Prediction

SVC for Iris Classification

- 3 types of irises in total: Setosa, Versicolour, Virginica
- 4 features: Sepal Length, Sepal Width, Petal Length and Petal Width



https://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html

SVC for Iris Classification

- Import libraries

```
from sklearn import svm, datasets
import sklearn.model_selection as model_selection
from sklearn.metrics import accuracy_score
```

- Load dataset

```
iris = datasets.load_iris()
X = iris.data[:, :2]
y = iris.target
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y,
    train_size=0.60, test_size=0.40, random_state=101)
```

SVC for Iris Classification

- Construct SVC classifiers on the training set

```
rbf = svm.SVC(kernel='rbf', gamma=0.5, C=0.1).fit(X_train, y_train)
poly = svm.SVC(kernel='poly', degree=3, C=1).fit(X_train, y_train)
```

- Perform predictions on the test set

```
poly_pred = poly.predict(X_test)
rbf_pred = rbf.predict(X_test)
```

- Accuracy evaluation. Many metrics can be used

Questions?