INT354
Machine Learning Foundations

Machine Learning Classifiers using Scikit-learn

Introduction to scikit-learn

- **Machine Learning in Python**
- **Simple and Effective tools for data mining and data Analysis**
- **Built on Numpy, Scipy, Matplotlib**
- **Open source, commercially usable - BSD license**
- **Accessible to everybody, reusable in various contexts**

How to Install?

Python Idle

- **pip install scikit-learn**

Anconda

- **conda install scikit-learn**

Jupyter Note Book

- **!pip install scikit-learn**

No Free Lunch Theorem

- **No Classifier works best across all possible scenarios.**

Choosing a classification Algorithm?

Parameters for choosing a classification algorithm

- **Performance may depends on**
  - **Number of features**
  - **Number of samples**
  - **The amount of noise in dataset**
  - **Classes are linearly separable or NOT.**

Selection of Features

Features to classify dissimilar Object

Tiger Vs Leopard

Easy??

Selection of Features

Features to classify similar Object

Jaguar Vs Leopard

??

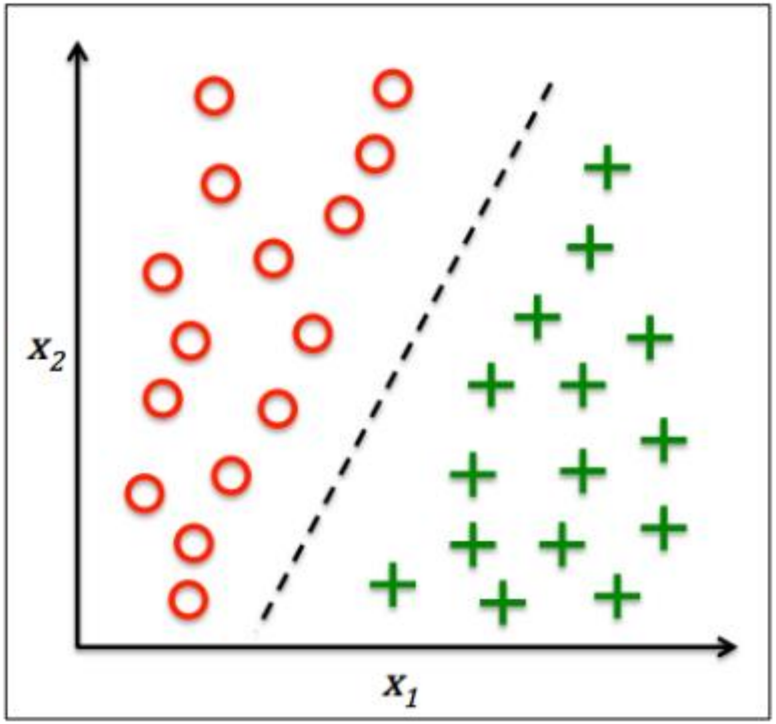
Training a machine learning algorithm

Training a machine learning algorithm can be summarized in 5 main steps.

1. **Selection of features**
2. **Choosing a performance Metrics**
3. **Choosing a classifier and Optimization Algorithm.**
4. **Evaluating the performance of model**
5. **Tuning the algorithm.**

Perceptorn

- **2 class classifier**
- **Linear separable patterns**



Training Perceptron using scikit-learn

- **Major Steps involved**
  1. **Load dataset**
  2. **Divide the data in train and test**
  3. **Feature Scaling**
  4. **Train Perceptron model**
  5. **Check accuracy**
  6. **Plot Decision Boundary**

Available Datasets in scikit

- load_boston()
- load_iris()
- load_diabetes()
- load_digits()
- load_linnerud()
- load_wine()
- load_breast_cancer()

Load Dataset

```
from sklearn import datasets
Iris=datasets.load_iris()
x=iris.data
y=iris.target
```

Split data in train and test

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test =
train_test_split(x,y,test_size=0.2,random_state=0)
print(np.shape(x))
print(np.shape(x_train))
print(np.shape(x_test))
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
sc.fit(x_train)
x_train_std=sc.transform(x_train)
x_test_std=sc.transform(x_test)
```

Note:

- fit() method estimates the sample mean(μ) and standard deviation (σ) for each feature dimension from the training data.
- transform() method standardized the training data and test data using the estimated parameters.

Train Perceptron Model

```
from sklearn.linear_model import Perceptron
ppn=Perceptron(eta0=0.1, random_state=0)
ppn.fit(x_train_std, y_train)
```

Note:

- **If learning rate is too large, then algorithm will overshoot the global cost minimum.**
- **If learning rate is too small, then algorithm requires more epochs until convergence.**

Prediction on new data

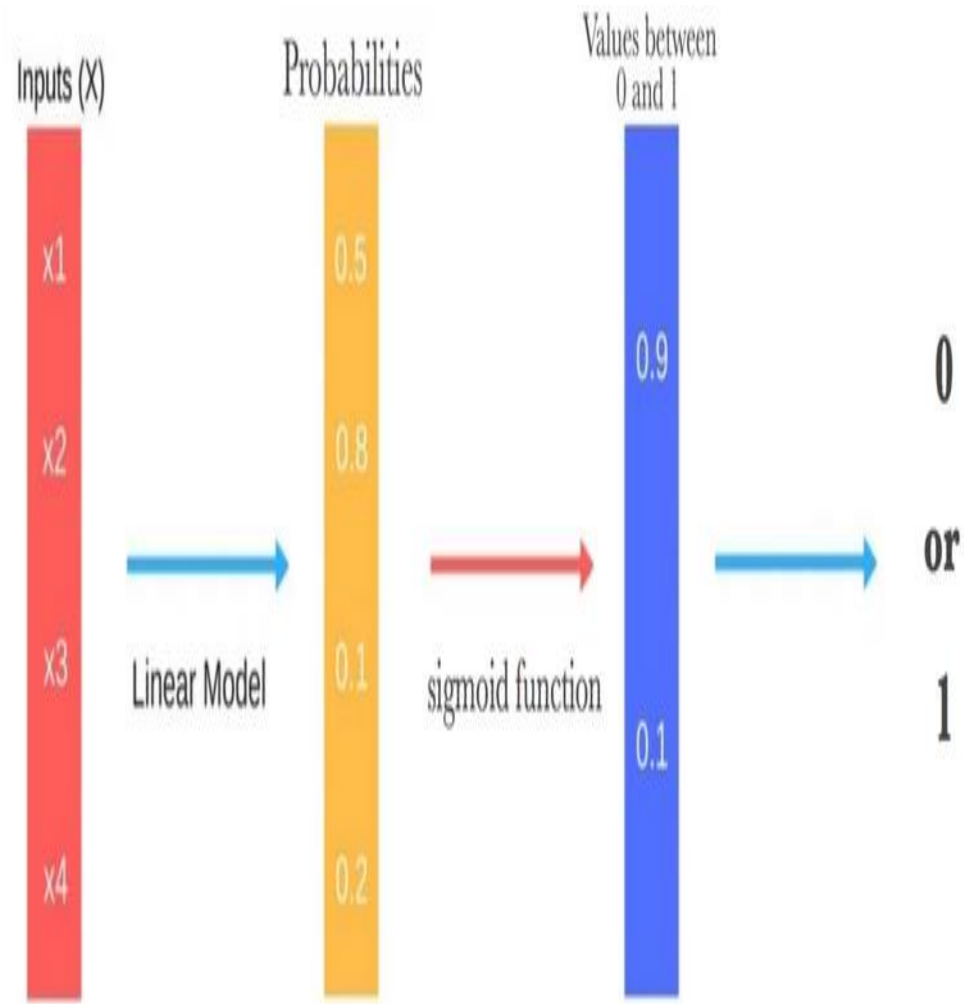
```
y_pred=ppn.predict(x_test_std)
print('Misclassified Samples: %d' %(y_test != y_pred).sum())
```

Compute Accuracy

from sklearn.metrics import accuracy_score
print(‘Accuracy: %f ‘%accuracy_score(y_test,y_pred))

Logistic Regression and Conditional Probabilities

- Measures the relationship between a target and one or more independent variables by plotting the dependent variable's probability score.
- For example:
  - To predict whether an email is spam or not.
  - Whether the tumor is malignant or not.



Odds Ratio

$$\text{Odds\_Ratio}=\frac{P}{1-P}$$

Where P stands for the Probability of positive event.

$$logit(p) = log\frac{p}{(1-p)}$$

$$logit\big(p(y=1|x)\big) = wTx$$

To predict the probability whether a sample belongs to certain class, take the inverse of logit function.

$$\emptyset(z) = \frac{1}{1+e^{-z}}$$

Where z is net input.

$$z = wTx = w_0 + w_1x_1 + \cdots + wmxm$$

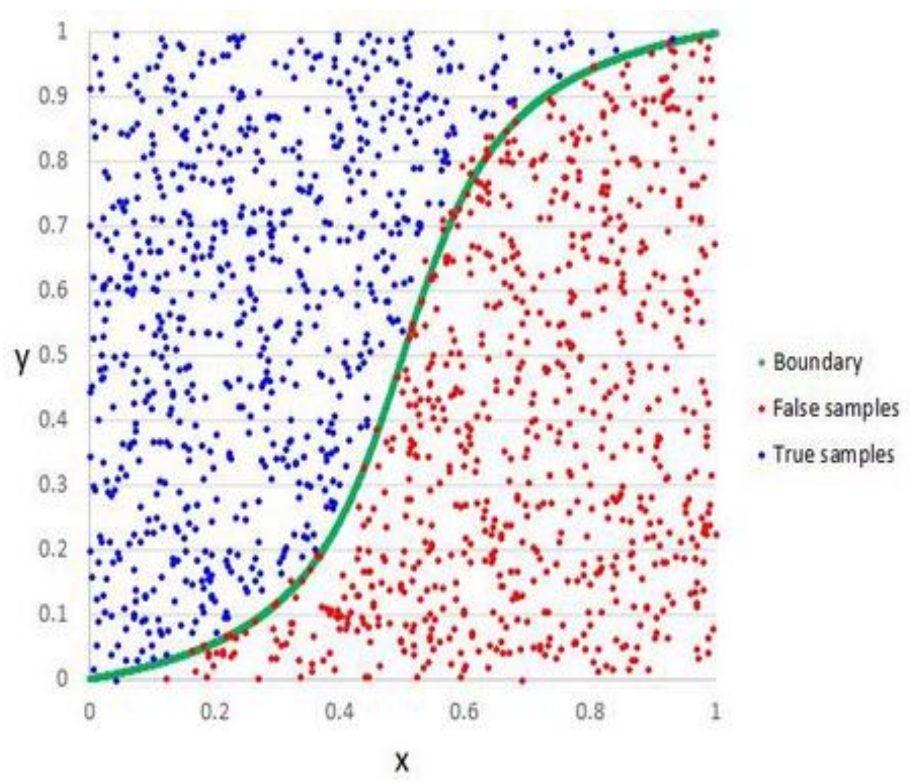
Activation Function in Logistic Regression

Sigmoid Function

$$f(x) = \frac{1}{1+e^{-x}}$$

Decision Boundary

- **Set threshold to predict the class of sample.**
- **Can be linear or non-linear.**
- **Increase polynomial order to get complex decision boundary.**



Cost Function

$$j(w) = \sum_{i=1}^{n} -log\left(\emptyset\left(z^{(i)}\right)\right) - \left(1 - y^{(i)}\right)log\left(i - \emptyset\left(z^{(i)}\right)\right)$$

$$j(\emptyset(z), y; w) = \begin{cases} -log\left(\emptyset(z)\right) & if\ y = 1 \\ -log\left(1 - \emptyset(z)\right) & if\ y = 0 \end{cases}$$

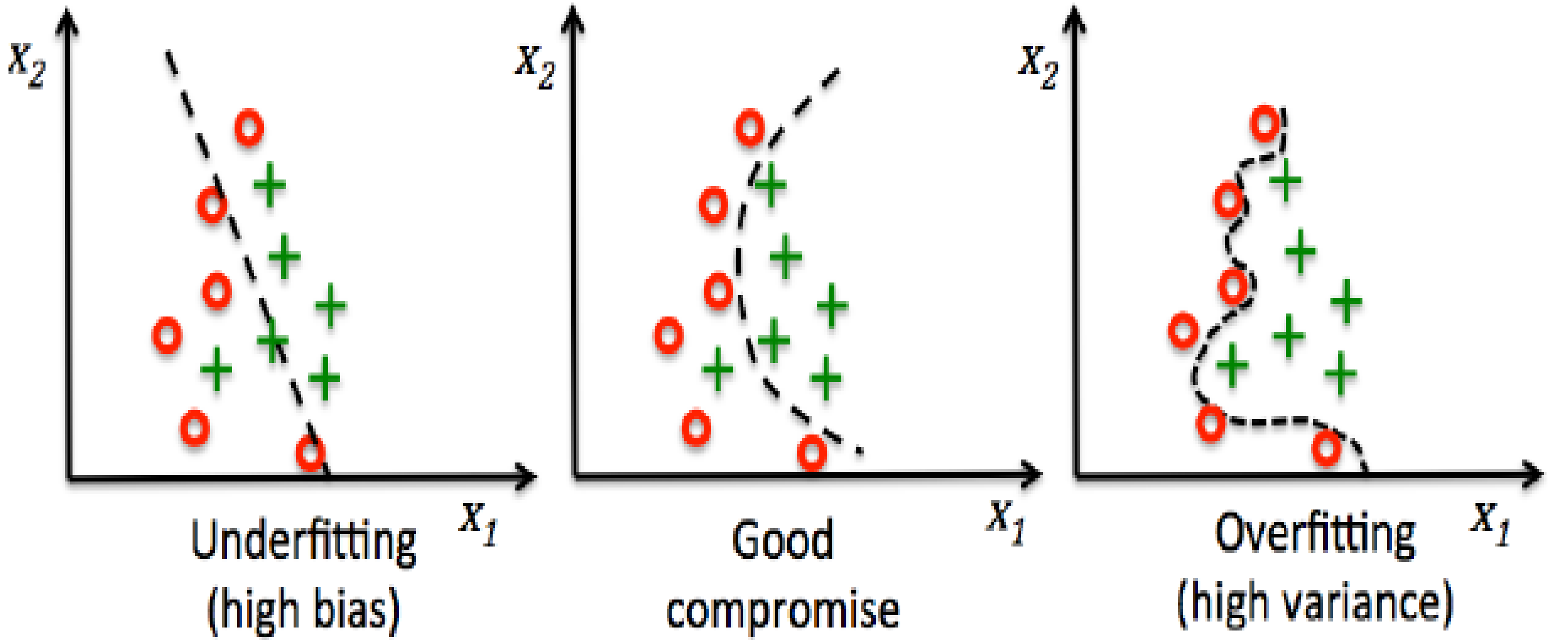
Weight Updating in Logistic Regression

$$wj = wj + \eta \sum_{i=1}^{n} \left(y^{(i)} - \phi\left(z^{(i)}\right)\right) x^{(i)}$$

$$w := w + \Delta w$$

$$\Delta wj = -\eta \frac{\partial J}{\partial wj} = \eta \sum_{i=1}^{n} \left(y^{(i)} - \phi(z^{(i)}) x^{(i)}\right)$$

Over-fitting and Under-fitting



Note:

- **Bias measures how far off the predictions from the correct values.**
- **Variance measures the consistency of the model prediction for a particular sample instance**

Regularization

- Handle collinearity
- Filters out noise from data.
- Prevent over-fitting.
- Introduces additional information (bias) to penalize parameter weights.

L2 Regularization

$$\frac{\lambda}{2}||w||^2 = \frac{\lambda}{2}\sum_{j=1}^{m} w^2{}_j$$

Here, ƛ is regularization parameter.

To apply regularization, add regularization term to the cost function.

$$j(w) = [\sum_{i=1}^{n} -log\left(\phi\left(z^{(i)}\right)\right) - \left(1 - y^{(i)}\right)log(i - \phi\left(z^{(i)}\right))] + \frac{\lambda}{2}||w||^2$$

By increasing the value of ƛ, regularization strength can be increased.

Training logistic regression model with scikit-learn

```
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression(C=1000, random_state=0)
lr.fit(x_train_std, y_train)
```

Advantages of logistic regression

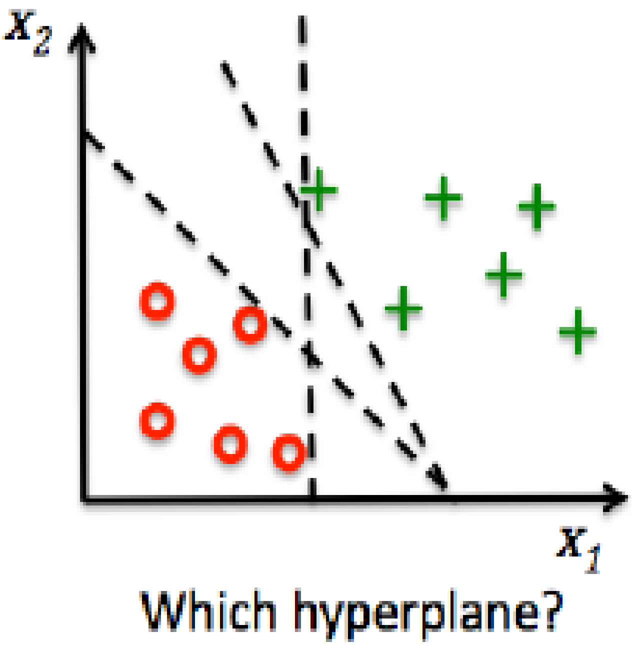
- **Doesn't require input features to be scaled.**
- **Doesn't require any tunning.**
- **Easy to implement.**

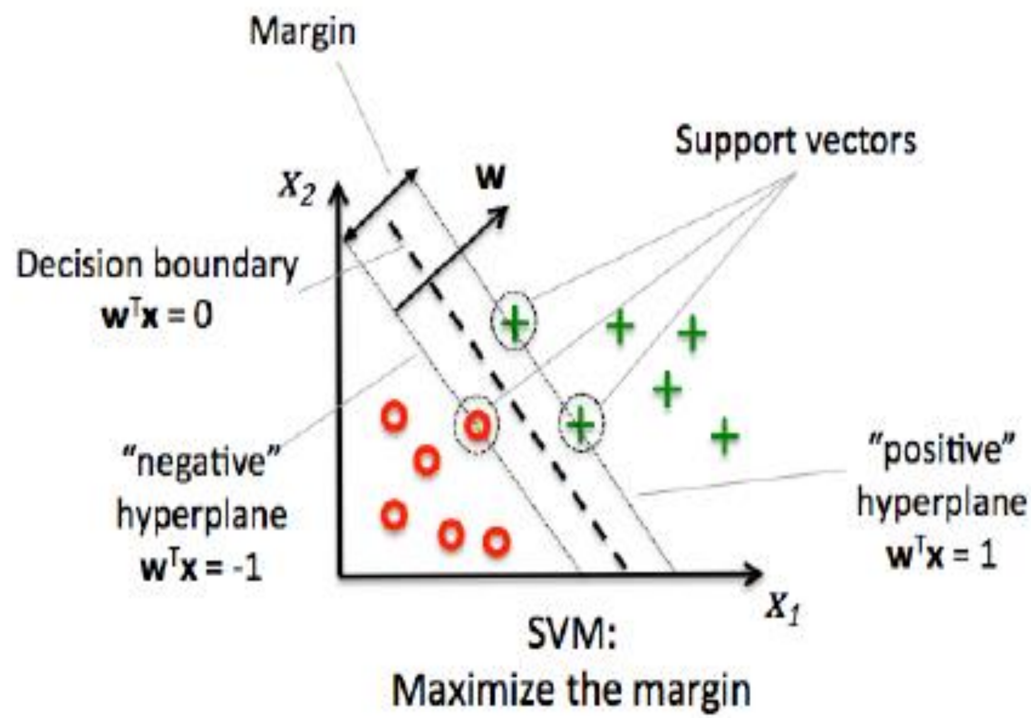
Disadvantages of logistic regression

- **Can't solve non-linear problems.**

Support Vector Machine

- **Objective is to maximize the margin.**





Maximum margin

$$w_0 + wTxp_{os} = 1 \qquad (1)$$
$$w_0 + wTxne_g = -1 \qquad (2)$$

Subtract eq. 1 and 2

$$w^T(x_{pos} - xne_g) = 2$$

Normalize this by the length of vector w,

$$||w|| = \sqrt{\sum_{j=1}^{m} w^2{}_j}$$

So, equation becomes:

$$\frac{w^T(xpo_s - xne_g)}{||w||} = \frac{2}{||w||}$$

Left side of equation is called margin.

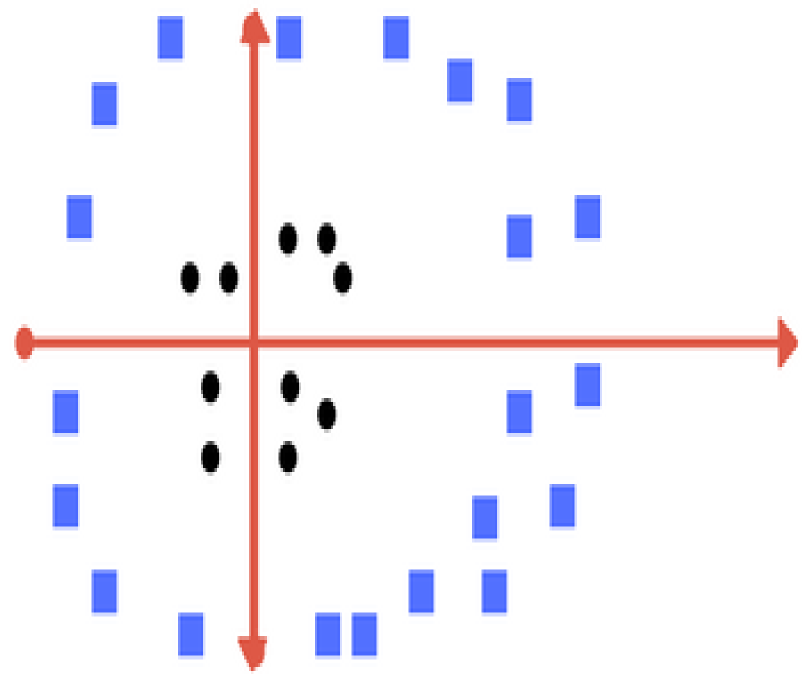
Non-linear Separable Data

Non-linear separable case

- **Slack variables are added for nonlinearly separable data to allow convergence.**
- **Positive values slack variables are added to the linear constraints:**

$$\boldsymbol{w}^T \boldsymbol{x}^{(i)} \geq 1 \; if \;\; y^{(i)} = 1 - \xi^{(i)}$$

$$\boldsymbol{w}^T \boldsymbol{x}^{(i)} < -1 \; if \;\; y^{(i)} = 1 + \xi^{(i)}$$

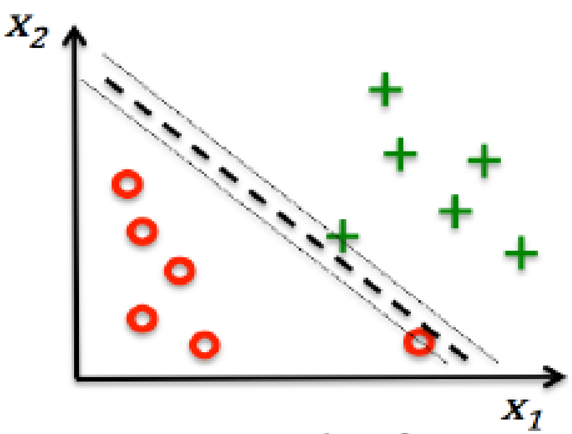


Objective Function

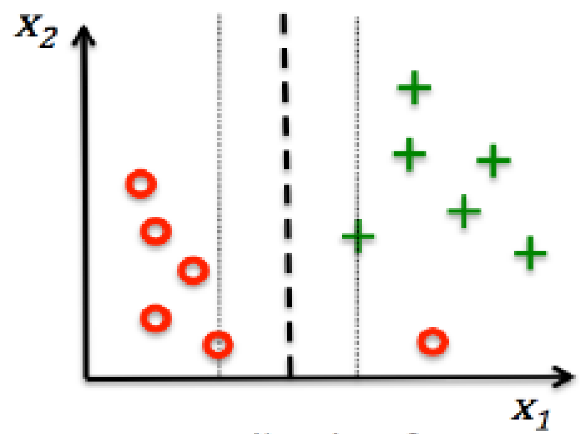
Minimize

$$\frac{1}{2}||w||^2 + C(\sum_i \xi^{(i)})$$

- **C controls the penalty for misclassification.**
- **C is used to control the width of margin.**



Large value for parameter C



Small value for parameter C

Support Vector Machine (Kernels)

- **Linear Kernel SVM**
- **Polynomial Kernel SVM**
- **Radial Kernel SVM**

Linear Kernel SVM

- **Dot-product**

$$K(x, xi) = sum(x * xi)$$

- **Defines similarity or distance measure between new data and the support vectors.**

Polynomial Kernel SVM

$$K(x, xi) = 1 + sum(x * xi)^\wedge d$$

- **d is the degree of polynomial.**

Radial Kernel SVM

$$K(x, xi) = exp(-gamma * sum((x - xi^2))$$

- **Good default value for gamma is 0.1**
- **Gamma lies in range of 0 to 1.**

SVM Advantages

- **Works well with unstructured data like text and images.**
- **The kernel trick is real strength of SVM.**
- **The risk of over-fitting is less in SVM.**
- **It scales relatively well to high dimensional data.**

SVM Disadvantages

- **Choosing a "good" kernel function is not easy.**
- **Long training time for large datasets.**
- **Difficult to understand and interpret the final model.**


COMING UP