Support Vector Machines

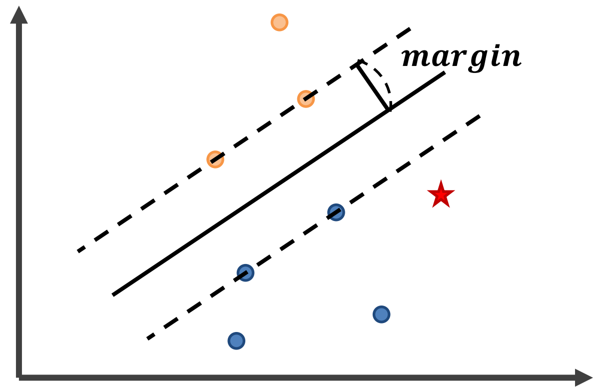
Overview

- What is support vector machines(SVMs)?
- Remind: Hyperplane
- Linear SVMs
- Soft margin SVMs
- Non-linear SVMs

What is support vector machines ?

▪ Support Vector Machines (SVMs)

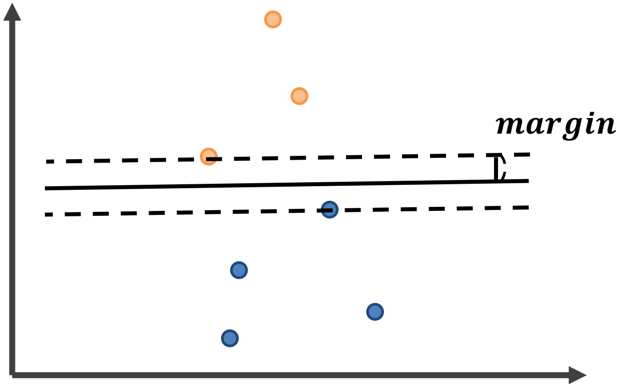
- Vector space classification (using hyperplane)
- Large margin classifier
- Binary classifier (typical)

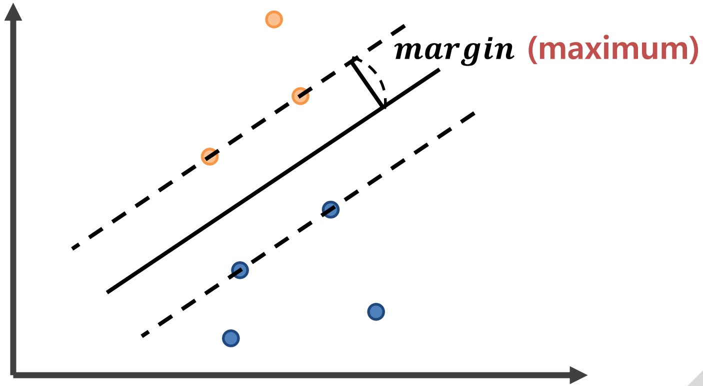


What is support vector machines ?

▪ Strategy of SVMs

1. Calculate hyperplanes that can classify classes
2. Find the hyperplane farthest from any point
3. Classify data based on selected hyperplane

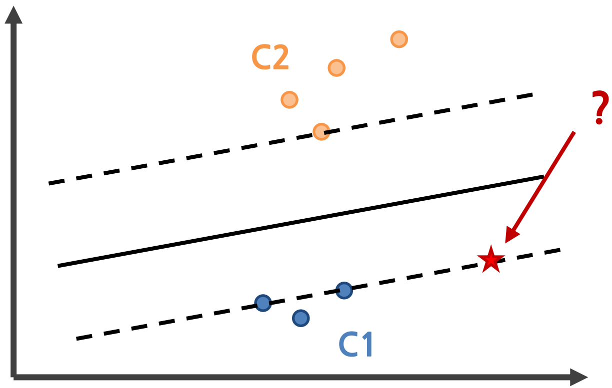




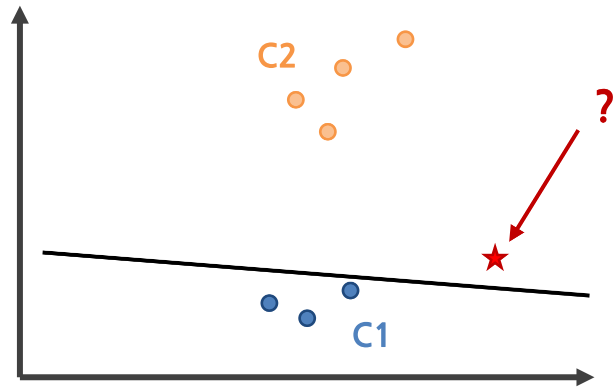
What is support vector machines ?

▪ Why choose a maximum margin

- Enable clear classification
- E.g., maximum margin vs. others



[maximum margin]

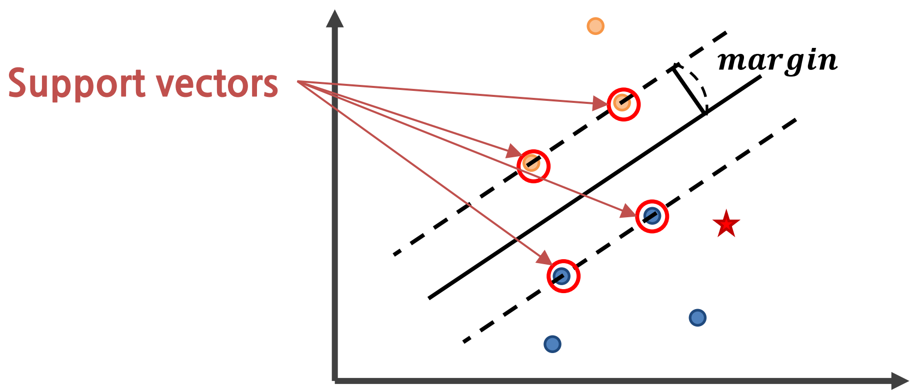


[others]

What is support vector machines ?

▪ Support vectors

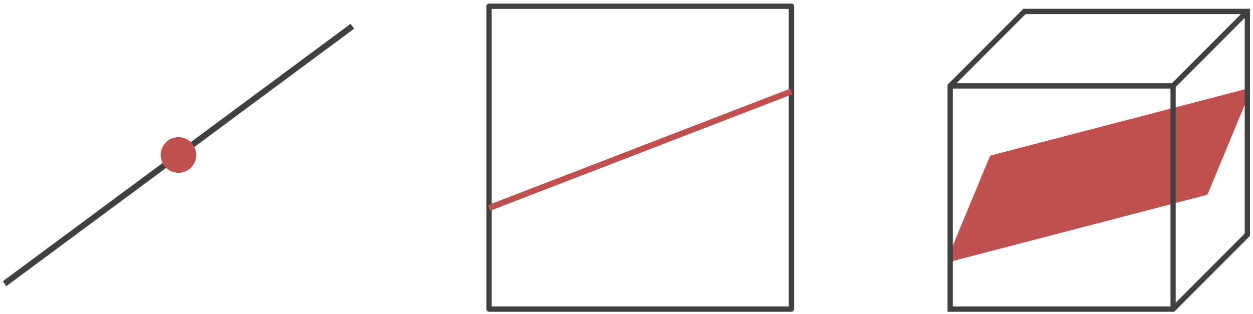
- Vectors that determine the maximum margin
- Vectors on margin lines are called support vectors



Remind: Hyperplane

Hyperplane

- An n-dimensional generalization of a plane
- The hyperplane is an n-dimensional representation of $n-1$ dimensions



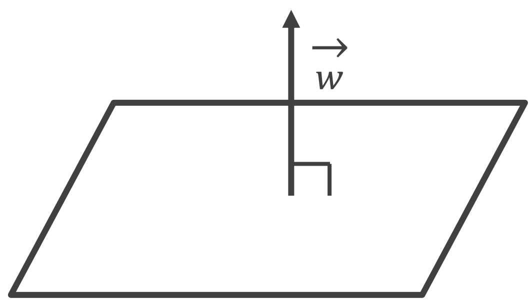
[1 dimension] [2 dimensions] [3 dimensions]

Remind: Hyperplane

How to get

- A hyperplane H in $\mathbb{R}^n$ is the set of points $(x_1, x_2, \dots, x_n)$ that satisfy a linear equation

$$\vec{w}^{\mathrm{T}}\vec{x} + b = 0$$



Remind: Hyperplane

- **What is $\vec{w}^{\mathrm{T}}\vec{x}$?**

- **Linear equation :** $y = ax + b$

$$y - ax - b = 0$$

It's just a different expression !

$$\vec{w}\begin{pmatrix}-b\\-a\\1\end{pmatrix}, \vec{x}\begin{pmatrix}1\\x\\y\end{pmatrix}$$

$$w^{\mathrm{T}} \cdot x = (-b) * 1 + (-a) * x + 1 * y$$

$$= y - ax - b$$

Linear SVMs

▪ How to choose hyperplane

- First, Margin calculation required

1. Functional margin

   - Calculate margin as the result of the hyperplane function

$$y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) = |(\boldsymbol{w}^T\boldsymbol{x}_i + b)|, \boldsymbol{x}_i \in DataSet$$

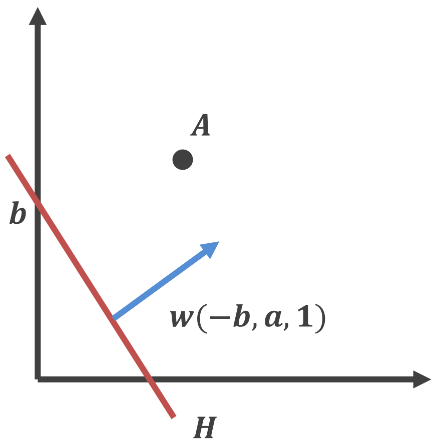
   - There is a problem that the margin can be changed easily

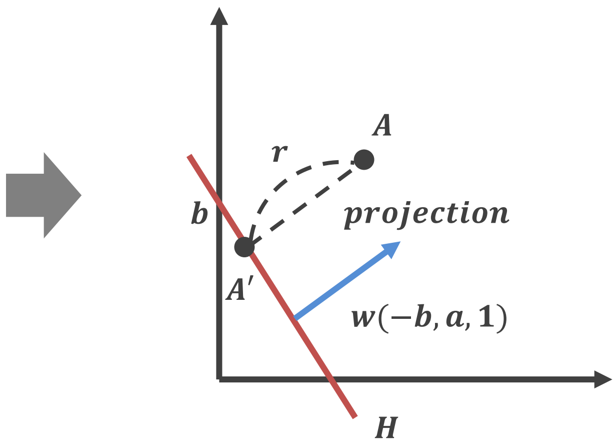
Linear SVMs

▪ How to choose hyperplane

2. Geometric margin

- Euclidean distance between point and hyperplane





Linear SVMs

▪ How to choose hyperplane

- Unit vector : $\boldsymbol{u} = {}^{\boldsymbol{w}}/_{|\boldsymbol{w}|}$
- Orthogonal vector : $r * \boldsymbol{u}$
- Projected vector : $\boldsymbol{x}' = \boldsymbol{x} - yr\,{}^{\boldsymbol{w}}/_{|\boldsymbol{w}|}$
- $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}' + \boldsymbol{b} = \mathbf{0}$

$$\boldsymbol{w}^{\mathrm{T}}\left(\boldsymbol{x} - yr\,{}^{\boldsymbol{w}}/_{|\boldsymbol{w}|}\right) + b = 0$$

$$r = y\,{}^{(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b)}/_{\boldsymbol{w}}$$

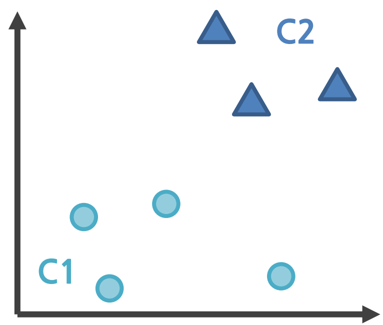
Linear SVMs

▪ How to choose hyperplane

- Find the hyperplane with the maximum margin

1. We have a dataset $\mathcal{D}$ and you want to classify it

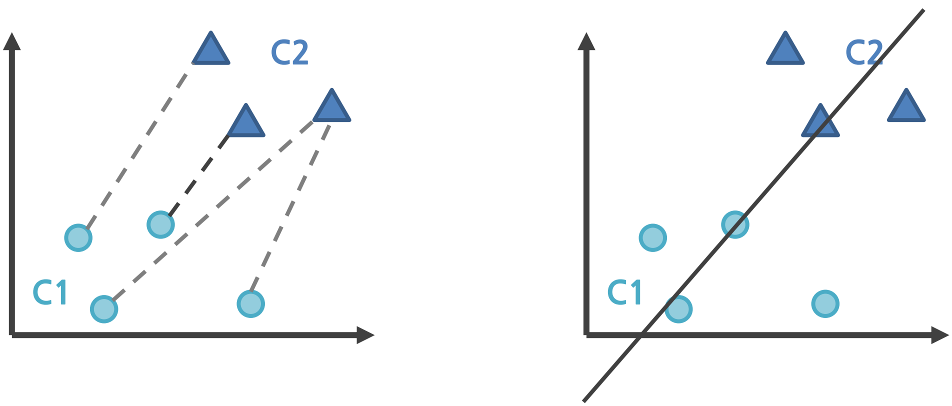
$$\mathcal{D} = \left\{(x_i, y_i) | x_i \in \mathbb{R}^d, y_i \in \{-1,1\}\right\}_{i=1}^{n}$$



Linear SVMs

- ## How to choose hyperplane

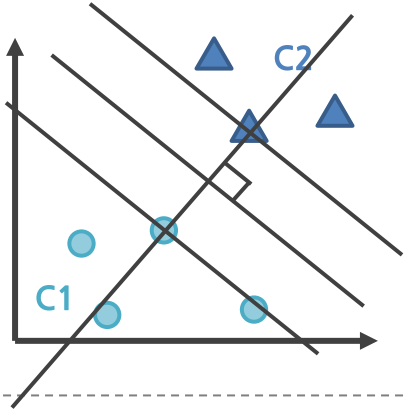
2. **Find the minimum distance between data with different class labels**



Linear SVMs

▪ How to choose hyperplane

3. Find a hyperplane with the maximum margin perpendicular to the hyperplane connecting the two vectors



Linear SVMs

- ## Mathematical summary

1. **We have a dataset $\mathcal{D}$ and you want to classify it**

$$\mathcal{D} = \left\{(x_i, y_i) | x_i \in \mathbb{R}^d, y_i \in \{-1,1\}\right\}_{i=1}^{n}$$

Linear SVMs

▪ Mathematical summary

2. **We need to select two hyperplanes separating the data with no points between them**

$for\ x_i\ having\ the\ class\ -1$

$w \cdot x_i + b \leq -1$

$for\ x_i\ having\ the\ class\ 1$

$w \cdot x_i + b \geq 1$

$And\ multiply\ both\ sides\ by\ y_i, and\ then\ we\ get\ it$

$y_i(w \cdot x_i + b) \geq 1\ \ for\ \forall i\ (1 \leq i \leq n)$

Linear SVMs

▪ Mathematical summary

3. Maximize the distance between the two hyperplanes

$unit\ vector\ \boldsymbol{u} : {}^{\boldsymbol{w}}/_{||\boldsymbol{w}||}$

$vector\ \boldsymbol{k} = m \cdot \boldsymbol{u}$

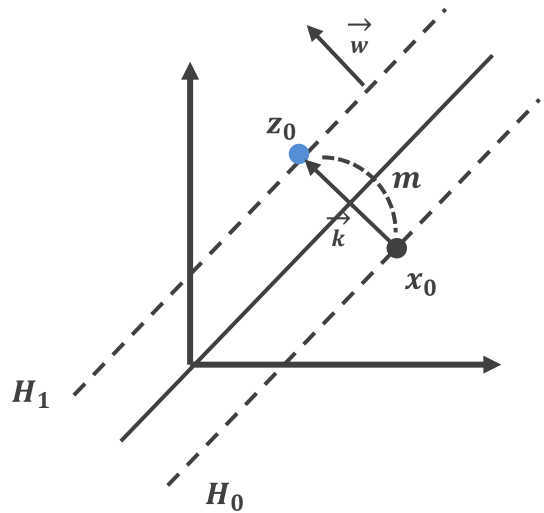
$vector\ \boldsymbol{z_0} = \boldsymbol{k} + \boldsymbol{x_0}$

$in\ H_1, w \cdot z_0 = -b + \delta$

$w \cdot (x_0 + k) = -b + \delta$

$w \cdot x_0 + m||w|| = -b + \delta$

$m = {}^{2\delta}/_{||w||}$



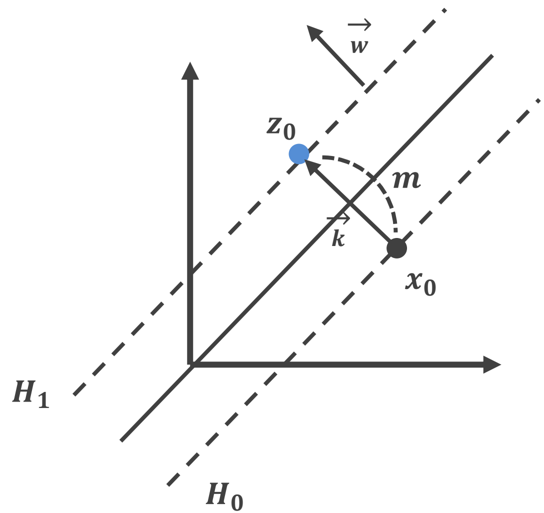
Linear SVMs

▪ Mathematical summary

3. Maximize the distance between the two hyperplanes

$$m = {}^{2\delta}/_{||w||} \ is\ maximized,$$

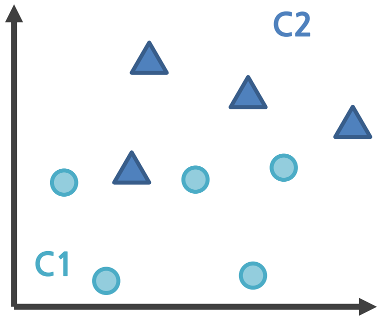
$$oppositely, \frac{1}{2} w^{\mathrm{T}} \cdot w \ is\ minimized$$



Linear SVMs issue

▪ Weakness of linear SVMs

- When data can’t be classified linearly,

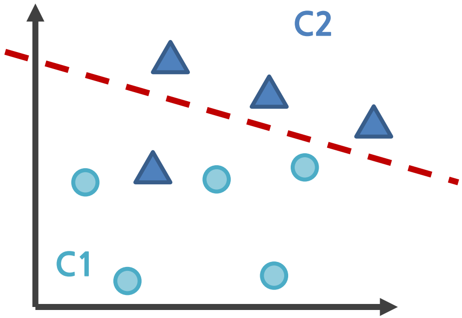


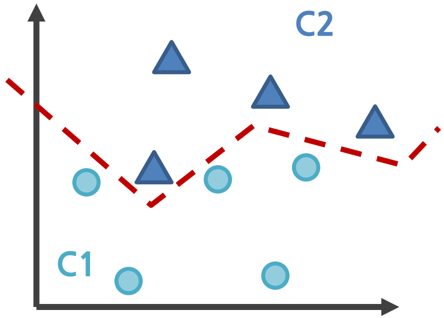
The problem is that this is a common case !

Linear SVMs issue

- **How to solve it ?**

1. Allow some errors
2. Using non-linear hyperplane (Decision boundary)





Soft Margin SVMs

▪ Strategies

- **Allow some errors**
- **A penalty is given for errors: slack variables ξ_i**

$$ {}^{1}\!/_{2}\, w^{\mathrm{T}} \cdot w + C \sum_{i} \xi_i \;\; is\ minimized $$

$$ and\ for\ all\ \{(\boldsymbol{x_i}, y_i)\}, y_i\left(\boldsymbol{w^T} \cdot \boldsymbol{x_i} + b\right) \geq 1 - \xi_i $$

$$ \xi_i \geq 1 - y_i\left(\boldsymbol{w^T} \cdot \boldsymbol{x_i} + b\right) $$

$$ if\ \xi_i = 0, correct\ classification $$

$$ else\ if\ 0 < \xi_i < 1, correct, but\ exceeded $$

$$ else\ \xi_i \geq 1, misclassified $$

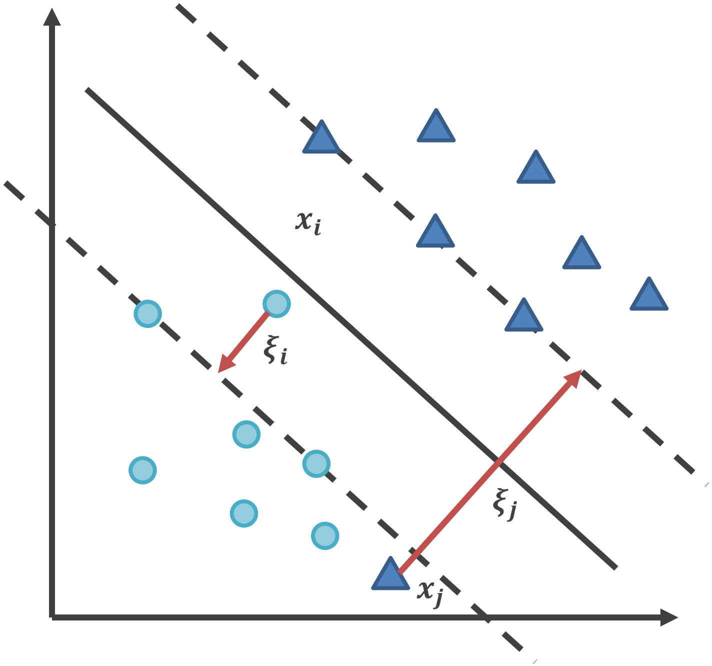
Soft Margin SVMs

How much error does it allow ?

- Tuning parameter: C $(regularization\ term)$
- The threshold for the errors
- Typically, C is a user input parameter
- If C is too large, underfitting occurs
- If C is too small, overfitting occurs

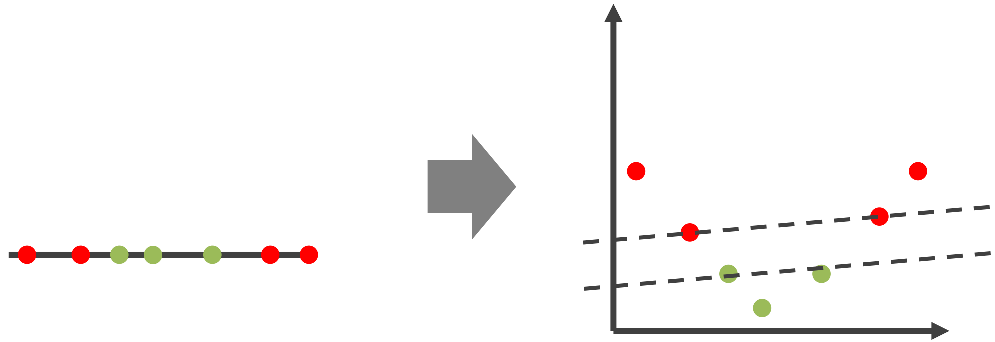
Soft Margin SVMs

- ## Example figure



Non-linear SVMs

- **It does not allow errors**
  - Use a strong margin as is
  - How ? $Kernel\ trick: K(x, y)$
  - Map a dataset to a higher dimensional space



Non-linear SVMs

- How to apply ***kernel trick*** ?

- Must be converted to applicable form first
- Lagrange dual problem
  - ✓ Converting a minimization problem to a maximization problem
  - ✓ Satisfy *KKT condition* to reduce duality gap

    (For more information, search Karush-Kuhn-Tucker conditions)

Non-linear SVMs

- **How to apply *kernel trick* ?**

  - **Lagrange dual problem**

$$\min_{w,b} ||\boldsymbol{w}||$$
$$s.t. (wx_j + b)y_j \geq 1, \forall j$$

Transformation

$$L(w,b,\alpha) = \frac{1}{2} w \cdot w - \sum_j \alpha_j [(wx_j + b)y_j - 1]$$

$$= \sum_j \alpha_j - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j x_i x_j y_i y_j$$

$$\therefore \max_{\boldsymbol{\alpha} \geq 0} \boldsymbol{L(x, \alpha)}$$

Non-linear SVMs

- ## How to apply *kernel trick* ?

$$\varphi(x_i)\varphi(x_j) = K(x_i, x_j)$$

$$L(w, b, \alpha) = \sum_j \alpha_j - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

$$w = \sum_j \alpha_j \varphi(x_j) y_j \,, b = y_j - w x_j \Rightarrow b = y_j - \sum_i \alpha_i \varphi(x_i) y_i \varphi(x_j)$$

$$f(\varphi(x)) = sign\left(\sum_i \alpha_i y_i K(x_i, x) + y_j - \sum_i \alpha_i y_i K(x_i, x_j)\right)$$

Non-linear SVMs

▪ How to apply *kernel trick* ?

- Typically, choose from three main kernels

1. Quadratic kernel

$$K(x, y) = (xy + 1)^p$$

2. Radial basis function (rdf)

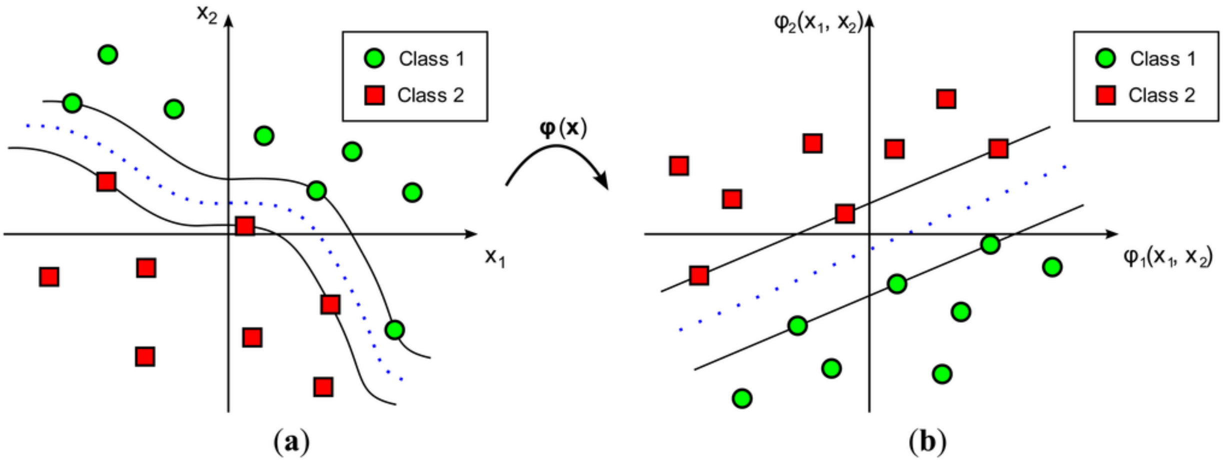
$$K(x, y) = e^{-\frac{|x-y|^2}{2\sigma^2}}$$

3. Hyperbolic tangent

$$K(x, y) = \tanh(axy + \beta)\,, commonly\ \alpha = 2, \beta = 1$$

Non-linear SVMs

- The result of the *kernel trick*



SVM 분류 모델

- 입력 데이터에서 단순한 초평면으로 정의되지 않는 복잡한 모델을 만들 수 있도록
  Support Vector Machine의 확장한 모델

- 직선과 초평면은 유연하지 못하여 저차원 데이터셋에서 선형 분류 모델이 매우 제한적이다.

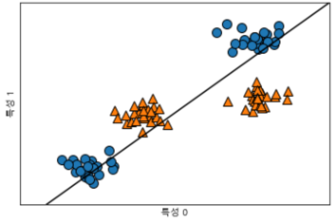
- 선형 모델을 유연하게 만드는 방법
  1. 비선형 특성 추가
  2. 커널 기법

SVM 분류 모델

비선형 특성 추가

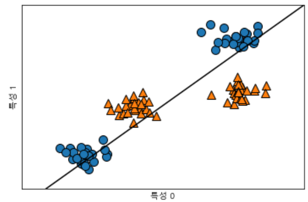
- 특성끼리 곱하거나 특성을 거듭제곱하여 직선과 초평면이 유연해지도록 새로운 특성 추가

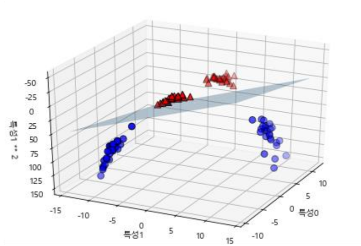
```
>>>from sklearn.datasets import make_blobs
>>> X, y = make_blobs(centers=4, random_state=8)
    y = y % 2
>>>from sklearn.svm import LinearSVC
>>>linear_svm = LinearSVC().fit(X, y)
>>>mglearn.plots.plot_2d_separator(linear_svm, X)
   mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
   plt.xlabel("특성 0")

     plt.ylabel("특성 1")
```



SVM 분류 모델

```
# 두 번째 특성을 제곱하여 추가합니다.
>>> X_new = np.hstack([X, X[:, 1:] ** 2])
>>> from mpl_toolkits.mplot3d import Axes3D, axes3d
# 3차원 그래프
>>> linear_svm_3d = LinearSVC().fit(X_new, y)
>>> coef, intercept = linear_svm_3d.coef_.ravel(), linear_svm_3d.intercept_
# y == 0인 포인트를 먼저 그리고 그다음 y == 1인 포인트를 그립니다.
>>> mask = y == 0
# 선형 결정 경계 그리기
>>> figure = plt.figure()
    ax = Axes3D(figure, elev=-152, azim=-26)
    xx = np.linspace(X_new[:, 0].min() - 2, X_new[:, 0].max() + 2, 50)
    yy = np.linspace(X_new[:, 1].min() - 2, X_new[:, 1].max() + 2, 50)

>>> XX, YY = np.meshgrid(xx, yy)
    ZZ = (coef[0] * XX + coef[1] * YY + intercept) / -coef[2]
    ax.plot_surface(XX, YY, ZZ, rstride=8, cstride=8, alpha=0.3)
    ax.scatter(X_new[mask, 0], X_new[mask, 1], X_new[mask, 2], c='b',
       cmap=mglearn.cm2, s=60, edgecolor='k')
    ax.scatter(X_new[~mask, 0], X_new[~mask, 1], X_new[~mask, 2], c='r', marker='^',
       cmap=mglearn.cm2, s=60, edgecolor='k')
    ax.set_xlabel("특성0")
    ax.set_ylabel("특성1")
    ax.set_zlabel("특성1 ** 2") X_new = np.hstack([X, X[:, 1:] ** 2])
```

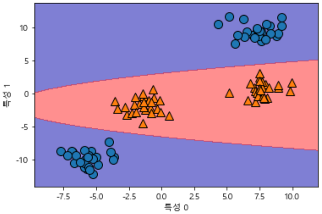




SVM 분류 모델

원래의 특성에 투영

```
>>> ZZ = YY ** 2
    dec = linear_svm_3d.decision_function(np.c_[XX.ravel(), YY.ravel(), ZZ.ravel()])

    plt.contourf(XX, YY, dec.reshape(XX.shape), levels=[dec.min(), 0, dec.max()],

        cmap=mglearn.cm2, alpha=0.5)

    mglearn.discrete_scatter(X[:, 0], X[:, 1], y)

    plt.xlabel("특성 0")

    plt.ylabel("특성 1")
```



SVM 분류 모델

충북대학교 SW중심대학사업단

- 커널 기법

  - 수학적 기교로 데이터를 확장하지 않고 고차원에서 분류기 학습

  - 다항식 커널 : 원래 특성의 가능한 조합을 지정된 차수까지 모두 계산

  (ex) 특성1 ** 2 × 특성2 ** 5)

  - 가우시안 커널(or RBF) : 모든 차수의 모든 다항식 고려하지만 특성의 중요도는 고차항이 될수록 작아짐

  - 서포트 벡터 : 두 클래스 사이의 경계에 위치하여 경계를 구분하는데 영향을 주는 훈련 데이터

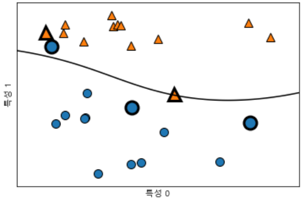
  - 매개변수

  1. gamma : 하나의 훈련 샘플에 미치는 영향의 범위(값이 작을수록 넓은 영역)

  2. C : 각 포인트의 중요도를 제한

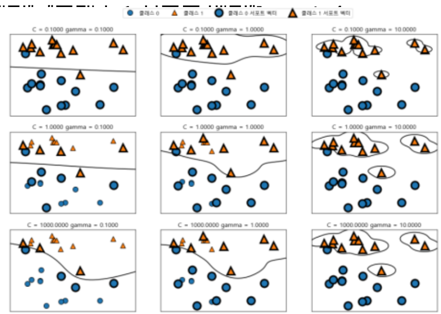
SVM 분류 모델

```
>>> from sklearn.svm import SVC
>>> X, y = mglearn.tools.make_handcrafted_dataset()
>>> svm = SVC(kernel='rbf', C=10, gamma=0.1).fit(X, y)
    mglearn.plots.plot_2d_separator(svm, X, eps=.5)
    mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
# 서포트 벡터를 굵은 테두리로 표시
>>> sv = svm.support_vectors_
# dual_coef_의 부호에 의해 서포트 벡터의 클래스 레이블이 결정됩니다.
>>> sv_labels = svm.dual_coef_.ravel() > 0
>>> mglearn.discrete_scatter(sv[:, 0], sv[:, 1], sv_labels, s=15,
markeredgewidth=3)
    plt.xlabel("특성 0")
    plt.ylabel("특성 1")
```



SVM 분류 모델

```
>>> fig, axes = plt.subplots(3, 3, figsize=(15, 10))
>>> for ax, C in zip(axes, [-1, 0, 3]):
        for a, gamma in zip(ax, range(-1, 2)):
            mglearn.plots.plot_svm(log_C=C,
            log_gamma=gamma, ax=a)
axes[0, 0].legend(["클래스 0", "클래스 1",
    "클래스 0 서포트 벡
    loc=(.9, 1.2))
```



Other issue

▪ Multiclass SVMs

- It is generally a binary classifier
- It can classify multiclass in a naïve approach
- Recursively classify 1:N
- Data point that is not classified or has multi-class labels may exist

Example – performing OCR with SVMs

- Perform Optical Character Recognition (OCR)

- Purpose
  - Process paper-based documents by converting printed or handwritten text into an electronic form

- This is a difficult problem due to the many variants in handwritten style and printed fonts

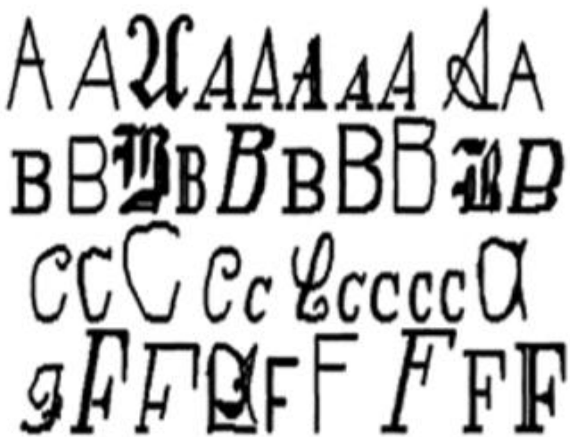
- Errors or typos can result in embarrassing or costly mi stakes in a business environmen

Example – performing OCR with SVMs

- Step 1 – collecting data
  - Dataset
    - Letter dataset
    - Can be downloaded from UCI Machine Learning Data Repository
      - http://archive.ics.uci.edu/ml
  - Characteristics of the dataset
    - The dataset contains 20,000 examples of 26 English al phabet capital letters as printed using 20 different ran domly reshaped and distorted black and white fonts

Example – performing OCR with SVMs

- Step 1 – collecting data
  - The following figure provides an example of some of the printed glyphs



Example – performing OCR with SVMs

- Step 2 – exploring and preparing the data
  - Import the CSV data file

    > letters <- read.csv("letterdata.csv")

  - Confirm that we have received the data with the 16 feat ures that define each example of the letter class

    > str(letters)

```
> letters <- read.csv("letterdata.csv")
> str(letters)
'data.frame':  20000 obs. of 17 variables:
 $ letter: Factor w/ 26 levels "A","B","C","D",..
 $ xbox  : int  2 5 4 7 2 4 4 1 2 11 ...
 $ ybox  : int  8 12 11 11 1 11 2 1 2 15 ...
 $ width : int  3 3 6 6 3 5 5 3 4 13 ...
 $ height: int  5 7 8 6 1 8 4 2 4 9 ...
```

Example – performing OCR with SVMs

- Step 2 – exploring and preparing the data
  - The first 16,000 records (80 percent) to build the mod el

    ```
    > letters_train <- letters[1:16000, ]
    ```

  - The next 4,000 records (20 percent) to test

    ```
    > letters_test  <- letters[16001:20000, ]
    ```

  - The data have already randomized, so no need to pe rform random function

Example – performing OCR with SVMs

- Step 3 – training a model on the data
  - When it comes to fitting an SVM model in R, there are se veral outstanding packages to choose from
  - The e1071 package from the Department of Statistics at the Vienna University of Technology
    - Provides an R interface to the award winning LIBSVM library, a wid ely used open source SVM program written in C++
  - The klaR package from the Department of Statistics at the Dortmund University of Technology
    - Provides functions to work with this SVM implementation directly within R
  - kernlab package

Example – performing OCR with SVMs

- Step 3 – training a model on the data

Support vector machine syntax

using the `ksvm()` function in the `kernlab` package

Building the model:

```
m <- ksvm(target ~ predictors, data = mydata,
          kernel = "rbfdot", C = 1)
```

- `target` is the outcome in the `mydata` data frame to be modeled
- `predictors` is an R formula specifying the features in the `mydata` data frame to use for prediction
- `data` specifies the data frame in which the `target` and `predictors` variables can be found
- `kernel` specifies a nonlinear mapping such as `"rbfdot"` (radial basis), `"polydot"` (polynomial), `"tanhdot"` (hyperbolic tangent sigmoid), or `"vanilladot"` (linear)
- C is a number that specifies the cost of violating the constraints, i.e., how big of a penalty there is for the "soft margin." Larger values will result in narrower margins

The function will return a SVM object that can be used to make predictions.

Making predictions:

```
p <- predict(m, test, type = "response")
```

- `m` is a model trained by the `ksvm()` function
- `test` is a data frame containing test data with the same features as the training data used to build the classifier
- `type` specifies whether the predictions should be `"response"` (the predicted class) or `"probabilities"` (the predicted probability, one column per class level).

The function will return a vector (or matrix) of predicted classes (or probabilities) depending on the value of the type parameter.

Example:

```
letter_classifier <- ksvm(letter ~ ., data =
  letters_train, kernel = "vanilladot")
letter_prediction <- predict(letter_classifier,
  letters_test)
```

Example – performing OCR with SVMs

- Step 3 – training a model on the data
  - Call the ksvm() function on the training data and specify the linear (that is, vanilla) kernel using the vanilladot option

```
> install.packages("kernlab")
> library(kernlab)
> letter_classifier <- ksvm(letter ~ ., data = letters_train,
                            kernel = "vanilladot")
```

Example – performing OCR with SVMs

- Step 3 – training a model on the data
  - Result of ksvm() function

```
> letter_classifier
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 1

Linear (vanilla) kernel function.

Number of Support Vectors : 7037

Objective Function Value : -14.1746 -20.0072 -23.5628 -6.2009 -7.5524
-32.7694 -49.9786 -18.1824 -62.1111 -32.7284 -16.2209...

Training error : 0.130062
```

Example – performing OCR with SVMs

- Step 4 – evaluating model performance
  - The predict() function allows us to use the letter classification mod el to make predictions on the testing dataset

```
> letter_predictions <- predict(letter_classifier, letters_test)
```

  - Because we didn't specify the type parameter, the type = "respon se" default was used
  - This returns a vector containing a predicted letter for each row o f values in the test data
  - Using the head() function, we can see the following result

```
> head(letter_predictions)
[1] U N V X N H
Levels: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

Example – performing OCR with SVMs

- Step 4 – evaluating model performance
  - To examine how well our classifier performed, we need to compare the predicted letter to the true letter in the testing dataset
    - table() function

```
> table(letter_predictions, letters_test$letter)
letter_predictions   A   B   C   D   E
                 A 144   0   0   0   0
                 B   0 121   0   5   2
                 C   0   0 120   0   4
                 D   2   2   0 156   0
                 E   0   0   5   0 127
```

Example – performing OCR with SVMs

- Step 4 – evaluating model performance
  - The following command returns a vector of TRUE or FALSE values, indicating whether the model's predicted letter agrees with the actual letter in the test dataset
    - > agreement <- letter_predictions == letters_test$letter
  - From result, we see that the classifier correctly identified the letter in 3,357 out of the 4,000 test records

```
> table(agreement)
agreement
FALSE  TRUE
643 3357
```

Example – performing OCR with SVMs

- Step 5 – improving model performance
  - By using a more complex kernel function, we can map the dat a into a higher dimensional space, and potentially obtain a be tter model fit
    - Gaussian RBF kernel

```
> letter_classifier_rbf <- ksvm(letter ~ ., data = letters_train, kernel = "rbfdot")
```

    - Next, we make predictions as done earlier

```
> letter_predictions_rbf <- predict(letter_classifier_rbf, letters_test)
```

    - Finally, we'll compare the accuracy to our linear SVM

```
> agreement_rbf <- letter_predictions_rbf == letters_test$letter
> table(agreement_rbf)

agreement_rbf
FALSE  TRUE
  275  3725
```

Referencec

- [1] An Introduction to Information Retrieval, Stanford press
- [2] An SVM-Based Classifier for Estimating the State of Various Rotating Components in Agro-Industrial Machinery with a Vibration Signal Acquired from a Single Point on the Machine Chassis, Sensors, MDPI
- [3] SVM – Understanding the math, www.svm-tutorial.com
- [4] Linear Algebra, LadislauFernandes, Youtube
- [5] Learning: Support Vector Machine, MIT OpenCourseWare, Youtube

Q n A