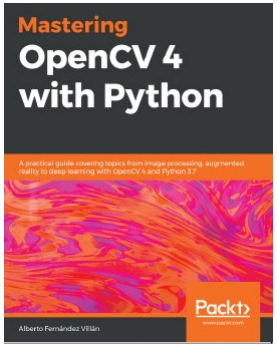
OpenCV Python

디지털영상처리/컴퓨터비전

Introduction to Support Vector Machine



Mastering OpenCV 4 with Python, Alberto, Packt, Pub. 2019

교재 10장의 일부..

2024년 1학기

서경대학교 김진헌

차례

1

- 1. Quick review of SVM
  - 1.1 어떻게 풀 것인가?
  - 1.2 RBF(Radial Basic Function) Kernel
  - 1.3 Misclassification의 문제, C
  - 1.4 SVM의 정리 및 장단점

제일 어려움. 대략 설명!!

- 2. OpenCV SVM 함수
  - 2.1 cv::ml::SVM Class Reference
  - 2.2 get(), set() methods
  - 2.3 create() 함수
  - 2.4 train() 함수
  - 2.5 predict() 함수

중요!!

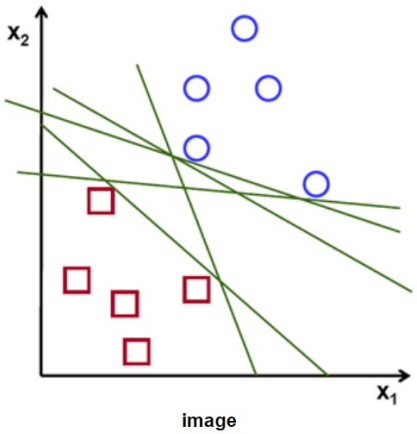
- 3. SVM 단순 실험
- 4. 필기체 문자 분류 실험 knn 과 비교(?)
- 5. C, gamma 및 학습량 비율에 따른 성능 비교 실험 → 필기체 문자 분류 프로그램의 연장
  - SVM_02 실험과의 비교 → 사소한 문제 발생 사실 발견
- 참고 1: scikit의 사례
- 참고 2: A Gentle Introduction To Method Of Lagrange Multipliers
  - An example: 주어진 조건을 만족하며 어떤 함수의 최대/최소값 찾기

자료 보관 차원. 설명 없음

- 검토 → 조별 레포트 출제

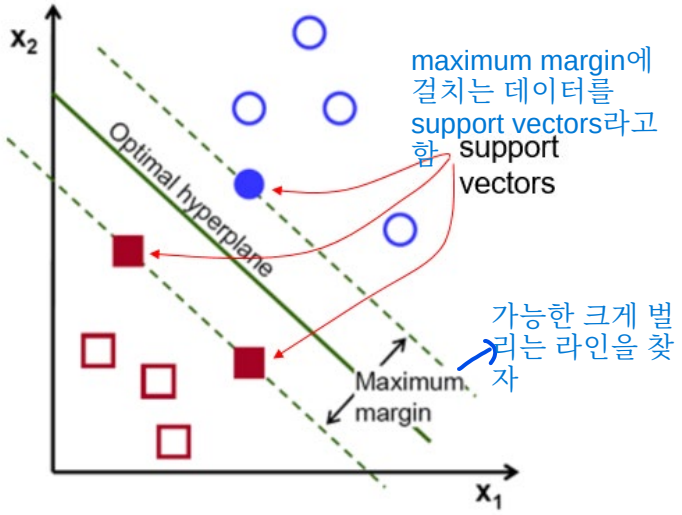
1. Quick review of SVM

OpenCV SVM 튜토리얼 링크



image

- 2종의 데이터를 구분하기 위한 문제를 생각해 보자
  - kNN에서는 test를 위해 모든 데이터와의 거리를 측정해야 했다. 거리 측정에 시간이 많이 소요되고, 모든 학습데이터를 저장하느라 많은 메모리도 필요하다.
- 주어진 사례처럼 선형 분리 가능하다면 그렇게 많은 데이터가 필요할까?



$$f(x) = ax_1 + bx_2 + c$$

- 두 종의 학습 데이터를 2개 영역으로 구분 짓는 선의 방정식 *f(x)*를 찾았다고 생각해 보자.
  - 즉, a, b, c 값을 찾았다는 이야기.
- *if f(X) ⟩ 0*
  - *X*는 파란색 그룹에 속한다. *X*는 (x_1, x_2)좌표 값.
- *else*
  - *X*는 빨간색 그룹에 속한다.

- SVM이 하는 일은 학습 데이터의 간극을 될 수 있으면 멀리 떨어뜨리는 직선을 찾는 일이 될 것이다.
- Support vector는 학습 데이터 중에서 파란색으로 채워진 원, 빨간색으로 채워진 네모의 좌표(특징 값 벡터)를 일컫는다.
- 학습할 때 가까이에 있는 반대의 그룹 데이터만 있으면 된다.
- 서포트 벡터만이 분류에 관여 ➔ 연산량이 적다

1.1 어떻게 풀 것인가?

3 클래스들이 Linearly Separable할 때

OpenCV SVM 튜토리얼 링크

- 이쯤에서 약간 다른 표기법을 사용한 분리 가능한 2차 공간에서의 SVM 문제를 소개한 링크(2. 데이터가 선형으로 분리되는 경우)를 살펴 보기로 하자. 링크에서 몇 가지 용어를 주목해 보자.
- Decision Boundary = hyperplane(line)
- MMH(Maximum Marginal Hyperplane, 최대 마진 초평면) decision boundary에서 떨어진 면적(?)
- support vector들의 수식 상 표현 $\Longrightarrow$ $y_i(w_0+w_1x_1+w_2x_2) = 1, \ \forall i$ (7)
- Soft Margin vs Hard Margin - 실제로는 안 씀
  - 서로 겹쳐도 허용하자(?) - 마진의 허용범위를 널널하게 하자
- Slack variable ξ_i 를 도입한 초평면 표현식 $y_i(w_0+w_1x_1+w_2x_2) \geq 1-\xi_i, \ \xi_i \geq 0 \ \ for \ \forall i$
- Margin을 최대화하기 위해서는 우측 값을 최소화해야 한다.

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum \xi_i$$ Soft margin을 적용한 목적 함수

$$s.t. \ y_i(w_0+w_1x_1+w_2x_2) \geq 1-\xi_i, \ \xi_i \geq 0 \ \ for \ \forall i$$

아래는 OpenCV 자료에서 발췌.

What happened is, first two hyperplanes are found which best represents the data. For eg, blue data is represented by $w^Tx + b_0 > 1$ while red data is represented by $w^Tx + b_0 < -1$ where w is **weight vector** ($w = [w_1, w_2, \ldots, w_n]$) and x is the feature vector ($x = [x_1, x_2, \ldots, x_n]$). b_0 is the **bias**. Weight vector decides the orientation of decision boundary while bias point decides its location. Now decision boundary is defined to be midway between these hyperplanes, so expressed as $w^Tx + b_0 = 0$. The minimum distance from support vector to the decision boundary is given by, $distance_{support\ vectors} = \frac{1}{\|w\|}$. Margin is twice this distance, and we need to maximize this margin. i.e. we need to minimize a new function $L(w, b_0)$ with some constraints which can expressed below:

$$\min_{w,b_0} L(w, b_0) = \frac{1}{2}\|w\|^2 \text{ subject to } t_i(w^Tx + b_0) \geq 1 \ \forall i$$ Hard margin을 적용한 목적 함수

where t_i is the label of each class, $t_i \in [-1, 1]$.

어떻게 해도 분리가 되지 않는다 - > n+1 차원으로 하자!

도움 될만한 SVM 한글 자료 링크

OpenCV SVM 튜토리얼 링크

4 클래스들이 Linearly Non-separable할 때

- 만약 선형 분리가 되지 않는다면 함수에 변형(예를 들어 x^2)을 가하면 분리가 될 수도 있다. 혹은 1차원을 2차원 데이터로 확장(예를 들어 $f(x)=(x, x^2)$)할 수 있다.
  - For example, consider an one-dimensional data where 'X' is at -3 & +3 and 'O' is at -1 & +1. Clearly it is not linearly separable. → $f(x)=(x, x^2)$ 함수를 통해 데이터의 차원을 확장하면,
  - Then 'X' becomes (-3,9) and (3,9) while 'O' becomes (-1,1) and (1,1). This is also linear separable.

아래는 링크(4. 데이터가 선형으로 분리되지 않는 경우) 에서 발췌.

- 한마디로 데이터의 차원을 올리면 곧은 직선 혹은 평평한 면(plane)으로 선형분리되지 않던 데이터가 차원 확장된 초평면(hyper planes)에서 선형 분리 가능하게 될 수 있다는 것이다.
- 비선형 맵핑을 통해 고차원으로 변환하면 분리가능한 초평면을 찾기 쉬워진다. 문제는 MMH를 구하기 위한 계산비용이 많이 소요된다는 것이다.
- 이 때문에 수학적 기교를 사용하는 이른바 Kernel Traick을 사용한다.
- 데이터 투플을 고차원으로 보낸 뒤 벡터의 내적을 계산하는 것과 내적을 한 뒤 고차원으로 보내는 것은 결과적으로 같은 값이기 때문이다. 커널함수(Kernel Function)는 다음의 벡터 내적으로 정의 된다.

$$K(X_i, X_j) = \varphi(X_i)\cdot\varphi(X_j)$$

따라서, 알고리즘에서 $\varphi(X_i)\cdot\varphi(X_j)$ 이 있는 모든 곳에 $K(X_i, X_j)$ 로 대체할 수 있다. 이러한 커널 트릭을 이용하여 모든 계산은 원 데이터의 차원에서 이루어지게 된다. 대표적인 커널함수는 다음과 같다.

$$Polynomial\ kernel\ of\ degree\ h:\ K(X_i, X_j) = (X_i\cdot X_j+1)^h$$

$$Gaussian\ radial\ basis\ function\ kernel:\ K(X_i, X_j) = e^{-\|X_i\cdot X_j\|^2/2\sigma^2}$$

$$Sigmoid\ kenerl:\ K(X_i, X_j) = tanh(\kappa X_i\cdot X_j-\delta)$$

도움 될만한 SVM 한글 자료 링크

5 클래스들이 Linearly Non-separable할 때

OpenCV SVM 튜토리얼 링크

- 아래는 OpenCV자료에서 발췌한 것으로 고차원에서의 벡터 내적이 원 자료에서 내적을 취한 후 고차원으로 보내는 것과 같다는 것을 보여 주는 사례이다.
- 고차원(kernel)의 공간에서의 dot product(내적) 연산을 저차원 입력 특징 공간에서 계산으로 실행하는 방안이 있다.
- 2차원 평면의 2개의 점을 가정하고, $p = (p_1, p_2)$ and $q = (q_1, q_2)$
- 이들을 3차원으로 매핑시키는 다음과 같은 함수를 ϕ라고 가정하자.

$$\phi(p) = (p_1^2, p_2^2, \sqrt{2}p_1 p_2)\phi(q) = (q_1^2, q_2^2, \sqrt{2}q_1 q_2)$$

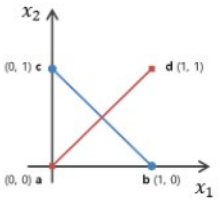
- 아래와 같이 두 점 간의 dot product 연산을 하는 커널(kernel) 함수, K(p, q)를 다음과 같이 정의해 보자.

내적

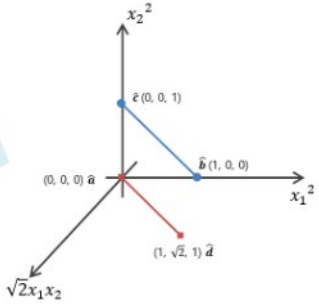
dot product 연산을 .으로 표기함.

$$\begin{aligned} K(p,q) = \phi(p).\phi(q) &= \phi(p)^T\phi(q) \\ &= (p_1^2, p_2^2, \sqrt{2}p_1 p_2).(q_1^2, q_2^2, \sqrt{2}q_1 q_2) \\ &= p_1 q_1 + p_2 q_2 + 2p_1 q_1 p_2 q_2 \\ &= (p_1 q_1 + p_2 q_2)^2 \\ \phi(p).\phi(q) &= (p.q)^2 \end{aligned}$$

dot product



$$\varphi(x) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix}$$



- 이것이 의미하는 것은 3차원의 dot product 연산이 2차원 공간에서는 곱셉의 제곱으로 이루어 진다는 것이다.

=> 연산 시간을 줄일 수 있다

- 이것은 더 높은 차원에도 적용된다. → 저차원으로 고차원의 연산을 할 수 있다.

1.2 RBF(Radial Basic Function) Kernel

The RBF kernel on two samples $\mathbf{x} \in \mathbb{R}^k$ and $\mathbf{x}'$, represented as feature vectors in some *input space*, is defined as[2]

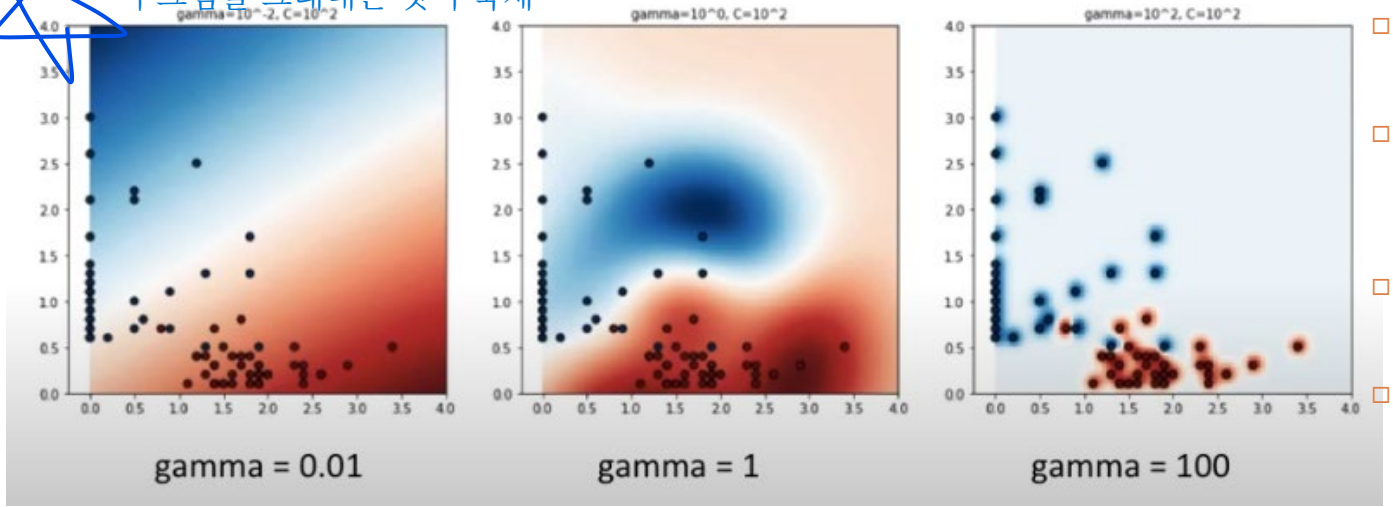
$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right) \Rightarrow K(\mathbf{x}, \mathbf{x}') = \exp(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2)$$

$\|\mathbf{x} - \mathbf{x}'\|^2$ may be recognized as the squared Euclidean distance between the two feature vectors. σ is a free parameter. An equivalent definition involves a parameter $\gamma = \frac{1}{2\sigma^2}$:

두 벡터의 거리(유사성)을 0~1로 측정.

Since the value of the RBF kernel decreases with distance and ranges between zero (in the infinite-distance limit) and one (when $\mathbf{x} = \mathbf{x}'$), it has a ready interpretation as a similarity measure.

이 그림을 그래내는 것이 숙제



- 특정 샘플(서포트벡터)을 랜드마크로 지정해서
- 각 샘플이 랜드마크와 얼마나 유사한지를 구하는 유사도 함수를 대입하여
- 나온 결과값을 해당 샘플의 새로운 특성으로 추가하고,
- 이를 기준으로 결정 경계를 만든다..

- 감마가 작을 수록 학습 데이터 하나 하나가 분류 동작에 미치는 영향의 범위가 넓어진다. → underfitting 초래 위험성 증가
- 감마가 클 수록 결정 경계의 구부러짐이 심해진다. → overfitting 초래 위험성 증가
- 감마는 서포트벡터로 선정되었을 때 샘플의 영향 반경의 역수로 간주할 수 있다. ← 음수 지수승이기 때문…

Gamma의 영향

1.3 Misclassification의 문제, C

도움 될만한 SVM 한글 자료 링크

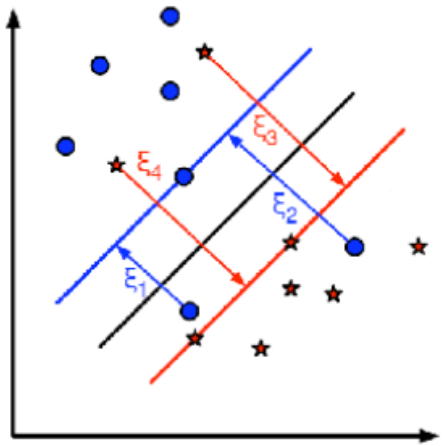
OpenCV SVM 튜토리얼 링크

- Maximum margin을 극대화하는 decision boundary를 찾는 것은 충분치 않다. ← margin을 줄이고 오 분류를 줄이는 것이 더 낫기 때문이다.
- Margin을 최대화하고, 동시에 오 분류를 최소화하기 위한 최소화해야 할 함수는 다음과 같이 정의하는 것이 유리하다.

$$min\ ||w||^2 + C(distance\ of\ misclassified\ samples\ to\ their\ correct\ regions)$$

- 그림에서 4개의 오 분류된 점들 각각이 해당 영역과의 거리를 최소로 하기 위해서는 다음 함수를 최소화해야 할 것이다.

$$\min_{w,b_0} L(w, b_0) = ||w||^2 + C\sum_i \xi_i \text{ subject to } y_i(w^T x_i + b_0) \geq 1 - \xi_i \text{ and } \xi_i \geq 0\ \forall i$$



- C 값(상수)이 크다: 위 최소화 함수에서 첫 번째 항은 경시되고, 두 번째 항이 중시되는 것을 의미한다. 즉, 모델링은 정확치 않아도 오 분류를 최소화하는 것을 의미한다. Overfitting이 생길 수 있다.
- C 값(상수)이 작다: 위 최소화 함수에서 두 번째 항은 경시되고, 첫 번째 항이 중시되는 것을 의미한다. 즉,모델링(초평면을 정확히 정의)은 정확히 하고 오 분류가 생기는 것을 각오하는 것이다. Underfitting이 생길 수 있다.

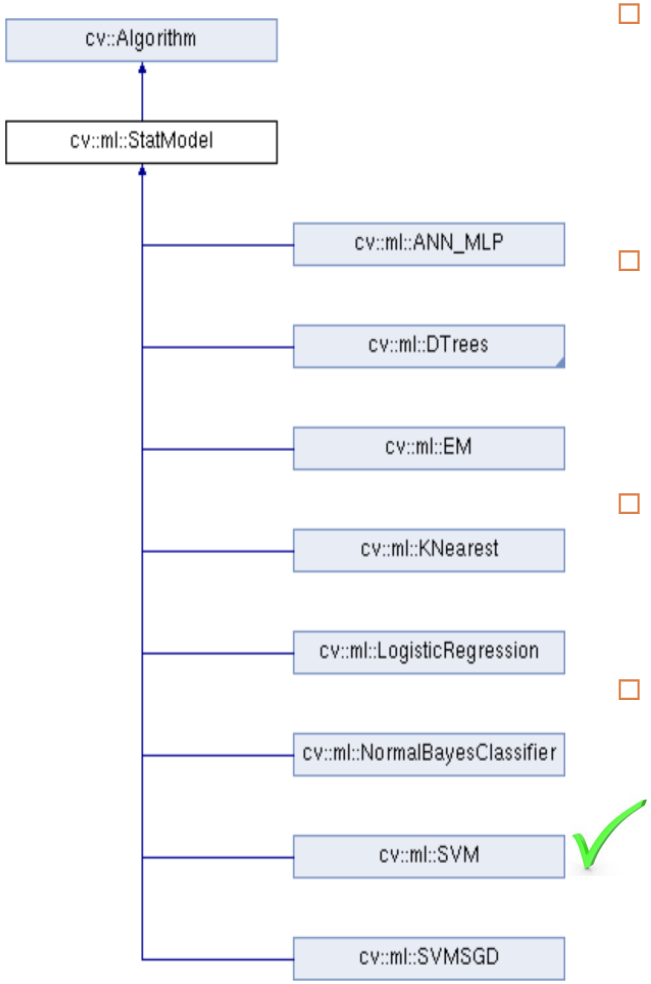
➔ 이것들은 학습 단계(SVM 모델링)에서의 결정에 불과하다. Testing 단계에서 어떤 선택이 더 좋은 결과를 냈는지는 상황에 따라 달라질 수 있다.

1.4 SVM의 정리 및 장단점

https://datamites.com/blog/support-vector-machine-algorithm-svm-understanding-kernel-trick/

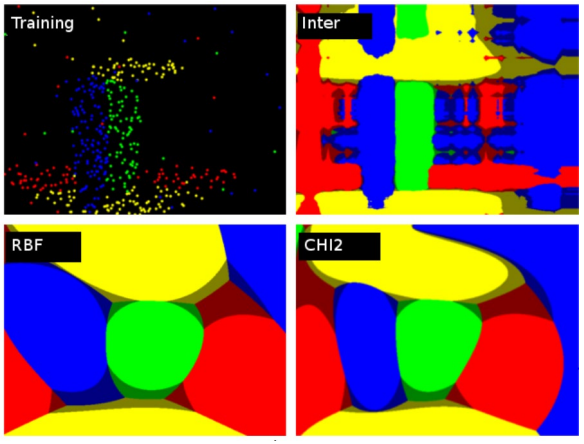
- SVM 특징 정리
  - SVM은 supervised learning 기법으로 학습한다.
  - SVM의 학습 목표는 클래스간의 마진을 최대화하는 optimal hyperplane을 찾는 것이라 정의할 수 있다.
  - SVM의 decision boundary는 전형적으로 linear이다. 비록 비선형 커널을 사용해서 non-linearly separable data를 분리하지만…
  - 원리적으로는 optimal binary classifier이다. 원래는 2종만 구분하는 것이지만 multi-class classification에도 잘 적용되도록 확장되었다.
  - 나중에 이는 regression이나 clustering 문제로 확장되었다.
  - SVM은 커널 기반의 방법 중의 일부 사례이다. 이것은 특징 벡터를 커널 함수를 이용해 고차원으로 매핑시키고, 이 공간- 학습데이터에 들어 맞는 최적 초평면(hyper plane)- 에서 선형 분리 함수를 구축한다.
- **Advantages of Support Vector Machine**
  - Training of the model is relatively easy
  - The model scales relatively well to high dimensional data
  - SVM is a useful alternative to neural networks
  - Trade-off amongst classifier complexity and error can be controlled explicitly
  - It is useful for both Linearly Separable and Non-linearly Separable data
  - Assured Optimality: The solution is guaranteed to be the global minimum due to the nature of Convex Optimization
- **Disadvantages of Support Vector Machine**
  - Picking right kernel and parameters can be computationally intensive
  - In Natural Language Processing (NLP), a structured representation of text yields better performance. However, SVMs cannot accommodate such structures(word embedding)

2. OpenCV SVM 함수



- Support Vector Machines (SVM) tutorial
  - Understanding SVM
  - OCR of Hand-written Data using SVM
- 모델 생성
  - cv.ml.SVM_create() -〉retval
  - 파라미터 자동 설정: trainAuto()
- 모델 학습(training)
  - cv.ml_StatModel.train(trainData[, flags]) -〉 retval
- 예측(testing)
  - cv.ml_StatModel.predict(samples[, results[, flags]]) -〉 retval, results

2.1 cv::ml::SVM Class Reference



KernelType에 따른 2차 평면 상의 4개의 class 분류 결과

□ KernelType

CUSTOM	Returned by SVM::getKernelType in case when custom kernel has been set
LINEAR	Linear kernel. No mapping is done, linear discrimination (or regression) is done in the original feature space. It is the fastest option. $K(x_i, x_j) = x_i^T x_j$.
POLY	Polynomial kernel: $K(x_i, x_j) = (\gamma x_i^T x_j + coef0)^{degree}, \gamma > 0$.
RBF	Radial basis function (RBF), a good choice in most cases. $K(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}, \gamma > 0$.
SIGMOID	Sigmoid kernel: $K(x_i, x_j) = \tanh(\gamma x_i^T x_j + coef0)$.
CHI2	Exponential Chi2 kernel, similar to the RBF kernel: $K(x_i, x_j) = e^{-\gamma \chi^2(x_i, x_j)}, \chi^2(x_i, x_j) = (x_i - x_j)^2/(x_i + x_j), \gamma > 0$.
INTER	Histogram intersection kernel. A fast kernel. $K(x_i, x_j) = min(x_i, x_j)$.

감마값이 커지면 경계의 강도 증가 → overfitting 위험 증가

□ SVM Type

C_SVC	C-Support Vector Classification. n-class classification ($n \geq 2$), allows imperfect separation of classes with penalty multiplier C for outliers.
NU_SVC	ν-Support Vector Classification. n-class classification with possible imperfect separation. Parameter ν (in the range 0..1, the larger the value, the smoother the decision boundary) is used instead of C.
ONE_CLASS	Distribution Estimation (One-class SVM). All the training data are from the same class, SVM builds a boundary that separates the class from the rest of the feature space.
EPS_SVR	ϵ-Support Vector Regression. The distance between feature vectors from the training set and the fitting hyper-plane must be less than p. For outliers the penalty multiplier C is used.
NU_SVR	ν-Support Vector Regression. ν is used instead of p. See [50] for details.

2.2 get(), set() methods

virtual double	**getC** () const =0
virtual **cv::Mat**	**getClassWeights** () const =0
virtual double	**getCoef0** () const =0
virtual double	**getDecisionFunction** (int i, **OutputArray** alpha, **OutputArray** Retrieves the decision function. More...
virtual double	**getDegree** () const =0
virtual double	**getGamma** () const =0
virtual int	**getKernelType** () const =0
virtual double	**getNu** () const =0
virtual double	**getP** () const =0
virtual **Mat**	**getSupportVectors** () const =0 Retrieves all the support vectors. More...
virtual **cv::TermCriteria**	**getTermCriteria** () const =0
virtual int	**getType** () const =0
virtual **Mat**	**getUncompressedSupportVectors** () const =0 Retrieves all the uncompressed support vectors of a linear SVM

virtual void	**setC** (double val)=0
virtual void	**setClassWeights** (const **cv::Mat** &val)=0
virtual void	**setCoef0** (double val)=0
virtual void	**setCustomKernel** (const **Ptr< Kernel >** &_kernel)=0
virtual void	**setDegree** (double val)=0
virtual void	**setGamma** (double val)=0
virtual void	**setKernel** (int kernelType)=0
virtual void	**setNu** (double val)=0
virtual void	**setP** (double val)=0
virtual void	**setTermCriteria** (const **cv::TermCriteria** &val)=0
virtual void	**setType** (int val)=0

2.3 create() 함수

- cv.ml.SVM.create(), cv.ml.SVM_create()
  - Creates empty model.
    - Use StatModel::train to train the model. Since SVM has several parameters, you may want to find the best parameters for your problem, it can be done with SVM::trainAuto.
  - 일단 생성하고 파라미터는 set() 함수로 바꾼다…

```
model = cv2.ml.SVM_create()
model.setGamma(gamma)    # gamma=0.50625
model.setC(C)    # C=12.5

# SVM kernel type                          #model.setKernel(cv2.ml.SVM_RBF)
model.setKernel(cv2.ml.SVM_LINEAR)         #model.setKernel(cv2.ml.SVM_POLY)
                                           #model.setKernel(cv2.ml.SVM_SIGMOID)

# C_SVC: C-Support Vector Classification. n-class classification (n ≥ 2),
# allows imperfect separation of classes with penalty multiplier C for outliers.
model.setType(cv2.ml.SVM_C_SVC)

model.setTermCriteria((cv2.TERM_CRITERIA_MAX_ITER, 100, 1e-6))
```

hyperplane(line) 찾는 과정

2.4 train() 함수

- cv.ml.StatModel.train(samples, layout, responses) ✓

samples training samples

layout See ml::SampleTypes.

ROW_SAMPLE Python: cv.ml.ROW_SAMPLE	each training sample is a row of samples
COL_SAMPLE Python: cv.ml.COL_SAMPLE	each training sample occupies a column of samples

responses vector of responses associated with the training samples. (= label ↷)
그동안 알고 있는 label과 같은 개념이다. 즉, 주어진 학습 데이터들 개별의 클래스의 번호이다.

```
model.train(samples, cv2.ml.ROW_SAMPLE, responses)
```

2차 평면에 존재하는 학습 데이터의 개수가 5개라고 하고, 레이블을 “1”혹은 “-1”로 부여했다면 다음 사례의 결과를 볼 수 있다.

```
Training data set: num=5, samples.shape=(5, 2)
labels는 responses와 같은 의미: labels.shape=(5,), labels=[ 1  1 -1 -1 -1]
```

cv.ml.StatModel.train(trainData[, flags])

trainData training data that can be loaded from file using TrainData::loadFromCSV or created with TrainData::create.

flags optional flags, depending on the model. Some of the models can be updated with the new training samples, not completely overwritten (such as NormalBayesClassifier or ANN_MLP).

설명 생략…

2.5 predict() 함수

- cv.ml.StatModel.predict(samples[, results[, flags]]) -> retval, results
- Predicts response(s) for the provided sample(s)

samples The input samples, floating-point matrix

results The optional output matrix of results.

flags The optional flags, model-dependent. See cv::ml::StatModel::Flags

UPDATE_MODEL	
RAW_OUTPUT	makes the method return the raw results (the sum), not the class label
COMPRESSED_INPUT	
PREPROCESSED_INPUT	

```
ret, results = svm_model.predict(test_data)
```

= label

2차원 형태로 나옴 -> 1차원으로 바꿔야 함

results.shape=(test_data의 길이, 1)

labels(다른 표현으로는 responses)와 같은 차원으로 만들려면 ravel()이나, flatten()함수를 써야 한다.

만약 5개의 길이로 이루어진 test_data라면 다음과 같은 사례의 데이터를 볼 수 있다.

```
results.ravel()=[ 1.  1. -1. -1. -1.]
results.flatten()=[ 1.  1. -1. -1. -1.]
```

3. SVM 단순 실험

아래 데이터 5개로 학습시키고,
1) 학습한 것은 분류가 잘되는지,
2) 학습하지 않은 데이터에 대해서는 어떻게 분류해 낼 것인지를 테스트 결과를 색상(노란색, 남색)으로 표현한다.

5개의 2차원, 2종 데이터로 SVM을 학습시켜 보고, 그 모델의 분류 능력을 검증해 본다.

svm01_introduction.py

```
1) Training data set: num=5, samples.shape=(5, 2),
labels는 responses와 같은 의미: labels.shape=(5,), labels=[ 1  1 -1 -1 -1]
[[500.  10.]] 1 blue
[[550. 100.]] 1 blue
[[300.  10.]] -1 green
[[500. 300.]] -1 green
[[ 10. 600.]] -1 green
sample(2차원 좌표) label dot_color
```

← 2종 레이블

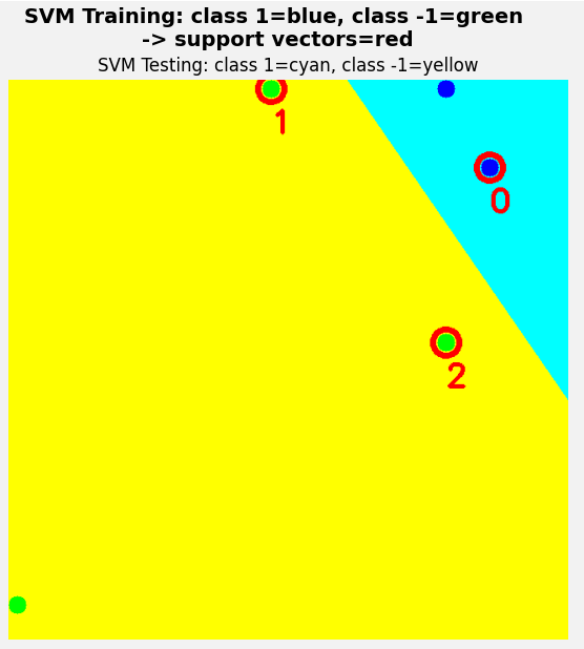
```
# 1) Set up training data:
2) Initialize the SVM model: svm_model = cv2.ml.SVM_create()
3) Train the SVM: svm_model.train(samples, cv2.ml.ROW_SAMPLE, labels)
4) 일단 학습한 데이터에 대한 predict 결과를 보기로 하자.
packed_return_values = svm_model.predict(samples)
4.1) packed return values of predict(): type(packed_return_values)=<class 'tuple'>
4.2) unpacked return values of predict(): ret=0.0, results.shape=(5, 1)
results.ravel()=[ 1.  1. -1. -1. -1.]
results.flatten()=[ 1.  1. -1. -1. -1.]
```

학습 결과와 일치한다.
labels == results

```
6) 레이블 1 평면은 BG(cyan), 레이블 -1 평면은 GR(yellow) 색상으로 표현한다.
class 1 학습 데이터는 blue 색상으로 표현된다.
class -1 학습 데이터는 green 색상으로 표현된다.
학습데이터 중 support vector들은 빨간색 원으로 마킹한다.
support vectors in (x, y) form:
0: (550, 100)
1: (300, 10)
2: (500, 300)
```

좌표 전 공간에 대한 test 실행

5개의 학습 데이터로 test 실행



4. 필기체 문자 분류 실험

svm02_digits_recog_deskew_hog.py

- KNN에서 실험했던 것처럼 deskew 보정과 hog 추출한 결과를 가지고 SVM으로 분류 처리를 시행해 본다.
- KNN의 사례와 같이 hog 기술자는 144개의 벡터를 사용한다.
- 전반적으로 아래에 보인 바와 같이 KNN에서 K를 잘 선택했을 때 보다도 우수하다.

```
90% 학습시켰을 때: SVM=99.00%, 2-NN=98.6(best), 1-NN:97.6(worst)
80% 학습시켰을 때: SVM=98.70%, 9-NN=97.7(best), 1-NN:97.3(worst)
70% 학습시켰을 때: SVM=98.87%, 7-NN=98.0(best), 1-NN:97.6(worst)
60% 학습시켰을 때: SVM=98.70%, 7-NN=98.0(best), 2-NN:97.5(worst)
50% 학습시켰을 때: SVM=98.60%, 7-NN=97.9(best), 2-NN:97.3(worst)
40% 학습시켰을 때: SVM=98.50%, 6-NN=97.8(best), 2-NN:97.0(worst)
30% 학습시켰을 때: SVM=98.00%, 4-NN=97.4(best), 1-NN:97.0(worst)
20% 학습시켰을 때: SVM=98.78%, 4-NN=97.4(best), 1-NN:96.7(worst)
10% 학습시켰을 때: SVM=98.13%, 4-NN=96.1(best), 8-NN:95.7(worst)
```

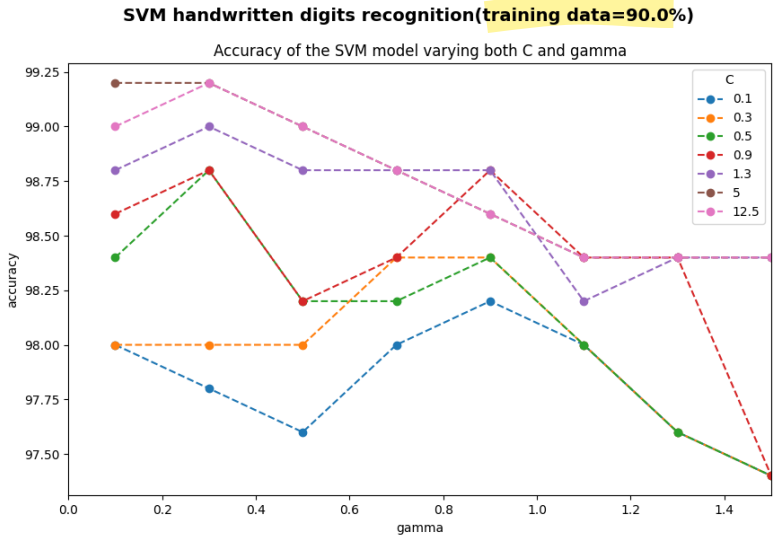
skew
찌그러진 글자를 올바르게

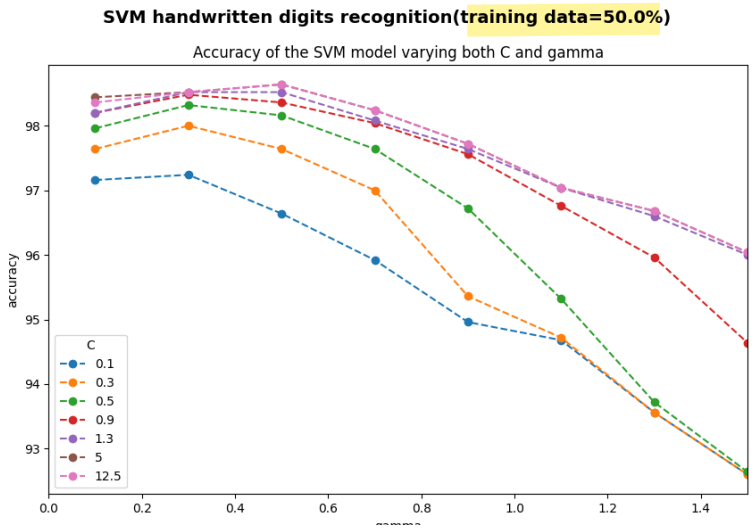
```
partition = int(0.9 * len(hog_descriptors))      # 90%를 학습, 10%를 테스트로 사용한다.
model = svm_init(C=12.5, gamma=0.50625)
model.train(hog_descriptors_train, cv2.ml.ROW_SAMPLE, labels_train)
```

```
hog descriptor size=144
5.1) 학습데이터: <class 'numpy.ndarray'> (4500, 144) float32
5.2) 학습레이블: <class 'numpy.ndarray'> (4500,)
Training SVM model ...
Evaluating model ...
Percentage Accuracy: 99.00 %
```

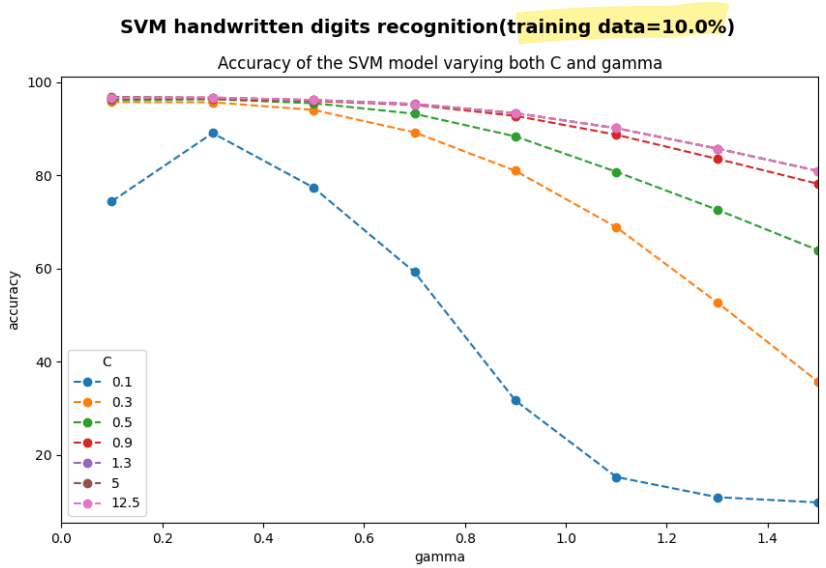
5. C, gamma 및 학습량 비율에 따른 성능 비교 실험

svm03_digits_recog_deskew_hog_varying_c_gamma.py





c : 간격을 얼마나 넓힐 것인가
gamma : 커지면 타이트, 작으면 루즈




```
C= 5.0, gamma=0.1: accuracy=99.20
C= 5.0, gamma=0.3: accuracy=99.20
C=12.5, gamma=0.3: accuracy=99.20
C= 1.3, gamma=0.3: accuracy=99.00
C= 5.0, gamma=0.5: accuracy=99.00
```

```
C= 5.0, gamma=0.5: accuracy=98.64
C=12.5, gamma=0.5: accuracy=98.64
C= 1.3, gamma=0.3: accuracy=98.52
C= 1.3, gamma=0.5: accuracy=98.52
C= 5.0, gamma=0.3: accuracy=98.52
```

fact: C, 감마가 크면 overfitting 위험성 증가

C와 gamma는 적절하게 선정해야 한다는 것 외에 **다른 결론을 도출할 수 있을까요?**

```
C= 5.0, gamma=0.1: accuracy=96.84
C= 5.0, gamma=0.3: accuracy=96.71
C=12.5, gamma=0.3: accuracy=96.71
C=12.5, gamma=0.1: accuracy=96.67
C= 1.3, gamma=0.1: accuracy=96.64
```

적어도 C=5.0/12.5 gamma는 0.1/0.3 정도가 좋아 보입니다. 참고서나 아래 값에서 99.2% 달성하였음.

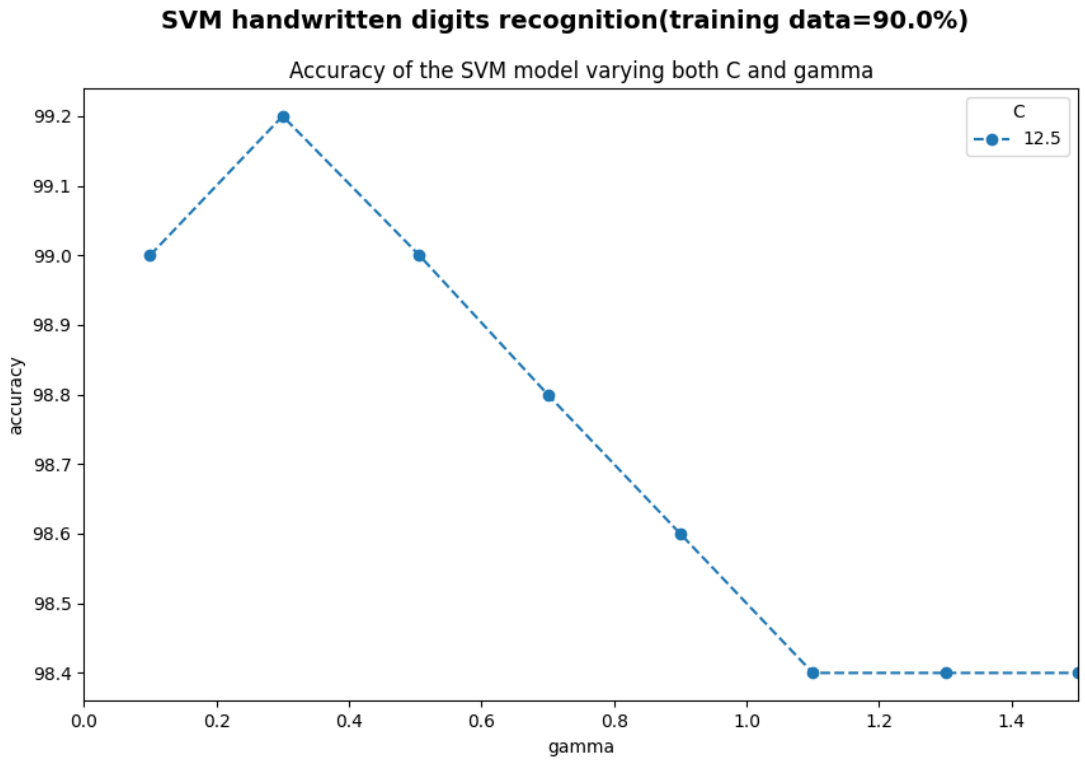
$C=12.5$ and $\gamma=0.50625$.

SVM_02 실험과의 비교

svm03_digits_recog_deskew_hog_varying_c_gamma.py

주요 수행 코드

```
rand = np.random.RandomState(1234)
gamma_list = [0.1, 0.3, 0.50625, 0.7, 0.9, 1.1, 1.3, 1.5]
C_list = [12.5]  C=12.5 and γ=0.50625.
```



의문 사항: SVM_03 예제는 99.2% 정확도인데, SVM_02 예제에서는 99.0% 정확도이다.

의심점: shuffle 과정에서 서로 다른 데이터를 학습데이터와 테스트 데이터로 쓸 수도 있다.

반론: RandomState(1234)의 seed 값이 같다. 이는 KNN에서도 계속 써 오던 값이다.

* 그 동안의 추정: seed가 같으면 난수 발생 패턴도 같다. True or False?

```
C=12.5, gamma=0.3: accuracy=99.20
C=12.5, gamma=0.1: accuracy=99.00
C=12.5, gamma=0.5: accuracy=99.00
C=12.5, gamma=0.7: accuracy=98.80
C=12.5, gamma=0.9: accuracy=98.60
```

참고 1: scikit의 사례

- SVM hyper parameters: scikit learn의 사례
  - C(cost) : 이론에서 배운 주요 파라미터로써 어느 정도의 오차를 허용할 지에 대한 파라미터입니다.
  - kernel : 어떤 커널함수를 사용할지에 대한 파라미터입니다. 'linear', 'sigmoid', 'rbf', 'poly'가 활용됩니다.
  - degree : 어느 차수까지의 다항차수로 분류를 할 지에 대한 파라미터입니다. 커널함수가 'poly' 일 때 사용됩니다.
  - gamma : 곡률 경계에 대한 파라미터입니다. 'rbf', 'poly', 'sigmoid'일 때 튜닝하는 값입니다.
  - coef0 : 상수값으로써 'poly', 'sigmoid'일 때 튜닝을 진행합니다.

참고 2: A Gentle Introduction To Method Of Lagrange Multipliers

어떤 문제를 어떻게 푸느냐? -SVM에서 목적함수의 최적화에 사용

https://machinelearningmastery.com/a-gentle-introduction-to-method-of-lagrange-multipliers/

- Lagrange multipliers는 어떤 함수의 local minima 혹은 local maxima을 어떤 조건(equality or inequality constraints)하에서 찾아내는 간결한 방법이다.

문제의 정의

- Minimize f(x)
- Subject to:
  - g_1(x) = 0
  - g_2(x) = 0
  - ⋯
  - g_n(x) = 0

풀이 방법

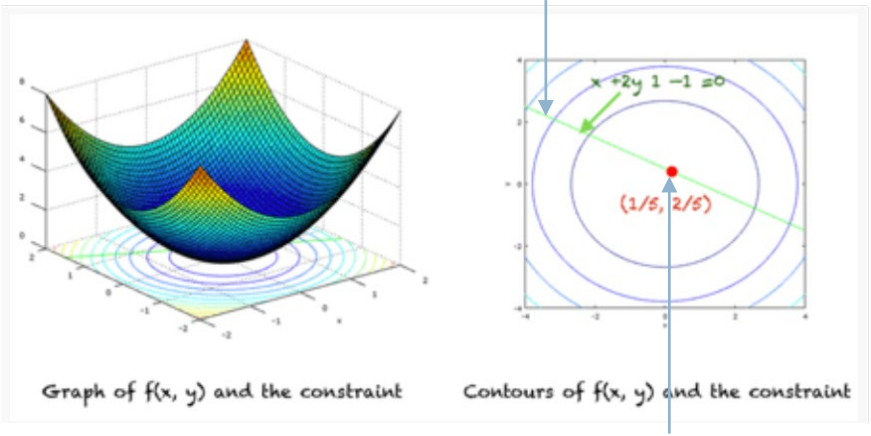
- 라그랑제 함수 (Lagrangian)를 정의한다.
  - $L(x, \lambda) = f(x) + \lambda\_1 g\_1(x) + \lambda\_2 g\_2(x) + \cdots + \lambda\_n g\_n(x)$
    - Here λ represents a vector of Lagrange multipliers, i.e., $\lambda = [\lambda\_1, \lambda\_2, \cdots, \lambda\_n]^T$
- 극점을 찾기 위해서는 아래 2개의 식을 만족하는 함수의 변수를 찾는다.
  - $\nabla x L = 0$ ⇨ 라그랑제함수를 x로 미분한 것이 0이되어야 한다.
    - $\partial L/\partial x\_j = 0$ (for j = 1..m)
  - $\partial L/\partial \lambda\_i = 0$ (for i = 1..n) ⇨ 라그랑제함수를 λ_i 로 미분한 것이 0이되어야 한다.
    - g_i(x) = 0 (for i = 1..n)
- 결론적으로 총 m+n개의 등식을 만족하는 변수를 찾으면 된다.
  - m = number of variables in domain of f
  - n = number of equality constraints.

An example: 주어진 조건을 만족하며 어떤 함수의 최대/최소값 찾기

https://machinelearningmastery.com/a-gentle-introduction-to-method-of-lagrange-multipliers/

- 문제:
  - Minimize: f(x) = x^2 + y^2
  - Subject to: x + 2y − 1 = 0
- 해법
  - 라그랑제 함수(Lagrangian) 정의
    - $L(x, y, \lambda) = x^2 + y^2 + \lambda(x + 2y - 1)$
  - 이로부터 3개의 미분식이 유도된다.
    - $\partial L/\partial x = 0,\ 2x + \lambda = 0$ (1)
    - $\partial L/\partial y = 0,\ 2y + 2\lambda = 0$ (2)
    - $\partial L/\partial \lambda = 0,\ x + 2y - 1 = 0$ (3)
  - (1)과 (2)식으로부터.
    - $\lambda = -2x = -y$ (4)
  - 식(4)를 식(3)에 대입하면
    - $x = 1/5,\ y = 2/5$

녹색직선이 제한 조



minimum

검토

- **SVM 문제 1** – 1.2 RBF(Radial Basic Function) Kernel
  - 1.2절의 gamma에 따른 분류 특성을 보여주는 사례와 같은 프로그램의 작성
    - 단계 1: gamma에 따라 몇 개의 그림을 한 화면에 matplotlib로 도시
    - 단계 2: 가능한 클래스를 2~4개 정도 선택가능하도록 설계 바람.
  - 선택사항: 여기부터는 OpenCV의 Trackbar로 구현
    - 단계 3: trackbar로 감마를 제어 (감마 값은 화면에서 문자로 출력)
    - 단계 4: 트랙 바를 하나 더 추가하여 C값의 설정도 바꿀 수 있음(화면에 값 출력)
    - 단계 5: 트랙 바 추가하여 커널 함수를 고를 수 있었으면 좋겠음(2.1절 사례처럼)
      - 일부 데이터에 대해 적용이 불가할 수도 있음.
  - 레포트: 단계 몇 번, 몇 번을 구현했는지 서두(문제 바오 다음)에 밝히기 바람.
- **SVM 문제 2** – 5절의 SVM_02 실험과의 비교에서 차이가 발생하는 원인 규명
  - 레포트: Seed가 난수 발생의 재연을 하는가 or 안 하는가를 서두에 먼저 밝히기 바람.