

2020 FALL ESC :: Week 1 (cv)

- 2조: 오정현 유은영 이성우 이승준 임선우

1. image classification

- 1.1 data driven approach
- 1.2 K-Nearest Neighbor
- 1.3 Cross-Validation
- 1.4 linear classifier

2. loss functions and optimization

- 2.1 SVM
- 2.2 Regularization
- 2.3 crossentropy loss(softmax)
- 2.4 optimization

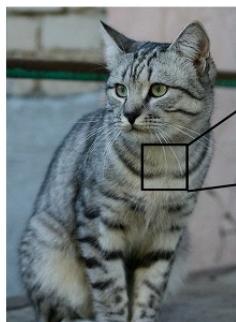
1. image classification

1.1 data driven approach

첫번째 강의는 이미지분류(image classification) 문제를 소개한다.

이미지분류는 이미지를 입력받아 카테고리 내에서 정답을 고르는 문제로 computer vision 분야에서 주요하게 다룬다.

The Problem: Semantic Gap



This image by [nikita](#) is licensed under [CC-BY 2.0](#)

[148 112 180 151 184 99 286 69 96 180 132 210 124 97 93 87]
[91 98 182 186 184 79 98 183 185 185 223 234 158 185 183 182]
[76 85 98 185 128 185 87 96 95 99 115 112 168 183 99 85]
[86 91 61 64 69 91 88 85 191 187 189 99 75 84 94 95]
[114 148 89 53 55 69 64 54 64 87 112 129 98 74 84 91]
[114 148 89 53 55 69 64 54 64 87 112 129 98 74 84 91]
[128 137 144 149 189 95 86 79 62 65 63 63 68 73 86 181]
[128 137 144 149 189 95 86 79 62 65 63 63 68 73 86 181]
[127 125 148 147 133 127 126 131 111 76 89 75 41 64 72 84]
[115 114 189 127 158 148 131 118 113 189 188 92 74 65 72 78]
[115 114 189 127 158 148 131 118 113 189 188 92 74 65 72 78]
[63 77 88 81 77 79 182 123 117 115 117 125 125 138 115 87]
[62 65 82 88 78 71 88 181 124 126 119 181 187 114 116 110]
[62 65 82 88 78 71 88 181 124 126 119 181 187 114 116 110]
[87 65 75 87 186 95 69 45 76 138 126 187 92 94 185 112]
[157 178 157 128 93 86 114 132 112 97 69 95 78 82 99 94]
[157 178 157 128 93 86 114 132 112 97 69 95 78 82 99 94]
[128 112 98 117 158 144 128 115 146 187 182 93 87 85 72 79]
[122 121 182 82 82 86 94 117 145 148 153 182 58 78 92 187]
[122 164 148 181 71 56 78 83 93 183 119 133 102 61 69 84]

What the computer sees

An image is just a big grid of numbers between [0, 255]:

e.g. 800 x 600 x 3
(3 channels RGB)

인간의 인지능력은 뛰어나기에 우리는 위 사진이 고양이라는 것을 쉽게 알 수 있다. 하지만 컴퓨터가 이미지를 바라보는 방식은 우리와는 많이 다르다.

컴퓨터는 우리처럼 전체적인 시각에서 이미지를 바라보지 않고, 아주 많은 숫자들의 모임으로 바라볼 뿐이다.

가령, 위의 고양이 이미지는 800 x 600 x 3의 크기를 가지는 이미지다. 여기서 800x600은 이미지의 2차원 크기를 나타내고, 뒤에 있는 3은 컴퓨터에서 색상을 표현할 때 주로 사용하는 RGB표현법에 의해 나타난 숫자다. (RGB : Red, Green, Blue) 즉, 위 이미지의 800x600의 픽셀 하나하나가 3개의 RGB 값으로 이루어져있다는 것이다.

결국 컴퓨터는 위 이미지를 1,440,000개(800x600x3)의 숫자로 인식하는데, 컴퓨터가 이 모든 pixel 값을 가지고 있다면 고양이를 잘 분류할 수 있을까?

답은 No. Why? 밑의 사진들을 보자.

Challenges: Illumination





[This image is CC0 1.0 public domain](#)



[This image is CC0 1.0 public domain](#)



[This image is CC0 1.0 public domain](#)



[This image is CC0 1.0 public domain](#)

Challenges: Background Clutter



[This image is CC0 1.0 public domain](#)



[This image is CC0 1.0 public domain](#)

Challenges: Occlusion



[This image is CC0 1.0 public domain](#)

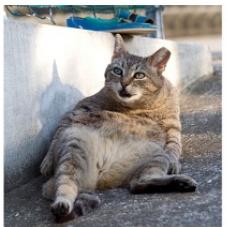


[This image is CC0 1.0 public domain](#)



[This image by jonsen is licensed under CC-BY 2.0](#)

Challenges: Deformation



[This image by Umberto Salvagni is licensed under CC-BY 2.0](#)



[This image by Umberto Salvagni is licensed under CC-BY 2.0](#)



[This image by pete rock is licensed under CC-BY 2.0](#)



[This image by Iam This is licensed under CC-BY 2.0](#)

- view point의 문제. 다른 각도에서 찍으면, 같은 고양이더라도 픽셀값이 전혀 다른 값들로 바뀐다.
- 빛의 강도의 차이, 배경의 차이, 부분적으로 가려지거나 이상한 포즈를 취하는 등...

물론 인간은 이런 경우에도 위 이미지들이 고양이라는 것을 알 수 있다. 그러나 컴퓨터에게 이것들을 인식하게 만드는 것은 어렵다.

이에 우리는 컴퓨터가 이런 상황들에서도 고양이를 인식할 수 있도록, 알고리즘을 robust하게 설계해야 한다.

그렇다면 다양한 이미지들을 분류하는 알고리즘을 어떻게 설계할 수 있을까?

예를 들어 비행기, 자동차, 새, 고양이, 사슴의 다섯 가지 사물과 동물을 구별해내는 프로그램을 만들려면 어떤 알고리즘을 써야 할까?

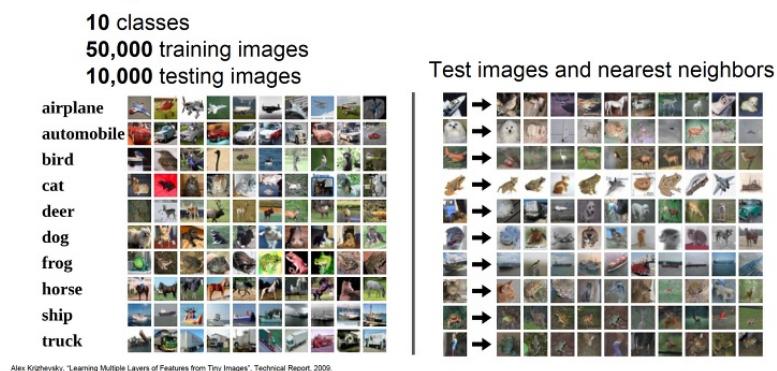
- 1) 우선 다섯 가지 사물과 동물의 다양한 사진들을 모은다(collect a data set).
- 2) 컴퓨터에게 5가지 class의 사진들을 각 class별로 label을 달아서 보여주고, 각각의 사진들을 학습시킨다. 학습하는 과정을 예로 들면, 사슴 사진을 많이 주고 사슴의 평균적 특징을 뽑아내는 식이다. 3) 분류(classification)가 잘 되는지 확인해본다.

이런 식의 Train&Predict는 라벨이 달려있는 학습이미지 data set를 기반으로 하기 때문에 **data-driven approach**라고 부른다.

1.2 KNN(K-Nearest Neighbor)

첫번째 Learning Algorithm은 KNN(K-Nearest Neighbor)이다.

Example Dataset: CIFAR10



Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 2 - 19 April 6, 2017

example data set은 CIFAR10 으로, 해당 데이터에는 10개의 class, 각 class별 training images 5000장, testing images 1000장으로 총 6만장의 사진이 있으며, 이미지들의 정답 라벨도 있다. 각 사진은 32x32x3(RGB) 크기로 이루어져 있다.

First classifier: Nearest Neighbor

```
def train(images, labels):
    # Machine learning!
    return model
```

→ Memorize all data and labels


```
def predict(model, test_images):
    # Use model to predict labels
    return test_labels
```

→ Predict the label of the most similar training image

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 2 - 17 April 6, 2017

KNN 알고리즘은 train 과정에서 모든 이미지와 라벨을 저장하고, predict 과정에서 train에서 모아 둔 이미지 중에서 test image와 가장 유사한 이미지를 골라서 반환하는 식으로 진행된다.

이때 유사도를 어떻게 판단할까?

첫 번째 방법은 **L1 distance(Manhattan distance)**으로, 각 이미지의 픽셀 간의 차이를 기준으로 한다.

Distance Metric to compare images

$$\text{L1 distance: } d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

test image				training image				pixel-wise absolute value differences			
56	32	10	18	10	20	24	17	46	12	14	1
90	23	128	133	8	10	89	100	82	13	39	33
24	26	178	200	12	16	178	170	12	10	0	30
2	0	255	220	4	32	233	112	2	32	22	108

= add 456

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 2 - 20 April 6, 2017

위는 4×4 행렬을 예시로 L1 distance 연산을 한 결과다. test 이미지와 training image 사이의 대응되는 픽셀 값을 빼고 절댓값을 취해 준 후 이 값을 모두 합친 것이 유사도를 나타내게 된다.

이것을 수식으로 나타내면 $d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$ 가 된다.

(I_1, I_2 가 각각 test와 train data가 된다. 순서는 절댓값 취해주기 때문에 상관없음. 또한 예제의 경우 p 는 $32 \times 32 \times 3 = 3072$ 가 될 것이다.)

실제로 predict할 때는 test데이터를 입력 받으면 test데이터 하나에 대한 모든 training 데이터와의 L1 distance를 구하고, 이 값이 가장 작은 사진을 정답으로 선택하게 될 것이다.

```
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N """
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for """
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred
```

Nearest Neighbor classifier

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 2 - 21

April 6, 2017

- 코드 참고: 앞서 본대로 train과 predict 함수로 이루어져 있다. train 함수(method)에서는 데이터를 그대로 저장하기만 하고, predict 함수에서 num_test 변수에 test이미지의 행(row)값을 numpy의 shape 함수를 이용하여 구해서 저장해준다. 그 밑의 for문은 행의 개수(num_test)만큼 돌고, distances에는 열 단위(axis=1 부분)로 sum을 해주고 있다.

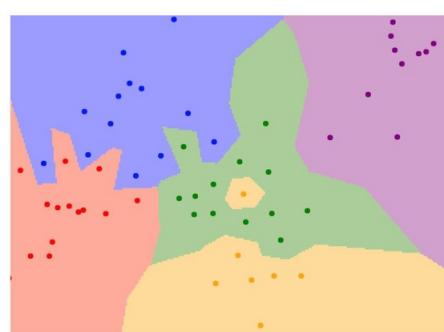
지금까지 KNN 알고리즘을 L1 distance를 이용해 구현해보았다. 이제 성능 평가를 해보자.

- 컴퓨터과학 분야에서 알고리즘의 성능평가는 최악의 경우를 나타내는 Big-O notation 표기법을 이용해 나타내며, 다양한 평가 요소가 있지만 보통은 얼마나 시간이 걸리는지를 기준으로 하는 time complexity를 척도로 한다.
- 해당 알고리즘의 train 함수의 time complexity는 $O(1)$ 이다. 이미지를 단순 저장만 하기 때문에 train이 빠를 수밖에 없다. 그러나, predict 함수의 time complexity는 $O(n)$ 이다. 여기서 n 은 데이터의 개수를 말하고, 우리는 predict를 할 때마다 데이터 전체를 훑어야 하기 때문에 $O(n)$ 이 되는 것이다.

결론적으로, L1을 이용한 KNN 분류 알고리즘의 성능은 좋지 않다. 보통 사람들은 학습이 느리더라도 predict가 빠르길 바라는데, 이 알고리즘은 학습은 빠르고 predict가 너무 느리기 때문이다.

그래도 일단 만들었으니까 KNN classifier를 시각화 해보자.

What does this look like?



위 이미지는 CIFAR10처럼 10개가 아닌 5개의 카테고리를 가진 데이터 셋을 KNN으로 분류한 결과를 나타낸 것이다. 각 색깔들 사이에

는 자연스럽게 선이 생기게 되는데, 이 선을 decision boundary라고 부른다. 이 그래프에서 볼 수 있는 KNN classifier의 문제는 두 가지다.

- 1) 초록색 영역 안에 있는 작은 노란색 부분이 초록색이 되어야 하지 않나?
- 2) 들쭉날쭉한 decision boundary. 이는 실제 예측시에도 불안한 모습을 보여줄 수 있다.

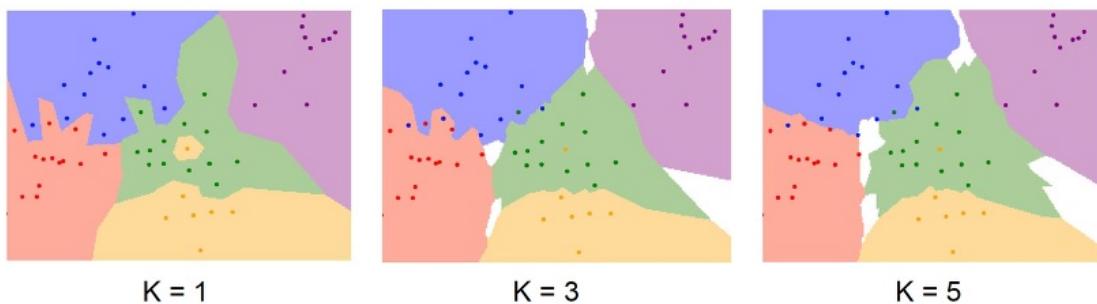
위의 노란 섬 문제의 원인도 아마 들쭉날쭉한 decision boundary가 생긴 원인과 비슷할 것이다. 이 문제를 해결해보자.

사실 KNN에서 K는 Nearest Neighbour을 얼마나 뽑을지 나타내는 숫자다.

우리가 지금까지 썼던 KNN은 사실 1NN이었던 것! 지금까지는 1등의 의견만 들어서 빠죽빠죽한 decision boundary가 나왔다. K를 높은 수로 잡아주게 되면 더 많은 의견을 들을 수 있게 된다.

K-Nearest Neighbors

Instead of copying label from nearest neighbor,
take **majority vote** from K closest points



K가 1보다 클 때는 majority vote, 다수결 투표를 이용한다. 가령 K가 100이라면 열 개의 결과 중 가장 많은 표를 받은 class를 정답으로 하는 것이다.

- K=3 일 때를 보면, K=1일 때 있던 노란 부분이 없어지고 빠죽하던 부분이 덜해졌다.(하얀 공간은 다수결에서 한 번도 승리하지 못한 부분)
- K=5일 때는 더 부드러워진 부분도 있지만 오히려 빠죽해진 부분도 있다. K가 크다고 무조건 좋은 건 아니라는 걸 알 수 있다.

What does this look like?



Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 2 - 30

이는 KNN을 CIFAR10 데이터 셋에 적용해 본 결과로, 왼쪽이 test data, 오른쪽이 predict 결과다. K=100이기 때문에 열 가지의 후보가 나왔음.

초록색은 정답이고 빨간색이 오답으로, 오답의 개수가 6개로 정답의 개수보다 많다. 그렇다, KNN은 이미지 분류에 별로 좋지 않다. 그렇다면, 유사도를 판단하는 방법을 바꾸면 어떨까?

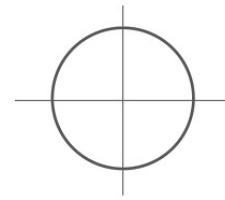
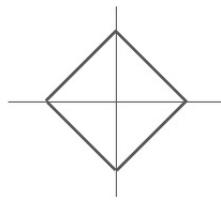
L1 distance가 아닌 L2 distance(Euclidean distance)를 사용해보자.

L1 (Manhattan) distance

L2 (Euclidean) distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$



수식을 보면 $I_1 - I_2$ 를 제곱하고 다시 루트를 써운 것이다.

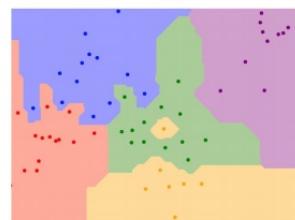
제곱을 했기 때문에 L1의 절댓값 부호는 사라지고, 그래프의 모양도 부드러워졌다.

두 식이 만들어내는 그래프의 모양이 다르므로 결과 또한 달라질 것이다. 더 general한 결과가 나오지 않을까?

참고-그래프가 둑글둥글하다고 L2가 무조건 좋은 것은 아님 : L1은 좌표 축을 경계로 각이 저있고 L2는 좌표 축과 상관없이 일관된 원형을 보여준다. 따라서 입력 데이터의 feature가 중요한 의미를 지니면 L1이 적합할 것이고(L2는 부드럽게 퍼버리기 때문에 상세한 데이터를 무시하는 듯한 상황이 생길 수 있음), 그보다는 그냥 평균적인(general) 결과를 얻고 싶다면 L2 distance가 더 적합할 것이다.

L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



K = 1

L2 (Euclidean) distance

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$



K = 1

두 이미지를 비교해보면 decision boundary의 모습이 조금 다르다. L1 distance의 경우 decision boundary가 좌표축을 따라가는 성질을 보이고 있다. L1은 좌표계에 종속적(dependent)이기 때문이다. 반면 L2 distance의 경우 좌표계에 큰 영향을 받지 않는 모습이다.

- <http://vision.stanford.edu/teaching/cs231n-demos/knn/>

서로 다른 알고리즘을 선택함으로써 나타나는 기하학적 변화를 시각화해 살펴보는 것은 직관적인 판단력을 늘려준다. 이 사이트로 들어가면 KNN 알고리즘에서 우리가 설정할 수 있는 parameter들을 조절해보면서 그래프의 움직임을 실시간으로 볼 수 있다!

- 어떤 방법으로 데이터간의 차이를 결정할지, K를 몇으로 할지 이 두 가지가 KNN classifier에서의 하이퍼파라미터

하지만 KNN은 이미지 분류에서 쓰지 않는다.



k-Nearest Neighbor on images **never used**.

- Very slow at test time
- Distance metrics on pixels are not informative



(all 3 images have same L2 distance to the one on the left)

- 첫 번째 이유는 언급했듯이 prediction이 매우 느리다는 것
- 두 번째 이유는 픽셀들간의 차(difference)가 이미지의 유사성(혹은 차이)를 잘 나타내지 못하기 때문

위 슬라이드에는 오리지널 이미지에 조금씩 변화를 준 이미지들이 있다. 그런데 오리지널 이미지에 대한 오른쪽 세 개의 이미지들의 L2 distance 값은 모두 같게 나온다. 실제로는 어느정도 다르다고 볼 수 있는 사진들이 완전히 같은 사진들로 인식되는 상황이 발생한 것이다.

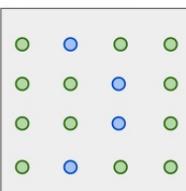
k-Nearest Neighbor on images **never used**.

- Curse of dimensionality

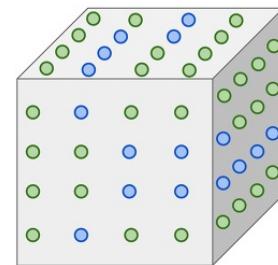
Dimensions = 1
Points = 4



Dimensions = 2
Points = 4^2



Dimensions = 3
Points = 4^3



- 세 번째는 '차원의 저주'(Curse of dimensionality)이다. 알고리즘이 잘 동작하도록 만들기 위해서는 우리가 다룰 data의 차원제곱 만큼의 학습데이터(training data)가 필요하다는 것이다. 여기서 차원은 dimension이라고도 하고 feature의 개수라고 볼 수도 있다. CIFAR10의 사진에서 dimension은 3072가 된다. CIFAR10에서 카테고리의 개수가 10개이니, KNN 알고리즘을 잘 동작하도록 하려면 10의 3072제곱만큼의 training data가 필요하다!!

1.3 Cross-Validation

Cross Validation(교차검증)

data set을 train, validation, test set으로 나누어 진행하는 교차검증.

필요한 이유

교차검증이 어떤 아이디어를 개선한 방법인지 알아보자.

- Idea#1: 주어진 training dataset에 가장 좋은 결과를 보여주는 파라미터를 선택한다.

Setting Hyperparameters

Idea #1: Choose hyperparameters
that work best on the data

Your Dataset

예측 모델의 목적은 갖고 있는 데이터를 잘 예측하는 것이 아니라
모델이 접해보지 못한, 새로운 데이터를 예측하는 것이다.

그림과 같은 방식을 활용하면 training set에만 잘 작동하는 모델을 만들어 overfitting(과적합)을 초래한다.

- Idea #2: 주어진 데이터를 train, test set으로 나누어 test set을 검증한다.

Idea #2: Split data into **train** and **test**, choose
hyperparameters that work best on test data

train

test

단순히 train set과 test set으로 나누어 검증하면 모델은 test set에 과적합된다.

마찬가지로 우리의 목적은 주어진 test set을 예측하는 것이 아니기 때문에 부적합한 방법이다.

Idea #3: Split data into **train**, **val**, and **test**; choose
hyperparameters on val and evaluate on test

Better!

train

validation

test

- Idea #3: 데이터를 train, validation, test set 세부분으로 나눈다.

validation set으로 하이퍼파라미터를 결정하고 이를 test set에서 평가한다.

즉 test set을 고정한 채로 Validation set을 활용해 하이퍼파라미터를 정하고 이를 test set에 평가한다.

모델을 검증할 때 단일한 test set을 활용하여 과적합이 발생하는데,

이를 validation set을 활용하여 개선한 것이다.



Setting Hyperparameters

Your Dataset

Idea #4: Cross-Validation: Split data into **folds**,
try each fold as validation and average the results

fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test

Useful for small datasets, but not used too frequently in deep learning

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 2 - 41

April 6, 2017

- Idea #4: K - fold Cross validation

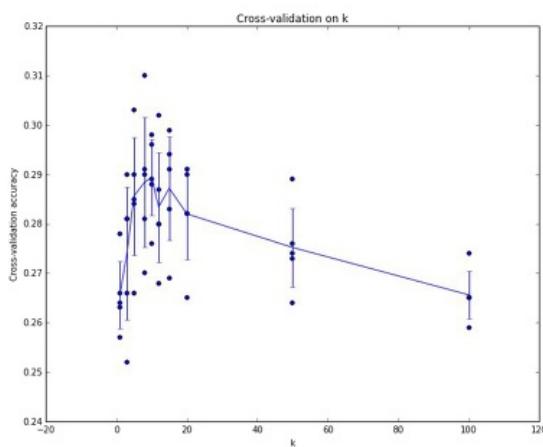
이 방법은 training set을 k개의 fold(subset)으로 나누고, 각각의 fold를 Validation set으로 하는 iteration을 거친 후, 나온 결과들(정확도)의 평균을 통해 최적의 파라미터를 도출하는 방식이다.

위 그림에서는 k가 5인 경우에 해당하고, 두 번째의 training set은 fold1,2,3,5가 training set이고 fold 4가 validation에 해당되며 fold가 중복되지 않게 지정한다.

이는 크기가 작은 데이터셋에 한정해서 유용하게 쓰인다. 실제로 많은 양의 데이터를 필요로 하는 딥러닝에서는 활용도가 떨어진다.

- 장점 - test set을 고르게 지정하므로 일반화할 수 있다.
- 단점 - 연산 비용 및 시간이 오래 걸린다.

Setting Hyperparameters



Example of
5-fold cross-validation
for the value of **k**.

Each point: single
outcome.

The line goes
through the mean, bars
indicated standard
deviation

(Seems that $k \approx 7$ works best
for this data)

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 2 - 42

April 6, 2017

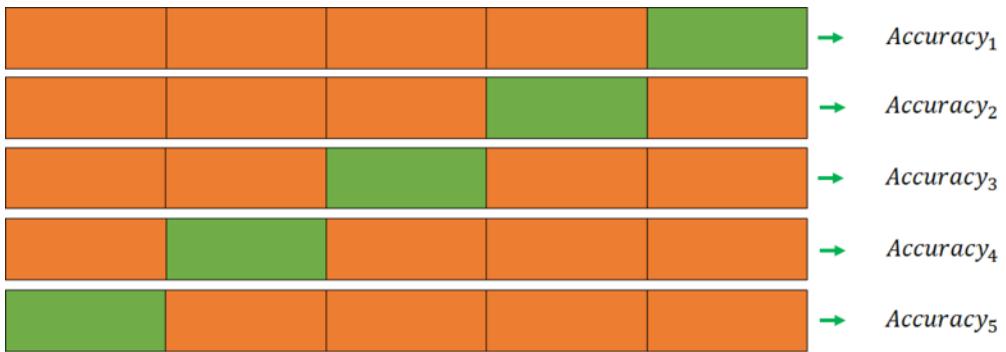
위 그림은 fold = 5 일 때 어떤 하이퍼파라미터 K 가 가장 좋은지를 보여주는 그래프이다.

x축은 k값을 나타내고 y축은 k값에 따른 accuracy의 변화를 나타낸다..

5개의 fold로 나누어서 각 k값마다 y축 방향으로 5개의 점들이 있고 이 점들의 평균값이 이어져있다.

K = 7 일 때 가장 높은 정확도를 보여준다!

K - Cross Validation의 K는 어떻게 정할까?



$$Accuracy = \text{Average}(Accuracy_1, \dots, Accuracy_k)$$

training set을 k 개의 fold로 나누다면 k 를 정하는 기준은 무엇일까?

정확한 룰은 없지만 일반적으로 k 개의 fold로 나누었을 때 각 데이터 표본이 충분히 커서 전체 데이터를 대표할 수 있어야 한다.

- 보통 $k = 10$ 이나 $k = 5$ 로 실시하는 경우가 대부분이다. k 가 커질수록 training set과 새로운 resampling subset이 비슷해질 것이고, 그럴수록 bias 가 줄어든다.

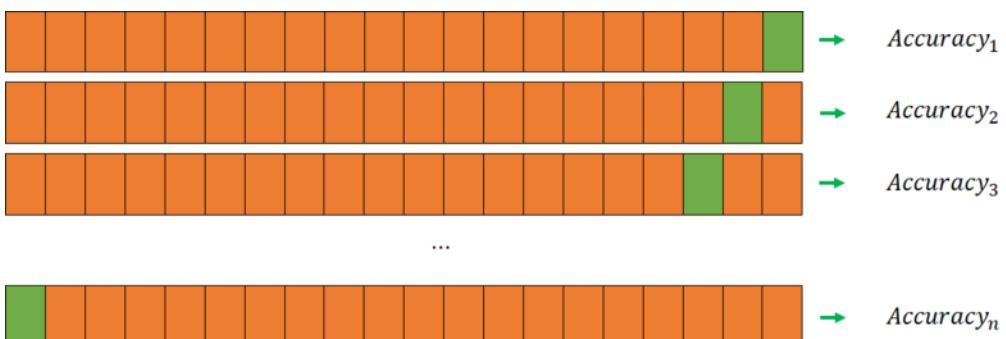
다른 교차 검정기법

KNN 알고리즘은 이미지 분류 분야에서 활용되기 어렵지만

Cross-Validation은 머신러닝에서 test error를 구할 때 널리 활용되는 기법이므로

K -fold CV 외에 어떤 종류의 Cross Validation이 있는지 간략히 알아보자!

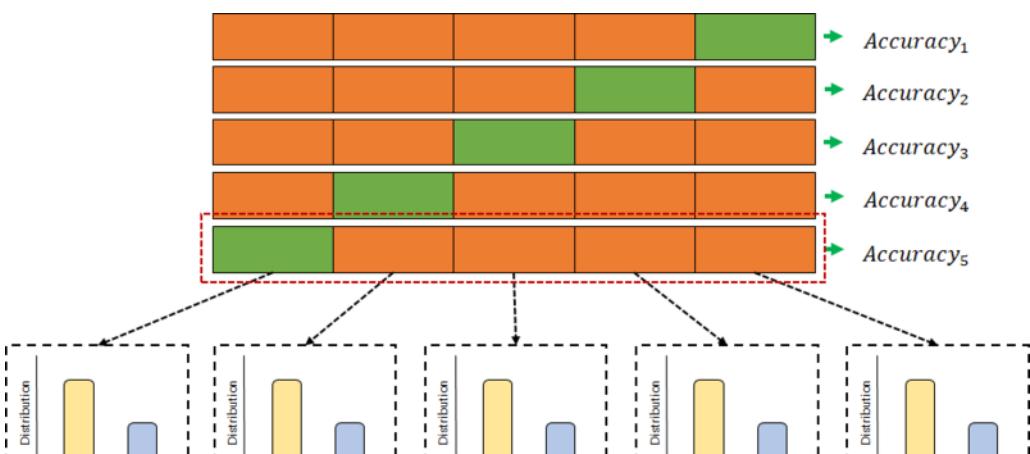
- LOOCO



$$Accuracy = \text{Average}(Accuracy_1, \dots, Accuracy_n), \quad \text{where } n = \# \text{ of data sample}$$

- Leave-one-out cross validation은 줄여서 LOOCV라고도 부르며, $k = n$ 인 k -fold CV와 같은 개념이다.
- 모델 검증에 하나의 데이터만을 사용하고 나머지를 모두 훈련하는 방식이므로 training set과 resampling subset이 거의 동일하고 예측력이 좋다.
- 데이터의 낭비를 최소화하여 예측력이 높다는 장점이 있지만 그만큼 비용과 시간이 훨씬 많이 소요된다.

- Stratified k-fold CV





$$Accuracy = \text{Average}(Accuracy_1, \dots, Accuracy_k)$$

- k-fold cv 에서는 train, validation 셋이 랜덤하게 나눠진다.
- 데이터가 불균형한 경우 train, validation set의 label 비율이 달라진다.
- 이러한 경우에 활용하는 것이 계층화된 k-fold 교차검정이다.

이를 활용하여 imbalance한 데이터를 label의 비율에 맞게 train, validation 셋을 나눌 수 있고 궁극적으로 안정성을 높일 수 있다.

참고 : <https://machinelearningmastery.com/k-fold-cross-validation/> <https://m.blog.naver.com/PostView.nhn?blogId=ckdgus1433&logNo=221599517834&proxyReferer=https%3A%2Fwww.google.com%2F>

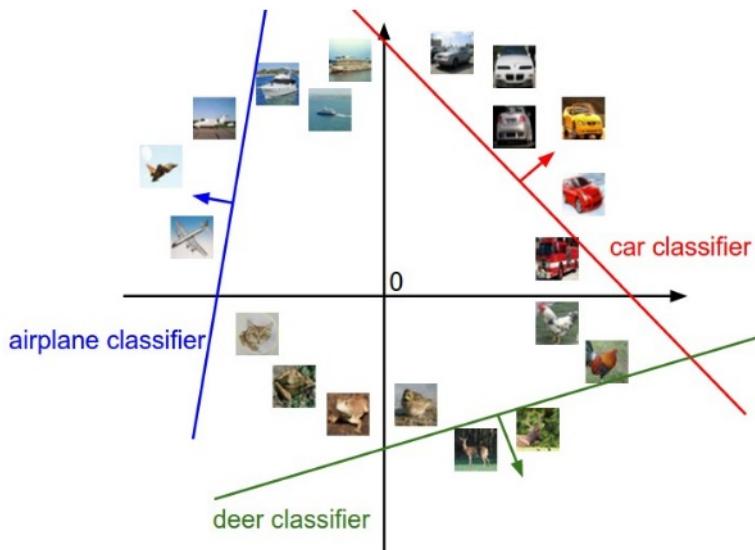
1.4 Linear Classifier

Linear Classification(선형 분류)

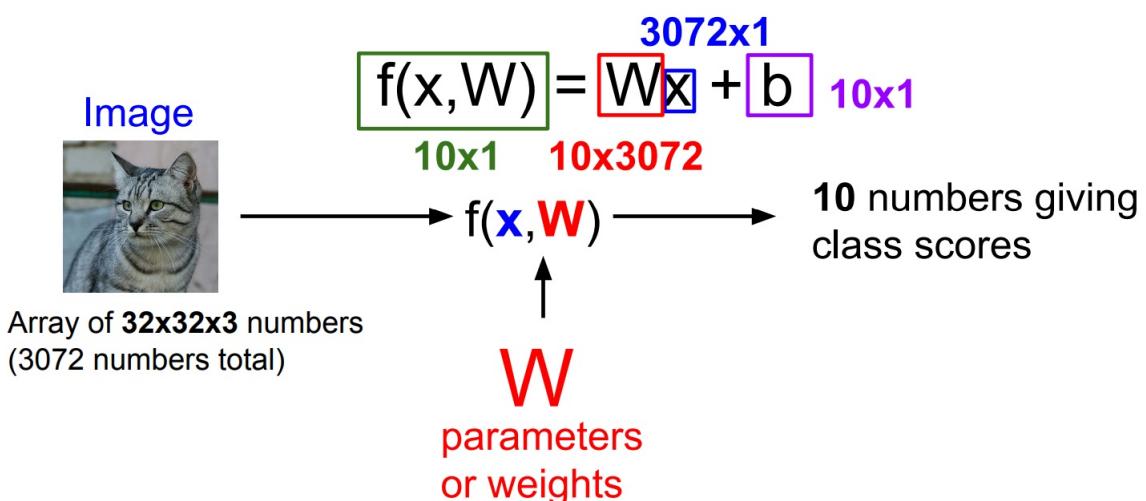
linear classification이란 쉽게 말해서 선을 이용하여 집단을 두개 이상으로 분류하는 모델이다.

선을 한 개 사용해서 두 개의 class로 분류 할 경우 binary classification이라고 한다. 하지만 실제 세상에서는 binary하게 분류하는 문제는 거의 없고 입력값에 대해 여러 class 중 하나를 택해 분류 하는 문제가 대다수다.

아래 그림은 3가지 클래스에 대해 linear classification한 예시이다.



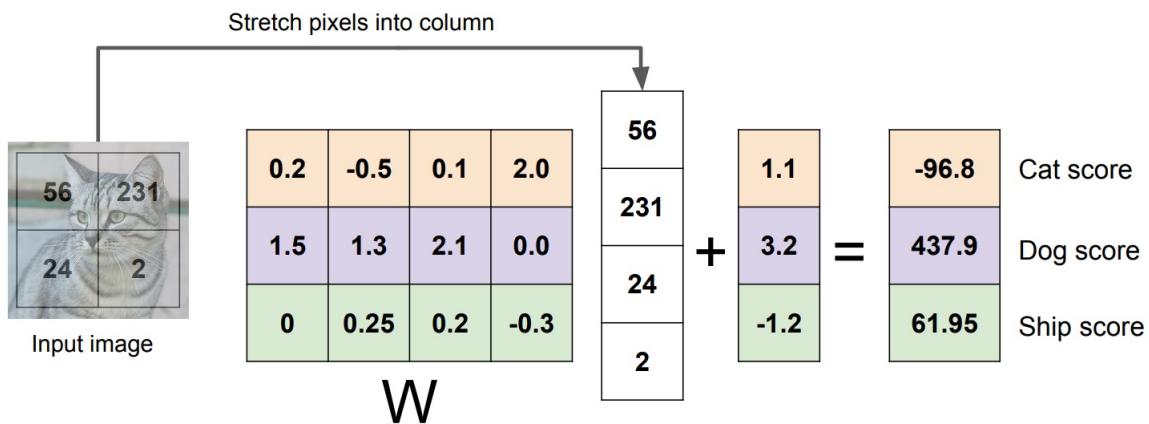
Linear Classifier(선형분류기)



그렇다면 Linear Classifier는 어떻게 작동할까? 간단한 Linear Classifier를 하나 생각해보자.

- CiFAR10의 경우 RGB 컬러 이미지의 차원은 $32 \times 32 \times 3$ (width x height x channel)이다. 이 3차원 shape를 가진 이미지를 1차원의 shape를 가진 모양으로 바꿔주면 3072차원의 벡터가 나온다.
- 3072차원의 벡터에 Linear Classifier의 Weight matrix와 matrix multiplication을 해주면 우리가 부르고자 하는 class 마크이 결과가

- 이미지를 Linear Classifier의 weight matrix와 matrix multiplication을 통해, 각각의 흐림한 클래스로 나온다. 여기서 class의 총 개수가 10개 이므로, Wx 의 결과값은 10개가 나온다.
- 여기에 bias 텀을 더해주게 되면, Classifier가 예측한 값이 나오는 것이다.



이처럼 이미지를 stretch(3차원 shape를 1차원 shape로)한 뒤, Weight matrix와 곱한 뒤 bias를 더해 나온 최종 score로 표현한다. 위의 경우, 점수가 가장 높은 것이 dog이니까, dog으로 예측했다고 보면 된다. 그 후 Classifier가 내놓은 결과 값에 대해 제대로 분류를 했나 평가를 하기 위해 정답 레이블과 비교한다.

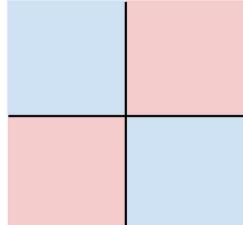
(이때 비교를 해주는 함수가 바로 뒷장에서 다룬 **Loss function**)

Linear Classifier의 문제점은?

Hard cases for a linear classifier

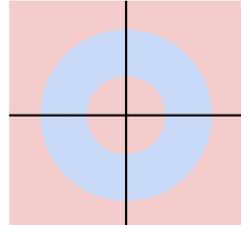
Class 1: number of pixels > 0 odd

Class 2: number of pixels > 0 even



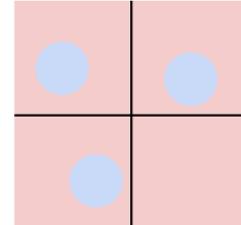
Class 1: $1 \leq L2 \text{ norm} \leq 2$

Class 2: Everything else



Class 1: Three modes

Class 2: Everything else



Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 2 - 59

April 6, 2017

선형분류기는 많은 경우에 이미지 분류를 잘 해내지 못하는데... 밑의 경우, 하나의 선으로 두 class를 분류하기는 어렵다.

- 사분면에서 opposite한 위치에 decision boundary가 있는 경우
- 원형의 decision boundary가 있는 경우
- 여러 개의 독립적인 decision boundary가 있는 경우

참고: <https://worthreading.tistory.com/44> , <https://worthreading.tistory.com/46?category=777078> , <https://chacha95.github.io/2018-11-15-Deeplearning1/>

2. Loss functions and optimization

앞의 1장에서 우리는 데이터에 기반한 접근을 하려고 노력했었고, 그에 따른 방법으로 kNN, Linear Classifier 등을 배웠다. 그리고 이러한 방법의 좋고 나쁨을 정량화할 수 있는 Loss function을 정의하려고 한다. 그 이후 Loss function을 최소화하는 방법을 통해서 파라미터를 효율적으로 찾는 방법을 알아볼 것이다.

2.1 SVM

먼저 Loss function중에서 SVM을 소개하려고 한다. SVM은 정답인 스코어와 오답인 스코어를 비교해서 얼마나 차이가 많이 나는지를 보는 방법이다. 차이가 크지 않은 경우는 loss를 0으로 취급하게 된다.

Suppose: 3 training examples, 3 classes.

With some W the scores $f(x, W) = Wx$ are:



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Losses:	2.9	0	12.9
	= max(0, 5.1 - 3.2 + 1) + max(0, -1.7 - 3.2 + 1) = max(0, 2.9) + max(0, -3.9) = 2.9 + 0 = 2.9	= max(0, 1.3 - 4.9 + 1) + max(0, 2.0 - 4.9 + 1) = max(0, -2.6) + max(0, -1.9) = 0 + 0 = 0	= max(0, 2.2 - (-3.1) + 1) + max(0, 2.5 - (-3.1) + 1) = max(0, 6.3) + max(0, 6.6) = 6.3 + 6.6 = 12.9

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

위의 예제는 Linear classifier를 통해서 score를 계산한 것이다. 계산을 하지 않고 숫자의 크기만을 보고 loss가 어떻게 될지 짐작해보면 (cat이 입력인 경우), car의 score가 cat보다 크기 때문에 ($5.1 > 3.2$) loss가 0보다 큰 값이 생길 것임을 알 수 있고, frog의 경우에는 -1.7이므로 cat의 score가 크기 때문에 loss는 0이 될 것이라는 것을 짐작할 수 있다. 즉 SVM은 score의 상대적인 크기 비교를 통해서 loss를 계산한다는 것을 간단하게 확인해볼 수 있다.

구체적인 수치를 통해서 한번 더 확인해보자. 고양이 사진을 입력으로 받았을 때의 스코어를 보면 각각 cat = 3.2, car = 5.1, frog = -1.7이 나온 것을 확인할 수 있다. 여기서 정답인 score는 cat = 3.2이다. 다른 오답인 score와 정답인 score의 차이에 1을 더한 값이 0보다 큰 경우에는 값을 그대로 linear하게 출력하고, 0보다 작은 경우는 loss는 그냥 0으로 출력하게 된다. 그래서 car와 cat을 비교한 값 ($5.1 - 3.2$)에 1을 더한 값이 2.9이므로 2.9를 loss로 출력한 것을 확인할 수 있다. 이렇게 각 입력값에 대해서 loss를 계산하고, 이를 모두 더한 값이 최종 losses가 된다.

여기서 만약 car의 score가 약간의 변동이 생긴다면 어떻게 될까?

정답은 "결과는 거의 변함이 없다"이다. car의 score의 경우를 자세히 살펴보면, 입력으로 받은 3개의 사진 모두에서 다른 클래스보다 높은 score를 보이고 있는 것을 알 수 있다. 그렇기 때문에 car의 score에 약간 변동이 생겨도 다른 클래스와의 상대적인 차이에는 거의 변함이 없을 것이기 때문에 loss가 0인 결과는 그대로 0이 나을 것이다. 여기서 SVM은 데이터의 변동에 둔감하다는 특성을 발견할 수 있다. score가 몇점인지 관심이 있는 것이 아니라 상대적으로 score가 높은 수치인지 아닌지에 중점을 두고 있기 때문이다.

그리고 만약 score값들이 0에 매우 가까워지면 loss는 어떻게 될까?

cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Losses:	2.9		

the SVM loss has the form:

$$\begin{aligned}
 L_i &= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \\
 &= \max(0, 5.1 - 3.2 + 1) \\
 &\quad + \max(0, -1.7 - 3.2 + 1) \\
 &= \max(0, 2.9) + \max(0, -3.9) \\
 &= 2.9 + 0 \\
 &= 2.9
 \end{aligned}$$

$$\max(0, 0-0+1) + \max(0, 0-0+1) = 2$$

$$\text{cat, car, frog} \rightarrow 2 + 2 + 2 / 3 = 2$$

$$\text{loss} = C - 1 \text{ (sanity check)}$$

위의 예제를 통해서 생각해보면, 클래스가 3개인 경우에 loss를 구하는 식은 $\max(0, 0 - 0 + 1) + \max(0, 0 - 0 + 1)$ 이 될 것으로 값은 2가 될 것이다. 클래스를 확장해서 생각해보면 개별 loss는 클래스에서 1을 뺀 값이 나오게 될 것임을 추측해볼 수 있다. 이를 이용해 본인이 코딩한 SVM loss function에, score를 0에 가까운 값을 대입하여 클래스 -1의 값이 나오는지 확인해서 올바르게 코딩했음을 확인하는 방법으로도 쓸 수 있다. 이를 sanity check라고도 부른다고 한다.

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Losses:	2.9		

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Before:

$$\begin{aligned}
 &= \max(0, 1.3 - 4.9 + 1) \\
 &\quad + \max(0, 2.0 - 4.9 + 1) \\
 &= \max(0, -2.6) + \max(0, -1.9) \\
 &= 0 + 0 \\
 &= 0
 \end{aligned}$$

With W twice as large:

$$\begin{aligned}
 &= \max(0, 2.6 - 9.8 + 1) \\
 &\quad + \max(0, 4.0 - 9.8 + 1) \\
 &= \max(0, -6.2) + \max(0, -4.8) \\
 &= 0 + 0 \\
 &= 0
 \end{aligned}$$

여기서 우리가 구하고 있는 W 에 대해서 조금 더 생각해보면, W 에 2배를 해준 값을 이용해서 SVM을 계산해보아도 기존의 W 를 가지고 구할 때와 같은 결과가 생기는 경우가 있음을 확인할 수 있다. 즉 이 W 는 유일하지 않은 값임을 알 수 있다.

이때까지는 사실 train set에 중점을 두고 W 를 구했다. 하지만 우리가 관심있는 것은 "예측"이기 때문에 test set에 조금 더 중점을 두고 싶다. 즉 train set에도 맞고 test에서도 맞는 W 를 구해야 한다. 그래서 여기서 Regularization이라는 개념이 도입된다.

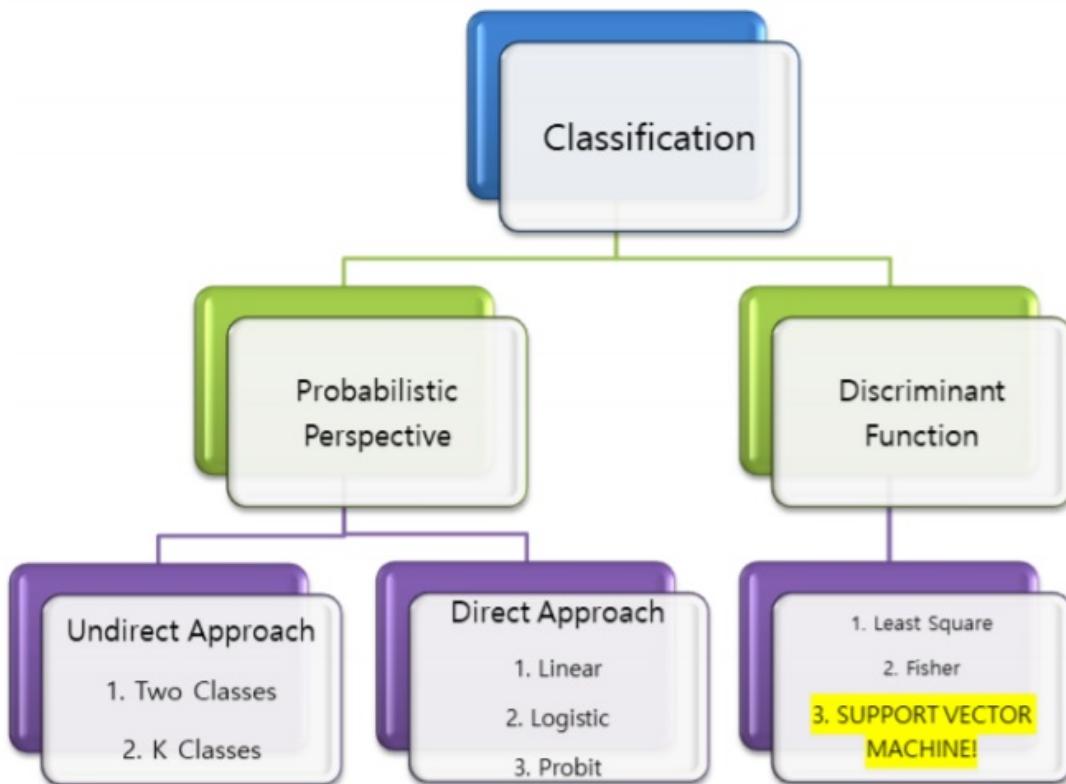
SVM 모델 관련 설명 (임선우)

SVM이 3강의 중요 주제가 아니고 COMPSCI 231N 수강생들한테는 선수 과목으로 보여 이 자체 대해서는 설명을 하지 않았다. 그러나 ESC 회원들이 2,3강 내용을 많이 접했을 것이라 생각하여 부가설명으로 SVM이 무엇인지 5~7분 설명 분량으로 작성하였다 (수식은 거의 제하였다).

Citing이 없는 부분 출처 : https://github.com/YonseiESC/ESC-20SPRING/blob/master/%EC%84%B8%EC%85%98%20%EC%9E%90%EB%A3%8C/week4/Week4_slides.pdf (본인 작성 2020 봄 ESC 4주차 발표자료)

2.1.a SVM은 데이터 내에 Discriminant Function을 그리는 방식

- SVM은 회귀모형으로도 쓰이나 주로 분류모형으로 쓰인다.



SVM : No consideration of $p(C_k|X)$

- 분류 모델로 ISL, ESL, 여름 ESC에서 배웠을 Logistic Regression, Softmax Regression, LDA, QDA 등과 SVM은 결정적인 차이가 있다.
- 위 이미지에서의 **probabilistic perspective**은 $P(C_k|X)$, 즉 데이터가 주어졌을 때 어떤 class에 속하는지에 관한 확률 모델을 제공하며 실제 분류하는 것은 연구자의 Decision criteria에 근거하여 할 것이다.
 - 가장 확률값이 큰 클래스로 결정을 내리거나 배경지식에 따라 특정 클래스에 관한 오분류를 심각하게 생각한다면 **argmax**가 아닌 기준으로 삼을 수도 있다.
- 허나, SVM은 확률 모델을 정립하지 않고 바로 한 클래스로 배정한다. (물론 여기에서도 hyperparameter가 있어 이에 따라 어떤 점이 다른 클래스에 배정될 수도 있지만)

2.1.b Hard Margin SVM의 기본 아이디어

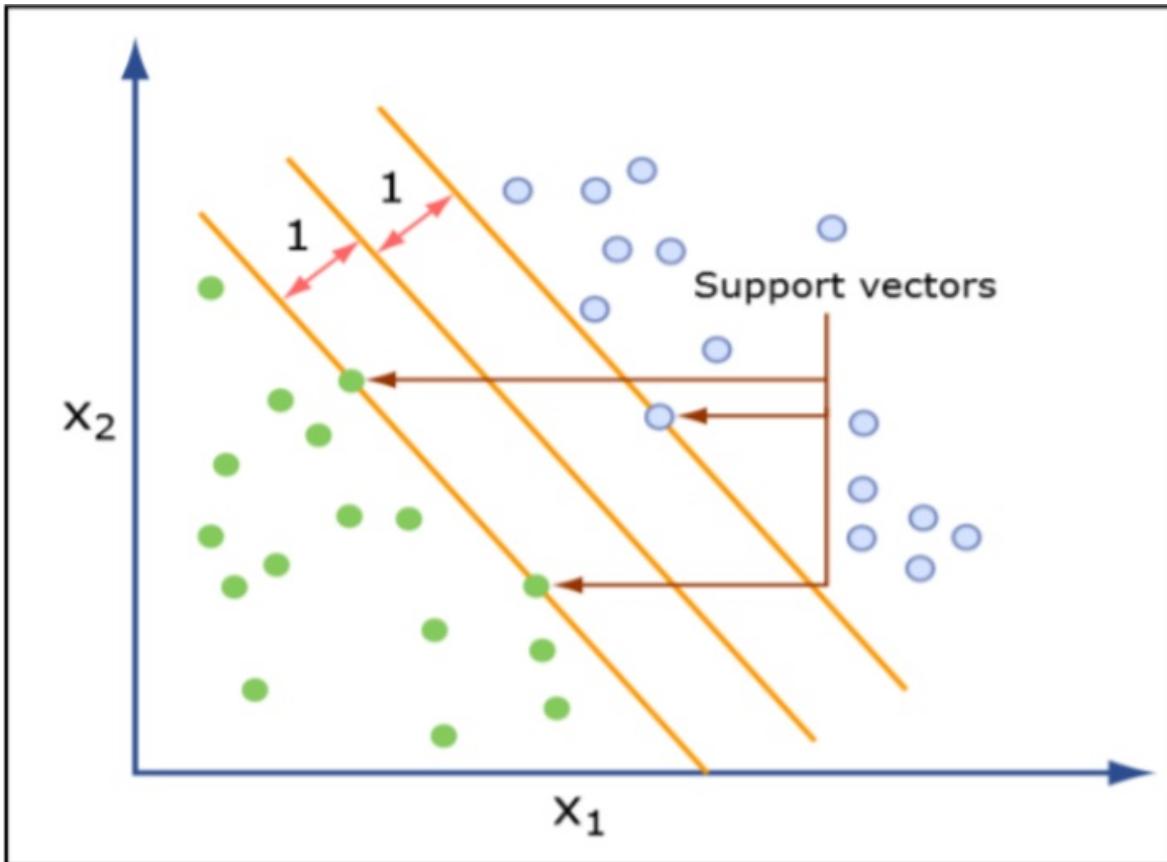


Image by MIT OpenCourseWare.

https://ocw.mit.edu/courses/sloan-school-of-management/15-097-prediction-machine-learning-and-statistics-spring-2012/lecture-notes/MIT15_097S12_lec12.pdf p8

먼저 Hard Margin (훈련 데이터에서 error가 0) SVM을 알아보자. 기본 아이디어는 초록색 class, 파란색 class 중간에 고속도로를 끓는 것이다!

양쪽 class를 전쟁의 진영이라고 생각해보자.

이 때, 각 class의 최전선에 있는 data들이 있을 것이다.

그 최전선 벡터(즉 data point)를 기준으로 정가운데에 hyperplane을 세우면 그것이 곧 support vector machine이라 하는 것이다.

2.1.c 용어설명

"Good Linear Discriminant Function"을 알아보자!

"Target" $t (= y)$: $t_n = 1$ or -1 : labeled!

"Estimator" of t : $y(x) = w^T x + b$

"Class1" $y(x) = w^T x + b > 0$: 한 쪽 면

"Class2" $y(x) = w^T x + b < 0$: 다른 한쪽 면

"Discriminant Function" : $y(x) = w^T x + b = 0$: 판별함수

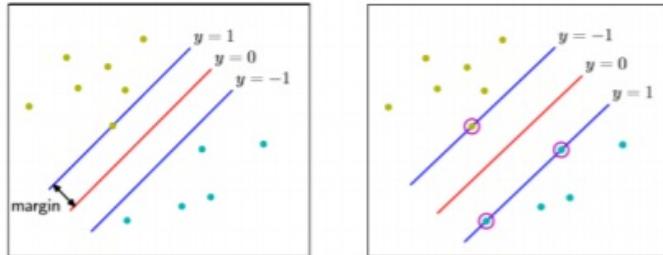
1) Target (y 변수)는 $-1, 1$ 로 코딩한다. 초록색 점은 $t_n = -1$, 파란색 점은 $t_n = 1$ 이라 하겠다. 이는 정답으로 아래의 **estimate / estimator**와 구별해야 한다.

2) Estimator는 $y(x) = w^T x + b$ 를 사용한다. 이 때, 정답지는 1 또는 -1 인데 반해 estimator는 위 그림과 같이 1 을 초과하거나 -1 미만일 수도 있다. **Estimated value**와 **Decision**의 차이로 보면 될 것이다.

3) 중간 노란색 선을 **Discriminant Function**이라 한다. test data에서는 노란색 선들 중간에 얼마든지 label (정답)을 모르는 데이터가 올 수 있다. 그러므로 test data에서는 $y(x) > 0$ 이면 1번 클래스, $y(x) < 0$ 이면 2번 클래스라고 부른다.

Support Vector Definition

- Margin:



- Perpendicular distance between boundary and the closest data point (left)
- Maximizing margin leads to a particular choice of decision boundary
- Determined by a subset of the data points
 - Which are known as *support vectors* (circles)

<https://cedar.buffalo.edu/~srihari/CSE574/Chap7/7.1-SVMs.pdf> p11

4) Margin : Discriminant Function과 각 class의 최전선 간의 수직 거리 (= 최소거리). hyperplane과 각 class의 support vector들 간 거리는 hyper plane과 파란색 선들간의 거리와 동일하다.

5) Support vector : 최전선에 놓여 있는 벡터!

2.1.d 어떻게 Support vector machine을 세워 놓을까?

즉, 어떻게 Discriminant Function 을 세우면 평면과 Support Vector 의 거리를 최대화할 수 있는지 알아야 한다!

spoiler: 평면과 각 클래스의 최전선에 있는 벡터 간의 거리를 최대화하도록 한다!

누가 support vector machine 이 무엇이냐고 돌발 질문을 던진다면 **spoiler**라고 쓰인 문구를 먼저 생각하고 점점 구체화하면 무리 없을 것이라 생각한다.

지금 구한 이 거리에는 절대값이 분자에 있다! (미분불가능한 문제) 절대값을 없애야
미분가능한 Optimization Problem 이 된다!

해결법 : multiply by $t_n \in (-1, 1)$

- $t_n = 1$ (adult, conservative,..) : Want $w^T X_n + b \geq 1$
- $t_n = -1$ (children, democratic, ..) : Want $w^T X_n + b \leq -1$
- $t_n(w^T(x_n) + b) \geq 1, \forall n \in (1, 2, \dots, N)$
- $\therefore \text{Margin} = \min \frac{t_n(w^T(x_n) + b)}{\|w\|}$
- So, Maximum Margin Solution : $\text{argmax}_{w,b} \min \frac{t_n(w^T(x_n) + b)}{\|w\|}$

다른 줄들은 시간관계상 생략하고 4째 줄 margin의 식과 5째 줄만 보자.

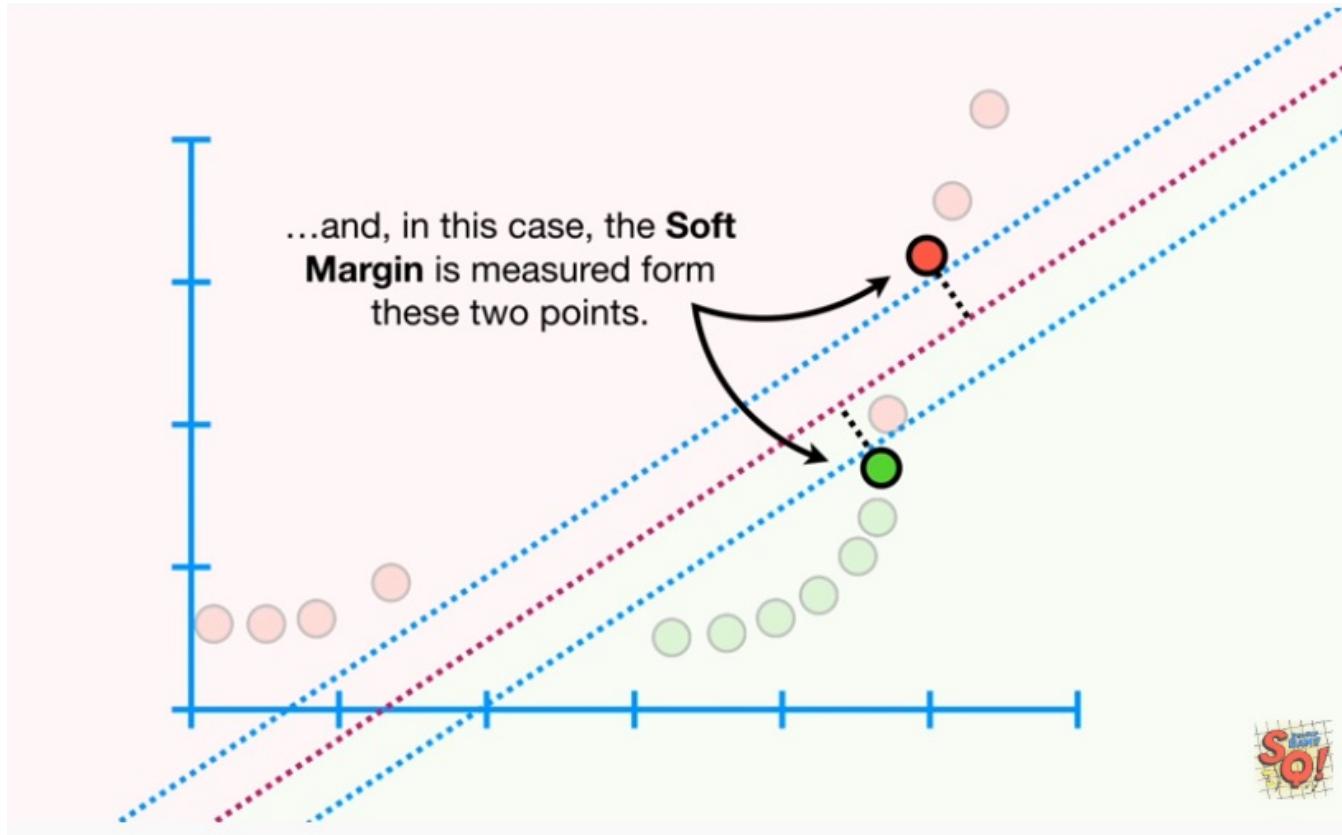
4째 줄 $Margin = \min\left(\frac{t_n(w^T(x_n) + b)}{\|w\|}\right)$ 에서 **min** 뒤에 있는 식은 **hyperplane**과 데이터 **vector**들 간의 거리이다.

이를 **최소화** : 최전선 점까지의 거리

5째 줄은 그 거리를 **최대화**하는 것이 최적의 support vector machine classifier라는 뜻이다.

2.1.e Soft margin vs Hard margin

hard margin으로 하면 margin이 너무 좁아 유용하지 않다. 그러므로 train error를 어느 정도 허용하는 soft margin classifier도 생각할 수 있다.



images : <https://www.youtube.com/watch?v=efR1C6CvhmE>

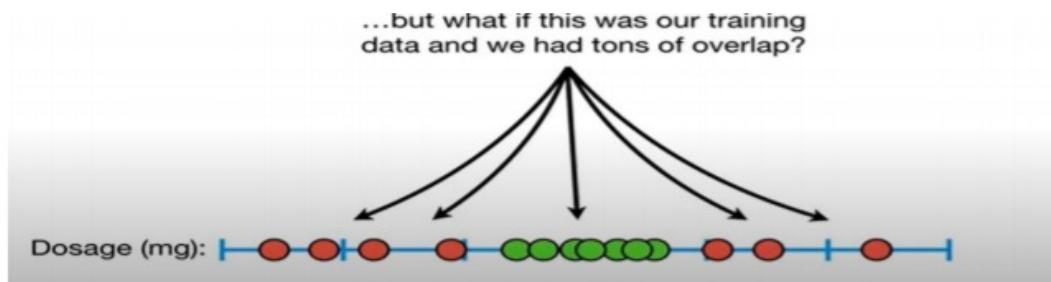
- **Soft Margin** : Distance between the Discriminant Function and the Frontier observation, Allowing Misclassification IN SAMPLE.
- 적절한 Soft Margin 은 **Validation** 을 통해 정한다! (Optimal Problem 0| X)

Hard Margin vs Soft Margin (Bias Variance Tradeoff 시/소)

- **Hard Margin** : Bigger Variance, Smaller Bias
- **Soft Margin** : Bigger Bias, Smaller Variance. (Bias 를 허용하되 Variance 를 줄이겠다)
- **Soft Margin** 은 **Ridge, Lasso** 와 분야는 다르나 참 비슷한 게 많다! (아이디어 뿐 아니라 나중에 Error 의 Penalty Term 도!)

hard margin vs soft margin에서 bias variance tradeoff가 있다.

2.1.f Kernel Method

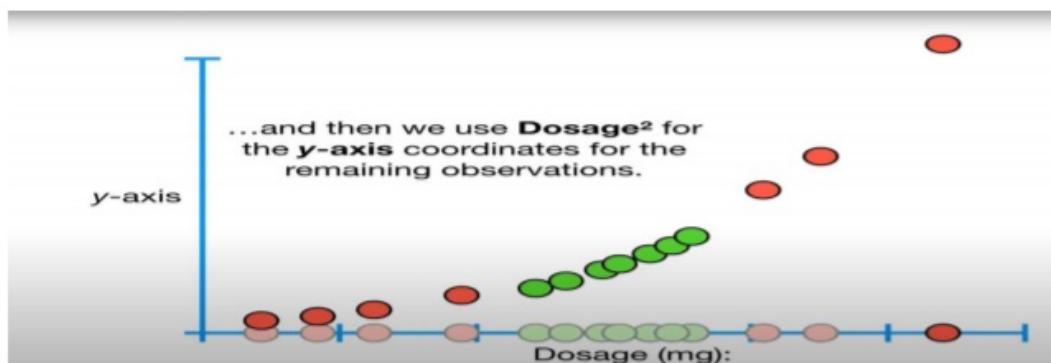


4-1. Problems. 어떡해?

후보 1) 절반정도의 misclassification 을 발생시킨다

후보 2) 이건 Support Vector Machine 이 아니야라고 포기한다

4-2. 해결방안 : SVM 은 이것도 선형으로 분리할 수 있어 (...) ?!?!?



- 해결방안 : Input Space에서 차원을 확장한 Feature Space에서 Linearly Separable Hyperplane !!
- 주의점 : 우리가 Client 한테 보고할 때는 고차원(Augmented Space)에서 나눈 경계를 저차원에 환원시켜 보고한다. 이 그림에서는 Curvature 경계가 된다.

images source : <https://www.youtube.com/watch?v=efR1C6CvhmE>

2.1.g. Hinge Loss vs Logistic Regression의 Cross Entropy Loss의 비교

Comparison of SVM and LR cost functions

SVM

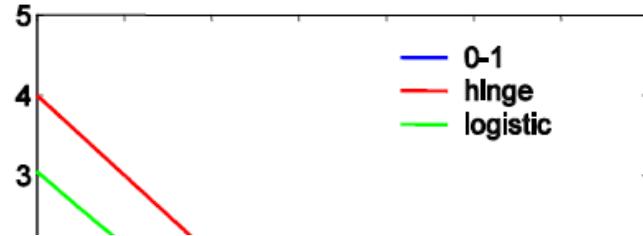
$$\min_{\mathbf{w} \in \mathbb{R}^d} C \sum_i^N \max(0, 1 - y_i f(\mathbf{x}_i)) + \|\mathbf{w}\|^2$$

Logistic regression:

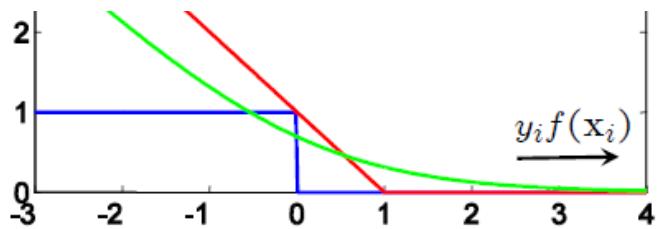
$$\min_{\mathbf{w} \in \mathbb{R}^d} \sum_i^N \log(1 + e^{-y_i f(\mathbf{x}_i)}) + \lambda \|\mathbf{w}\|^2$$

Note:

- both approximate 0-1 loss



- very similar asymptotic behaviour
- main difference is smoothness, and non-zero values outside margin
- SVM gives **sparse** solution for α_i



<http://www.robots.ox.ac.uk/~az/lectures/ml/2011/lect4.pdf> p15

0-1 loss, hinge loss, logistic loss를 살펴본다.

1) 먼저 0-1 loss를 기본적으로 생각할 수 있겠다.

- 그러나 미분불가능, 게다가 불연속 (파란 선은 연속으로 수직 선 이어놨지만 사실 indicator function은 불연속함수)
- 고로, 수학적으로 좋은 특징을 가진 hinge loss, cross entropy loss를 많이 사용한다.

hinge loss, logistic regression loss의 식에서 "+"를 기준으로 왼쪽은 training error, 오른쪽은 penalty term이다. 이들 loss function들은 0 - 1 loss와 유사한 형태를 지닌다

2) SVM에서의 C는 training data에서의 오분류로 인한 penalty에 대해 신경을 얼마나 쓰는지를 나타낸다.

- $\uparrow C : \sum \max(0, 1 - y_i f(x_i))$ 를 크게 만들어야 전체 식 최소화 가능 : Hard Margin Classifier에 가깝다 \leftrightarrow Big variance Small Bias
- $\downarrow C : ||w||^2$ 를 크게 만들어야 전체 식 최소화 가능 : Training error는 조금 생겨되 괜찮다 \leftrightarrow Soft Margin Classifier \leftrightarrow Big Bias Small Variance
 - 식을 보면 0과 linear function의 maximum function으로 이는 위 그림에서의 빨간색 piecewise linear 선으로 나타난다.

3) Logistic Regression에서의 hyperparameter λ 는 penalty term쪽에 붙었다.

- 훈련데이터에서 정분류 $\leftrightarrow y_i$ 와 $f(x_i)$ 의 부호가 같아 $\leftrightarrow -y_i f(x_i) < 0 \leftrightarrow \log(1 + e^{-y_i f(x_i)}) \approx 0$
- 훈련데이터에서 오분류 $\leftrightarrow y_i$ 와 $f(x_i)$ 의 부호가 달라 $\leftrightarrow -y_i f(x_i) > 0 \leftrightarrow \log(1 + e^{-y_i f(x_i)})$ big

2.1.h 각각의 특징을 통해 무엇을 쓸지 정하기

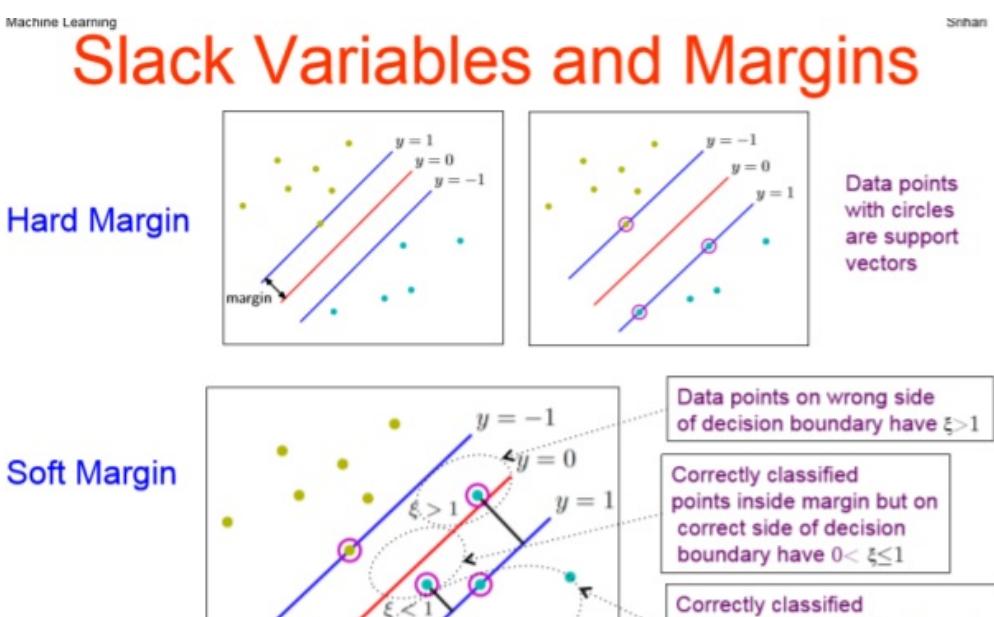
http://www.cs.toronto.edu/~kswersky/wp-content/uploads/svm_vs_lr.pdf p23

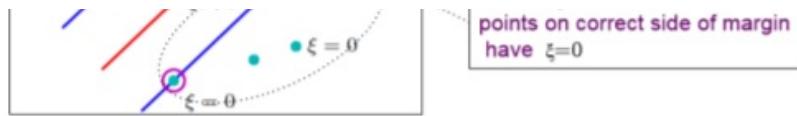
Logistic Regression :

- 해석력 : Odds ratio (조건부 성공확률, 조건부 실패확률의 ratio)
- 해당하는 Bayesian method 존재

SVM :

- SVM 모델을 훈련시켰다. 만약 Support Vector들 내부 (최전선 지키는 support vector들 안쪽의 안정 영역)에 test data가 오면 Error가 없다.





<https://cedar.buffalo.edu/~srihari/CSE574/Chap7/7.2-SVM-Overlap.pdf>

- SVM은 Sparse Kernel 방식이라 불리는데, 이는 분류모델을 세울 때 모든 점이 고려되기는 하지만 (minimum 연산은 거쳐야 하니) 결국에 **Discriminant Function** 식으로 들어가는 것은 support vector들뿐으로 **sparse**
- 고로, 훈련 데이터 sampling에 따라 모델이 적게 변하므로 모델 분산이 비교적 적음
- Dual (쌍대) 문제, kernel trick으로 고차원 model space M 에서도 분류 : 어려운 쌍대이론, kernel을 공부하면 이해가 증진될 것이라 생각.

2.1.i Multiclass SVM

설명이 매우 훌륭한 <https://cedar.buffalo.edu/~srihari/CSE574/Chap7/7.3-SVM-Multiclass.pdf> 참조하면 좋을 것이라 생각.

- One Versus All (OVA) 랑 All Versus All (AVA)로 나뉜다.
- OVA는 가령 한 vs 중 vs 일 분류문제가 있다고 하면 한 vs 중일처럼 먼저 one vs rest로 나누는 방식 (Logistic Regression에서의 OVA는 20년 여름 첫 세션 발표자료 36p에 나와 있기도 하다)

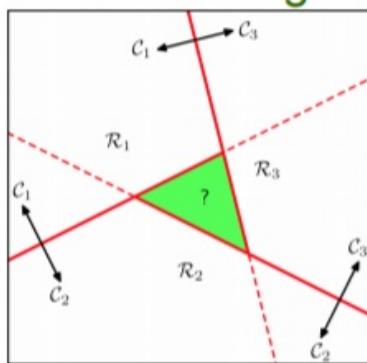
https://github.com/YonseiESC/ESC-20SUMMER/blob/master/DL_SESSION/week1/%5BDL%5DWeek1%20session.pdf

K

- AVA는 K 개 Class가 있다면 $(^2)$ 개의 2 class SVM을 훈련시키고 vote를 붙이는 방식이다.

Machine Learning Srihari Multiclass SVMs: All-versus-All (AVA)

- Train $K(K-1)/2$ different 2-class SVMs on all possible pairs of classes
 - Classify test points according to which class has highest number of votes
 - Again leads to ambiguities in classification



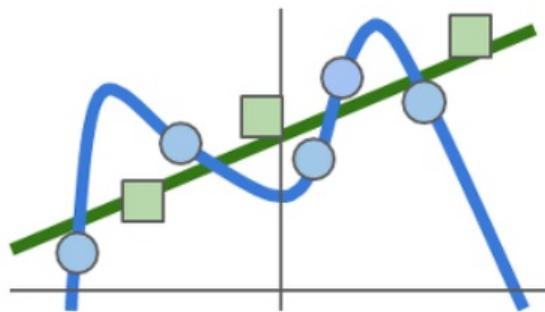
현재까지 SVM의 기본적 내용을 많이 압축하였다.

- 이번 post에서 많이 생략된 수식 전개 방식은 2020 봄 ESC 4주차 발표자료 https://github.com/YonseiESC/ESC-20SPRING/blob/master/%EC%84%B8%EC%85%98%20%EC%9E%90%EB%A3%8C/week4/Week4_slides.pdf를 참고하면 좋을 것이라 생각한다.
- 만약 하나의 통일된 자료를 보고 싶다면 <https://cedar.buffalo.edu/~srihari/CSE574/Chap7/7.1-SVMs.pdf>로 공부하는 것이 최고라고 개인적으로 생각한다. 6장은 kernel method, 7.1은 SVM 전반적 내용, 7.2는 soft margin, 7.3은 multiclass SVM을 다룬다.

2.2 Regularization

$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)$$

Data loss: Model predictions should match training data



여기서 파란 동그라미가 train set이고, 초록 네모가 test set이다. 위처럼 train set만을 생각하고 W 를 구하게 되면, 파란선과 같은 경우를 생각해볼 수 있다. 여기서 test set을 입력값으로 받는 경우에는 오차가 크게 발생할 것이라는 것을 알 수 있다. train set에 맞는 W 는 여러 개 존재할 수 있다는 것을 알고 있기 때문에, 그들 중에서 test set에도 잘 맞는 W 를 찾는 것이 Regularization의 목적이라고 볼 수 있다. 이를 위해서 우리는 Loss term에 페널티항을 부과하게 된다.

$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i) + \lambda R(W)$$

Data loss: Model predictions should match training data

Regularization: Model should be “simple”, so it works on test data



Occam's Razor:
“Among competing hypotheses, the simplest is the best”
 William of Ockham, 1285 - 1347

여러 가정중에서 가장 단순한 것이 좋다는 Occam's Razor의 아이디어로 만들어진 페널티항은 람다값이 커질수록 강하게 규제해서 함수가 단순한 식에 가까워지게 만들고, 작아질수록 약하게 규제해서 본래의 복잡한 식에 가까워지도록 만들어지도록 설계되었다.

Regularization

λ = regularization strength
 (hyperparameter)

$$L = \frac{1}{N} \sum_{i=1}^N \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

In common use:

L2 regularization $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 regularization $E1$ $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

Max norm regularization (might see later)

Dropout (will see later)

Fancier: Batch normalization, stochastic depth

Regularization을 하는 방법은 여러가지가 있지만, 페널티항을 부과하는 방법으로써 대표적으로 L2, L1, Elastic net 등이 있다. 여기서 L2와 L1에 대해서 간단히 언급을 하고 넘어가려고 한다. 이 두가지는 모델의 복잡도를 판단하는 기준이 다르다.

L2 regularization이 제일 일반적으로 많이 쓰이는 방법이라고 한다. 이는 norm을 고려하는 방법이기 때문에 벡터의 성분 전체를 고려해서 규제를 한다는 특징이 있다. 즉 X 의 모든 요소가 영향을 주는 것을 선호하는 것이다. L1 regularization은 0의 개수에 따라서 모델의 복잡도를 다루기 때문에 전체 차수의 값을 0에 가깝게 유도하는 특징이 있다. 그래서 요소가 0이 많은 것을 더 선호하는 경향이 있다고 한다.

2.3 Crossentropy loss(softmax)

SVM 말고도 대표적으로 많이 사용하는 Loss function이 바로 Softmax이다. 이를 설명하기 위해서 다시 위의 고양이를 입력으로 받은 linear classifier의 예제를 가져왔다.

Softmax Classifier (Multinomial Logistic Regression)



scores = unnormalized log probabilities of the classes.

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{where} \quad s = f(x_i; W)$$

Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

$$L_i = -\log P(Y = y_i | X = x_i)$$

cat **3.2**

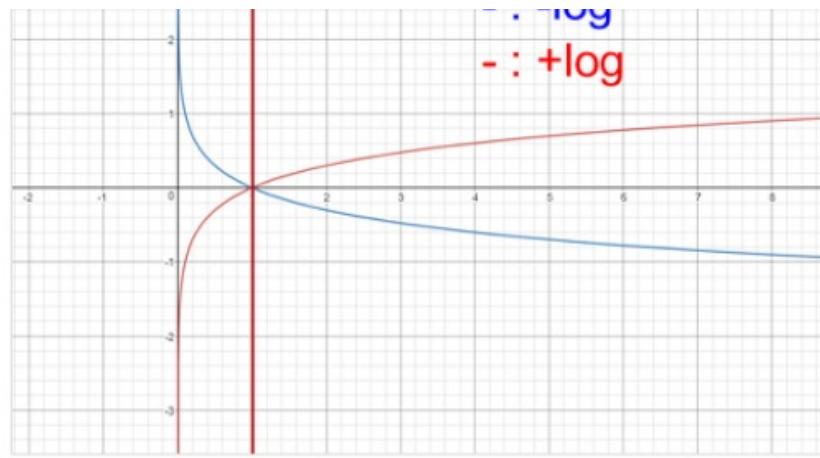
car **5.1**

frog **-1.7**

in summary: $L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$

위 식을 보면, 모든 스코어를 \exp 을 취한 뒤 각각 전체합으로 나눠주는 것으로 구할 수 있다. 이렇게 만들면 결과값으로 확률값을 얻을 수 있다. 그 뒤에 나오는 확률값을 $-\log$ 취해주는 것으로 loss값으로 만들어준다.





왜 $-\log$ 를 이용하는지 쉽게 알아보기 위해서 그림을 가져왔다. 우리는 확률값을 대입할 것이기 때문에 0과 1사이에서 y 값이 어떻게 나오는지 확인하면 된다. $-\log$ 의 경우에는 0과 1 사이의 값에서 반비례하는 모양으로 그려지는 것을 확인할 수 있다. score가 높은 값을 대입하게 되면 y 값이 작은 값이 나오게 될 것이고, 우리는 이를 loss로 취급하게 되는 것이다.

Softmax Classifier (Multinomial Logistic Regression)



$$L_i = -\log\left(\frac{e^{s_i}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

cat

3.2

exp

24.5

car

5.1

164.0

0.18

frog

-1.7

unnormalized log probabilities

0.13	→	$L_i = -\log(0.13) = 0.89$
0.87		
0.00		

probabilities

구체적인 예제를 가져와서 계산과정을 한번 살펴보았다. 이처럼 softmax는 score값이 얼마나 나오느냐에 따라서 SVM과는 달리 바로 loss에 영향을 미치는 것을 확인할 수 있다. 즉 데이터에 민감하다는 사실을 알 수 있다.

그리고 score를 0에 가깝게 만들면 $\log(\text{클래스개수})$ 가 되는 것을 확인할 수 있다. 여기서도 SVM처럼 디버그 용도로 이 방법을 사용할 수 있다.

이렇게 loss를 계산하는 방법을 통해서 W 가 좋은지 안 좋은지를 판단할 수 있게 되었다. 이제는 좋은 W 를 그림 어떻게 찾아갈 것인가 하는 문제가 남았다.

2.4 Optimization

이제는 좋은 W 를 어떻게 찾아갈지에 대한 방법론에 대한 이야기를 할 차례이다. 여기서는 경사를 따라서 내려가는 gradient descent를 설명하려고 한다.

Strategy #2: Follow the slope

In 1-dimension, the derivative of a function:

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

In multiple dimensions, the **gradient** is the vector of (partial derivatives) along each dimension

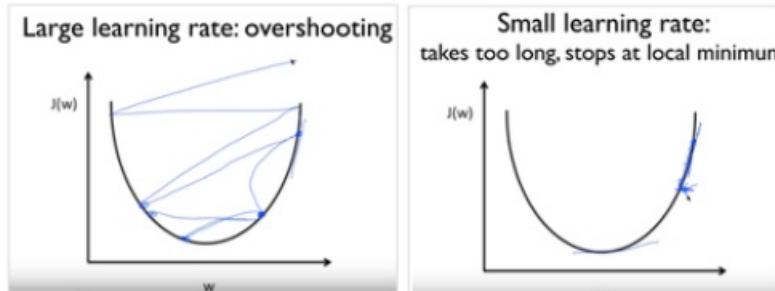
The slope in any direction is the **dot product** of the direction with the gradient
The direction of steepest descent is the **negative gradient**

1차원인 경우에는 위와 같은 미분으로 계산하게 된다. 다차원인 경우에는 편미분된 벡터의 합으로 나오게 된다. 다차원인 경우에는 성분 각각에 대해서 numerical하게 일일히 계산할 수도 있지만, 입력값이 엄청 큰 경우에는 일일히 계산하는 것이 오래 걸릴 수 있다. 그래서 우리가 관심 있는 것은 loss값이므로 analytic gradient를 구하는 것으로 많이 이용한다.(이에 관해서는 다음 강의에서 더 자세하게 나온다고 함) analytic gradient를 구하는 것이 유용하기 하나 예러가 많이 나올 가능성이 있다. 그래서 추가로 보통 numerical gradient로 디버깅 및 점검을 하는 방법이 있다고 한다. 이를 gradient check라고 보통 부른다.

Gradient Descent

```
# Vanilla Gradient Descent

while True:
    weights_grad = evaluate_gradient(loss_fun, data, weights)
    weights += - step_size * weights_grad # perform parameter update
```



적절한 step size를 정하고 이를 gradient에 곱해준 값을 원래의 W 에 빼줘서 업데이트를 시킨다. 여기서 step size는 hyper parameter이다. 시행착오로 적절한 값을 찾아줘야 한다. 하지만 여기서도 전체 N 을 사용하여 W 를 업데이트 하는 것이기 때문에 학습 속도가 느린 편이다. 그래서 여기서 또 발전한 방법이 Stochastic Gradient Descent이다.

Stochastic Gradient Descent (SGD)

$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(x_i, y_i, W) + \lambda R(W)$$

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^N \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

Full sum expensive
when N is large!

Approximate sum using a **minibatch** of examples
32 / 64 / 128 common

```
# Vanilla Minibatch Gradient Descent
```

```
while True:
    data_batch = sample_training_data(data, 256) # sample 256 examples
    weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
    weights += - step_size * weights_grad # perform parameter update
```

이는 전체 데이터셋을 사용하지 않고 나누어서 사용하는 방법이다. 보통 2의 승수로 많이 나눠서 사용한다고 한다. 256개로 나눈다고 가정했을 때, 256개의 데이터셋만 가지고 W 를 업데이트하고 이후 또 다음 256개의 데이터셋으로 W 를 업데이트 하는 방식으로 이루어진다.

이외에도 많은 학습방법이 있지만 (Adam, RMSprop 등) 다른 방법은 다음 강의에서 더 자세히 다룰 예정이라고 한다.