Lecture2) Image Classification

Image Classification

- input image → assign category, label
- 이미지는 숫자로 나타난다.
  - 가로 * 세로 * 채널(RGB, 3)
- 문제
  - semantic gap:
    이미지는 0~255 사이의 숫자로 나타내는데, 픽셀값과 semantic idea(사진의 의미)의 차이가 있음
  - Viewpoint variation: 똑같은 개체를 어디서 찍느냐에 따라 픽셀값이 크게 변동
  - Illumination: 빛의 양
  - Deformation: 생물의 경우, 움직이며 형태가 변할 수 있다.
  - Occulusion: 다른 물체에 의해 일부가 가려지는 경우
  - Background Clutter: 개체의 색이나 무늬가 주변과 유사한 경우
  - Intraclass variation: 하나의 종은 다양한 형태(색, 무늬)로 나타날 수 있다.
- The solution: “find edge, corners, boundaries → find cat”
  - ⇒ 매 class 마다 동일한 작업을 해야하므로 좋지 않음
- Data driven approach ⇒ ML approach

  1. Collect a dataset of images and labels
  2. Use Machine Learning to train a classifier
  3. Evaluate the classifier on new images

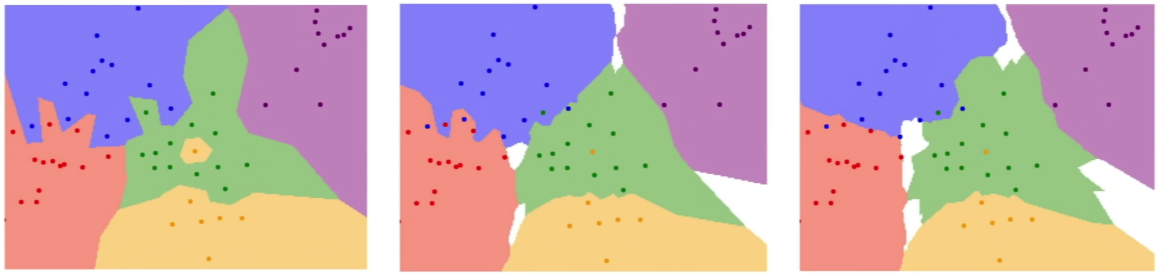
  - 일일히 cat과 dog를 구별하는 방법에 대한 로직을 짜는 것이 아닌, 데이터를 ML classifier에 넣어서 학습시키는 방법
  - Two Tasks
    - Train
      - input images, labels
    - Predict
      - make predictions for images

Nearest Neighbor

- 절차
  - 모든 학습 데이터를 기억하게 한다.
  - 학습된 데이터들 중, 가장 유사한 이미지의 레이블을 따라가도록 함
- Metric
  - 두 이미지의 픽셀 값의 차이를 계산하기 위해 metric을 선정해야한다.
  - 어떤 metric을 사용하느냐에 따라, 다르게 분류된다.
  - Distance Metric
    - L1 distance(= Manhattan distance): 두 이미지간의 픽셀값의 차이를 더한다.
- 속도
  - Train: O(1)

- Test: O(N)
  - 하나의 test image의 클래스를 찾기 위해, 모든 train dataset과의 거리 비교를 수행해야한다.
- 실제로 사용하기 위해서는, “학습속도가 느려도, 테스트 속도가 빠른게” 좋다.

- K-Neareset Neighbors
  - 가장 가까운 K개 포인트가 어떤 label에 속해있느냐에 따라, 다수결로 test sample의 label을 결정
  - **Majority vote**



K = 1 K = 3 K = 5

  - K가 클수록, decision boundary를 smooth해진다.
  - 정확성이 높지 않다.
  - Distance Metric
    - L1 distance(Manhattan): $|x_1 - x_2|$
    - L2 distance(Euclidean): $\sqrt{(x_1 - x_2)^2}$

    → 둘은 기하학적, 위상학적으로 다르다. 어떤 데이터에든 사용하기 좋다. 이미지에 사용하는 경우,
    두 이미지에 대한 각각의 픽셀값의 거리를 더해준다. 픽셀값이 유사할수록, 거리가 짧아진다.
  - L1, L2 dist 특징
    - L1 dist: 좌표계의 형태에 따라 좌표계의 방향을 따라 분포하는 성질이 있다.
    - L2 dist: 좌표계의 방향에 상관없이, 자연스러운 테두리를 보여준다.
  - Hyperparameters
    - KNN의 경우, K, distance metric 등이 hyperparameter에 해당한다.
    - 학습을 통해 배우지 않아도 되는 값들, 어떻게 알고리즘을 짜느냐에 따라 사람이 선택하는 값이다.
    - hyperparameter는 데이터를 통해 배울 수 없으며, 실험적으로 최적의 값을 구해야한다.
      문제와 데이터에 따라, 최적 hyperparameter가 다르다.
  - L1 distance의 결과가 L2 distance의 결과보다 좋은 경우
    - L1 dist가 좌표계에 의존적이므로, 데이터가 좌표계에 의존성이 있다면 L1 dist가 더 좋을 수 있다.
    - 각 축에 대한 elements에 의미가 있는 경우를 예시로 들 수 있다.
    - 예시) 문제: employee 분류 / 축: 연봉, 시간 등

Setting Hyperparameters

- Hyperparameter를 찾아내는 방법? → K를 번갈아가면서 실험해보기
- 방법1) 모든 데이터를 학습데이터로 사용하기
  - 모든 데이터셋을 학습과정에 쓴다면, 학습데이터에 대한 분류 정확도는 100% 일 것이다.
    그러나 학습데이터 외의 데이터가 들어왔을 때, 에러율이 높을 것이다.
  - train 데이터셋에 대해 오버피팅된다.

    - 머신러닝에서 중요한 것은 학습데이터를 얼마나 잘 분류해내는지가 아닌,
      “학습 후, unseen data(학습에 사용되지 않은 데이터)를 얼마나 잘 분류해 내는가” 이다.
- 방법2) 데이터를 train과 test 데이터셋으로 나누고, train 데이터셋만 학습시키기
    - 데이터를 train과 test 데이터셋으로 분류하고, 여러 K 값을 넣어서 train data를 학습시켜본 후,
      test 데이터셋에서 가장 좋은 성능을 내는 K를 hyperparameter로 선정
    - 그러나 이 방법도 여전히 test 데이터셋에 대해 오버피팅될 가능성이 있다.
      완전한 raw data, unseen data가 들어오면 에러율이 높을 것이다.
- 방법3) 데이터를 train, validation, test 데이터셋으로 나누고, train 데이터셋만 학습시키기
    - K를 번갈아가며 train 데이터셋을 학습시킨다. validation 데이터셋으로 각 모델의 정확도를 구하고, 가장 좋은 validation 정확도를 가지는 모델을 test 데이터셋에 대해 돌려본다.
      이때, test 데이터셋에 대한 모델의 정확도가 unseen data에 대한 정확도를 나타낸다.
    - (방법3)이 가장 이상적으로 unseen data에 대한 정확도를 구하는 방법이다.
- 방법4) Cross Validation
    - 데이터양이 적을 때 주로 사용한다.
    - test 데이터셋을 미리 분리해둔다. test 데이터셋을 제외한 나머지 데이터를 N개의 fold로 분류한다.
      한번의 실험에 하나의 fold를 validataion 데이터셋으로 사용하고, 나머지 folds를 train 데이터셋으로 사용하여 모델을 학습시킨다. 각 validation 데이터셋에 대한 정확도를 평균낸다.
      → validation accuracy의 평균값이 해당 하이퍼파라미터에 대한 신뢰성 높은 결과를 보인다.
    - 위의 과정을 여러 하이퍼파라미터에 대해서 수행하고, 가장 높은 validation acc avg를 도출하는 하이퍼파라미터 쌍을 채택하면 된다.
    - 장점: 데이터가 부족한 경우, 양적으로 도움이 된다. 각 fold에 대해 알고리즘이 각각 어떤 결과를 내는 지에 대한 분산도 확인할 수 있다. 정확도의 분포를 확인할 수 있다.
- training vs. validataion
    - KNN에서 validataion 과정은, train 데이터와 일일히 비교하게 된다. 그리고 근접한 K개의 이웃에 따라 해당 validation 원소의 label이 정해진다.
    - train 데이터셋은 정답이 함께 주어지지만, validation 데이터셋은 정답을 주지 않고, 모델이 맞춰보게 하는 것이다. 예측을 끝내면, 그 예측값과 정답을 distance metric등으로 비교하여 오차를 계산하게 된다.
- 데이터셋을 오랜 기간동안 모으다보면, 앞서 모은 데이터와 나중에 모은 데이터 간의 차이가 발생할 수 있으나, 대부분 데이터셋을 모으게 되면 확률분포 상에 존재하게 되므로, 큰 문제가 되지 않는다.

> 📌 - Train 데이터는 이미지와 label이 함께 주어진다.
> - Test 데이터는 이미지가 주어지고, label을 예측하기 위해 사용된다.

KNN Classifiers on Images? No!

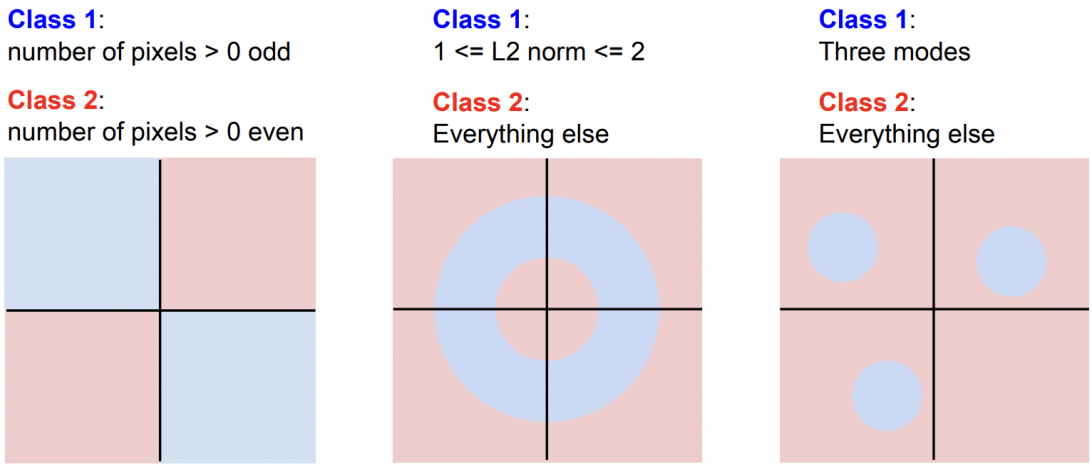
- KNN은 실생활의 이미지에서 잘 사용되지 않는다.
- 이유
    - test 하는 데에 시간이 오래 걸린다.
    - 픽셀에 대한 distance metrics이 유용하지 않다.
      여러 다른 validation 데이터에 대해, 우연하게도, 픽셀차의 합이 동일할 수 있다.
    - Curse of Dimensionality
- Curse of Dimensionality
    - 공간을 빽빽하게(densely) 채울수록,
    - 문제의 공간에서 학습 데이터가 많을수록,
    - 공간을 메꿀 수 있을 만큼의 양의 데이터를 찾아야하고, 그렇게 하기 쉽지 않다는 것

- Dimension이 추가될수록, 공간을 빽빽하게 채우기 위해 필요한 데이터의 수가 기하학적으로 늘어난다.
- KNN은 manifold 형태의 데이터에 대해 잘 작동하지 않는다. (3차원 이상에서 일어날 수 있는 일이다.)

Linear Classification

- 신경망은 선형분류(linear classification)을 여러겹 쌓은 것과 유사하다.
- image → $f(x, W)$ → class scores(각 카테고리에 속할 확률)
  - x: input data, 이미지, 픽셀의 행렬로 볼 수 있다.
  - W: weights, set of parameters(데이터에 의존적인 값들, 학습에 의해 결정됨)
    - **weights**: train 데이터셋에서 반드시 필요한 정보들을 추출한 값들의 집합.
      Test 시, train 데이터셋을 직접 사용하는 것이 아닌, train 데이터셋을 학습하여 도출한 값들을 사용하여 예측을 시도하는 것이다.
    - 행렬값으로 나타낼 수 있다.
  - b: bias term, 상수벡터로 나타난다.
    - train 데이터와 상호작용하지 않음. 각 class에 대한 선호정도를 나타낸다.
    - 데이터셋이 하나의 class에 대해 unbalance 하다면, 해당 class에 대한 bias term은 큰 값이 나타날 것이다.
    - 각 class에 대한 data independence scaling offset을 나타낸다.
  - $f(x, W) = Wx + b$
    - 연산결과는 하나의 컬럼벡터이다. 벡터의 각 원소는 각 label(class)에 해당할 확률을 의미한다.
    - inner product = dot product
- 선형분류기의 문제점: 하나의 선형분류기는 각 class에 대해 하나의 템플릿만을 학습함(관점1)
  - 다양한 모양에 대한 특징을 담기 어려움
  - 하나의 카테고리에 대해, 하나의 템플릿만을 학습한다.
  - ex) 말을 분류하려고 할 때, 머리가 왼쪽으로 가있는 말에 대해서만 분류기 학습이 이뤄질 수 있다.

  → 더 깊은 신경망을 사용하면, 하나의 카테고리에 대해 여러 템플릿을 학습할 수 있게 된다.
- 선형분류기의 다른 관점(관점2)
  - 고차원에서 보았을 때, 선형분류기가 어떤 데이터셋에 대한 분리지점으로 볼 수 있다는 것
  - 2차원에서 하나의 직선을 그려서, linear decision boundary로 해석할 수 있다.
    (데이터 분류기)
  - 선형분류기가 해내기 어려운 작업
    - 하나의 직선으로 클래스를 분리하기 불가능한 경우



-

- 과제

  example dataset: CIFAR-10(10 classes, 50000 training images, 10000 testing images)

  32*32*3

  - K-Nearest Neighbor
  - Linear classifiers: SVM, Softmax
  - Two-layer neural network
  - Image features