9. 이미지 기계학습

http://ahogrammer.com/2016/11/15/deep-learning-enables-you-to-hide-screen-when-your-boss-is-approaching/

9.1. 이미지 데이터

이미지에 기계학습을 적용하려면 텍스트를 TDM으로 바꿀때와 같이 표 형태의 정형 데이터로 바꿔줘야 한다. 이미지는 정형 데이터로 만들기가 텍스트보다는 간편하다.

9.1.1. 이미지 파일 형식

대표적인 이미지 파일 형식에는 JPEG, PNG, GIF가 있다.

JPEG는 사진 저장을 위한 손실 압축 방식의 표준이다. 흔히 jpg라고 한다. 손실 압축 방식이기 때문에 압축률이 높을 수록 화질이 낮다. 아래 그림의 경우 왼쪽에서 오른쪽으로 갈 수록 압축률이 낮고 화질이 높다. JPEG 파일은 편집과 저장을 반복하면 손실이 누적되서 점점 화질이 나빠진다.

GIF는 미국의 컴퓨서브(CompuServe)사가 개발한 비손실 압축 방식의 이미지 저장 형식이다. PNG는 GIF와 비슷한 국제 표준이다. 소프트웨어 특허로 묶여 있던 GIF를 대신해서 누구나 자유롭게 사용할 수 있도록 만들어졌다.

9.1.2. 일반적인 이미지 데이터의 구조

JPEG이든 GIF이든, PNG든 파일 형식에 관계 없이 데이터 분석을 위해 이미지를 불러오면 모두 동일한 형태로 변환해서 사용한다. 가장 널리 쓰이는 형식은 이미지를 픽셀 단위로 다루는 것이다. 그림의 점 하나를 픽셀이라고 한다.

각각의 점은 색깔을 나타내는 수치로 표현한다. 흑백의 경우 밝기의 정도를 0에서 255까지로 나타낸다. 0은 검정, 255는 흰색이다. 왜 255까지인가하면 8비트, 즉 2진수 8자리로 나타낼 수 있는 범위가 255까지이기 때문이다. 이정도면 사람의 눈에는 충분하기 때문에 굳이 16비트까지 쓰지는 않는다.

컬러의 경우에는 빛의 삼원색인 빨강(R), 초록(G), 파랑(B) 세 가지를 혼합해서 사용한다. 따라서 점 하나에 RGB를 나타내는 0~255 범위의 수치 3개로 표현한다. 보통은 표기를 간단히 하기 위해 16진수 표기를 많이 쓰는데 이때는 00~FF로 표시한다. 구글에서 color picker로 검색해서 색상코드를 확인해보자.

9.1.3. 이미지 행렬

점 하나를 1개(흑백) 또는 3개(컬러)의 수로 나타낸다면 전체 그림은 행렬로 나타낼 수 있다.가로 n, 세로 p 크기의 이미지이면 n행 p열인 행렬로 나타내는 것이다.

행렬 형태는 점들의 위치가 그대로 행렬에서 위치가 되므로 위치를 그대로 보존할 수 있다. 필터 등을 적용하기도 간단하다.

흑백은 1개의 행렬로 나타내지만, 컬러의 경우 RGB별로 3개의 행렬로 나타낸다.

9.1.4. 이미지 벡터

정형 데이터를 다루는 기계학습 모형은 데이터를 표에서 하나의 행, 즉 벡터로 다룬다. 따라서 이미지를 벡터로 바꾸면 기존의 기계학습 모형에 적용할 수 있다.

가로 n, 세로 p 크기의 이미지는 $n \times p$ 크기의 벡터로 나타낸다. 가로 20, 세로 10 크기의 흑백 그림은 크기 200인 벡터로 나타내는 것이다. 컬러의 경우에는 RGB가 모두 들어가야 하니 크기가 600인 벡터가 된다.

일반적으로는 행렬 형태로 전처리를 하고 벡터로 변환하여 최종 예측을 한다.

9.2. 이미지 처리

`Pillow` 라는 라이브러리는 이미지 처리할 때 사용되는 유용한 라이브러리이다.

공식문서 - (https://pillow.readthedocs.io/)

9.2.1. 이미지 파일 찾기

```
from PIL import Image
import os
```

우선 예제로 사용할 이미지들을 불러오자.

`os.listdir()` 를 통해 "images" 라는 폴더에 있는 파일목록을 가져오자.

```
images = os.listdir('./images')
```

```
images
```

```
['gecko_resized.jpg', 'gecko_rotated.jpg', 'gecko.jpg']
```

```
os.getcwd()
```

```
'/Users/yireysuh/Desktop/Quantlab/samsung_lectures'
```

9.2.2. 첫번째 이미지 선택

`os.path.join`을 통해서 사진 목록 중 첫 번째 이미지파일을 가져오자.

```
# 현재 작업 폴더의 "images" 에 첫 번째 사진을 지정한다.
image_path = os.path.join(os.getcwd(), 'images', images[0])
```

`Pillow` 의 `Image`를 통해 사진을 연다.

```
image = Image.open(image_path)
```

9.2.3. 이미지 크기 변환

```
image.resize((128, 128))  # 픽셀 단위
```

```
<PIL.Image.Image image mode=RGB size=128x128 at 0x10E564828>
```

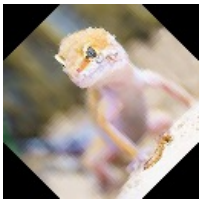
```
resize = image.resize((128, 128))
```

```
resize.save('./images/resized.jpg')
```

9.2.4. 이미지 회전

```
resize.rotate(45)  # 반시계방향 각도
```

```
<PIL.Image.Image image mode=RGB size=128x128 at 0x1CD76DE14A8>
```



```
rotate = resize.rotate(45)

rotate.save('./images/rotated.jpg')
```

9.2.5. 이미지 반전

상하반전

```
resize.transpose(Image.FLIP_TOP_BOTTOM)
```

```
<PIL.Image.Image image mode=RGB size=128x128 at 0x1CD76DE19B0>
```

좌우 반전

```
resize.transpose(Image.FLIP_LEFT_RIGHT)
```

```
<PIL.Image.Image image mode=RGB size=128x128 at 0x1CD76DE1B00>
```

90도로 돌리기

```
resize.transpose(Image.ROTATE_90)  # Image.ROTATE_180, Image.ROTATE_270
```

```
<PIL.Image.Image image mode=RGB size=128x128 at 0x1CD75E6A7B8>
```

9.2.6. 이미지 필터

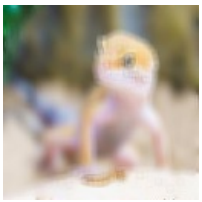
`Pillow` 의 `ImageFilter`를 통해 사진에 필터링을 할 수 있다.

```
from PIL import ImageFilter
```

블러 처리

```
resize.filter(ImageFilter.BLUR)
```

```
<PIL.Image.Image image mode=RGB size=128x128 at 0x1CD76DF3EB8>
```



엠보싱

```
resize.filter(ImageFilter.EMBOSS)
```

```
<PIL.Image.Image image mode=RGB size=128x128 at 0x1CD76DF3FD0>
```

윤곽

```
resize.filter(ImageFilter.CONTOUR)
```

```
<PIL.Image.Image image mode=RGB size=128x128 at 0x1CD76DFC0F0>
```

자세히

```
resize.filter(ImageFilter.DETAIL)
```

```
<PIL.Image.Image image mode=RGB size=128x128 at 0x1CD76DFC1D0>
```

날카롭게

```
resize.filter(ImageFilter.EDGE_ENHANCE)
```

```
<PIL.Image.Image image mode=RGB size=128x128 at 0x1CD76DFC358>
```

부드럽게

```
resize.filter(ImageFilter.SMOOTH)
```

```
<PIL.Image.Image image mode=RGB size=128x128 at 0x1CD76DFC198>
```

9.2.7. Convert

`.convert()`에 색상 표현 방식을 넘겨줘서 이미지를 원하는 색상표현으로 변경할 수 있다.

자주 사용되는 방식들은 다음과 같다

- L : 흑백
- RGB : Red, Green, Blue 삼원색을 이용한 색 표현
- RGBA : Red, Green, Blue 삼원색을 이용한 색 표현 방법 + 투명도
- CMYK : Cyan, Magenta, Yellow, Black 을 이용한 색 표현 방법

이미지에 `.convert()` 함수에 L, RGB, RGBA 등을 넘겨줘서 원하는 색상표현 방식으로 변환할 수 있다.

9.2.8. 흑백

`.convert()`에 'L'을 넘겨줘서 8 비트 그레이스케일로 변환할 수 있다.

```
resize.convert('L')  # ITU-R 601-2 luma transform
```

```
<PIL.Image.Image image mode=L size=128x128 at 0x1CD76DFC400>
```

9.2.9. draw text

사진에 텍스트를 한번 넣어보자.

```
from PIL import ImageDraw, ImageFont
```

우선 텍스트를 넣을 사진을 불러온다.

```
base = Image.open(os.path.join(os.getcwd(), 'images', images[1])).convert('RGBA')

# 흰색-투명으로 이미지와 같은 크기 이미지 하나 더 만들기
txt = Image.new('RGBA', base.size, (255,255,255,0))
d = ImageDraw.Draw(txt)


d.text((10,10), "Hello", fill=(255,255,255,128))  # 위치, 텍스트, 흰색-반투명

d.text((10,60), "World", fill=(255,255,255,255))  # 위치, 텍스트, 흰색-불투명

Image.alpha_composite(base, txt)
```

```
<PIL.Image.Image image mode=RGBA size=128x128 at 0x1CD76E039B0>
```

RGBA 형태의 그림은 **`.png`** 파일로 저장해야한다. **`.jpg`** 파일로 저장하면 오류가 발생한다.

9.2.10. PIL - 사진 여러 장을 pdf 로 생성하기

3 개의 이미지 파일을 불러오자.

```
im1 = Image.open('images/gecko.jpg')
im2 = Image.open('images/gecko_resized.jpg')
im3 = Image.open('images/gecko_rotated.jpg')

image_list = [im2, im3]

im1 = im1.convert('RGB')
```

이미지의 사이즈를 조정하고 싶다면 `im.resize(x 크기, y 크기)` 로 지정하면 된다.

`tiff` 파일을 먼저 생성을 해야 `pdf` 가 생성이 된다.

```
im1.save('images.tiff', save_all=True, append_images=image_list)
im1.save('images.pdf', save_all=True, append_images=image_list)
```

후에 `PyPDF2` 를 사용해 사진을 다른 PDF 페이지 사이에 끼워넣을 수 있다.

9.3. keras 이미지 기계학습 실습

keras는 딥러닝을 쉽게 할 수 있는 라이브러리다. 이미지 분석은 딥러닝이 특히 강점을 보이는 주제다.

PIL은 이미지를 한 장씩 열어야 하니 대량의 이미지를 열거나 닫을 때는 불편하다. keras에는 PIL 방식으로 이미지를 열 수 있는 기능을 내장하고 있다.

9.3.1. 이미지 다운로드 및 압축 해제

먼저 실습용 이미지를 다운로드하자.

```
from urllib.request import urlretrieve
urlretrieve('http://doc.mindscale.kr/km/unstructured/dog-vs-cat.zip',
            'dog-vs-cat.zip')
```

```
('dog-vs-cat.zip', <http.client.HTTPMessage at 0x7f85db4cab00>)
```

다음으로 압축을 해제한다.

```
from zipfile import ZipFile

with ZipFile('dog-vs-cat.zip') as z:
    z.extractall()
```

9.3.2. 이미지 불러오기

keras는 PIL과 마찬가지로 이미지를 불러올 수 있는 기능이 있다.

```
from keras.preprocessing.image import load_img
load_img('dog-vs-cat/train/cat/cat.1.jpg')
```

그러나 이미지를 하나씩 불러오면 처리하기 어려우므로 한 꺼번에 불러올 수 있는 방법도 있다. 학습용 데이터를 불러오자.

```
from keras.preprocessing.image import ImageDataGenerator
train = ImageDataGenerator().flow_from_directory(
    'dog-vs-cat/train',     # 이미지 디렉토리
    target_size=(100, 100), # 변환할 크기는 가로 100, 세로 100
    color_mode='rgb',       # 컬러는 rgb, 흑백은 grayscale. 생략하면 컬러로 처리한다
    class_mode='binary')    # 고양이 vs. 개로 binary 분류
```

학습시킨 모형의 성능 검증을 위한 검증용 데이터도 불러오자. 방법은 동일하다.

```
valid = ImageDataGenerator().flow_from_directory(
    'dog-vs-cat/validation',
    target_size=(100, 100),
    class_mode='binary')
```

9.3.3. 모형 만들기

이제 간단한 로지스틱 회귀분석 모형을 만들어보자.

```
from keras import Sequential
from keras.layers import Dense, Flatten
```

먼저 모형을 만든다.

```
model = Sequential()
```

데이터가 가로 세로 크기 100인 컬러 이미지이므로 입력 형태는 $100 \times 100 \times 3$가 된다. 모형의 앞단에 이 입력을 하나의 벡터로 만들어주는 레이어(layer)를 넣어준다.

```
model.add(Flatten(input_shape=(100, 100, 3)))
```

하나의 예측을 내놓는 dense 레이어를 추가한다. 활성화 함수(activation)를 시그모이드(sigmoid)로 설정하면 0~1범위의 출력을 내어놓는다. 이것을 "입력된 사진이 강아지(dog)일 확률"로 해석할 수 있다. 이것은 로지스틱 회귀분석과 동일한 모형이 된다.

```
model.add(Dense(1, activation='sigmoid'))
```

9.3.4. 요약 정보

만들어진 모형의 요약 정보를 보자.

```
model.summary()
```

9.3.5. 모형 설정

이제 모형을 학습시키기 전 설정을 한다. 손실 함수는 모형이 얼마나 잘 맞는지를 측정하는 역할을 한다. 흔히 쓰는 것은 다음 3가지이다.

- `mse`: 오차 제곱의 평균. 연속적인 예측을 할 때 사용한다.
- `binary_crossentropy`: 둘 중에 하나로 분류할 때 사용한다.
- `categorical_crossentropy`: 셋 이상으로 분류할 때 사용한다.

크로스엔트로피는 직관적으로 이해하기 어려운 지표이다. 우리가 성능을 확인하기 위해 추가로 정확도(`accuracy`)를 계산한다. 정확도는 전체 데이터에서 올바르게 분류한 사례의 비율을 나타낸다.

마지막으로 최적화 알고리즘은 실제 학습 방법을 결정한다. `rmsprop`이나 `adam`을 사용한다.

```
model.compile(loss='binary_crossentropy', metrics=['accuracy'], optimizer='adam')
```

9.3.6. 학습

이제 전체 데이터를 5회 학습시켜보자.

```
model.fit_generator(train, validation_data=valid, epochs=5)
```

```
Epoch 1/5
63/63 [==============================] - 8s 134ms/step - loss: 7.9314 - acc: 0.5025 - val_loss: 7.9712 - val_acc: 0.5000
Epoch 2/5
63/63 [==============================] - 8s 122ms/step - loss: 7.9323 - acc: 0.5024 - val_loss: 7.9712 - val_acc: 0.5000
Epoch 3/5
63/63 [==============================] - 8s 120ms/step - loss: 7.9634 - acc: 0.5005 - val_loss: 7.9712 - val_acc: 0.5000
Epoch 4/5
63/63 [==============================] - 8s 119ms/step - loss: 7.9945 - acc: 0.4985 - val_loss: 7.9712 - val_acc: 0.5000
Epoch 5/5
63/63 [==============================] - 8s 119ms/step - loss: 7.9478 - acc: 0.5015 - val_loss: 7.9712 - val_acc: 0.5000
```