Banana Classification - 한가희

전체 개요

1. 주제 선정 과정
   (1) 바나나 분류 선정 및 이유
   (2) 실제 AI 적용 사례

2. 데이터 수집
   (1) 데이터 수집: 수집 대상, 수집 출처, 수집 방법

3. Selenium을 이용하여 웹 크롤링 하기
   (1) 웹 크롤링이란??
   (2) 이미지 수집을 위해 selenium 설치 후 webdriver를 이용해 google 페이지에 접속 후 종료
   (3) 웹 크롤링 함수 정의 후 함수로 web에서 이미지 다운받아 원하는 경로에 저장하기

4. banana 이미지 분류 특성 선택

5. sklearn을 이용하여 학습하기
   (1) svm 모델로 학습하기
   (2) knn 모델로 학습하기

6. Summary

1.주제 선정 과정

(1)바나나 분류 선정 및 이유

AI를 이용하여 이미지를 분류를 학습하고 다 익은것과 안 익은것을 분류하여 기계가 바나나를 수확할 수 있도록 한다.
농장에 AI 기술을 적용시켜 일손부족 해결가능

(2) 실제 AI 적용 사례

출처: https://biz.chosun.com/site/data/html_dir/2019/07/10/2019071002946.html

2. 데이터 수집 및 분석 개요

(1)데이터 수집: 수집 대상, 수집 출처, 수집 방법

- 수집 대상: 덜 익은 바나나(초록), 잘 익은 바나나(보통), 많이 익은 바나나(갈색)
- 수집 방법: Selenium을 사용하여 web 크롤링
- 수집 출처: google

3. Selenium을 이용하여 웹 크롤링 하기

(1)웹 크롤링이란?

이미지 데이터 수집을 위해 웹 크롤링을 활용한다.
웹크롤링이란 웹 페이지를 그대로 가져와서 거기서 데이터를 추출해 내는 행위다.

-웹크롤링을 하기위해 파이썬의 selenium 모듈을 설치하여 webdriver를 이용해 google 페이지 접속 후 이미지 검색을 해준다.
이미지 검색을 위해 def crawling_image 함수를 정의하여 google에서 이미지 검색 후 내가 지정한 경로에 저장하는 함수로 만든다.

1) selenium 설치 후 webdriver를 이용해 google 페이지에 접속 후 종료

In [1]:
```
!pip install selenium
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: selenium in c:₩users₩21₩appdata₩roaming₩python₩python39₩site-packages (4.5.0)
Requirement already satisfied: trio-websocket~=0.9 in c:₩users₩21₩appdata₩roaming₩python₩python39₩site-packages (from selenium) (0.9.2)
Requirement already satisfied: trio~=0.17 in c:₩users₩21₩appdata₩roaming₩python₩python39₩site-packages (from selenium) (0.22.0)
Requirement already satisfied: certifi>=2021.10.8 in c:₩programdata₩anaconda3₩lib₩site-packages (from selenium) (2021.10.8)
Requirement already satisfied: urllib3[socks]~=1.26 in c:₩programdata₩anaconda3₩lib₩site-packages (from selenium) (1.26.9)
Requirement already satisfied: attrs>=19.2.0 in c:₩programdata₩anaconda3₩lib₩site-packages (from trio~=0.17->selenium) (21.4.0)
```

In [2]:
```
from selenium import webdriver
# chromedriver.exe 가 위치한 폴더 경로 지정하기
path = "C:/Users/21/Downloads/chromedriver.exe"
# 1.selenium으로 제어할 수 있는 크롬 브라우저 새창이 열린다.
driver = webdriver.Chrome(path)
```

```
C:₩Users₩21₩AppData₩Local₩Temp₩ipykernel_11984₩547220527.py:5: DeprecationWarning: executable_path has been deprecated, please pass in a Service object
  driver = webdriver.Chrome(path)
```

In [3]:
```
# 2.webdriver가 google 페이지에 접속하도록 명령
driver.get('https://www.google.com')
```

In [4]:
```
elem = driver.find_element("name", "q")
#입력 부분에 지운다-default로 값이 있을 수 있는 경우 고려
elem.clear()
#검색어 입력
elem.send_keys("Selenium")
#검색 실행
elem.submit()
#검색이 제대로 됐는지 확인
assert "No results found." not in driver.page_source
```

In [5]:
```
#브라우저 종료
driver.close()
```

2) 웹 크롤링으로 이미지 다운하기

In [6]:
```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import time
import urllib.request
from selenium.webdriver.common.by import By
```

In [7]:
```
def crawling_image(driver, search, number, save_dir):
    driver.get("https://www.google.co.kr/imghp?hl=ko&tab=wi&ogbl") # Google이미지검색

    elem = driver.find_element("name", "q")                        # 검색창 부분
    elem.send_keys(str(search))                                    # 검색어 입력
    elem.send_keys(Keys.RETURN)                                    # Enter키 쳐서 검
    SCROLL_PAUSE_TIME = 0.7
    # Get scroll height
    last_height = driver.execute_script("return document.body.scrollHeight")
    while True:
        # 스크롤을 내립니다.
        driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
        # 페이지를 넘기기 위해 기다립니다
        time.sleep(SCROLL_PAUSE_TIME)
        # 마지막으로 스크롤한 위치와 새로 스크롤한 위치를 비교합니다
        new_height = driver.execute_script("return document.body.scrollHeight")
        if new_height == last_height: # 스크롤이 끝까지 내려갔다면
            try:                      # 이미지 더보기 버튼을 클릭해 줌

                driver.find_elements(By.CSS_SELECTOR, ".mye4qd").click()
            except:                   # 더이상 이미지 더보기가 없으면 끝냄
                break
        last_height = new_height
    #images = driver.find_elements_by_css_selector(".rg_i.Q4LuWd")
    images = driver.find_elements(By.CSS_SELECTOR, ".rg_i.Q4LuWd")
    count = 0              # 이미지 파일 이름을 1,2,3...으로 하기 위해

    for image in images:
        try:
            image.click()   # 첫번째 이미지 선택
            time.sleep(0.1)   # 이미지 사이트 검색을 위해 3초간 쉬어줌
            imgURL = driver.find_element(By.CSS_SELECTOR, ".n3VNCb").get_attribute("
            # 아래줄의 폴더의 경로부분은 자신의 PC에 맞추어 변경해 주세요.
            outpath = save_dir # 이미지를 저장할 폴더
            outfile = str(count) + ".jpg"
            urllib.request.urlretrieve(imgURL, outpath+outfile)
            count = count + 1
            if count == number:
                break
            print("이미지 저장 {}회".format(count+1))
        except:
            pass
```

In [11]:
```
if __name__=='__main__':
    path = "C:/Users/21/Downloads/chromedriver.exe"
    driver = webdriver.Chrome(path)
    print('이미지 검색을 시작합니다.')
    number = 50                  # 저장할 사진 개수
    search_name = 'overripe banana'          # 검색어
    # 사진이 저장될 폴더 : 자신의 컴퓨터 이름으로 변경하여 사용하도록 함
    save_dir = "C:/Users/21/Downloads/overripe/image"
    crawling_image(driver, search_name, number, save_dir) # 크롤링을 시작합니다.
    driver.close()    # 마지막에는 드라이버를 닫아줌
```

```
C:₩Users₩21₩AppData₩Local₩Temp₩ipykernel_11984₩2011812675.py:3: DeprecationWarning:
executable_path has been deprecated, please pass in a Service object
  driver = webdriver.Chrome(path)
```

```
이미지 검색을 시작합니다.
이미지 저장 2회
이미지 저장 3회
이미지 저장 4회
이미지 저장 5회
```

In [13]:
```
if __name__=='__main__':
    path = "C:/Users/21/Downloads/chromedriver.exe"
    driver = webdriver.Chrome(path)
    print('이미지 검색을 시작합니다.')
    number = 50                  # 저장할 사진 개수
    search_name = 'banana'          # 검색어
    # 사진이 저장될 폴더 : 자신의 컴퓨터 이름으로 변경하여 사용하도록 함
    save_dir = "C:/Users/21/Downloads/ripe/image"
    crawling_image(driver, search_name, number, save_dir) # 크롤링을 시작합니다.
    driver.close()    # 마지막에는 드라이버를 닫아줌
```

```
C:₩Users₩21₩AppData₩Local₩Temp₩ipykernel_11984₩1877016020.py:3: DeprecationWarning:
executable_path has been deprecated, please pass in a Service object
  driver = webdriver.Chrome(path)
```

```
이미지 검색을 시작합니다.
이미지 저장 2회
이미지 저장 3회
이미지 저장 4회
이미지 저장 5회
이미지 저장 6회
이미지 저장 7회
이미지 저장 8회
이미지 저장 9회
```

4. banana 이미지 분류 특성 선택

지금까지 이미지 분류를 위한 이미지 데이터를 google Crawling을 통해 모았다.
다음 장에는 모은 데이터를 활용해 sklearn을 이용해 학습하기 전, 특성 추출을 해보자.
이 과정에서 이미지를 불러와 평균 색상을 구하고, 평균 색상을 특성으로 지정한다.
이미지 특성을 추출하여 벡터화 한 후, 샘플 이미지를 출력 해 볼 것이다.

1)이미지 하나씩 불러와 평균색상보기

라이브러리 불러오기

```
In [1]: import cv2
        import numpy as np
        import warnings
        warnings.filterwarnings('ignore')
```

특성 추출 - 바나나 익은 정도를 알기위한 특성 선택

```
In [4]: green_banana = cv2.imread("banana data/greensample.jpg")
        overripe_banana = cv2.imread("banana data/overripesample.jpg")
        ripe_banana = cv2.imread("banana data/ripesample.jpg")

        print (averagecolor(green_banana))
        print (averagecolor(overripe_banana))
        print (averagecolor(ripe_banana))
```

```
[139.80286139 206.86742847 188.72206739]
[104.27028099 143.81804121 165.84011184]
[172.71251886 228.24215834 243.09175733]
```

샘플이미지 보기

matplotlib의 pyplot을 이용하여 샘플이미지를 볼 수 있다.

cv2로 이미지를 불러오면 BGR로 불러오므로 cv2.cvtColor를 이용하여 RGB로 바꾸어야한다.

```
In [16]: from matplotlib import pyplot as plt

         green_banana = cv2.cvtColor(green_banana,cv2.COLOR_BGR2RGB)
         overripe_banana = cv2.cvtColor(overripe_banana,cv2.COLOR_BGR2RGB)
         ripe_banana = cv2.cvtColor(ripe_banana,cv2.COLOR_BGR2RGB)


         plt.figure(figsize=(10, 4))
         plt.subplot(131)
         plt.imshow(green_banana)
         plt.axis("off")
         plt.title("green banana")
         plt.subplot(132)
         plt.imshow(overripe_banana)
         plt.axis("off")
         plt.title("overripe_banana")
         plt.subplot(133)
         plt.imshow(ripe_banana)
         plt.axis("off")
         plt.title("ripe_banana")
```

```
Out[16]: Text(0.5, 1.0, 'ripe_banana')
```

5. sklearn을 이용하여 학습하기

(1) svm 모델로 학습하기

*서포트 벡터 머신(Support Vector Machine)

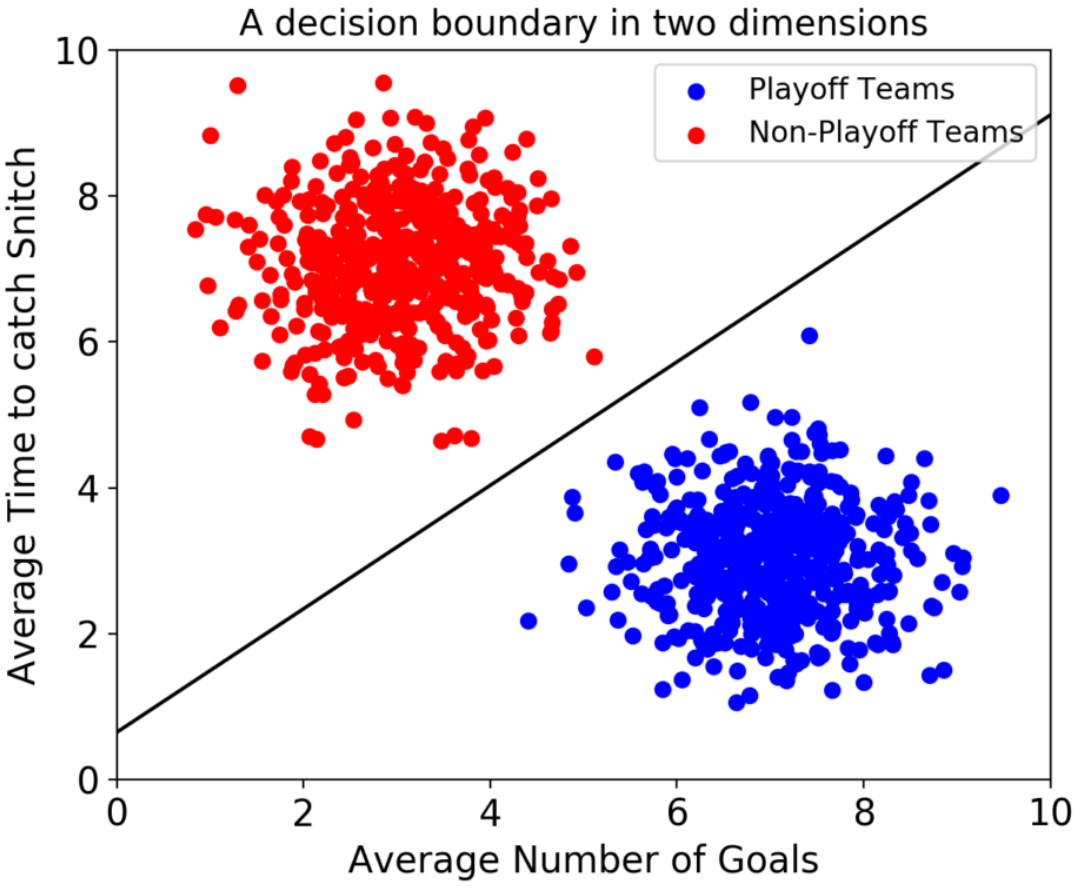
:분류 과제에 사용할 수 있는 강력한 머신러닝 지도학습 모델

서포트 벡터 머신이란

서포트 벡터 머신(이하 **SVM**)은 결정 경계(**Decision Boundary**), 즉 분류를 위한 기준 선을 정의하는 모델이다. 그래서 분류되지 않은 새로운 점이 나타나면 경계의 어느 쪽에 속하는지 확인해서 분류 과제를 수행할 수 있게 된다.

결국 이 결정 경계라는 걸 어떻게 정의하고 계산하는지 이해하는 게 중요하다는 뜻

예시)



참조: https://hleecaster.com/ml-svm-concept/

1)모델 학습을 위해 train / test 이미지 불러오기

train 이미지 불러오기

In [18]:
```
trainX = []
trainY = []
import os

path = "banana data/"
for label in ('green','overripe','ripe'):
    print ("Loading training images for the label: "+label)

    # 하위 폴더의 모든 이미지를 읽어옵니다.
    for filename in os.listdir(path+label+"/"):
        img = cv2.imread(path+label+"/"+filename)
        img_features = averagecolor(img)
        trainX.append(img_features)
        trainY.append(label)
```

```
Loading training images for the label: green
Loading training images for the label: overripe
Loading training images for the label: ripe
```

In [19]:
```
print (len(trainX))
print (len(trainY))
```

```
91
91
```

svm 모델로 학습하기

In [20]:
```
# SVM은 숫자 값을 사용하기 때문에 먼저 레이블을 숫자로 인코딩합니다.
from sklearn.preprocessing import LabelEncoder  # 레이블을 숫자로 인코딩
encoder = LabelEncoder()                        # 레이블을 숫자로 인코딩
encodedtrainY = encoder.fit_transform(trainY)  # 레이블을 숫자로 인코딩

from sklearn import svm
model = svm.SVC(gamma="scale", decision_function_shape='ovr')
model.fit(trainX, encodedtrainY)
```

Out[20]:
```
SVC()
```

In [21]:
```
print (encodedtrainY)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

svm모델에서는 숫자만 인식하기 때문에 레이블을 LabelEncoder()를 이용하여 숫자로 바꿔준 다음에 학습시킨다.

test이미지 불러온 후 예측 실행

In [22]:
```
realtestY = np.array(["green","green","green","green","green",
                      "overripe","overripe","overripe","overripe","overripe",
                      "ripe","ripe","ripe","ripe","ripe"])
```

In [23]:
```
import os
from sklearn.metrics import classification_report

path = "banana data/test/"
filenames = []
predictedY = []
for filename in os.listdir(path):
    img = cv2.imread(path+filename)
    img_features = averagecolor(img)
    prediction = model.predict([img_features])[0]

    # 예측을 코드화합니다.
    prediction = encoder.inverse_transform([prediction])[0]

    print (filename + ": " + prediction)
    filenames.append(filename)
    predictedY.append(prediction)

# 정확도 평가(sklearn 패키지는 유용한 보고서를 제공합니다)
print ()
print(classification_report(realtestY, predictedY))
```

```
01.jpg: green
02.jpg: green
03.jpg: green
04.jpg: green
05.jpg: green
06.jpg: overripe
07.jpg: overripe
08.jpg: overripe
09.jpg: green
10.jpg: overripe
11.jpg: ripe
12.jpg: ripe
13.jpg: ripe
14.jpg: ripe
15.jpg: overripe

              precision    recall  f1-score   support

       green       0.83      1.00      0.91         5
    overripe       0.80      0.80      0.80         5
        ripe       1.00      0.80      0.89         5

    accuracy                           0.87        15
   macro avg       0.88      0.87      0.87        15
weighted avg       0.88      0.87      0.87        15
```

svm 모델을 사용하기 위해 레이블을 숫자로 바꿔줬기 때문에
encoder.inverse_transform을 사용하여 숫자로 나온 예측값을 레이블 문자로 다시 바꿔준다.

svm 모델정확도 평가:

- green 바나나
  정밀도: 0.83, 재현율:1.00, f1-score:0.91
- overripe 바나나
  정밀도: 0.80, 재현율:0.80, f1-score:0.80
- ripe 바나나
  정밀도: 1.00, 재현율:0.80, f1-score:0.89
- 정확도: 0.87

정확도 평가 결과로 전체 f1-score 0.87이 나왔다.
green 바나나의 경우 0.91로 가장 평가지수가 높게 나왔고, ripe,overripe 순으로 지수가 높게 나왔다.

히트맵 이용하여 혼동행렬 그리기

혼동행렬 (Confusion Matrix)

		예측 결과	
		TRUE	FALSE
실제 정답	TRUE	TP (True Positive)	FN (False Negative)
	FALSE	FP (False Positive)	TN (True Negative)

혼동행렬 사용하는 이유: 분류모델에서 정확도만이 최적의 결정포인트가 아니므로 정밀도, 재현율과 같은 평가지표가 필요하다. 왜냐하면 모든 데이터의 비율이 공평하게 나누어져 있지 않기 때문이다.

ex) 100개의 라벨중 A가 50개 , B가 50개라면 A와 B를 구분하는 분류모델에서 정확도가 중요한 평가지표가 된다
but A가 90개, B가 10개 라벨의 모델이라면 무작정 A라고 분류해도 90%의 정확도를 나타내기 때문에 정확도가 최적의 평가지표가 아니라는 것을 알 수 있다.

해석 시 뒤가 예측
앞은 예측한 것이 맞음과 틀림을 나타낸다.

Accuracy (정확도) :전체 데이터 중 올바르게 예측한 비율

		현실	
		긍정(Positive)	부정(Negative)
예측	긍정(Positive)	**참긍정** (TP; True Positive)	**거짓긍정** (FP; False Positive)
	부정(Negative)	**거짓부정** (FN; False Negative)	**참부정** (TN; True Negative)

$$\frac{(TP+TN)}{(TP+TN+FP+FN)}$$

단점 : 예측하려고 하는 종속변수의 비율이 불균형할때 가치가 낮아진다.

정의 [편집]

		실제 정답	
		Positive	Negative
실험 결과	Positive	True Positive	False Positive (Type II error)
	Negative	False Negative (Type I error)	True Negative

통계적 분류 분야에서 정밀도(precision)와 재현율(recall)은 다음과 같이 정의된다:[1]

$$\text{Precision} = \frac{tp}{tp + fp}$$

$$\text{Recall} = \frac{tp}{tp + fn}$$

Precision(정밀도) :예측 결과가 긍정적일 때 현실도 실제로 긍정일 확률

정밀도가 중요한 경우**:** 스팸 메일

스팸인 줄 알았는데(Positive) 실제론 아니었다(False). → 중요한 이메일이 누락됐다. 때문에 거액의 거래를 놓쳤다. → False Positive (2종 오류)

이번 경우에는 스팸이라고 판단하는 "Positive" 예측에 신중을 기해야 하고, 실제로 스팸메일 중에서 잘못된 False Positive 예측이 얼마나 포함되었는지 판단하는 정밀도[TP/(TP+FP)]가 더 중요한 척도가 되는 것이다.

→ FP(False Positive)가 줄어들수록, 분모가 작아져서 숫자가 커지기 때문!

Recall (재현율)
:현실이 실제로 긍정일 때 예측 결과도 긍정적일 확률

재현율이 중요한 경우**:** 암진단

암이 아닌 줄 알았는데(Negative) 실제로 암이었다(True). → 치료시기를 놓쳤다. 아주 위험하다. → True Negative (1종 오류)

"Negative" 예측을 할 때 신중에 신중을 기해야 한다. 즉, 실제 결과에서 잘못된 False Negative 예측이 얼마나 포함되었는지 판단하는 재현율[TP/(TP+FN)]이 훨씬! 중요한 척도가 되는 것

F1 score :

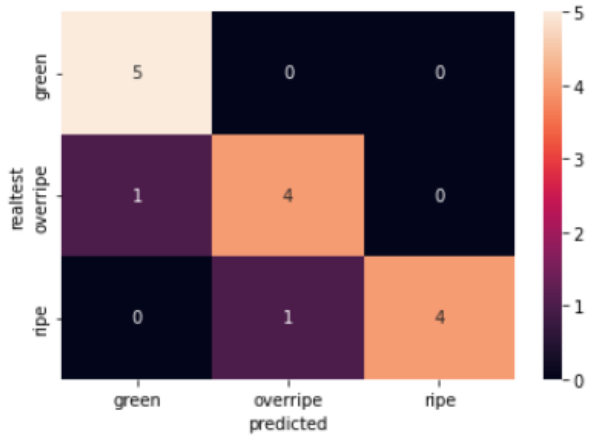
$$\frac{2PR}{P+R}$$

정밀도와 재현율(민감도)을 활용하는 평가용 지수
F1 값의 논리는 조화평균에 입각한 것으로, 정밀도와 재현율 사이에 하나가 높아지면 다른 하나가 낮아지는 상황이 자주 발생하기에 이를 보정하기 위해 개발

heatmap을 이용하여 혼동행렬을 보기 쉽게 시각화하였다.

In [24]:
```
import seaborn as sns
import pandas as pd

df = pd.DataFrame()

# 데이터 프레임 내에서 새 열 만들기
df['realtest'] = realtestY
df['predicted'] = predictedY

# pd.crosstab을 사용하여 실제 및 예측된 클래스의 빈도 계산
freq = pd.crosstab(df.realtest,df.predicted)

# sns.heatmap을 사용하여 히트맵 그리기
sns.heatmap(freq,annot=True,fmt="d")
```

Out[24]: <AxesSubplot: xlabel='predicted', ylabel='realtest'>



정답 별 예측 결과를 알아보고 항목별 비교를 위해 heapmap을 활용한다.

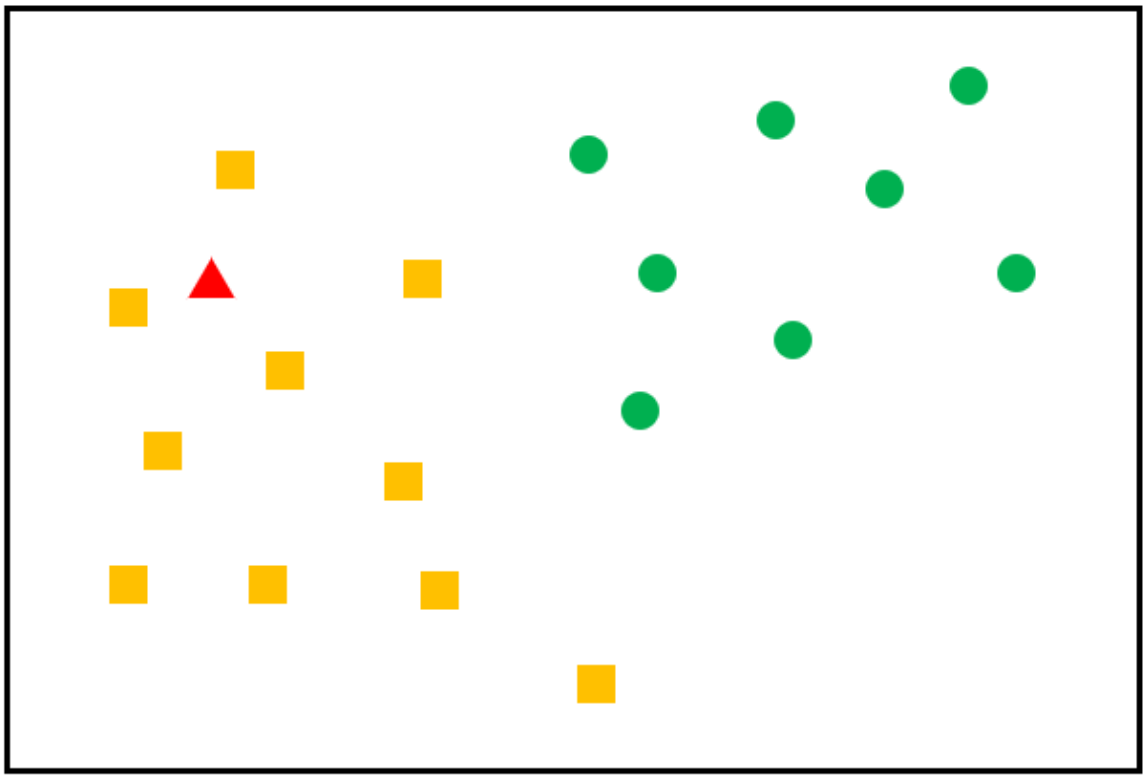
green 바나나의 정밀도 계산:
5/(5+1)=0.833
overripe 바나나의 재현율 계산:
4/(1+4)=0.80

(2) KNN 모델로 학습하기

K-최근접 이웃(K-NN) 알고리즘

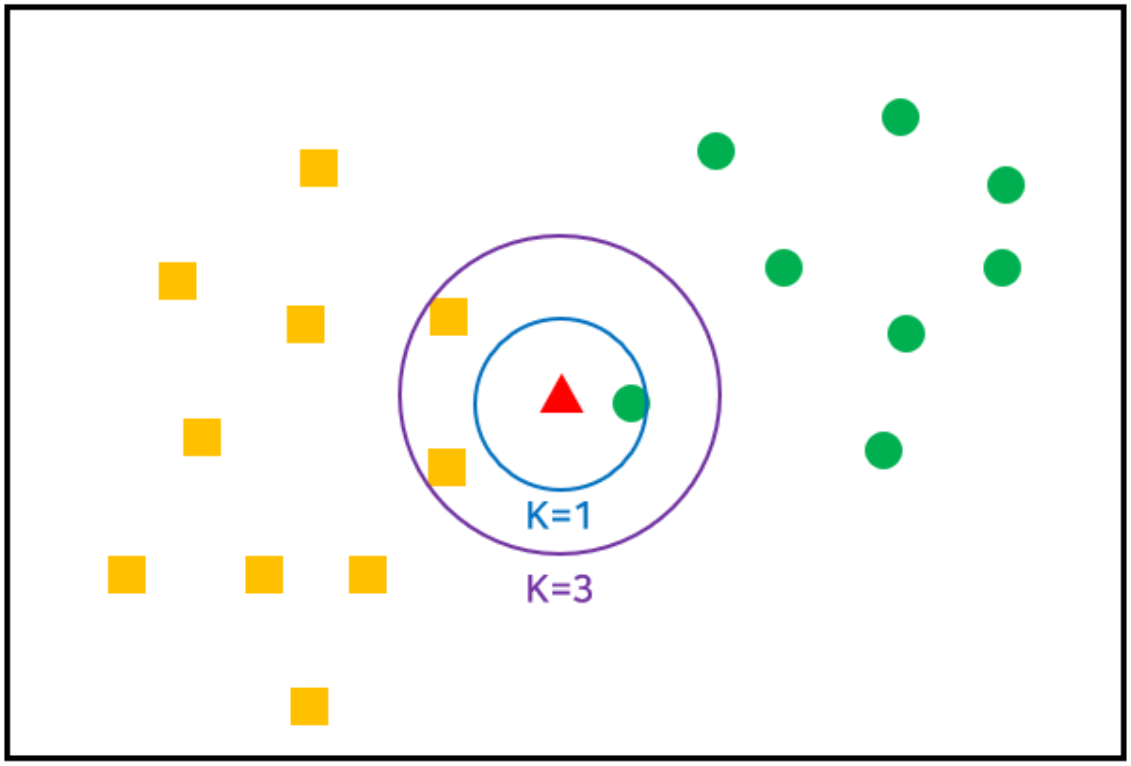
KNN알고리즘이란?

K-최근접 이웃(K-NN, K-Nearest Neighbor) 알고리즘은 가장 간단한 머신러닝 알고리즘으로, 분류**(Classification)** 알고리즘이다. 비슷한 특성을 가진 데이터는 비슷한 범주에 속하는 경향이 있다는 가정하에 사용



예를 들어 위와 같이 데이터가 주어져 있을 때, 빨간색인 세모 모양의 데이터는 초록색 그룹과 노란색 그룹 중 어디에 속한다고 말할 수 있을까?
주변에 가까운 데이터들이 모두 노란색이기 때문에 '노란색 그룹에 속할 것이다'라고 추측할 수 있다.
이처럼, 주변의 가장 가까운 **K**개의 데이터를 보고 데이터가 속할 그룹을 판단하는 알고리즘이 **K-NN** 알고리즘이다.

K-NN 알고리즘은, 단순히 훈련 데이터셋을 그냥 저장하는 것이 모델을 만드는 과정의 전부이다. 그리고 거리를 측정할 땐 유클리드 거리**(Euclidean distance)**를 사용한다. **K-NN** 알고리즘의 특징 중 하나는 **K**의 값에 따라 분류가 달라질 수 있다는 점이다. 예를 들어 아래와 같은 상황을 보면 **K = 1**인 경우에는 초록색 그룹이라고 판단하고, **K = 3**인 경우에는 노란색 그룹이라고 판단할 것이다.



장점: 단순하기 때문에 다른 알고리즘에 비해 구현하기가 쉽다. 또 훈련 데이터를 그대로 가지고 있어 특별한 훈련을 하지 않기 때문에 훈련 단계가 매우 빠르게 수행된다.

단점: 모델을 생성하지 않기 때문에 특징과 클래스 간 관계를 이해하는데 제한적이다. 모델의 결과를 가지고 해석하는 것이 아니라, 미리 변수와 클래스 간의 관계를 파악하여 이를 알고리즘에 적용해야 원하는 결과를 얻을 수 있기 때문이다. 또, 적절한 **K**의 선택이 필요하고, 훈련 단계가 빠른 대신 데이터가 많아지면 분류 단계가 느리다.
참조:https://rebro.kr/183

k=3인 KNN 알고리즘

k=3으로 설정하여 동작을 수행 후 정확도 평가는 f-score, precision, recall을 활용한다.

In [25]:
```
from sklearn.neighbors import KNeighborsClassifier

KNN = KNeighborsClassifier(n_neighbors=3)
KNN.fit(trainX, encodedtrainY)
```

Out[25]:
```
KNeighborsClassifier(n_neighbors=3)
```

In [26]:
```
import os
from sklearn.metrics import classification_report

path = "banana data/test/"
filenames = []
predictedY = []
for filename in os.listdir(path):
    img = cv2.imread(path+filename)
    img_features = averagecolor(img)
    prediction = KNN.predict([img_features])[0]

    # 예측을 코드화합니다.
    prediction = encoder.inverse_transform([prediction])[0]

    print (filename + ": " + prediction)
    filenames.append(filename)
    predictedY.append(prediction)

# 정확도 평가(sklearn 패키지는 유용한 보고서를 제공합니다)
print ()
print(classification_report(realtestY, predictedY))
```

```
01.jpg: green
02.jpg: green
03.jpg: green
04.jpg: green
05.jpg: green
06.jpg: overripe
07.jpg: green
08.jpg: overripe
09.jpg: overripe
10.jpg: overripe
11.jpg: ripe
12.jpg: ripe
13.jpg: ripe
14.jpg: ripe
15.jpg: green

              precision    recall  f1-score   support

       green       0.71      1.00      0.83         5
    overripe       1.00      0.80      0.89         5
        ripe       1.00      0.80      0.89         5

    accuracy                           0.87        15
   macro avg       0.90      0.87      0.87        15
weighted avg       0.90      0.87      0.87        15
```

정확도 평가:

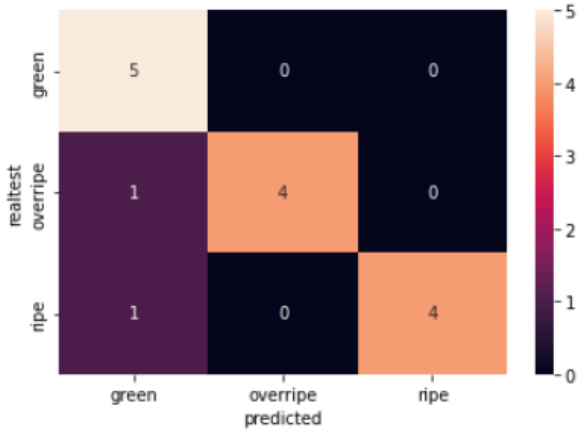
- green 바나나
  정밀도: 0.71, 재현율:1.00, f1-score:0.83
- overripe 바나나
  정밀도: 1.00, 재현율:0.80, f1-score:0.89
- ripe 바나나
  정밀도: 1.00, 재현율:0.80, f1-score:0.89
- 정확도: 0.87

정확도 평가 결과로 전체 f1-score 0.87이 나왔다.
green 바나나의 경우 0.83로 가장 평가지수가 가장 낮게 나왔고, ripe,overripe는 0.89로 같다.
위의 svm모델과 평가가 다르게 나옴.

히트맵 이용하여 혼동행렬 그리기

In [27]:
```
import seaborn as sns
import pandas as pd

df = pd.DataFrame()

# 데이터 프레임 내에서 새 열 만들기
df['realtest'] = realtestY
df['predicted'] = predictedY

# pd.crosstab을 사용하여 실제 및 예측된 클래스의 빈도 계산
freq = pd.crosstab(df.realtest,df.predicted)

# sns.heatmap을 사용하여 히트맵 그리기
sns.heatmap(freq,annot=True,fmt="d")
```

Out[27]: <AxesSubplot: xlabel='predicted', ylabel='realtest'>



k=5인 **KNN** 알고리즘

이번에는 k=5으로 설정하여 동작을 수행 후 정확도 평가는 f-score, precision, recall을 활용한다.

In [28]:
```
from sklearn.neighbors import KNeighborsClassifier

KNN2 = KNeighborsClassifier(n_neighbors=5)
KNN2.fit(trainX, encodedtrainY)
```

Out[28]:
```
KNeighborsClassifier()
```

In [29]:
```
import os
from sklearn.metrics import classification_report

path = "banana data/test/"
filenames = []
predictedY = []
for filename in os.listdir(path):
    img = cv2.imread(path+filename)
    img_features = averagecolor(img)
    prediction = KNN2.predict([img_features])[0]

    # 예측을 코드화합니다.
    prediction = encoder.inverse_transform([prediction])[0]

    print (filename + ": " + prediction)
    filenames.append(filename)
    predictedY.append(prediction)

# 정확도 평가(sklearn 패키지는 유용한 보고서를 제공합니다)
print ()
print(classification_report(realtestY, predictedY))
```

```
01.jpg: green
02.jpg: overripe
03.jpg: green
04.jpg: green
05.jpg: green
06.jpg: overripe
07.jpg: green
08.jpg: overripe
09.jpg: overripe
10.jpg: overripe
11.jpg: ripe
12.jpg: ripe
13.jpg: ripe
14.jpg: ripe
15.jpg: green

              precision    recall  f1-score   support

       green       0.67      0.80      0.73         5
    overripe       0.80      0.80      0.80         5
        ripe       1.00      0.80      0.89         5

    accuracy                           0.80        15
   macro avg       0.82      0.80      0.81        15
weighted avg       0.82      0.80      0.81        15
```

정확도 평가:

- green 바나나
  정밀도: 0.67, 재현율:0.80, f1-score:0.73
- overripe 바나나
  정밀도: 0.80, 재현율:0.80, f1-score:0.80
- ripe 바나나
  정밀도: 1.00, 재현율:0.80, f1-score:0.89
- 정확도: 0.80

정확도 평가 결과로 전체 f1-score 0.80이 나왔다.
green 바나나의 경우 0.73로 가장 평가지수가 가장 낮게 나왔고, ripe,overripe순을 커진다.

➔ **K=3**일때보다 평가지수 낮아짐**.** 성능이 더 좋지않다**.**

k=7인 **KNN** 알고리즘

이번에는 k=7으로 설정하여 동작을 수행 후 정확도 평가는 f-score, precision, recall을 활용한다.

In [30]:
```
from sklearn.neighbors import KNeighborsClassifier

KNN3 = KNeighborsClassifier(n_neighbors=7)
KNN3.fit(trainX, encodedtrainY)

import os
from sklearn.metrics import classification_report

path = "banana data/test/"
filenames = []
predictedY = []
for filename in os.listdir(path):
    img = cv2.imread(path+filename)
    img_features = averagecolor(img)
    prediction = KNN3.predict([img_features])[0]

    # 예측을 코드화합니다.
    prediction = encoder.inverse_transform([prediction])[0]

    print (filename + ": " + prediction)
    filenames.append(filename)
    predictedY.append(prediction)

# 정확도 평가(sklearn 패키지는 유용한 보고서를 제공합니다)
print ()
print(classification_report(realtestY, predictedY))
```

```
01.jpg: green
02.jpg: green
03.jpg: green
04.jpg: green
05.jpg: green
06.jpg: overripe
07.jpg: overripe
08.jpg: overripe
09.jpg: overripe
10.jpg: overripe
11.jpg: ripe
12.jpg: ripe
```

```
13.jpg: ripe
14.jpg: ripe
15.jpg: green

              precision    recall  f1-score   support

       green       0.83      1.00      0.91         5
    overripe       1.00      1.00      1.00         5
        ripe       1.00      0.80      0.89         5

    accuracy                           0.93        15
   macro avg       0.94      0.93      0.93        15
weighted avg       0.94      0.93      0.93        15
```

정확도 평가:

- green 바나나
  정밀도: 0.83, 재현율:1.00, f1-score:0.91
- overripe 바나나
  정밀도: 1.00, 재현율:1.00, f1-score:1.00
- ripe 바나나
  정밀도: 1.00, 재현율:0.80, f1-score:0.89
- 정확도: 0.93

정확도 평가 결과로 전체 f1-score 0.93이 나왔다.
overripe 바나나의 경우 1.00 로 가장 평가지수가 가장 낮게 나왔고, green,ripe순으로 작아진다.

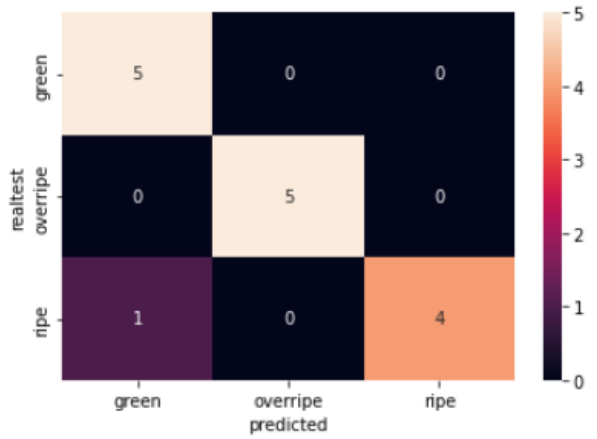
➔ **K=3**일때보다 평가지수 높아짐**.** 성능이 더 좋다**.**

- **KNN**모델의 **k**값을 증가시키며 실행해 보았을 때 **k=7**일 때 정확도 **0.93,** 다른 각각의 **f1 score**도 **k=3, k=5**보다 제일 높음을 볼 수 있었다**.**

히트맵 이용하여 혼동행렬 그리기

In [31]:
```
import seaborn as sns
import pandas as pd

df = pd.DataFrame()

# 데이터 프레임 내에서 새 열 만들기
df['realtest'] = realtestY
df['predicted'] = predictedY

# pd.crosstab을 사용하여 실제 및 예측된 클래스의 빈도 계산
freq = pd.crosstab(df.realtest,df.predicted)

# sns.heatmap을 사용하여 히트맵 그리기
sns.heatmap(freq,annot=True,fmt="d")
```

Out[31]: <AxesSubplot: xlabel='predicted', ylabel='realtest'>



혼동행렬을 보면 알 수 있듯이 실제 ripe바나나 이미지를 green바나나로 예측하였다.
하나의 이미지만 예측 실패.

6. Summary

Selenium을 이용하여 구글 이미지 크롤링을 해보았고, 원하는 실제 바나나의 사진이 아닌 캐릭터나 그림인 이미지도 많이 수집 되어서 다운로드 된 데이터를 확인하고 삭제하여 알맞은 데이터만 모으는 과정이 필요하였다.

svm모델과 k=3인 KNN모델을 비교하였을 때 두 모델 다 정확도는 0.87로 같지만 각 클래스의 f1 score가 다 다름을 볼 수 있었다.

또한, KNN모델의 k값을 증가시키며 실행해 보았을 때 k=7일 때 정확도 0.93, 다른 각각의 f1 score도 제일 높음을 볼 수 있었다.

다만, train 데이터의 양이 각 class마다 30개로 매우 적은 양의 이미지로 수행하여 성능이 좋게 나올 수 밖에 없었던 것 같다.