**[작성한 소스코드의 구동 환경]**

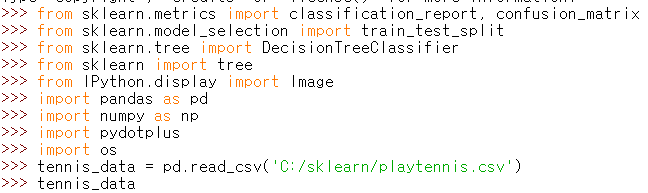
Python version : Python 3.7.0

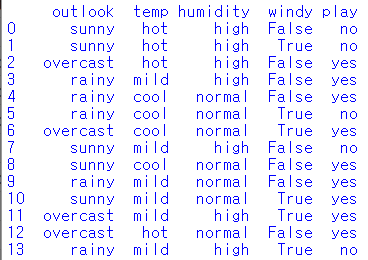
IDLE : Python 3.7 64-bit

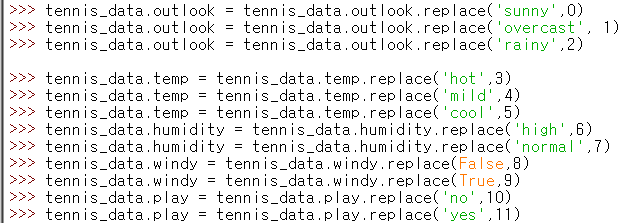
scikit-learn, numpy, pandas : Python 3.7.0 버전에 맞는 모듈 설치

anaconda : Anaconda3-2019.10-Windows-x86\_64

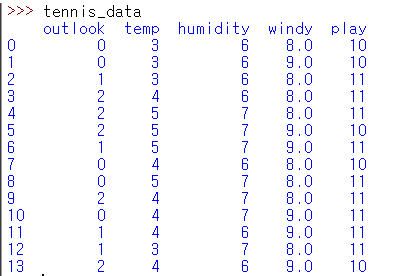
**[실험1]**

[작성한 코드에 대한 설명 및 각 실험의 결과 로그 캡처]

먼저 필요한 모듈을 import 해주고, tennis\_data라는 개체를 하나 생성해서 playtennis.csv라는 파일을 읽어와 저장해줍니다.

그러면 이러한 결과 값이 출력됩니다.

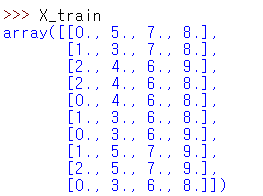
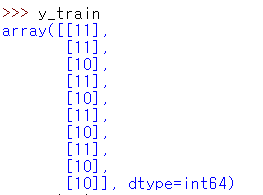
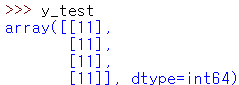
그 다음으로, 데이터 전처리 과정을 거치는데 문자열 타입에서 숫자 타입으로 바꿔 저장해줍니다.

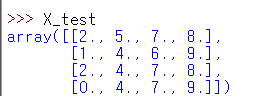


그러면 결과값은 이렇게 나옵니다.

데이터 프레임 형태로 추출하여, np.array 함수를 이용하여 추출한 데이터를 배열형태로 변환하여 저장합니다.

그리고 y에는 tennis\_data의 컬럼 값을 데이터프레임 형태로 추출하고 배열형태로 변환한후에 변수에 저장합니다.

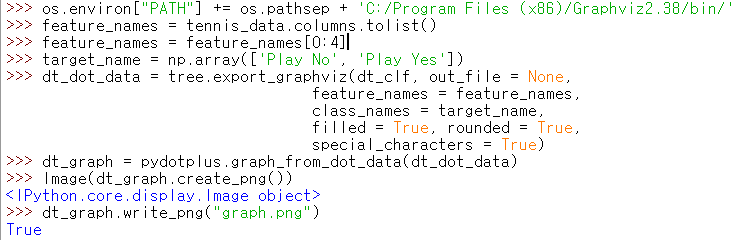
로드된 train\_test\_split 모듈을 이용하여 변수 X에 입력 4개의 컬럼의 데이터와 변수 y에 입력한 playtennis 컬럼의 데이터를, train(훈련)과 test를 구분해, 임의의 개수로 각각 변수 X\_train, X\_test, y\_train, y\_test에 저장합니다.



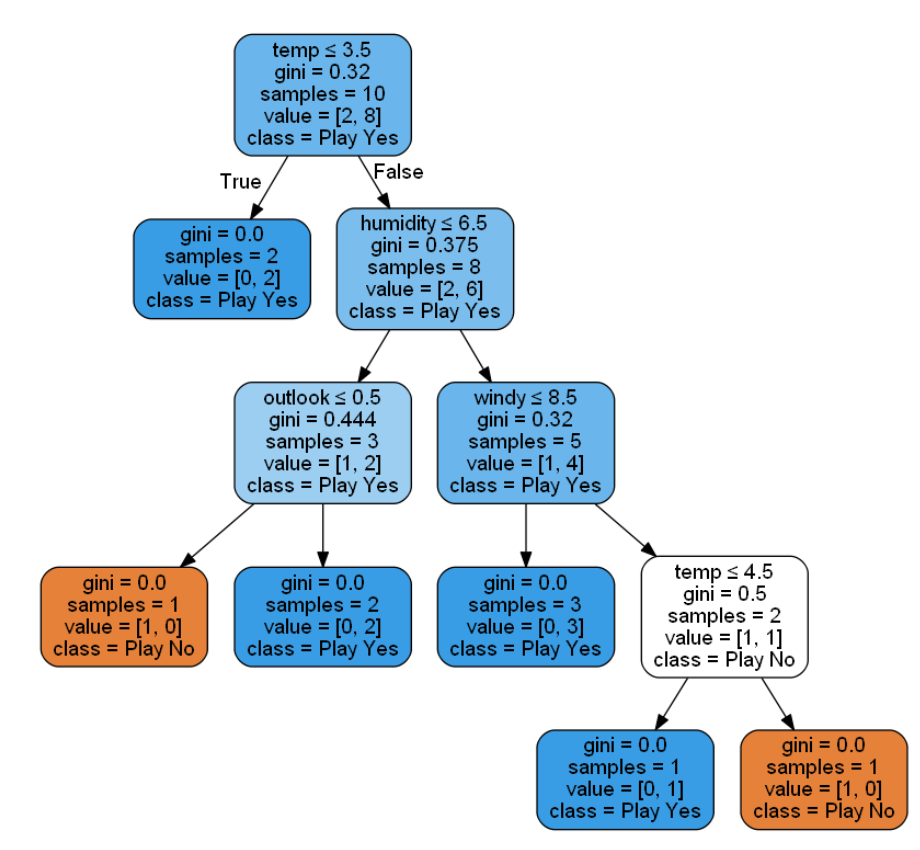
저장된 값을 출력해 봤습니다.

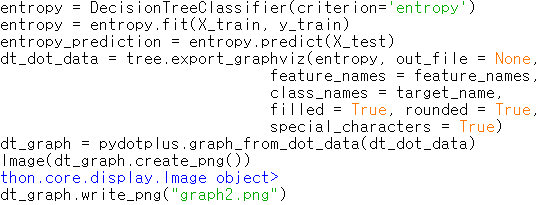
다음으로 로드된 의사 결정 트리 분류 모듈을 변수 dt\_clf에 저장했습니다. 그리고 의사 결정 트리 모듈이 저장된 변수 dt\_clf의 함수 fit()에 변수 X\_train, y\_train의 값을 입력해서 의사 결정 트리 분류 모델을 생성했습니다. 그런 다음 생성한 모델을 다시 변수 dt\_clf에 저장했습니다.

X\_test에 대한 클래스 예측 값을 변수 dt\_prediciton에 저장했습니다.

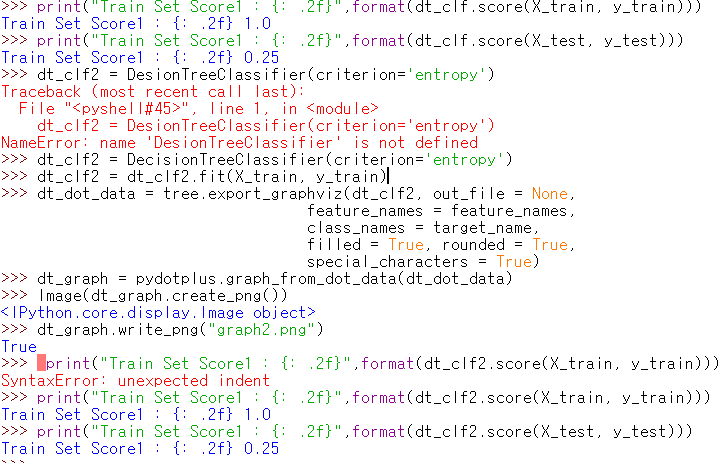
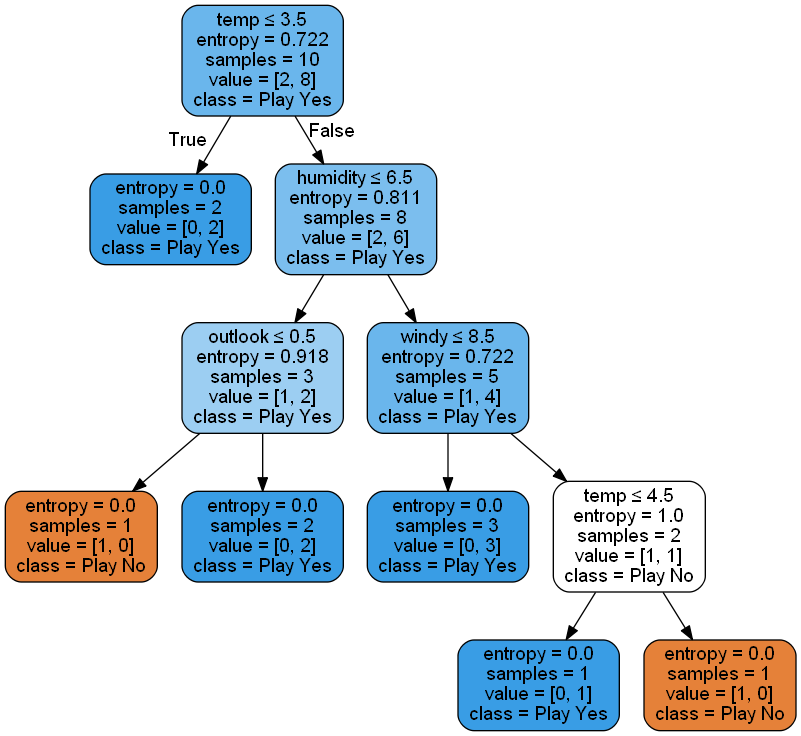
그래프를 표시해주는 Graphviz라는 소프트웨어를 설치했습니다. 그리고 경로를 지정해주고, 트리 표현함수에 입력되는 feature\_names를 입력하기 위해 각 컬럼을 list 형태로 변환해주었습니다.

슬라이싱해서 outlook,temp,humidity, windy의 컬럼명을 추출하여 저장해주었고, Play No와 Play Yes를 배열 형태로 변수에 저장해주었습니다. 그리고 완성된 트리를 export\_graphviz()를 이용해 내보내주고, 그래프를 png 형태로 저장했습니다.

**[트리 시각화 출력물] – 지니계수**

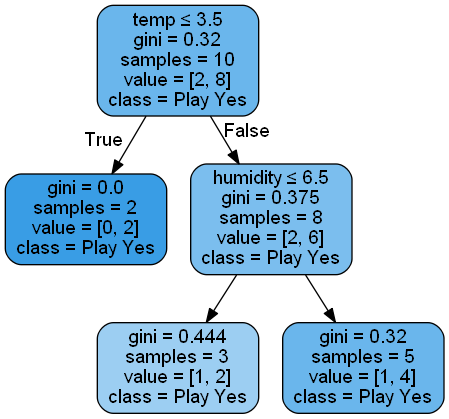
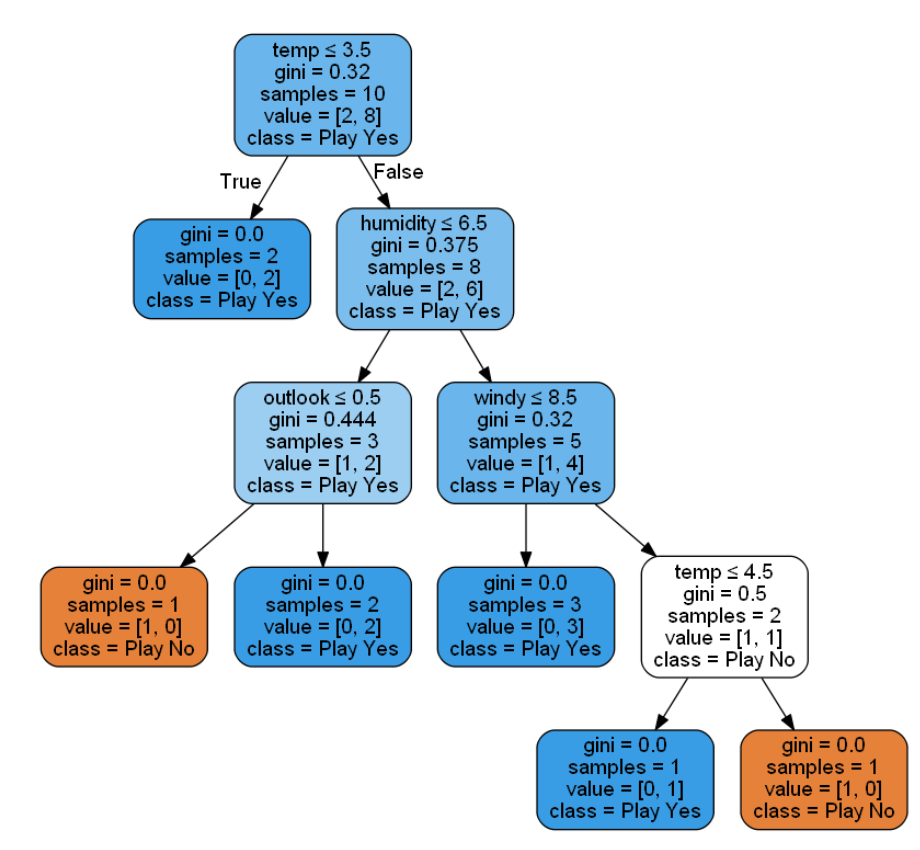
****

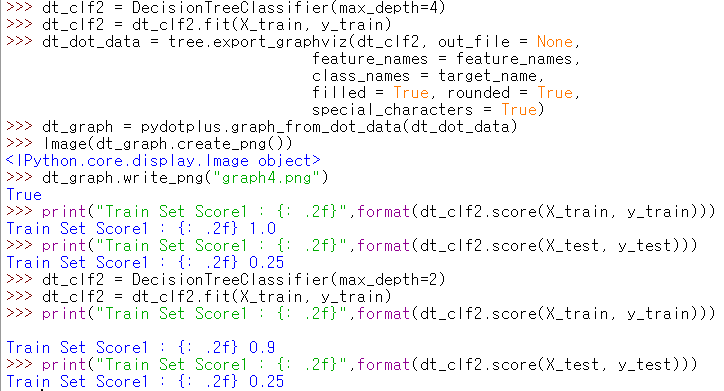
다른 것은 모두 같고 criterion만 entropy로 변경해주니 이렇게 출력물이 변했습니다.

******[트리 시각화 출력물] – 엔트로피**

**[실험결과 분석]**

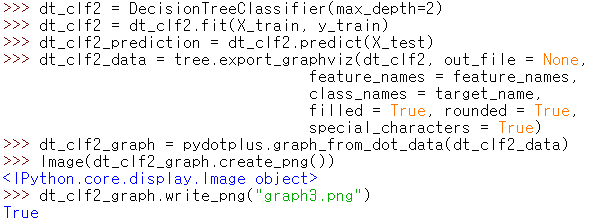
지니계수와 비교할 때 엔트로피와 지니계수의 값 차이만 존재할 뿐 동일한 결과를 출력하는 것을 확인했습니다. 트리의 깊이 뿐 아니라 점수를 출력할 때도 같았습니다.

**[트리 시각화 출력물] – max\_depth=2 & max\_depth=4**



max\_depth=4는 코드가 비슷하므로 생략하겠습니다.

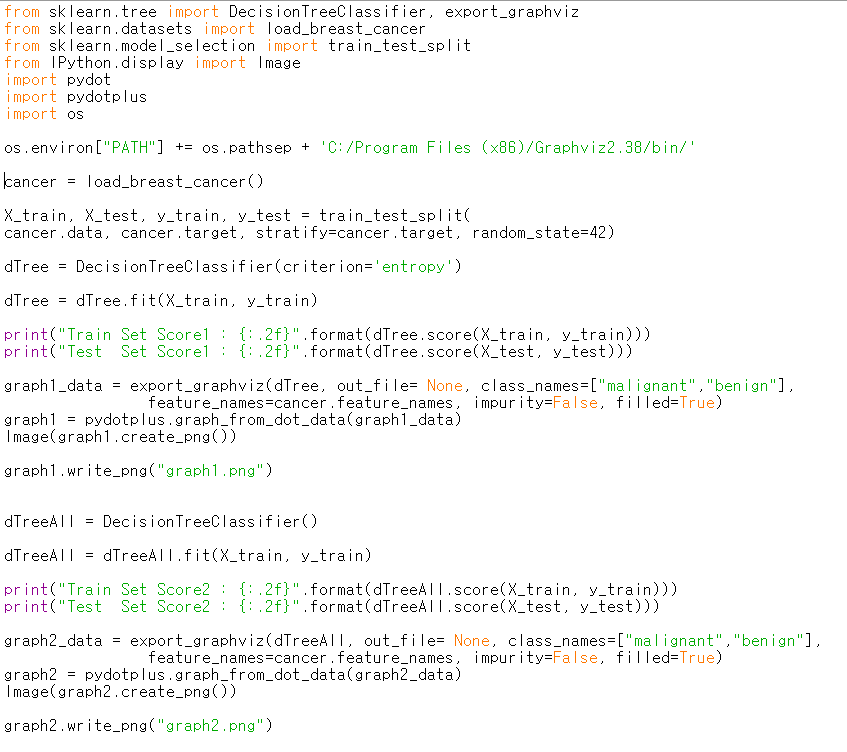
**[코드]**

****

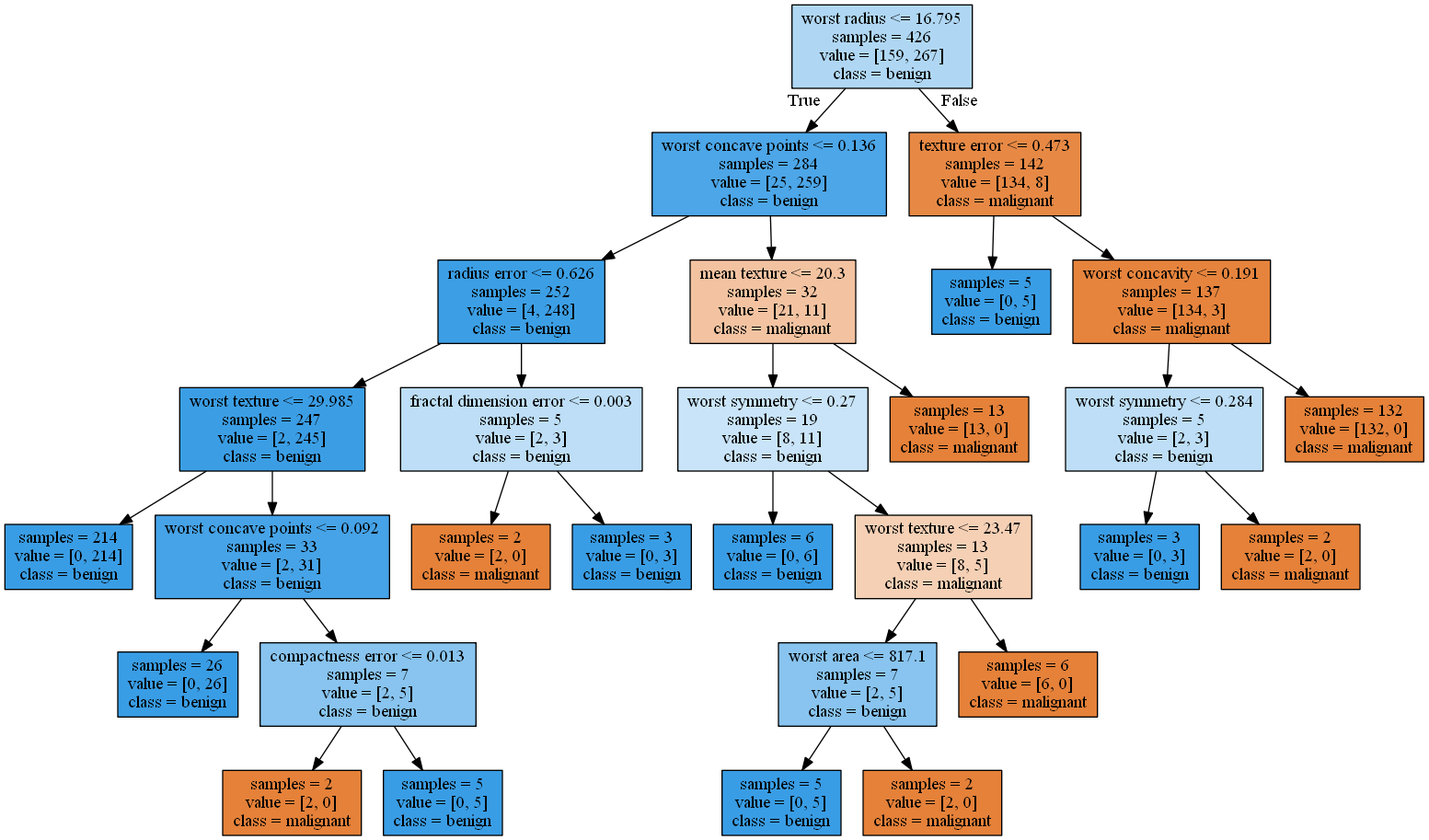
**[실험 결과 분석]**

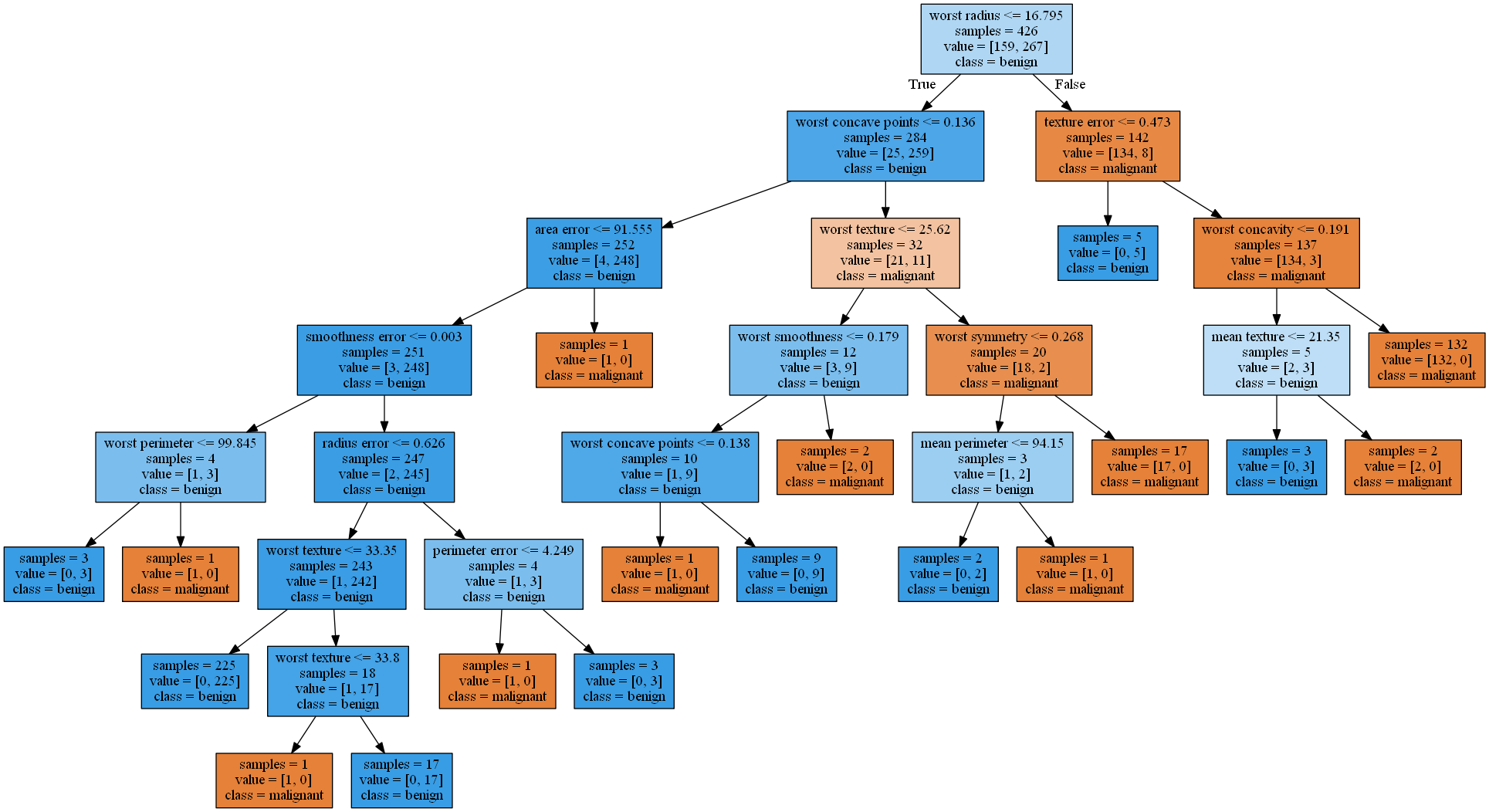
최대 깊이를 2로 주었더니, Play Yes와 Play No가 제대로 분류 되지 않고, 분류하다가 마는 경우가 되었습니다. 최대 깊이를 주지 않으면 계속 해서 분류 해나가기 때문에 조금 더 시간이 오래 걸릴 수는 있지만 데이터가 정확히 분류 되고 최대 깊이를 지정할 경우 분류가 되다가 말기 때문에 시간은 적게 소요되지만 정확도는 떨어짐을 확인하게 되었습니다.

**[실험2 ]**

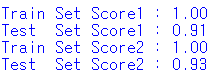
**[작성한 코드에 대한 설명 및 각 실험의 결과 로그 캡처]**

그 다음으로는 데이터를 유방암 샘플 데이터를 가지고 실험을 진행했습니다. 첫번째로는 지니계수대신 엔트로피를 써서 진행하고, 지니계수를 써서 진행하여 각각의 결과 값과 그래프를 비교해보기로 했습니다.

**[트리 결과물] -graph1.png 엔트로피 사용**

**[트리 결과물] -graph2.png 지니계수 사용**

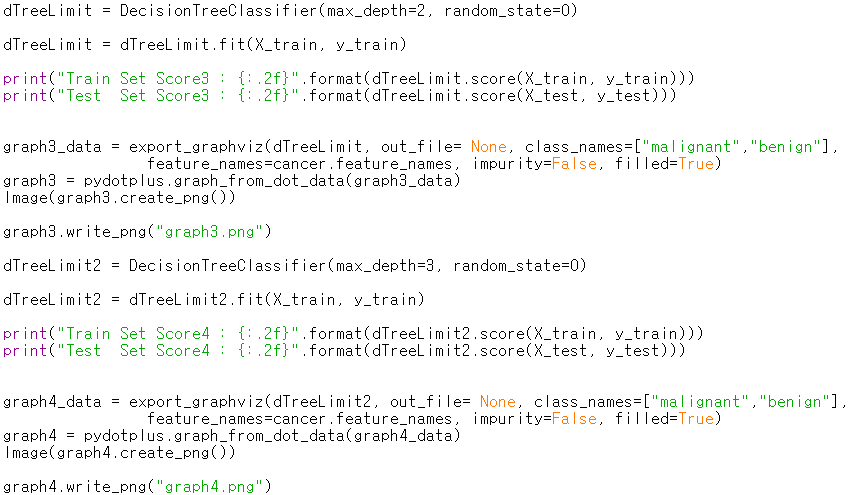
**[실험결과 분석]**



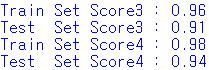
먼저 이렇게 하면, score는 test의 경우 두번째인 지니계수를 쓸 때 더 좋다는 것을 확인 할 수 있었고, train의 경우에는 같은 결과 값을 출력하는 것을 볼 수 있었습니다.

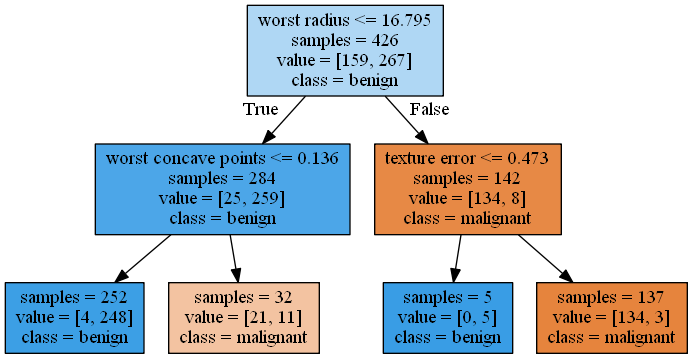
그래프의 경우 지니계수를 사용할 때 깊이가 더 깊은 것을 확인할 수 있었습니다. 그러나, test에서의 점수가 지니계수를 사용할 때 더 높게 나온 것을 보면 테스트를 진행할 때는 지니계수가 더 좋은 효율을 낸다는 것을 알게 되었습니다. 깊이가 작을 수록 족은 것이지만 가중치 값을 가지고 있기 때문에 깊이가 더 작다고 해서 항상 더 좋은 결과 값을 내는 것은 아니구나라는 생각을 하게 되었습니다.

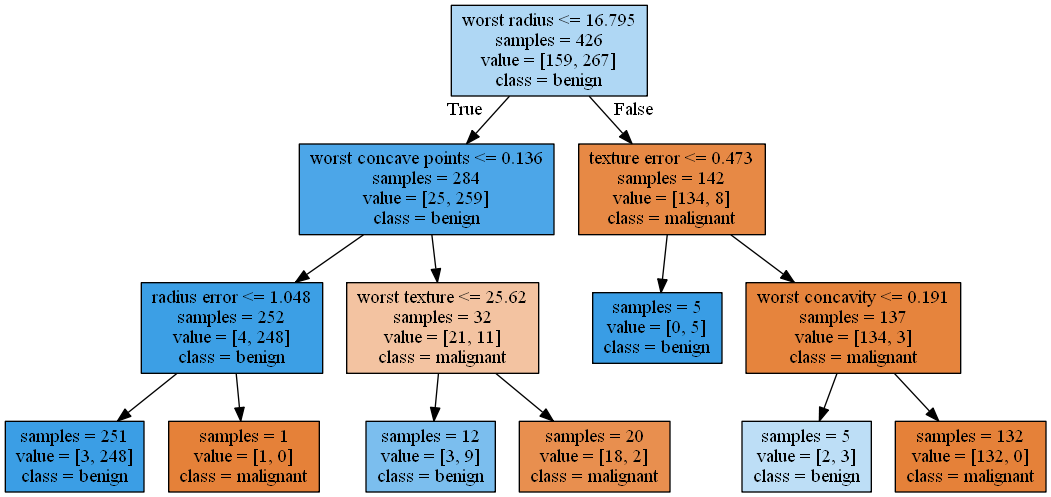
다음으로는 깊이에 대한 실험을 진행했습니다.

**[작성한 코드에 대한 설명 및 각 실험의 결과 로그 캡처]**

위의 코드에서 모두 지니계수를 사용하여 실험을 진행했고, 깊이를 2와 3일때로 주어 실험을 진행했습니다. 다른 코드는 모두 위와 동일하므로, 생략하겠습니다.

**[로그 출력물]**

**[트리결과물] – max\_depth 2**

**[트리결과물] – max\_depth 3**

**[실험결과 분석]**

깊이를 2로 주었을때, 분류해내는 것도 있지만, 정확한 분류를 시키기엔 아직 부족한 것을 확인 할 수 있습니다. 또한, 깊이를 3으로 주었을 때는 그것보다는 더 정확히 분류해내긴 하지만, 그럼에도 확실히 다 분류되진 않음을 볼 수 있습니다. 로그 출력물에서 볼 수 있다 싶이, 이러한 경우, 깊이가 너무 작을 때는 분류가 되는 수준이 좀 부족하기에 train, test 모든 값에서 깊이를 더 줄 경우보다 작게 나오는 것을 확인 할 수 있지만, 깊이를 3으로 줄 때는 위에서 진행한 최대 깊이를 지정하지 않았을 경우보다 train의 경우 점수가 낮으나, test의 경우에는 깊이 자체가 작아 점수가 더 높게 나오는 것을 확인 할 수 있습니다. 적당한 깊이를 설정하는 것이 더 효율적인 알고리즘을 만든다는 것을 깨닫게 되었습니다.

[실험3] – **과제 모두 수행 후에 추가해서 실습한 내용**

실험 1,2를 진행하다보니 진짜 tensorflow를 사용하여 실습을 진행해보고 싶은 마음이 생겨 이렇게 실험 3도 진행하게 되었습니다. 이것은 추가적으로 진행했지만, 과제를 수행하며 관심이 생겨 작성한 것이어서 보고서에도 추가해봤습니다.

**[환경 구축]**

Python : Python 3.7.0

moudles : numpy, pandas, matplotlib.pyplot, tensorflow

Anaconda : JupyterNoteBook, Anaconda3

먼저 저는 anaconda가 이미 깔려있던 상태이기 때문에, jupyterNoteBook을 이용하여 실습을 진행하기 위해, anaconda prompt에서 tensorflow를 설치해주는 과정을 거쳤습니다.

Anaconda prompt 명령어는 이렇게 작성했습니다.

1 - (base) C:\Users\tjsdu>conda update -n base conda

2 - (base) C:\Users\tjsdu>conda update –all

3 - (base) C:\Users\tjsdu>conda install tensorflow

4 - (base) C:\Users\tjsdu>ipython

Python 3.7.4 (default, Aug 9 2019, 18:34:13) [MSC v.1915 64 bit (AMD64)]

Type 'copyright', 'credits' or 'license' for more information

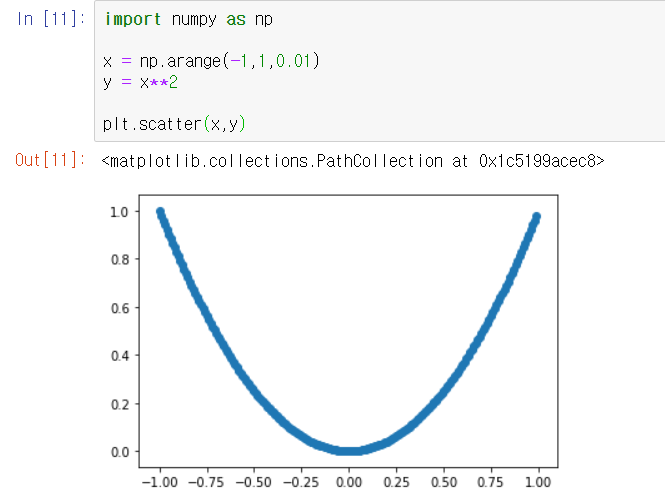
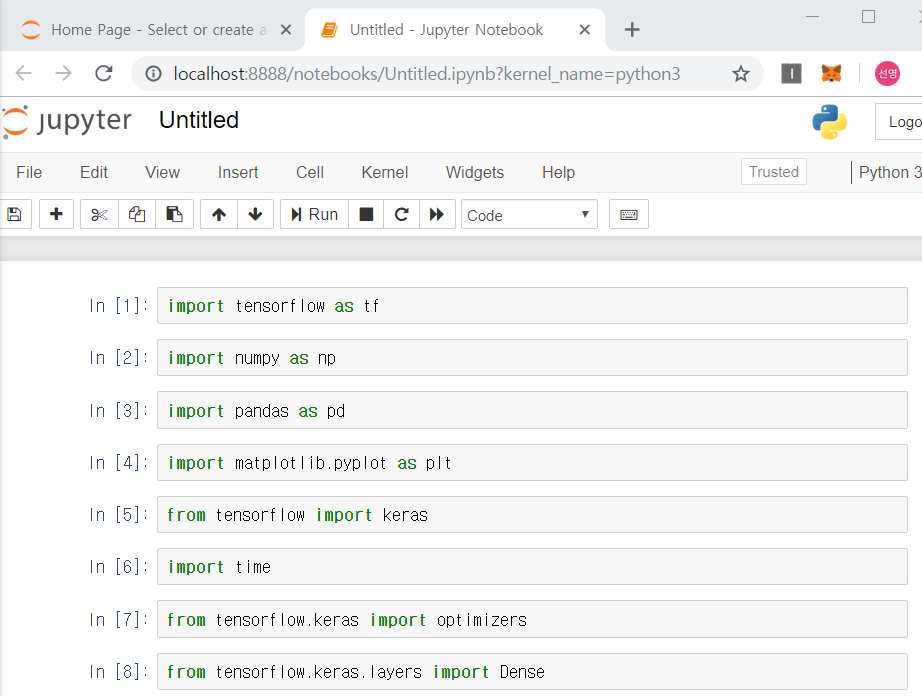
IPython 7.8.0 -- An enhanced Interactive Python. Type '?' for help.

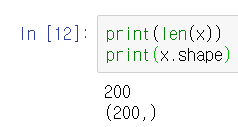
In [1]: import tensorflow as tf //ipython에서 tensorflow를 사용할 수 있도록 import

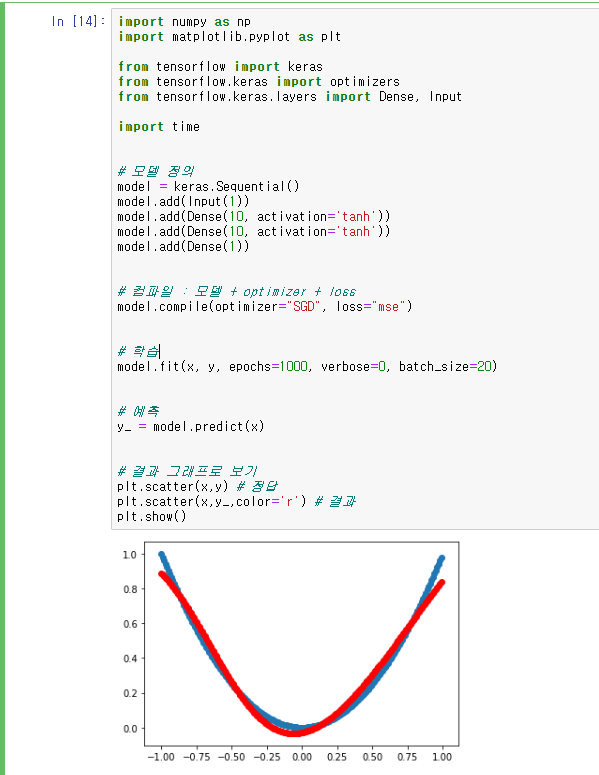
In [2]: exit

5 - (base) C:\Users\tjsdu>jupyter notebook

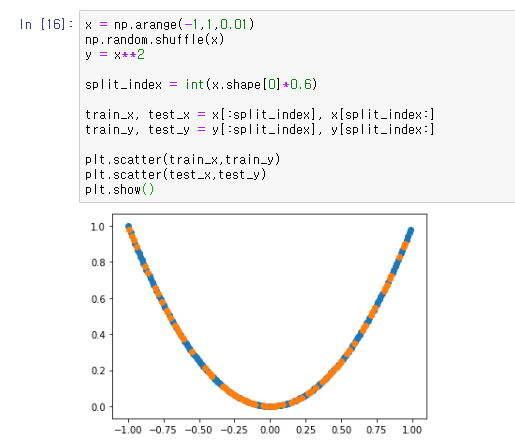
이렇게 해주면 jupyter notebook이 실행됩니다.

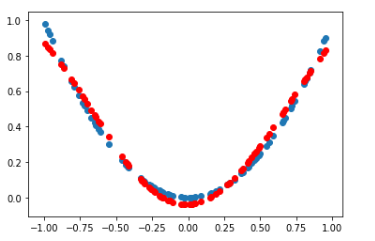
주피터 노트북을 실행한 후 새로운 python3를 만들어서 이렇게 실습에 필요한 모듈들을 import 해주었습니다. 제대로 설치가 완료되어 있기에, 오류메세지는 나타나지 않았습니다.

1차원 데이터 x와 x^2의 1차원 데이터 y를 준비해서 matplotlib를 사용하여 그래프로 나타내주었습니다. 데이터를 준비하는 과정입니다.

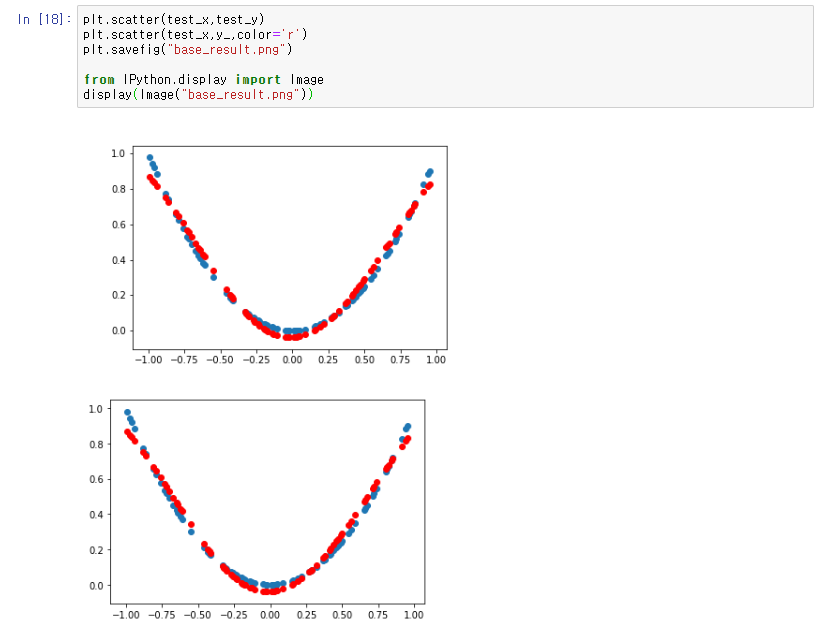
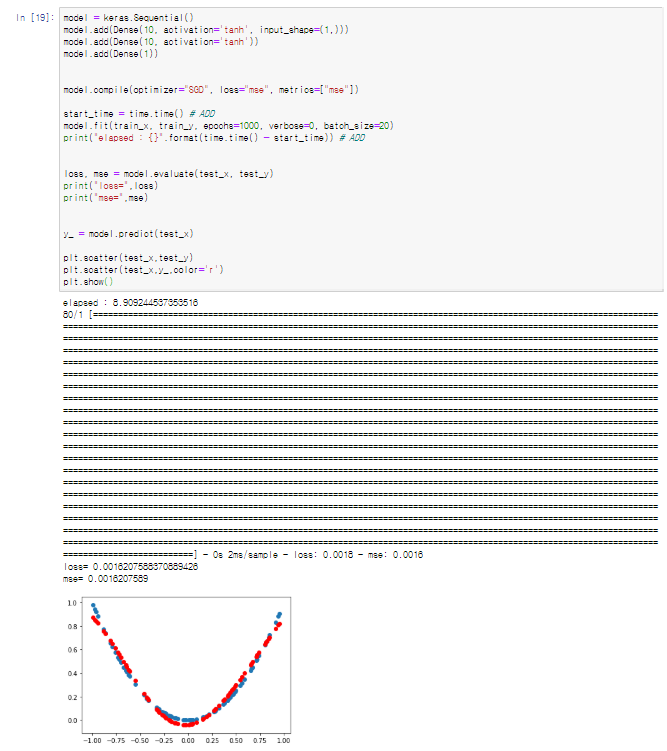
1차원 데이터 x는 200개입니다.

이번에는 모델을 정의하고, 컴파일하여, 그 모델을 학습하여 예측한 그래프를 나타내는 실습을 진행했습니다. 그러면 처음 데이터와 학습시킨 결과가 나옵니다.

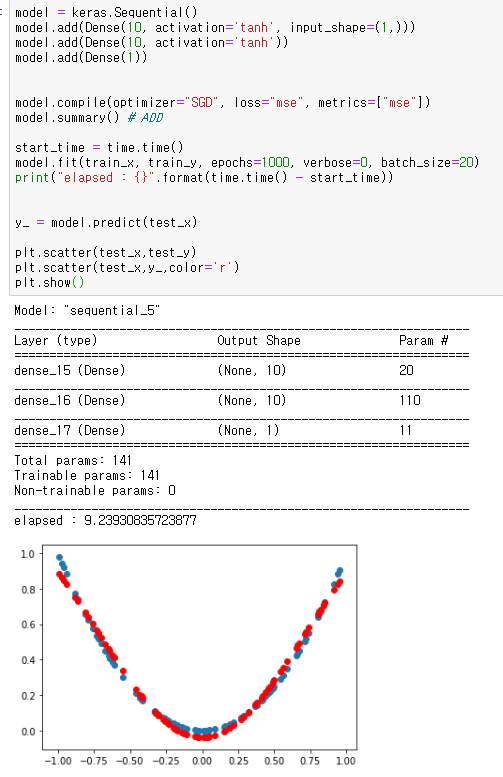
데이터를 train과 test로 나누어 준비합니다. 테스트셋으로 평가하기 위해서입니다.



train\_x와 train\_y로 학습시키고, evaluate()와 predict()에서는 test\_x, test\_y를 사용하였습니다. 그려진 그래프를 보면 test\_x의 점 사이에 빈 간격이 보이는데 그 간격이 train\_x에 해당합니다.

만들어진 그래프를 저장해주었습니다.

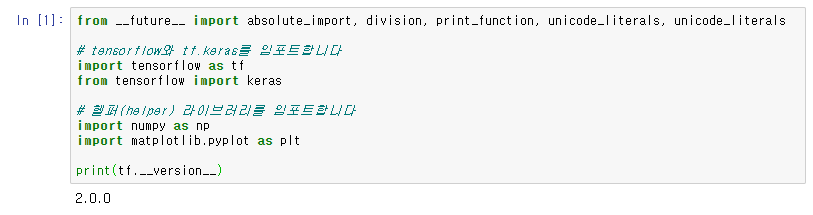
그리고 학습 시간을 출력하기 위해 time 모듈을 이용하여 elpased로 경과된 시간이 초 단위로 보이게 하였고, 학습 시간에 소요된 시간들을 그래프로 나타내주었습니다.

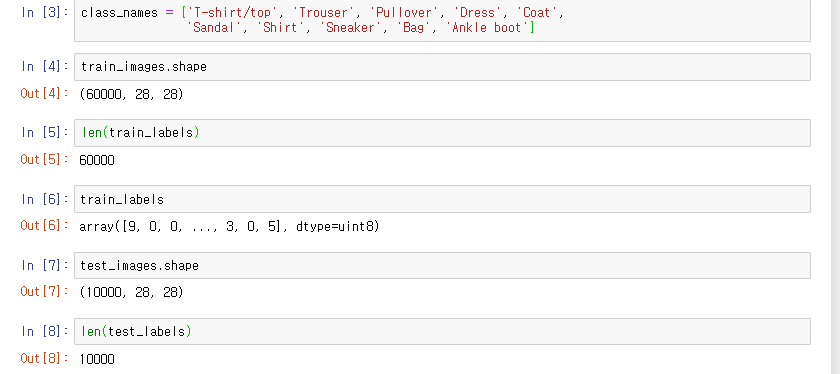
model.summary()의 네트워크 모양을 보기 위해서 작성된 코드입니다. 노드 10개,10개를 갖는 은닉층 2개가 있는데 모든 레이어는 activation(w\*x+b)의 형태로, 첫번째 은닉층은 20((1+1)\*10)=(입력수+1)\*10)개의 웨이트를 두번째 은닉층은 110=((10+1)\*10))개의 웨이트를 가집니다.

[실험4]

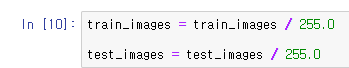
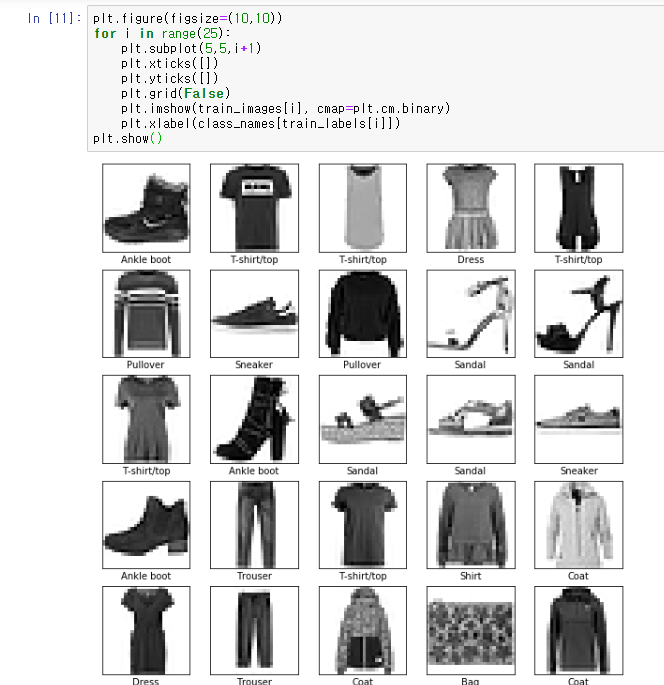
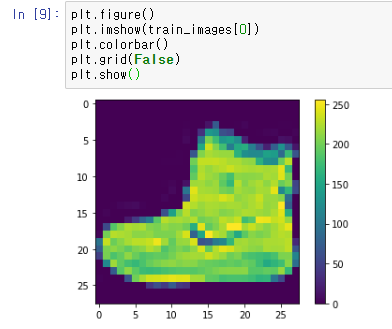
여기까지 튜토리얼로 나와있는 학습 예제들을 학습해보니, 조금 더 keras나 텐서플로우 명령어 숙지가 필요함을 느끼게 되어 추가적으로 코드 학습을 진행했습니다. Tensorflow에 나와있는 예제를 가지고 실습을 진행해봤습니다.

<https://www.tensorflow.org/tutorials/keras/classification?hl=ko>

먼저 import를 하고 tensorflow가 제대로 import되었는지를 확인하기 위해 version을 체크해봤습니다.

그리고 keras 데이터 중 fashion\_mnist라는 데이터를 로드해보았습니다. load\_data() 함수를 호출하면 네 개의 넘파이 배열이 반환된다고 합니다.

각 이미지는 하나의 레이블에 매핑되어있고, 클래스 이름이 들어있지 않아 이미지를 출력할 때 사용하려고 별도의 변수를 만들어 저장합니다. 또한 각 레이블은 0~9사이이며, 테스트 셋은 10000개, 이미지는 28\*28 픽셀이라고 합니다.

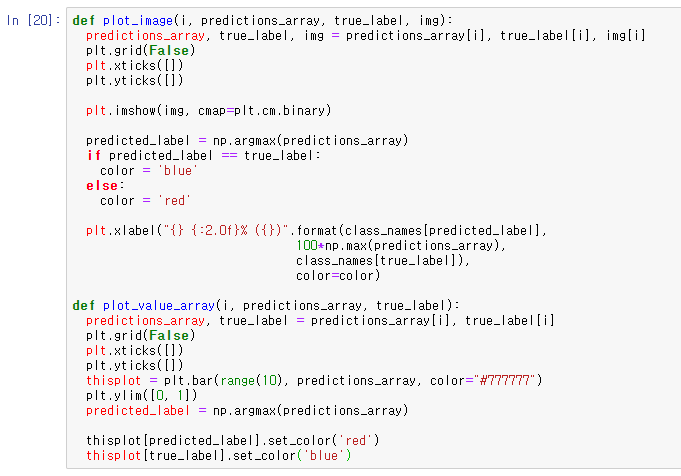
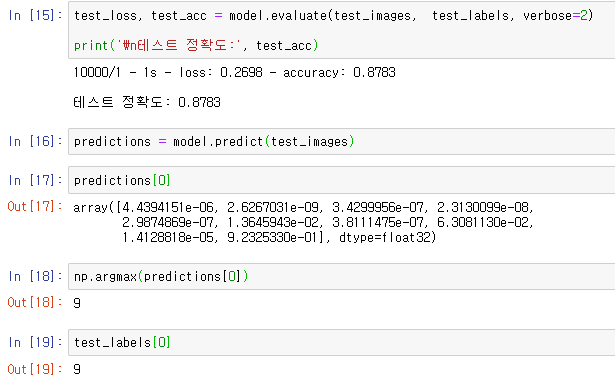
네트워크를 훈련하기 전에 데이터를 전처리해주었습니다. 픽셀값이 0~255 사이이기 때문에 255로 나누어 신경망 모델 주입 전 train\_images와 test\_images의 값을 0-1사이로 조정해주었습니다.

훈련 세트의 25개 이미지와 클래스 이름을 출력해보았습니다. 이렇게 네트워크 구성과 훈련할 준비를 마쳤습니다.

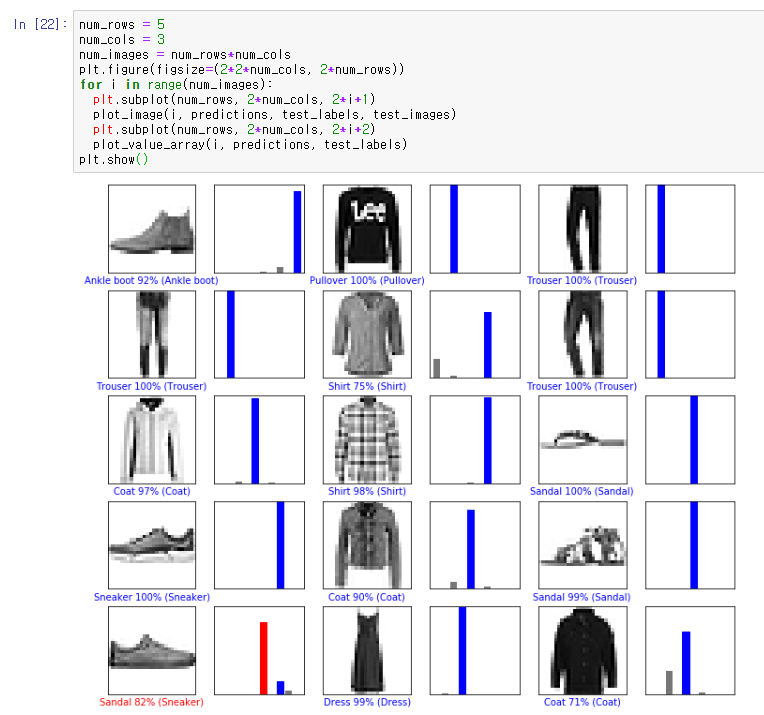
학습할 모델을 만들어줍니다. 첫번째 층에서는 2차원 배열의 이미지 포맷을 1차원 데이터로 변환해줍니다. (Flatten) 그리고, relu와 softmax 를 적용하여 층을 구성합니다. 10개의 확률을 반환해서 현재 이미지가 10개 클래스 중 하나에 속할 확률을 출력해줍니다.

그 다음으로 컴파일을 진행하는데, 손실함수는 loss의 값으로 훈련하는 동안 모델의 오차를 측정하고, 이 값이 최소화되도록 학습시킵니다. 옵티마이저는 데이터와 손실 함수를 바탕으로 모델의 업데이트 방법을 결정합니다. 지표는 훈련 단계와 테스트 단계를 모니터링하며, 분류된 이미지의 비율인 정확도를 사용한다고 합니다. 그리고 5번에 걸쳐 train\_images와 train\_labels 배열을 주입해 매핑하고 예측, 학습합니다.

모델이 훈련 되고 나면, 손실과 정확도 지표가 출력됩니다.

정확도를 출력해보니 0.88정도가 나오며, 훈련된 모델을 사용해서 예측해보니 이렇게 나왔습니다. 테스트 셋에 있는 각 이미지 레이블을 예측해 보니 이러한 값들이 나왔고, 가장 신뢰도가 높은 레이블을 찾으니 9번이 나왔습니다.

10개의 클래스에 대한 예측을 모두 그래프로 나타내보기 위해 이러한 함수를 작성했습니다.

몇개의 이미지 예측을 출력해 봤는데, 올바르게 예측된 레이블은 파란색이 나오고 잘못 예측된 레이블은 빨강색이 나온다고 합니다. 그리고 숫자는 신뢰도인데, 신뢰도 점수가 높을 때도 잘못 예측할 수 있다고 합니다. 여기서는 sandal이라는 레이블이 잘못 예측된 것을 볼 수 있습니다.

이렇게 tensorflow와 keras는 이미지를 예측하고, 학습시키는 데도 사용 할 수 있는 것을 예제를 실습해보며 확인해봤습니다.