"AI News That's Fit to Print"

The AI Newsletter

Bimonthly

Late Edition

K-water연구원 AI연구센터에서 정기적으로 발간하는 뉴스레터입니다. AI 분야의 뉴스 뿐 아니라 코딩실습과 기술정보 등 다양한 내용과 난이도로 담아냈습니다.

AI Newsletter No.12B

AI LAB, MONDAY, DEC 2, 2024

AI Tech

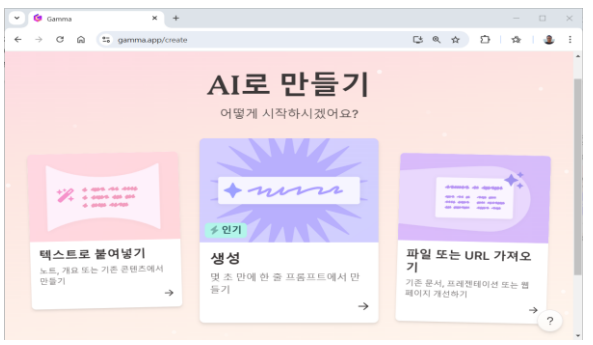
Tips

AI로 PPT 빠르게 완성하기, Gamma

최근 '감마(Gamma)'라는 AI 프리젠테이션 제작 기술에 대한 반응이 뜨겁다. Gamma는 사용자가 손쉽게 프리젠테이션, 문서, 웹페이지 등을 생성할 수 있도록 돕는 AI 도구이다. 직관적인 인터페이스 덕분에 템플릿이나 애니메이션 효과를 쉽게 추가할 수 있어 비주얼을 강조하는 자료를 만들기에도 좋다. 특히 자료를 드래그 앤 드롭 방식으로 배치하고, 웹을 통한 공동 작업이 가능해 실시간 피드백을 받아 PPT를 빠르게 개선할 수 있는 것이 강점이다. 감마는 영문과 한글을 포함한 모든 콘텐츠 디자인을 지원하며, 사용자가 원하는 PPT 종류와 주제를 입력하면 자동으로 목차를 추천하고 전체 내용을 생성해 준다. 사용자는 이를 기반으로 내용을 수정하고, 템플릿 종류와 테마를 선택하여 최종적으로 PPT를 완성할 수 있다.

QUICK TIPS Gamma를 사용하는 간단한 방법

- Gamma 홈페이지에 접속(https://gamma.app/ko)
- 무료로 가입하기 → Google 계정으로 로그인



Gamma에서 ppt 작성 방식을 선택하는 화면

Gamma에서 AI를 이용해 자동 생성한 PPT 작성 결과

파이썬 비동기 프로그래밍

비동기 프로그래밍은 작업이 완료될 때까지 기다리지 않고 즉시 다음 작업으로 넘어가며, 주로 네트워크 요청, 파일 입출력, 데이터베이스 쿼리와 같은 I/O-bound 작업에서 사용된다. 비동기 프로그래밍에서는 하나의 작업이 완료될 때까지 다른 작업을 수행할 수 있으므로 동시에 여러 작업을 처리하는 것처럼 보인다. 코루틴(Coroutines)은 일반적인 함수처럼 값을 반환하는 대신, 중간에 특정 시점에서 await 키워드를 통해 실행을 일시 중지하고 다른 작업을 실행할 수 있는 특별한 함수이다. 코루틴은 async def로 정의되며, 호출 시 즉시 실행되지 않고 코루틴 객체를 반환한다. 코루틴 내에서 await 키워드를 사용하여 다른 비동기 작업이 완료될 때까지 기다릴 수도 있다. 이벤트 루프(Event Loop)는 비동기 작업을 스케줄링하고 실행하는 역할을 한다. Python의 asyncio 모듈을 사용하면 쉽게 이벤트 루프를 사용할 수 있다. 다음은 비동기 프로그래밍의 기본 구조를 보여주는 예제이다.

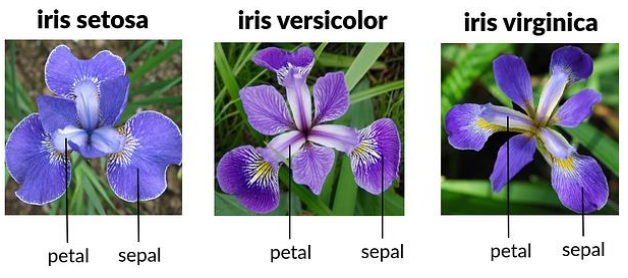
```
import asyncio
# 비동기 함수 정의
async def async_function_1():
print("Function 1: 시작") # 함수 1 시작 알림 출력
await asyncio.sleep(2) # 비동기적으로 2초 대기
print("Function 1: 완료") # 대기 완료 후 함수 1 완료 알림 출력

# 또 다른 비동기 함수 정의
async def async_function_2():
print("Function 2: 시작") # 함수 2 시작 알림 출력
await asyncio.sleep(1) # 비동기적으로 1초 대기
print("Function 2: 완료") # 대기 완료 후 함수 2 완료 알림 출력

# 메인 비동기 함수 정의
async def main():
# 두 개의 비동기 함수를 동시에 실행
await asyncio.gather(async_function_1(), async_function_2())
asyncio.run(main()) # 이벤트 루프를 시작하고 main 함수를 실행

# 실행 결과:
Function 1: 시작 --> 코드 실행과 동시에 이 메시지 출력
Function 2: 시작 --> 코드 실행과 동시에 이 메시지 출력
Function 2: 완료 --> 비동기적으로 1초 대기 후 이 메시지 출력
Function 1: 완료 --> 비동기적으로 2초 대기 후 이 메시지 출력
```

Hands-on AI Project

설명가능한 인공지능, XAI 적용해 보기

이번 Hands-on에서는 Iris 데이터셋을 대상으로 랜덤 포레스트 모델을 이용한 분류 문제를 풀어보고, 모델의 결과를 XAI 기법 중 하나인 SHAP으로 해석하는 방법을 다룬다. 실습 코드는 구글 코랩 환경을 기준으로 작성되었다.

Iris 데이터셋은 인공지능 분야에서 자주 활용되는 유명한 데이터셋이다. 총 150개 샘플로, Setosa, Versicolor, Virginica라는 세 가지 붓꽃 종의 꽃잎 길이(petal length), 꽃잎 너비(petal width), 꽃받침 길이(sepal length), 꽃받침 너비(sepal width)로 구성되어 있다. 랜덤 포레스트 모델은 회귀 및 분류 문제에 활용 가능한 트리 기반의 앙상블 모델로 복수의 개별 모델 출력을 종합하여 최종 결과를 도출한다. 내부에 개별 모델이 100개 들어있는 랜덤 포레스트 모델을 이용해 사과, 배, 토마토를 분류하는 문제를 생각해 보자. 학습을 마친 후 어떤 과일을 보고 각각 70, 20, 10개의 개별 모델들이 사과, 배, 토마토로 판단했다면, 랜덤 포레스트 모델의 최종 분류 결과는 사과가 된다. XAI는 블랙박스로 인식되는 AI 모델의 결과를 설명하기 위한 방법론으로, LIME, SHAP 등 다양한 기법이 존재한다. 이번호의 Hands-on에서 사용할 SHAP(Shapley Additive exPlanations)은 게임이론을 기반으로 모델의 출력에 대한 각 feature의 기여도를 샤플리 값(Shapley value)을 이용하여 정량화 한다. 모델에 상관없이 결과를 해석할 수 있는 특성으로(model-agnostic) 활용도가 높다. 이제 본격적인 실습으로 들어가 보자!



※ 전체 Source 코드 및 결과는 분량상 생략, 아래 링크를 참고해주세요. **구글 코랩에서 열기**

Hands-on AI Project

① 실습에 활용할 패키지와 클래스, 함수를 불러온다. 사용된 패키지는 구글 코랩 환경에 모두 내장되어 있어 따로 설치할 필요는 없다.

```
import shap
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
```

② 실습에 활용할 Iris 데이터셋을 불러온다. Iris 데이터셋은 다양한 소스에서 불러올 수 있지만, 실습에서는 scikit-learn 패키지에 기본적으로 저장되어 있는 Iris 데이터셋을 사용한다.

```
# iris 데이터셋을 불러온 뒤 feature를 X로, target을 y로 선언
iris = load_iris()
X = iris.data
y = iris.target
```

③ scikit-learn 패키지에서 불러온 train_test_split 함수를 이용하여 Iris 데이터셋을 학습 및 테스트 데이터로 분리한다. 전체 데이터셋 중 70%를 학습에, 30%는 테스트에 사용한다. 무작위 샘플링이 이루어지도록 shuffle은 True로, random_state는 임의로 1111로 설정한다.

```
# 전체 데이터셋에서 학습(70%) 및 테스트(30%) 데이터를 무작위 추출
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=1111, shuffle=True)
```

④ train_test_split 함수가 Iris 데이터셋을 어떻게 학습 및 테스트 데이터로 나누었는지 확인해보자.

```
df2 = pd.DataFrame(X_train,columns=iris.feature_names)   # 데이터프레임 생성
df2['species'] = iris.target_names[y_train]              # 붓꽃 종 컬럼 추가
df2.head()                                               # 상위 5개 행을 확인
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.2	3.4	1.4	0.2	setosa
1	6.4	2.7	5.3	1.9	virginica
2	4.8	3.4	1.9	0.2	setosa
3	6.5	3.0	5.5	1.8	virginica
4	4.9	3.1	1.5	0.2	setosa

⑤ scikit-learn 패키지의 RandomForestClassifier 클래스를 불러온다. 랜덤 포레스트 모델을 선언하고 학습 데이터로 학습을 진행한다. 랜덤 포레스트 모델은 다양한 하이퍼파라미터를 가질 수 있지만, 실습에서는 개별 모델의 수(n_estimator)만 10으로 설정하고 나머지는 기본값으로 설정한다.

```
# 모델 선언 및 데이터 학습
rfc = RandomForestClassifier(n_estimators=10, random_state=1111)
rfc.fit(X_train, y_train)
```

⑥ 학습을 마친 랜덤 포레스트 모델의 성능을 확인해보자. Scikit-learn 패키지에서 불러온 accuracy_score 및 classification_report 함수를 활용한다. 랜덤 포레스트 모델은 93%의 분류 정확도(accuracy)를 보였다. 각각의 붓꽃 종에 대한 분류 성능을 classification report의 정밀도(precision), 재현율(recall), f1 score와 같은 성능 지표로 더 자세하게 확인할 수 있다.

```
y_pred = rfc.predict(X_test) # 테스트 데이터를 사용하여 분류 결과 도출
 # 분류 결과의 정확도 평가
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred, target_names=iris.target_names)
# 모델 성능 표출
print(f"Accuracy: {accuracy:.2f}")
print("\nClassification Report:\n", report)
```

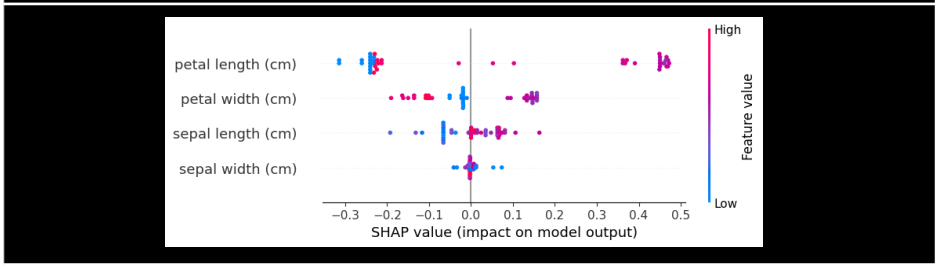
```
Accuracy: 0.93

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        14
  versicolor       0.86      1.00      0.92        18
   virginica       1.00      0.77      0.87        13

    accuracy                           0.93        45
   macro avg       0.95      0.92      0.93        45
weighted avg       0.94      0.93      0.93        45
```

⑦ 이제 랜덤 포레스트 모델이 예측한 결과를 SHAP 기법을 적용하여 확인해보자. shap 패키지의 TreeExplainer 클래스를 사용하여 각 feature의 랜덤 포레스트 모델 출력에 대한 기여도를 정량화 한다.

```
# SHAP_values를 계산합니다.
explainer = shap.TreeExplainer(rfc)
shap_values = explainer.shap_values(X_test)
reshaped_shap_values = shap_values.transpose(2, 0, 1)
```

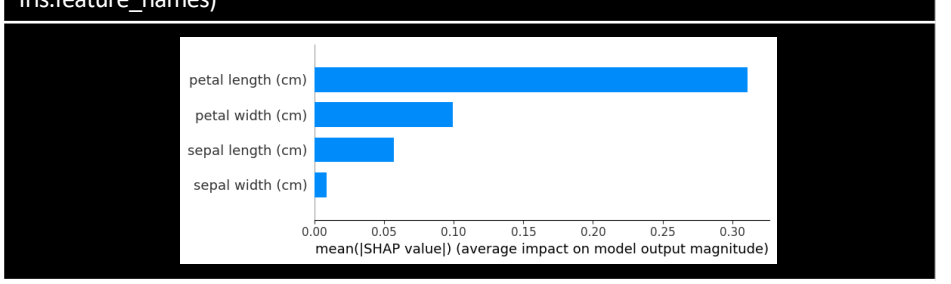
⑧ 계산된 샤플리 값을 시각화해 보자. Shap 패키지는 다양한 시각화 함수를 제공하는데, 여기서는 summary_plot 함수를 사용하여 랜덤 포레스트 모델이 Versicolor 종으로 판단한 경우 각 feature의 기여도를 확인해보겠다. 물론 네 가지 feature를 모두 참고했지만, 가장 큰 영향을 끼친 feature는 꽃잎 길이(petal length)가 된다. 대체로 petal length가 작을수록 Versicolor 종이 아닌 것으로, 클수록 Versicolor 종으로 판단했다고 해석할 수 있다.

```
# versicolor 종의 분류에는 어떤 feature가 큰 영향을 끼쳤는지 확인
shap.summary_plot(reshaped_shap_values[1], X_test,
feature_names=iris.feature_names)
```



⑨ summary_plot 함수에 plot_type='bar'를 추가해 보자. Versicolor 종의 식별에 꽃잎 길이, 꽃잎 너비, 꽃받침 길이, 꽃받침 너비 순으로 영향을 준 것을 확인할 수 있다. Versicolor 종을 예시로 확인해보았지만, 다른 종들도 모두 꽃잎의 길이가 가장 큰 영향을 끼쳤다. 만약 꽃받침 너비(sepal width)만 주어진다면 종 분류는 거의 불가능할 것으로 예상된다.

```
# versicolor 종의 분류에는 어떤 feature가 큰 영향을 끼쳤는지 확인
shap.summary_plot(reshaped_shap_values[1], X_test, plot_type='bar', feature_names=
Iris.feature_names)
```



이제 실습 서두의 그림으로 돌아가 보자. 붓꽃 종을 구분하려면 꽃잎(petal)과 꽃받침(sepal) 중 어떤 것을 확인하는 것이 좋은가? 이것으로 이번 호의 실습을 마치도록 하겠다.