**산학협력 프로젝트 개인 결과보고서**

**주제 : UIPATH 와 머신러닝을 활용한 자동 입찰 시스템**

소속 : 경북대학교 IT대학 컴퓨터학부

이름 : 박주홍

학번 : 2015117246

산학협력 기업 : UIPATH

**# 목차**

**1. 서론 및 설명**

**2. 베이지안 모델 (초기모델)**

**3. 데이터 전처리**

**4. 상위 데이터 선택하기**

**5. pname 모델 만들기**

**6. pmake 모델 만들기**

**7. 모델과 토크나이저 불러와서 사용하기**

**8. 실제 사용 사례**

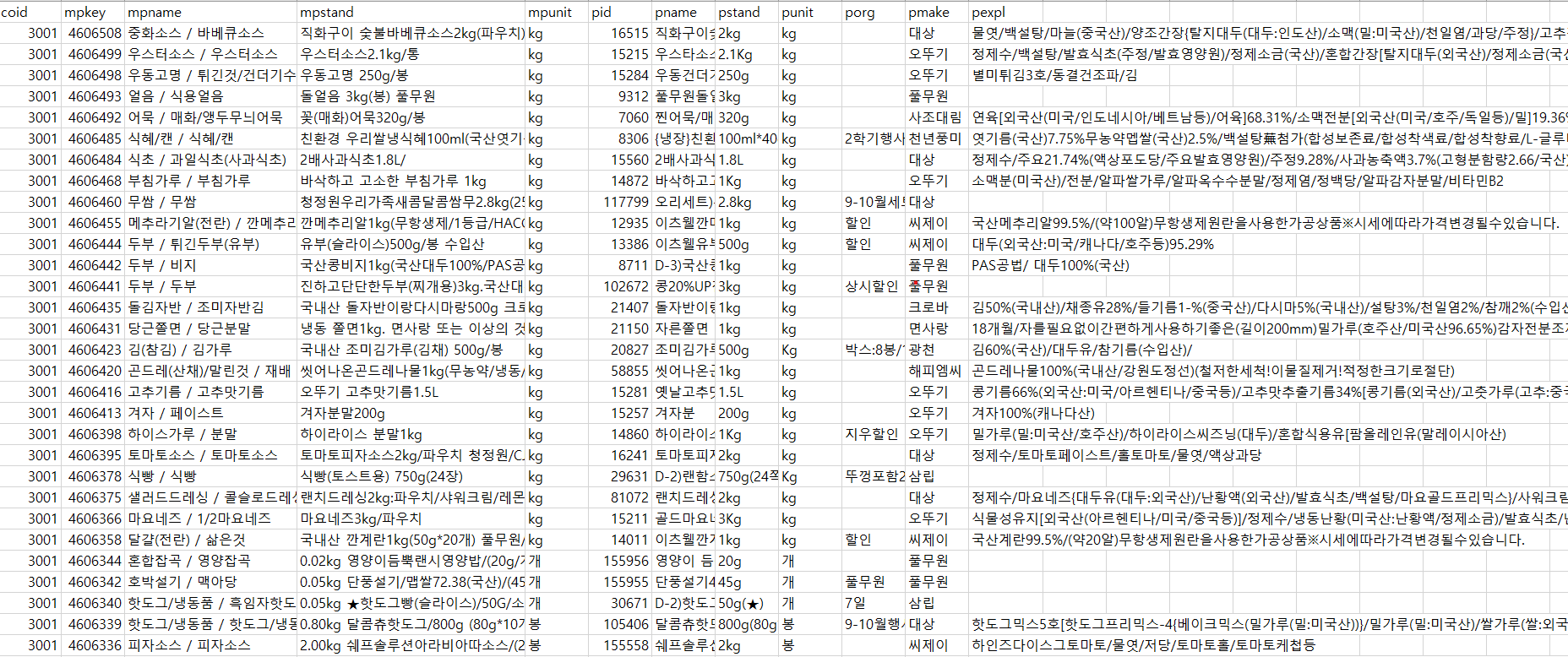
**1. 서론 및 설명**

본 프로젝트는 크게 UIPATH RPA + 기계학습 + 서버로 나뉩니다.

저는 프로젝트에서 텍스트 기반의 기계학습을 맡아서 수행하게 되었습니다.

아래는 저희가 UIPATH 에서 제공받은 Excel 형식 데이터입니다.

총 535412 개의 데이터로 구성되어 있습니다.



좀 더 자세히 보기 위해서 한 개의 데이터를 가져와서 설명하겠습니다.

EMB00000fb83be3

여기서 중심적으로 봐야하는 것은 mpname, mpstand, pname, pmake 입니다.

pname 은 실제 제공된 제품명을 의미합니다.

pmake 는 해당 제품을 판매하는 실제 판매사를 의미합니다.

mpname 은 학교 급식실에서 필요로 하는 제품의 카탈로그를 의미합니다.

mpstand 는 학교 급식실에서 쓴 문구로 어떠한 제품이 필요하다는 내용을 담고 있습니다. 제각각 형식이 다르며 의미 없는 문구와 수식어가 많이 담겨있습니다.

사람의 경우는 mpname 과 mpstand 를 보고서 적합한 pname 과 pmake 를 유추할 수 있지만 그 양이 많기 때문에 시간이 매우 오래 걸리게 됩니다.

따라서 기계학습을 통해 학교 급식실에서 특정한 형식 없이 작성한 문구에서 적합한 제품명과 해당 제품의 판매사를 유추하는 것을 목표로 하였습니다.

**2. 베이지안 모델 ( 초기 모델 )**

처음 사용한 모델은 베이지안 모델이었습니다. 베이즈 정리를 사용한 모델로 P(A|B) = P(B|A) \* P(A) / P(B) 와 같이 Prior 확률과 Likelihood 를 안다면 Posterior 확률을 구할 수 있다는 간단한 방법으로 구현한 텍스트 모델입니다. 최종 결과로 사용된 모델이 아니므로 간단하게나마 결과만 보이지면 다음과 같이 나오게 됩니다.

|  |
| --- |
| 입력 mpstand + 입력 mpname  = 파인애플 과일 쥬스 과즙 음료 과즙 망고 농축액 파인애플 농축액 천년 풍 밉다 자 연속 망고 파인애플  추측 pname = 골드파인애플골든엔골드  실제 pname = 냉장자연속의파인애플망고천년풍밉다  입력 mpstand = 망고 농축액 파인애플 농축액 천년 풍 밉다 자 연속 망고 파인애플  추측 pmake = 천년풍밉다  실제 pmake = 천년풍밉다  (pname, pname 이 나오게 된 확률, pmake, pmake 가 나오게 된 확률) =  ('골드파인애플골든엔골드\n', -284.98850180902974) ('천년풍밉다\n', -174.09845111372786) |

결과는 잘 나왔지만 모델을 추출할 수 없었고 매번 전처리를 해줬어야 했습니다. 또한 모든 카테고리별로 확률이 분산되었기 때문에 모든 확률이 log 를 취했음에도 작은 값으로 표현되었고 그 중에서 그나마 확률이 가장 큰 값을 나타내는 카테고리를 추측 결과로 나타냈기 때문에 저희가 원하는 형태의 확률 형태인 0~100% 사이의 값으로 나타내기까지의 표현 방법 구현이 미숙하였습니다. 따라서 다른 모델을 사용하는 것으로 결정하였습니다.

더불어 데이터 전처리를 미리 해야겠다는 필요성을 느꼈습니다. 또한 기존에는 pname 을 추측하기 위해서 mpstand 와 mpname 을 사용하고, pmake 를 추측하기 위해서 mpstand 만을 사용했는데 굳이 mpstand 만 사용할 필요 없이 mpname 도 사용해도 되겠다는 피드백을 얻었습니다.

**3. 데이터 전처리**

데이터를 사용하기 전에 전처리를 하는 것이 필요했습니다. 의미 없는 문구와 학습을 하는데 필요 없는 문장들을 제외하기 위해서 자연어 처리 작업을 했습니다. konlpy 를 사용해서 필요한 요소들만 남게끔 새로운 CSV 파일을 만들었습니다.

EMB00000fb83be4

위의 예시를 다시 보면 데이터의 특성을 잘 나타내는 파라미터만을 사용하기 위해서, 예를 들어 숫자는 그 데이터의 특성을 잘 나타내는 파라미터가 아니라고 판단하여 제외하였습니다.

- pname 의 내용은 "중화소스 / 바베큐소스"입니다.

'/' 와 같은 요소를 포함해서 의미 없는 문구를 없애기 위해 "Punctuation" 과 "Number", "Alpha", "Foreign" 에 해당되는 형태소들을 제외하고 문장을 재구성하였습니다. 그 결과로 "중화 소스 바베큐 소스" 가 나왔습니다.

|  |  |
| --- | --- |
| 전처리 전 | "중화소스 / 바베큐소스" |
| 제외 요소 | "Punctuation", "Number", "Alpha", "Foreign" |
| 전처리 후 | "중화 소스 바베큐 소스" |

- mpstand 의 내용은 "직화구이 숯불바베큐소스2kg(파우치) : 스모크오일/불맛나는 바베큐소스. 청정원 또는 그이상의것" 입니다. 여기서 "Josa", "Eomi", "Punctuation", "Foreign", "Number", "Alpha" 를 제외하고 문장을 재구성하였습니다.

그 결과로 "직화 구이 숯불 바베큐 소스 파우치 스모크 오일 불맛나 는 바베큐 소스 청정 원 또는 그 이상 의 것" 이 나왔습니다.

|  |  |
| --- | --- |
| 전처리 전 | "직화구이 숯불바베큐소스2kg(파우치) : 스모크오일/불맛나는 바베큐소스. 청정원 또는 그이상의것" |
| 제외 요소 | "Josa", "Eomi", "Punctuation", "Foreign", "Number", "Alpha" |
| 전처리 후 | "직화 구이 숯불 바베큐 소스 파우치 스모크 오일 불맛나 는 바베큐 소스 청정 원 또는 그 이상 의 것" |

- pname 의 내용은 "직화구이숯불바베큐소스2kg" 입니다. 여기서 "Josa", "Eomi", "Punctuation", "Foreign", "Number", "Alpha" 를 제외하고 문장을 재구성하였습니다. 또한 서로 같은 제품임에도 띄어쓰기로 인해서 다른 제품으로 분류될 것을 방지하기 위해서 띄어쓰기를 사용하지 않고 나타내기로 결정했습니다.

그 결과로 "직화구이숯불바베큐소스" 가 나왔습니다.

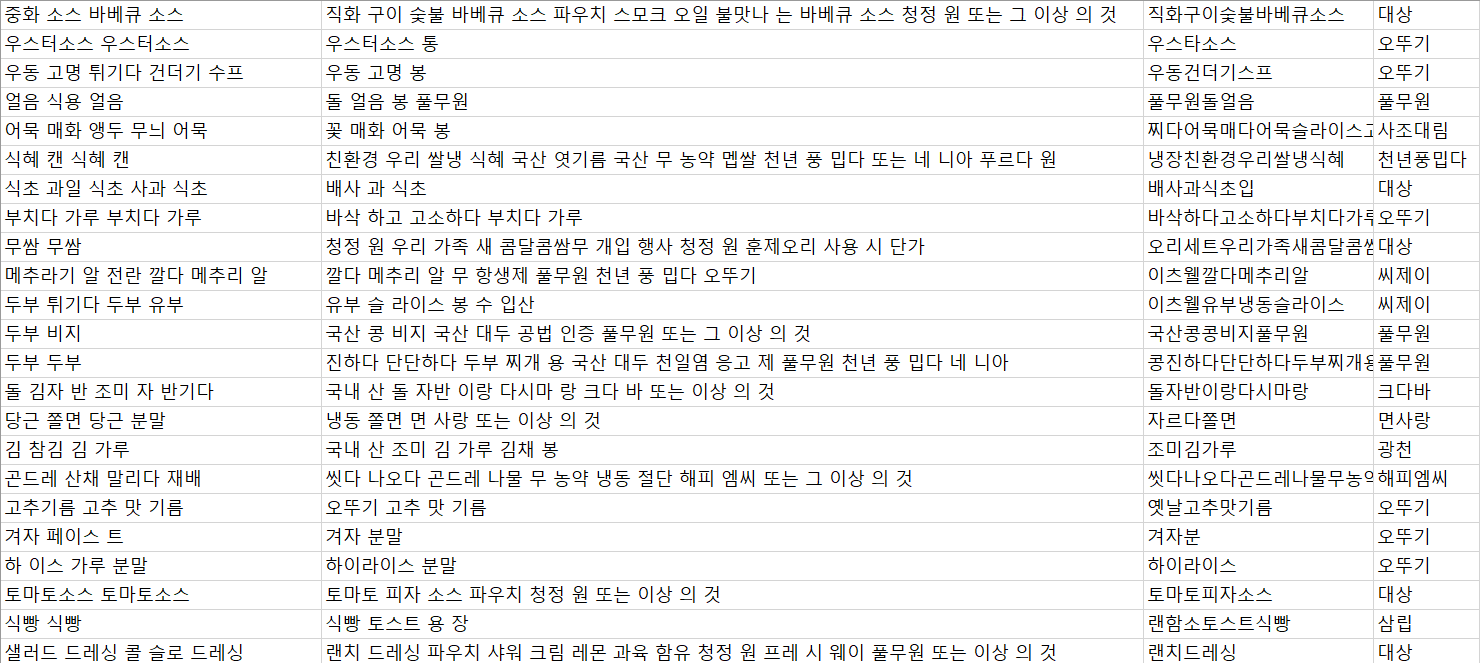
|  |  |
| --- | --- |
| 전처리 전 | "직화구이숯불바베큐소스2kg" |
| 제외 요소 | "Josa", "Eomi", "Punctuation", "Foreign", "Number", "Alpha" |
| 추가 작업 | 띄어쓰기 없애기 |
| 전처리 후 | "직화구이숯불바베큐소스" |

- pmake 의 내용은 "대상" 입니다. 이 예제에서는 잘 나타나지 않지만 다른 의미 없는 형태소들이 함께 적힌 경우가 있었습니다. 따라서 이 역시 "Josa", "Eomi", "Punctuation", "Foreign", "Number", "Alpha" 를 제외하고 문장을 재구성하였습니다. pname 과 마찬가지로 띄어쓰기로 인해서 다른 제조사로 분류될 것을 방지하기 위해서 띄어쓰기를 사용하지 않고 나타내었습니다.

그 결과로 "대상" 이 나왔습니다.

|  |  |
| --- | --- |
| 전처리 전 | "대상" |
| 제외 요소 | "Josa", "Eomi", "Punctuation", "Foreign", "Number", "Alpha" |
| 추가 작업 | 띄어쓰기 없애기 |
| 전처리 후 | "대상" |

전처리 작업이 완료된 후의 CSV 파일입니다.



**4. 상위 데이터 선택하기**

미리 말씀드리자면 머신러닝 모델을 2개를 만들어서 하나는 pname 을 추측하는, 나머지 하나는 pmake 를 추측하도록 하였습니다. 즉 첫번째 모델은 Label 로 pname 을 사용하였고, 두번째 모델은 Label 로 pmake 를 사용하였습니다.

Oracle DBMS 내에서 uipath 라는 Table 을 생성하였고 이렇게 정규화된 데이터를 Oracle DBMS 에 Import 시켰습니다. 그 이유로는 전체 데이터를 학습시키려고 하니 용량이 컸기 때문에 생성에 필요한 Dimension 이 너무나 커져서 제 노트북에서는 학습이 불가능하였기 때문에 상위 데이터를 선택할 수 밖에 없었습니다. 또한 Label 에 해당되는 데이터들이 많은 것은 충분히 많았지만 적은 것은 너무 적었기 때문에 가장 많이 출현한 Label 들 중에서 상위 데이터들만을 선택할 수 밖에 없었습니다.

|  |
| --- |
| pname 중에 특정 Label 에 속하는 데이터 중 갯수가 200개 넘는 데이터 추출 |
| select uipath.\*  from uipath, (select \*  from( select pname, count(pname) as pname\_count  from uipath  group by pname  order by pname\_count desc)  where pname\_count >= 200 ) subquery  where uipath.pname = subquery.pname; |
| pmake 중에 특정 Label 에 해당되는 데이터 중 갯수가 200개 넘는 데이터 추출 |
| select uipath.\*  from uipath, (select \*  from( select pmake, count(pmake) as pmake\_count  from uipath  group by pmake  order by pmake\_count desc)  where pmake\_count >= 200 ) subquery  where uipath.pmake = subquery.pmake; |

pname query 를 사용해서 총 159107 개의 데이터를 추출할 수 있었습니다.

{파슬리후레이크 : 1434}, {우스타소스 : 1284}, {순후추 : 1183}, ... {우리쌀떡국떡개 : 201}, {블루베리엔요 : 200}

pmake query 를 사용해서는 총 391117 개의 데이터를 추출할 수 있었습니다

{오뚜기 : 40765}, {대상 : 26579}, {씨제이 : 24081}, ... , {진주원예농협 : 201}, {준유통 : 200}

이와 같이 추출한 데이터들을 각각 CSV 파일로 저장하였습니다.

**5. pname 모델 만들기**

우선 pname 를 추측하는 모델을 만들었습니다. 단계는 다음과 같습니다.

(단계 1) 윗 단계에서 최종적으로 만든 CSV 파일을 불러옵니다.

(단계 2) 각 row 마다, mpname 열에 있는 데이터와 mpstand 열에 있는 데이터를 하나의 변수로 합칩니다.

(단계 3) pname 에 해당되는 keras tokenizer 를 만들고, mpbase 에 해당되는 keras tokenizer 를 만듭니다.

(단계 3-1) 각 tokenizer 마다 fit\_on\_texts() 함수를 사용하여 단어들의 인덱스를 구축합니다.

(단계 3-2) 각 tokenizer 마다 texts\_to\_sequences() 함수를 사용하여 문자열을 정수 인덱스의 리스트로 변환합니다. 그 결과로 pname\_sequences 와 mpbase\_sequences 가 나오게 됩니다.

(단계 4) 전체 159107 개의 데이터 중에서 학습에 사용될 데이터와 테스트에 사용될 데이터를 80 : 20 비율로 나눠줍니다. 따라서 127285 개의 데이터는 학습에 사용되고 31822 개의 데이터는 테스트에 사용됩니다.

(단계 5) mpbase 는 source 에 해당되고, pname 은 label 에 해당됩니다.

source 에 해당되는 데이터들은 정의된 vectorize\_sequences() 함수를 사용해서 정수 인덱스를 벡터로 변환시키고, label 에 해당되는 데이터들은 정의된 to\_one\_hot() 함수를 사용해서 정수 인덱스를 벡터로 변환시킵니다. 이 때 vectorize\_sequences() 와 to\_one\_hot() 의 dimension 은 각각 10000, 414 로 최대한 제가 사용하고 있는 PC 의 RAM 이 지원할 수 있는 dimension 을 사용하였습니다.

(단계 6) keras 의 model 과 layer 를 import 하여서 모델을 만듭니다.

model.add(layers.Dense(512, activation='relu', input\_shape=(10000, )))

model.add(layers.Dense(512, activation='relu'))

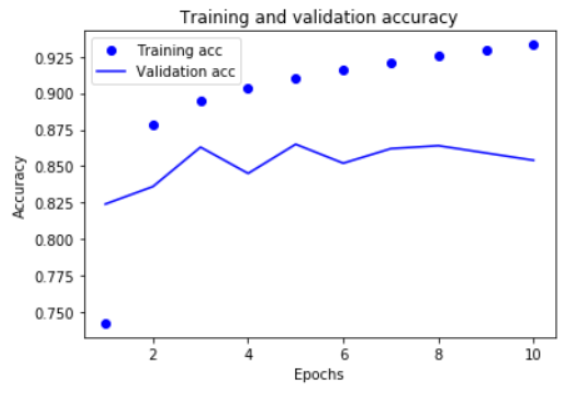
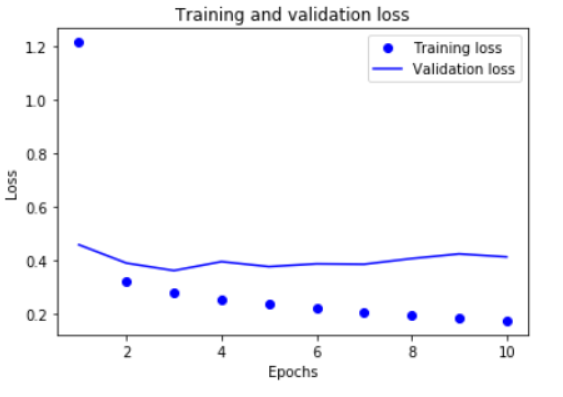
relu 활성화 함수는 0보다 작은 값이 나온 경우 0을 반환하고 0보다 큰 값이 나온 경우 그 값을 그대로 반환하게 합니다. 따라서 hidden layer 로 Relu 를 적용하여 정확도를 높였습니다.

model.add(layers.Dense(414, activation='softmax'))

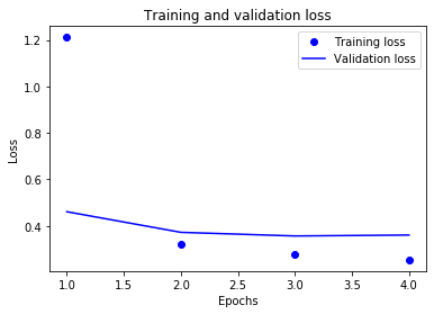
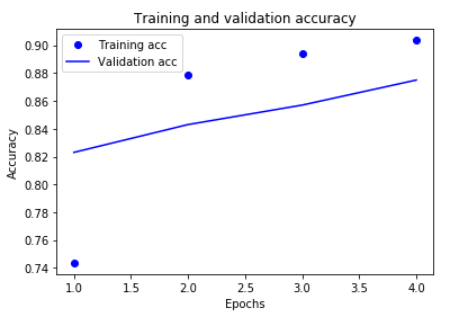
마지막 Dense 층의 크기가 414 로 각 입력 샘플에 대해서 414 차원의 벡터를 출력하게 됩니다. 또한 softmax 활성화 함수를 사용해서 각 입력 샘플 마다 414 개의 출력 클래스에 대한 확률 분포를 출력합니다.

(단계 7) 손실함수로는 categorical\_crossentropy 를 사용하여 네트워크가 출력한 확률 분포와 실제 레이블의 분포 사이의 거리를 측정하여 최소화시키는 방향으로 모델을 컴파일하였습니다.

(단계 8) 훈련 데이터에서 1000개의 샘플을 따로 떼어서 검증 세트로 설정하였고, epoch 를 10 으로 설정하고 측정한 결과는 다음과 같았습니다.



4번째 epoch 이후로 과대 적합이 발생하였기 때문에 epoch 를 4로 재설정하고 모델을 만들었습니다. 그 결과는 다음과 같습니다.



(단계 9) 테스트 데이터를 사용해서 모델의 정확도를 분석한 결과

31822/31822 [==============================] - 6s 191us/step

[0.3098828221737016, 0.888724781601383]

로 88.8% 의 정확도를 나타남을 확인할 수 있었습니다.

(단계 10) 이렇게 학습된 모델을 keras 의 load\_model 을 import 해서 model.save() 함수를 통해 따로 저장할 수 있었습니다.

(단계 11) 사용되는 tokenizer 를 keras\_preprocessing.txt 의 tokenizer\_from\_json 을 import 해서 to\_json() 함수를 사용하여 따로 저장할 수 있었습니다.

**6. pmake 모델 만들기**

pmake 의 경우도 위와 같은 절차를 따라서 만들었습니다.

다만 pmake query 문을 사용해서 얻은 데이터가 391117 개 였는데 이를 벡터화시키기 위해서 dimension 을 최대로 올려봐도 RAM 이 감당하지 못해서 위의 데이터 중에 랜덤하게 160001 개의 데이터를 추출해서 사용했습니다.

위의 절차에서 (단계 5)에 해당되는 내용이 다음과 같이 변경되어 작용했습니다.

: mpbase 는 source 에 해당되고, pmake 는 label 에 해당됩니다.

source 에 해당되는 데이터들은 정의된 vectorize\_sequences() 함수를 사용해서 정수 인덱스를 벡터로 변환시키고, label 에 해당되는 데이터들은 정의된 to\_one\_hot() 함수를 사용해서 정수 인덱스를 벡터로 변환시킵니다. 이 때 vectorize\_sequences() 와 to\_one\_hot() 의 dimension 은 각각 14000, 264 로 최대한 제가 사용하고 있는 PC 의 RAM 이 지원할 수 있는 dimension 을 사용하였습니다.

따라서 model 을 정의할 때도

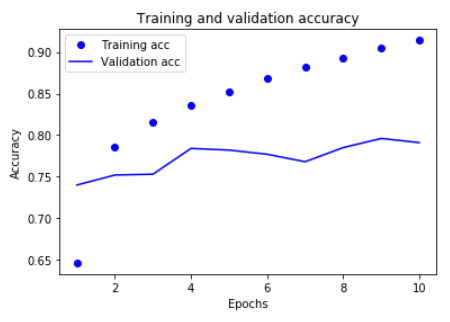
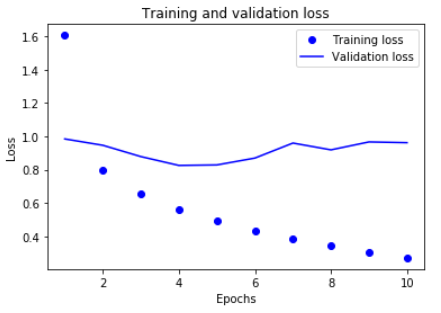
odel.add(layers.Dense(512, activation='relu', input\_shape=(14000, )))

model.add(layers.Dense(512, activation='relu'))

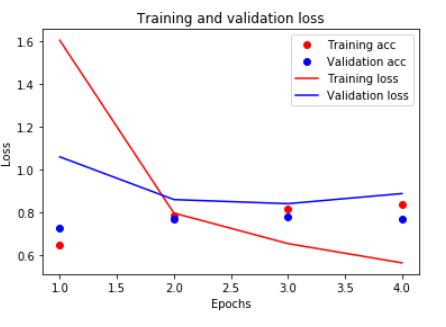
model.add(layers.Dense(264, activation='softmax'))

와 같이 정의하였습니다.

모델 훈련 시에 epoch 를 10 으로 설정하고 훈련한 결과 다음과 같이 나왔습니다.



마찬가지로 4번째 epoch 이후로 과대 적합이 발생하였기 때문에 epoch 를 4로 재설정하고 모델을 만들었습니다. 그 결과는 다음과 같습니다.



테스트 데이터를 사용해서 모델의 정확도를 분석한 결과

32001/32001 [==============================] - 8s 234us/step

[0.8657965094989829, 0.7752570232180245]

로 77.5% 의 정확도를 나타남을 확인할 수 있었습니다.

**7. 모델과 토크나이저 불러와서 사용하기**

keras.models 에서 load\_model 을 import 해서 load\_model() 함수를 통해 기존에 학습시켰던 모델들을 불러왔습니다.

keras\_preprocessing.text 에서 tokenizer\_from\_json 을 import 해서 json\_load() 함수를 통해 tokenizer 를 불러왔습니다. 같은 mpbase 더라도 pname 을 추측하기 위해서 만들어진 mpbase\_pname\_tokenizer 와 pmake 를 추측하기 위해서 만들어진 mpbase\_pmake\_tokenizer 가 다르기 때문에 애당초 다르게 저장되어 있었기 때문에, 다르게 불러와야 했습니다. 따라서 벡터화시키는데 필요한 dimension 역시 pname 의 경우는 10000 이었고, pmake 의 경우는 140000 이었기 때문에 각각 다르게 vectorize\_sequences\_pname 과 vectorize\_sequences\_pmake 로 정의하였습니다.

또한 konlpy 를 사용해서 자연어 처리를 하는 split() 함수를 만들었습니다. 이 함수는 위에서 자연어 처리를 했던 방식과 동일하게 앞으로 들어올 데이터들에 대해서 쓸모 없는 부분들을 제외해주는 함수입니다. 마찬가지로 "Josa", "Eomi", "Punctuation", "Foreign", "Number", "Alpha" 을 제외한 형태의 문장으로 변경해줍니다.

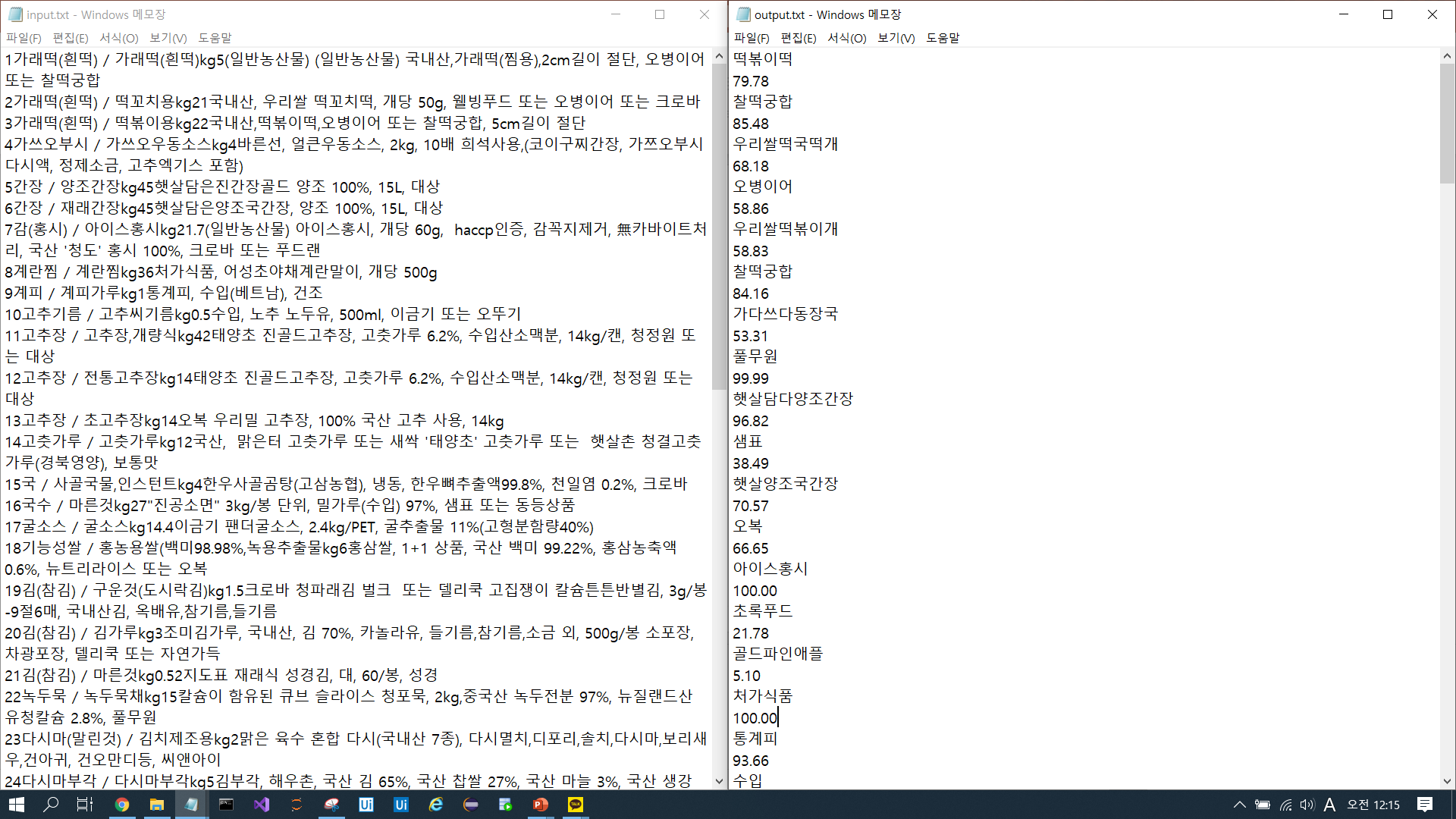
이 후에 들어오는 문장에 대해서 split() 함수를 통해 필요한 파라미터만 있는 문장으로 변경시키고 이를 각각 mpbase\_pname\_tokenizer 와 mpbase\_pmake\_tokenizer 를 통해 정수로 변환합니다. 그리고 각자 정수를 해당되는 vectorize 함수를 통해 각각의 벡터로 변환합니다. 각각의 벡터에 해당되는 model\_pname 과 model\_pmake 가 predict 해서 나온 결과를 변수에 저장합니다.

np.flip 과 np.argsort 를 사용해서 예상되는 가장 확률이 큰 pname 과 pmake의 index 를 추출합니다.

pname\_tokenizer 와 pmake\_tokenizer 는 dict 형식으로 되어 있으므로 이를 reverse 시켜서 그에 해당되는 label 값을 출력합니다.

**8. 실제 사용 사례**

아래는 지금껏 만든 머신러닝을 사용해서 실제로 eat.school.go.kr 에서 한 학교의 입찰 요구 정보를 읽어와서 어떠한 제품을 원하는지, 어떠한 제조사를 의미하는지를 정확도와 함께 나타낸 결과입니다.



Input file 의 첫번째 레코드를 보면 카테고리와 제품 설명부분이 함께 합쳐져 있습니다. ‘가래떡(흰떡) / 가래떡(흰떡) kg5 (일반농산물) (일반농산물) 국내산, 가래떡(찜용), 2cm 길이 절단, 오병이어 또는 찰떡궁합’

이를 머신에 넣어서 어떠한 제품을 의미하고 제조사는 무엇인지 결과를 얻게 되니 제품명은 ‘떡볶이떡’으로 79.78% 의 정확도를 나타내고 있고, 제조사는 ‘찰떡궁합’으로 85.48% 의 정확도를 나타내며 추측 결과값을 나타내고 있습니다.

하지만 Input file 의 8번째 레코드를 보면 ‘계란찜 /게란찜kg36처가식품, 어성초야채계란말이, 개당 500g’ 이라는 내용을 담고 있지만 해당 내용을 머신이 읽어서 추측하기를 제품명은 ‘골드파인애플’로 5.1% 의 정확도를 나타내고, 제조사는 ‘처가식품’ 으로 100% 의 정확도를 보이고 있습니다. 즉 자신이 배운 결과에 대해서는 매우 정확하게 결과를 도출해내지만 자신이 배우지 못한 내용에 대해서는 정확도가 매우 낮게 측정이 됩니다. 이러한 정확도를 높이기 위해서는 많은 양의 데이터를 학습시킬 수 있어야 하지만 개발에 사용되었던 PC 의 성능이 전체 데이터를 학습하기에는 무리가 있기 때문에 더 이상의 학습을 진행할 수 없었습니다.