

Gene Name Recognition

기말 프로젝트

김소현

1771089

2019.06.22

➤ Goal

'단백질 이름을 분간하는' Classifier를 만들어 본다. 즉, 특정 단어가 입력되면, 이 단어가 단백질인지 아닌지를 판별한다. 단백질 이름을 분간할 기준이 될 feature들을 생각해 보고, 그에 기반하여 feature extract하는 함수를 만들고, 이를 기반으로 Naive Bayes Classifier방식을 사용한다. 또한 feature 추출시에 Wikipedia 모듈을 이용, 해당 단어의 검색 결과를 기반으로 뽑아낸 feature를 반영한다.

➤ 시작하기에 앞서 참고자료에 나와있는 내용을 읽어보고 정리한다.

Biological Object의 인식은 굉장히 어렵다

명명 관습이 없으며, 약어를 과도하게 사용하고, 또 동의어나 동음이의어가 존재하기 때문이다.

'human T-cell leukaemia lymphotropic virus type 1 Tax protein'.같이, 여러 개의 단어로 이루어진 경우가 많은 것도 인식을 어렵게 하는 이유 중 하나. 로마자, 그리스어 혹은 '-'등 특수기호가 사용되기도 한다.

HAKENBERG, Jörg, et al. Systematic feature evaluation for gene name recognition. BMC bioinformatics, 2005, 6.1: S9.

해당 논문을 참고하여 어떤 분야들을 특징 추출에 반영했는지 알아보았다. 모든 내용을 이해할 수는 없었지만, 소개된 바에 의하면 토큰 그 자체, Unseen token, n-gram, 이전/이후 token, n-grams of tokens in window, 접두사, 접미사, stop words (코퍼스), POS tag결과, 대문자 유무 등을 반영했다. 수많은 특징들을 이용했으나, 내가 이해할 수 없는 개념도 있었기 때문에 이 중 이해할 수 있는 특징들을 최대한 구현해 보는 것을 목표로 하려 한다.

The most popular genes in the human genome - K. Krause and J. Krzysztofciak/Nature, 2017

가장 인기 있는 인간 유전자 이름 TOP10은 TP53, TNF, EGFR, VEGFA, APOE, IL6, TGFB1, MTHFR, ESR1, AKT1이라고 한다. 사전식으로 해서 해당 token을 포함하고 있는지 확인한다.

➤ 특징 추출 (위키피디아 제외)

1. 해당 토큰의 모든 알파벳이 대문자인가.

예시: TP53, TNF, EGFR, VEGFA, APOE, IL6, TGFB1, MTHFR, ESR1, AKT1

```
>>> def feature_extractor(word):  
    features={}  
    feature['isupper']=word.isupper()
```

2. 자주 쓰이는 토큰을 포함하고 있는가

논문 The most popular genes in the human genome - K. Krause and J. Krzysztofciak/Nature, 2017
을 참고하면 TP53, TNF, EGFR, VEGFA, APOE, IL6, TGFB1, MTHFR, ESR1, AKT1를 포함하는 단백질 이름이 가장 많다고 한다.

```
>>> def feature_extractor(word):  
    features={}  
    features['isupper']=word.isupper()  
    popular=['TP53', 'TNF', 'EGFR', 'VEGFA', 'APOE', 'IL6', 'TGFB1', 'MTHFR', 'ESR1', 'AKT1']  
    features['contains popular words']=False  
    for token in popular:  
        if(not features['contains popular words']):  
            features['contains popular words']= (token in word)  
    return features  
  
>>> feature_extractor('RTNF')  
{'isupper': True, 'contains popular words': True}
```

3. stop words에 포함되어 있지 '않은'가

stop words에 들어있는 단어들이 단백질 이름 일 리가 없다.

```
>>> stopwordslist=stopwords.words()  
>>> stopwordslist=stopwords.words('english')
```

stop words corpus를 불러오고

```
features['stopwords']=word in stopwordslist
```

feature extractor에 추가한다.

4. pos tag 에서 NN 이나 DT 로 태그 되는가

이는 HAKENBERG, Jörg, et al. Systematic feature evaluation for gene name recognition. BMC

bioinformatics, 2005, 6.1: S9. 논문에서 참고한 내용이다

```
#pos tag결과
pos=nltk.pos_tag([word])
if (pos==[(word, 'NN')]):
    features['NN']=True
    featrues['DT']=False
elif(pos==[(word, 'DT')]):
    features['NN']=False
    featrues['DT']=True
else:
    features['NN']=False
    features['DT']=False
```

5. 해당 단어가 숫자를 포함하는가

P35, Q8IXJ6, Q9ULW0와 같이 이름에 숫자를 포함한 gene들이 자주 눈에 띈다. 이 점을 반영하여 숫자를 포함하고 있는지 확인한다.


```
>>> def hasNumbers(inputString):
    return any(char.isdigit() for char in inputString)
```

우선 해당 스트링에 숫자가 들어있는지 확인하는 함수를 정의해야한다.

```
features['contain_number']=hasNumbers(word)
```

그 다음 해당 feature를 추가 해 준다.

➤ 위키피디아 모듈을 이용한 특징 추출

 명령 프롬프트 - pip install wikipedia

```
Microsoft Windows [Version 10.0.17134.829]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\yorco>pip install wikipedia
```

cmd 창에서 wikipedia모듈을 설치

```
>>> import wikipedia
```

import하여 Wikipedia 모듈의 기능들을 사용해 본다.

1. 위키피디아 검색 결과에 'protein'이나 'gene'이 포함되어 있는가

```
ny=wikipedia.page(word)
wikicontent=ny.content
features['wiki-proteni?']='protein' in wikicontent
featurese['wiki-genes?']='gene' in wikicontent
```

위키피디아 페이지 내용에 protein, gene이 포함되어 있다면 단백질 이름일 가능성이 높을 것이다.

그러나 만약, wikipedia에서 특정 키워드의 검색 결과가 유일하지 않은 경우는 어떻게 할까.

[예시]

```
>>> feature_extractor('P35')
Traceback (most recent call last):
  File "<pyshell#88>", line 1, in <module>
    feature_extractor('P35')
  File "<pyshell#88>", line 4, in feature_extractor
    ny=wikipedia.page(word)
  File "C:\Users\yorco\AppData\Local\Programs\Python\Python37-32\Lib\site-packages\wikipedia\wikipedia.py", line 276, in page
    return WikipediaPage(title, redirect=redirect, preload=preload)
  File "C:\Users\yorco\AppData\Local\Programs\Python\Python37-32\Lib\site-packages\wikipedia\wikipedia.py", line 299, in __init__
    self._load(redirect=redirect, preload=preload)
  File "C:\Users\yorco\AppData\Local\Programs\Python\Python37-32\Lib\site-packages\wikipedia\wikipedia.py", line 393, in _load
    raise DisambiguationError(getattr(self, 'title', page['title']), may_refer_to)
wikipedia.exceptions.DisambiguationError: "P35" may refer to:
P-35 radar
Seversky P-35
HMS P.35 (1917)
Browning Hi-Power
CDK5R1
Early 35 kDa protein
EIF3J
IL12A
Intel P35
Nissan P35
Papyrus 35
isotopes of phosphorus
```

예를 들어 'P35'를 검색하면 검색 결과가 많아 에러가 발생한다. 다양한 경우 중 원하는 페이지를 선택해야한다. 하지만 결과가 무수하게 많을 때 이것을 일일이 확인하는 것은 불가능에 가깝다.

그러므로, 이 때는 아쉽지만 랜덤으로 페이지 하나를 선택하도록 한다. import random을 한 뒤 다음과 같이 코드를 작성했다.

[수정한 코드]

```
try:
    #위키피디아 사용
    ny=wikipedia.page(word)
    wikicontent=ny.content
    #features['wiki-proteni?']='protein' in wikicontent
    #features['wiki-genes?']='gene' in wikicontent

except wikipedia.DisambiguationError as e:
    s=random.choice(e.options)
    ny=wikipedia.page(s)
    wikicontent=ny.content
    features['wiki-proteni?']='protein' in wikicontent
    features['wiki-genes?']='gene' in wikicontent

return features
```

수정된 함수를 이용해 제대로 추출이 됨을 확인한다.

['P35'로 다시 테스트]

```
>>> feature_extractor('P35')
{'NN': True, 'DT': False, 'contain number': True, 'isupper': True, 'contains popular words': False, 'stopwords': False, 'wiki-proteni?': False, 'wiki-genes?': False}
```

2. 앞 뒤에 특정 단어들이 위치하는가

Wikipedia 검색 결과 중에서도 정확도를 높이기 위해 특정 단어의 위치를 확인해 볼 가치가 있다. 예를 들면 아래는 단백질 'CDK5R1'을 위키피디아로 검색, content를 래핑 한 뒤 'concordance()' 함수를 적용 해 본 모습이다. (input은 'gene'으로 했다.)

(앞 뒤의 단어를 살피기 위해서는 래핑 후, bigrams())를 사용해 tuple list화 해 주어야한다.)

```
>>> ny = wikipedia.page("CDK5R1")
>>> teststrig=ny.content
>>> text=nltk.Text(nltk.word_tokenize(teststrig))
>>> text.concordance('gene')
Displaying 4 of 4 matches:
t in humans is encoded by the CDK5R1 gene . == Function == The protein encoded
ction == The protein encoded by this gene ( p35 ) is a neuron-specific activat
disease.In melanocytic cells CDK5R1 gene expression may be regulated by MITF
an CDK5R1 genome location and CDK5R1 gene details page in the UCSC Genome Brow
```

'gene' 앞에 해당 단백질의 이름이 위치하는 모습을 확인 할 수 있다. 본문에서 '유전자 CDK5R1'와 같은 식으로 언급했기 때문이다. 특정 단어가 'gene' 같은 단어와 합성되어 있다면 해당 단어는 단백질 이름일 확률이 굉장히 높다.

```

text=nltk.Text(nltk.word_tokenize(wikicontent)) #래핑
bigram_wiki=list(nltk.bigrams(text))
word_bf_gene = [a[0] for a in bigram_wiki if a[1]=='gene']
if(word in word_bf_gene):
    features['applied by gene?']=True
else:
    features['applied by gene?']=False

return features

```

위에서 알아낸 사실을 기반으로 feature에 해당 내용을 반영했다. wikipedia에서 가져온 페이지에서, '해당 단어 + gene' 조합이 나온다면 True값을 저장한다.

3. 주의 사항

그러나 혹시 모를 상황, wikipedia에 특정 단어의 검색 값이 발견되지 않았을 때의 상황도 대비해 두려고 한다.

```

>>> ny = wikipedia.page("rrreoeoeoweoweoeowweoeowoeowower")
Traceback (most recent call last):
  File "C:\Users\yoorco\AppData\Local\Programs\Python\Python37-32\Lib\site-packages\wikipedia\wikipedia.py", line 272
, in page
    title = suggestion or results[0]
IndexError: list index out of range

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "<pyshell#135>", line 1, in <module>
    ny = wikipedia.page("rrreoeoeoweoweoeowweoeowoeowower")
  File "C:\Users\yoorco\AppData\Local\Programs\Python\Python37-32\Lib\site-packages\wikipedia\wikipedia.py", line 275
, in page
    raise PageError(title)
wikipedia.exceptions.PageError: Page id "rrreoeoeoweoweoeowweoeowoeowower" does not match any pages. Try another

```

다음은 그 예시다, PageError가 발생한다. 에러 handling을 해 줄 필요가 있다.

[해당 경우를 대비해 수정해준 코드]

```

noresult=0
try:
    #위키피디아 사용
    ny=wikipedia.page(word)
    wikicontent=ny.content
    #features['wiki-proteni?']='protein' in wikicontent
    #features['wiki-genes?']='gene' in wikicontent

except wikipedia.DisambiguationError as e:
    s=random.choice(e.options)
    ny=wikipedia.page(s)
    wikicontent=ny.content
except wikipedia.PageError as ee:
    features['wiki-proteni?']=False
    features['wiki-genes?']=False
    features['applied by gene?']=False
    noresult=1 #goto문 대신 사용할 flag
if (noresult !=1):
    features['wiki-proteni?']='protein' in wikicontent
    features['wiki-genes?']='gene' in wikicontent

text=nltk.Text(nltk.word_tokenize(wikicontent)) #래핑
bigram_wiki=list(nltk.bigrams(text))
word_bf_gene = [a[0] for a in bigram_wiki if a[1]=='gene']
if(word in word_bf_gene):
    features['applied by gene?']=True
else:
    features['applied by gene?']=False

return features

```

[테스트]

```
>>> feature_extractor('rrreoeoeoweoeoweoeoweoeoweoeower')
{'NN': True, 'DT': False, 'contain number': False, 'isupper': False, 'contains popular words': False, 'stopwords': False, 'wiki-protein?': False, 'wiki-genes?': False, 'applied by gene?': False}
```

원래 에러가 발생했던 입력값으로 다시 한번 테스트 해 본다. 에러없이 결과가 도출된다.

➤ Data 가공하기 (training data, test data 만들기)

<http://geneontology.org/> 에서 goa_human.gaf파일을 다운받아 이 자료를 가공한다.

해당 파일을 열어보면 한 열 당 아래와 같은 형식을 갖추고 있다.

UniProtKB	A0A024RBG1	NUDT4B	GO:0003723	GO_REF:0000037 IEA
	UniProtKB-KW:KW-0694	F	Diphosphoinositol polyphosphate	phosphohydrolase
NUDT4B	NUDT4B protein	taxon:9606	20190601	UniProt

이 중 단백질 이름에 해당하는 것은 두번째, 여덟번째 필드에 해당한다. 해당 자료를 정규표현식을 이용해 추출해 내는 것을 목표로 한다. 물론 파일의 길이가 너무 커서 모든 경우를 확인하고 반영하지는 못할 수 있다. 그래도 최대한 정확하게, 충분한 자료를 뽑아내는 것을 목표로 했다.

```
>>> import nltk
>>> from nltk.corpus import *
>>> corpusroot="C:\\Users\\yorko\\Desktop\\imp\\"
>>> nc=PlaintextCorpusReader(corpusroot, 'goa_human.gaf', encoding='utf-8')
>>> raw=nc.raw()
```

우선 해당 파일을 불러온다.

```
>>> genelist=re.findall('UniProtKB ([A-Za-z0-9]+) [A-Za-z0-9-:]+',raw)
>>> len(genelist)
481418
```

정규표현식을 이용해 두번째 필드에 위치한 단백질 이름을 뽑아냈다.

```
>>> genelist=set(genelist)
>>> len(genelist)
19774
```

set()을 이용해 중복을 골라내면 2만개 정도가 남는다.

```
>>> genelist_t=re.findall('\\\\t[CPF]{1}\\\\t(.+?)\\\\s[A-Z]+',raw)
>>> len(genelist_t)
481418
>>> genelist_t[1]
'Diphosphoinositol polyphosphate phosphohydrolase'
```

여덟번째 필드의 이름을 골라낸다.

```
>>> genelist_t=set(genelist_t)
>>> len(genelist_t)
14623
```

마찬가지로 중복을 삭제하면, 만 오천 개 정도의 결과값이 남는다.

이제 모든 자료들을 합쳐 특징을 추출한다. list comprehension을 이용한다. 이 test set(혹은 training set)의 끝은 (특징, YES 혹은 NO)가 될 것이다. 위에서 뽑아낸 자료들은 모두 단백질 이름이 맞으므로 YES가 들어가게 된다.

```
proteinlist=[ (feature_extractor(token), 'YES') for token in genelist]
```

그러나 위의 자료 모두를 사용하면 약 3만개의 데이터를 처리해야 하며, 컴퓨터 사양 문제로 처리를 시도하자 과도한 시간이 소요됐다. 원활한 테스트를 위해 데이터의 양을 약 100개로 줄여 다시금 해당 작업을 개시했다.

[자료의 양을 '약' 100개로 잘라내는 코드]

```
genelist=genelist[:100]
```

다음으로는 '잘못된 예'로 학습할 자료를 구한다. 우선 feature 추출 내용중 stopwords corpus가 사용되었고 이 중 단백질의 이름이 포함되어 있을 리 없으므로 해당 단어들을 가져와 학습에 이용토록 한다. 또한 treebank의 word들을 가져와 학습시키도록 하고, 여러가지 단어로 구성된 단백질 이름 (예를들어 Diphosphoinositol polyphosphate phosphoh 등)과 구별 짓는데 도움을 주기 위해 webtext의 문장 몇 개를 가져와 학습시키도록 한다. negative 데이터는 positive만큼 선별해 고르지 않았기 때문에, 자세한 설명은 생략하도록 하겠다.

```
falselist=stopwords.words('english')
falselist.extend(treebank.words('wsj_0003.mrg'))
falselist=set(falselist)
falselist=list(falselist)
for i in range(20):
    falselist.append(webt[i+5])
random.shuffle(falselist)
#falselist=falselist[:100]
notgenes=[(feature_extractor(token), 'No') for token in falselist]
```

위 코드를 실행하던 중에 한가지 에러가 발생했다. webtext에서 가져온 sentence중 하나가 지나치게 길어 wikipedia의 예외가 발생해, feature_extractor에 예외처리 코드를 한가지 더 추가 해 주었다.

[disambiguation error2]

```
wikipedia.exceptions.DisambiguationError: "About" may refer to:
About (surname)
About.com
about.me
abOUT
About URI scheme
About box
About equal sign
About Face (disambiguation)
About Last Night (disambiguation)
About Time (disambiguation)
About us (disambiguation)
About You (disambiguation)
about to
All pages with titles beginning with about
```

또 다른 문제가 발생했다. 본래 계획했던 대로라면, disambiguation 에러가 발생했을 때 랜덤으로 페이지를 골라 진행하게 되어있다. 그러나, 특정 검색어는 '랜덤으로 선택한 페이지 도' disambiguation에러가 발생 할 수 있다.

이 에러를 발견하기 위해, 또 수정하기 위해 시행착오를 거듭한 결과 python은 exception내부에 서 한번 더 exception 처리 (try~ except문)을 사용 가능하다는 사실을 알아냈다. 어렵사리 exception을 핸들링 할 수 있었다.

두번째로 disambiguation 에러가 발생하면, 무한정 새로운 페이지를 선택할 수는 없으므로 마찬가지로 관련 feature를 false로 처리한다.

```
>>> len(notgenes)
100
>>> len(proteinlist)
100
>>> featureset=proteinlist+notgenes
>>> random.shuffle(featureset)
>>> featureset[5]
({'NN': False, 'DT': False, 'contain number': False, 'isupper': False, 'contains popular words': False, 'stopwords': False, 'wiki-proteni?': False, 'wiki-genes?': False, 'applied by gene?': False}, 'No')
```

negative data들과, positive data들의 정형화가 끝났으므로 featureset 리스트에 합친 뒤 섞어준다.

```
>>> index=int(len(featureset)*0.9)
>>> train_set=featureset[:index]
>>> test_set=featureset[index:]
```

그 뒤 9:1 정도로 분할 해 주면 해당 단계는 끝이 난다.

[수정된 코드]

```
def feature_extractor(word):
    features={}

    #pos tag결과
    pos=nltk.pos_tag([word])
    if (pos==[(word,'NN')]):
        features['NN']=True
        features['DT']=False
    elif(pos==[(word,'DT')]):
        features['NN']=False
        features['DT']=True
    else:
        features['NN']=False
        features['DT']=False

    #그 밖
    features['contain_number']=hasNumbers(word)
    features['isupper']=word.isupper()
    popular=['TP53', 'TNF', 'EGFR', 'VEGFA', 'APOE', 'IL6', 'TGFB1', 'MTHFR', 'ESR1','AKT1']
    features['contains popular words']=False
    for token in popular:
        if(not features['contains popular words']):
            features['contains popular words']= (token in word)
    features['stopwords']=word in stopwordslist

    noresult=0
    try:
        #위키피디아 사용
        ny=wikipedia.page(word)
        wikicontent=ny.content
        #features['wiki-proteni?']='protein' in wikicontent
        #features['wiki-genes?']='gene' in wikicontent

    except wikipedia.exceptions.DisambiguationError as e:
        try:
            s=random.choice(e.options)
            ny=wikipedia.page(s)
            wikicontent=ny.content
        except wikipedia.exceptions.DisambiguationError as e:
            features['wiki-proteni?']=False
            features['wiki-genes?']=False
            features['applied by gene?']=False
            noresult=1 #goto문 대신 사용할 flag

    except wikipedia.PageError as ee:
        features['wiki-proteni?']=False
        features['wiki-genes?']=False
        features['applied by gene?']=False
        noresult=1 #goto문 대신 사용할 flag

    except wikipedia.exceptions.WikipediaException as eee:
        features['wiki-proteni?']=False
        features['wiki-genes?']=False
        features['applied by gene?']=False
        noresult=1 #goto문 대신 사용할 flag

    if (noresult !=1):
        try:
            features['wiki-proteni?']='protein' in wikicontent
            features['wiki-genes?']='gene' in wikicontent
            text=nltk.Text(nltk.word_tokenize(wikicontent)) #래평
            bigram_wiki=list(nltk.bigrams(text))
            word_bf_gene = [a[0] for a in bigram_wiki if a[1]=='gene']
            if(word in word_bf_gene):
                features['applied by gene?']=True
            else:
                features['applied by gene?']=False
        except wikipedia.exceptions as f:
            features['wiki-proteni?']=False
            features['wiki-genes?']=False
            features['applied by gene?']=False

    return features
```

➤ **training**

마련한 데이터를 기반으로 학습을 시작한다. 계획했던 대로 NaiveBayesClassifier를 이용한다.

```
>>> classifier=nlTK.NaiveBayesClassifier.train(train_set)
>>> print(nltk.classify.accuracy(classifier,test_set)
)
1.0
>>> classifier.show_most_informative_features()
Most Informative Features
          isupper = True          YES : No      =      26.4 : 1.0
    contain number = True        YES : No      =       9.7 : 1.0
      wiki-proteni? = True        YES : No      =       5.1 : 1.0
          NN = True              YES : No      =       2.3 : 1.0
      wiki-genes? = False         No : YES     =       1.7 : 1.0
      stopwords = False          YES : No      =       1.5 : 1.0
      wiki-proteni? = False       No : YES     =       1.4 : 1.0
      wiki-genes? = True          YES : No      =       1.4 : 1.0
          DT = False             YES : No      =       1.0 : 1.0
    applied by gene? = False      YES : No      =       1.0 : 1.0
```

마련한 test data만으로는 정확도가 100%가 나왔다.

```
>>> classifier.classify(feature_extractor('P35'))
'YES'
>>> classifier.classify(feature_extractor('APPLE'))
'No'
>>> classifier.classify(feature_extractor('hello-nice to meet you'))
'No'
>>> classifier.classify(feature_extractor('APPLE2'))
'YES'
```

그 외에도 classifier가 혼동 할 수 있을 것 같은 예제 몇가지를 수동으로 테스트 해 보았다. 대부분 올바른 결과를 도출했으나, 단백질 이름과 유사하게 꾸며낸 input은 잘못된 결과를 도출했다.

➤ gni corpus에의 적용

classifier의 성능을 확인하기 위해 classifier를 이용해 gni corpus에서 단백질 이름을 골라내는 활동을 해 보려고 한다.

```
>>> croot="C:\Users\yvorco\Desktop\imp\gni"
>>> cp=PlaintextCorpusReader(croot, '.*.txt', encoding='utf-8')
>>> gniwords=cp.words()
>>> len(gni)

Squeezed text (4 lines).
>>> len(gniwords)
3390278
```

우선 gni corpus를 코퍼스 화 해 준다. 단어 수를 확인 해 본다면 3백만개 가량인데, 컴퓨터의 사양 문제로 이렇게 많은 데이터를 처리하는 것은 과도한 시간이 걸리므로 약 1000개 정도의 데이터만 처리해 보기로 했다.

[classifier결과가 'YES'가 나오면 gnigene 리스트에 추가된다]

```
>>> gniwords=gniwords[:1000]
>>> gnigene=[]
>>> for i in gniwords:
    result=classifier.classify(feature_extractor(i))
    if (result=='YES'):
        gnigene.append(i)
```

[에러발생]

```
Traceback (most recent call last):
  File "<pyshell#259>", line 2, in <module>
    result=classifier.classify(feature_extractor(i))
  File "<pyshell#209>", line 38, in feature_extractor
    ny=wikipedia.page(s)
  File "C:\Users\yvorco\AppData\Local\Programs\Python\Python37-32\lib\site-packages\wikipedia\wikipedia.py", line 270, in page
    results, suggestion = search(title, results=1, suggestion=True)
  File "C:\Users\yvorco\AppData\Local\Programs\Python\Python37-32\lib\site-packages\wikipedia\util.py", line 28, in __call__
    ret = self._cache[key] = self.fn(*args, **kwargs)
  File "C:\Users\yvorco\AppData\Local\Programs\Python\Python37-32\lib\site-packages\wikipedia\wikipedia.py", line 109, in search
    raise WikipediaException(raw_results['error']['info'])
wikipedia.exceptions.WikipediaException: An unknown error occurred: "We could not complete your search due to a temporary problem. Please try again later.". Please report it on GitHub!
```

처리중에 원인을 알 수 없는 에러가 발생했다. 일시적인 문제 발생이라고 하는데, 특별히 세분화해서 다뤄 줄 사항은 아닌 것 같아 except문 하나를 추가해 해당 결과의 feature를 false처리 해주기로 한다.

[1000개의 classifier 분류 결과]

```
>>> gnigene  
['eISSN2234', 'CNV', 'FISH', 'DNA', 'DNA', 'DNA', 'DNA', 'UV', '750K', 'V2', 'VUS']
```

1. eISSN2334는 단백질이 이름이 아닌 도서 번호로 추정된다.

도서번호인 ISSN은 자주쓰이며 또 단백질의 형태와 흡사하다. 사전식으로 ISSN을 포함하는 경우를 제외하거나, ISSN+숫자 형식을 띄면 단백질이 아니라고 classifier개선헌시 학습에 반영 해 주는게 좋을 것 같다.

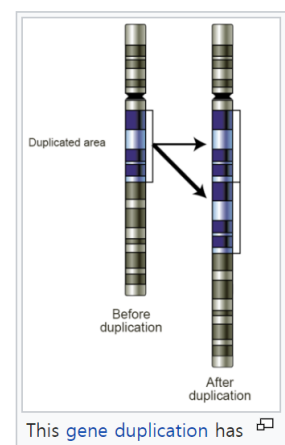
2. CNV는 검색결과 gene의 하나로 성공적인 분류를 했다.

Copy-number variation

From Wikipedia, the free encyclopedia
(Redirected from [Copy number variation](#))

Copy number variation (CNV) is a phenomenon in which sections of the genome are repeated and the number of repeats in the genome varies between individuals in the human population.^[1] Copy number variation is a type of [structural variation](#): specifically, it is a type of [duplication](#) or [deletion](#) event that affects a considerable number of base pairs.^[2] However, note that although modern genomics research is mostly focused on human genomes, copy number variations also occur in a variety of other organisms including *E. coli* and *S. cerevisiae*.^[3] ^[4] Recent research indicates that approximately two thirds of the entire human genome is composed of repeats^[5] and 4.8–9.5% of the human genome can be classified as copy number variations.^[6] In [mammals](#), copy number variations play an important role in generating necessary variation in the population as well as disease phenotype.^[1]

Copy number variations can be generally categorized into two main groups: short repeats and long repeats. However, there are no clear boundaries between the two groups and the



3.FISH

잘못된 분류. 대소문자 구성 여부가 분류에 큰 영향을 가지게 설정된 까닭으로 보인다.

4. DNA

잘 분류되었다고 하기에, 안 하기에 다소 모호하다.

5.UV

단백질이 아닌 자외선을 의미한다. 잘못 분류된 경우다.

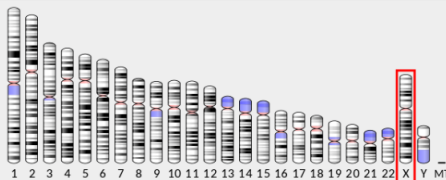
6.V2

Vasopressin receptor 2

From Wikipedia, the free encyclopedia
(Redirected from [V2 receptor](#))

Vasopressin receptor 2 (V2R), or **arginine vasopressin receptor 2** (officially called **AVPR2**), is a [protein](#) that acts as [receptor](#) for [vasopressin](#).^[5] AVPR2 belongs to the subfamily of [G-protein-coupled receptors](#). Its activity is mediated by the [G_s](#) type of [G proteins](#), which stimulate [adenylate cyclase](#).

AVPR2 is expressed in the [kidney tubule](#), predominantly in the membrane of cells of the [distal convoluted tubule](#) and [collecting ducts](#), in [fetal lung](#) tissue and [lung cancer](#), the last two being associated with [alternative splicing](#). AVPR2 is also expressed outside the kidney in vascular endothelium.^[6] Stimulation causes the release of [von Willebrand factor](#) and [factor VIII](#) from the endothelial cells.^[6] Because von Willebrand factor helps stabilize circulating levels of factor VIII, the vasopressin analog [desmopressin](#) can be used to stimulate the AVPR2 receptor and increase levels of circulating factor VIII. This is useful in the treatment of [hemophilia A](#) as well as [Von Willebrand disease](#).

AVPR2	
Available structures	
PDB	Ortholog search: PDB RCSB
List of PDB id codes [show]	
Identifiers	
Aliases	AVPR2, ADHR, DI1, DIR, DIR3, NDI, V2R, arginine vasopressin receptor 2
External IDs	OMIM: 300538 MGI: 88123 HomoloGene: 20064 GeneCards : AVPR2
Gene location (Human) [hide]	
	

성공적인 분류다.

7.VUS

Variant of uncertain significance

From Wikipedia, the free encyclopedia

A **variant of uncertain (or unknown) significance (VUS)** is an [allele](#), or variant form of a gene, which has been identified through [genetic testing](#), but whose significance to the function or health of an organism is not known.^[1] Two related terms are "Gene of uncertain significance" (GUS), which refers to a gene which has been identified through [genome sequencing](#), but whose connection to a human disease has not been established, and "Insignificant mutation", referring to a gene variant that has no impact on the health or function of an organism. The term "variant" is favored over "[mutation](#)" because it can be used to describe an allele more precisely. When the variant has no impact on health it is called a "benign variant". When it is associated with a disease it is called a "pathogenic variant". A "pharmacogenomic variant" has an effect only when an individual takes a particular drug and therefore it is neither benign nor pathogenic.^[1]

A VUS is most commonly encountered by people when they get the results of a lab test looking for a variant form - a [mutation](#) in a particular gene. For example, many people know that mutations in the [BRCA1](#) gene are involved in the development of [breast cancer](#) because of the publicity surrounding [Angelina Jolie](#)'s preventative treatment.^[2] Few people are aware of the immense number of other genetic variants in and around BRCA1 and other genes that may predispose to hereditary breast and ovarian cancer. A recent study of the genes [ATM](#), [BRCA1](#), [BRCA2](#), [CDH1](#), [CHEK2](#), [PALB2](#), and [TP53](#)

‘form of a gene’이라는 문장이 들어가 있는 것으로 보아 유전자와 아주 밀접한 관계를 가지는 용어임은 맞는 것으로 추정되나, 관련 지식이 없어 확실히 단언 할 수 없겠다.

➤ 오류의 원인 고찰

'APPLE2' 따위의 입력에 잘못된 결과를 도출하는 점으로 미루어 보아 몇가지 가설을 떠올려 볼 수 있다.

1. 정교하게 꾸며진 Negative data 의 학습이 부족했음

'A0A024RBG1' 와 'hello' 처럼, 학습에 이용한 positive data 와 negative data 의 형태가 판이하게 달랐다. 위의 show_most_informative_feature() 함수의 결과를 보면 분류를 하는 가장 큰 기준이 wikipedia 가 아닌 그 외의 feature 들이 대다수임을 알 수 있다. 이는 이미 1 차원적인 분류 (wikipedia 가 아닌, 대소문자 유무나 글자의 유무 등) 단계에서 분류를 끝낼 수 있는 학습을 했음을 의미한다. positive data 처럼 보이게 꾸며진 정교한 negative data 를 마련할 수 없었던 것이 오류의 원인이 아닐까 짐작된다.

2. feature 들의 단순함

굳이 wikipedia 검색 내용에 의존하지 않더라도, 그 외의 feature 들이 정교했다면 올바른 분류를 할 수 있었을 것이다. 그러나 wikipedia 를 제외한 feature 들은 대소문자 유무, pos tagging 결과, stopwords corpus 포함여부 등으로 단순하다. 이 feature 들만 본다면 'APPLE2'와 'P35'의 차이를 분간할 수 없는 것은 당연한 일이다.

3. feature 갯수의 한계, 단순한 학습량의 부족

컴퓨터 사양의 문제로 feature 의 개수나 처리 할 수 있는 데이터의 양에 한계가 있었다. 마련했던 데이터는 3 만개가 넘었으나, 실제로 학습시켜 본 것은 200 개 정도에 불과했다.

여유 시간이 없어 classifier 를 개선해 보진 못했지만, 만약 이후 개선을 노린다면 위와 같은 두가지 사항을 반영해 더 높은 정확도의 classifier 를 만들 수 있으리라 생각된다.