

Project #1

3조 김대원, 신의호, 문경희, 김수정

역할분담

- Problem1: 문경희, 김수정
- Problem2: 신의호
- Problem3: 김대원
- 보고서 및 발표: 김대원

프로젝트 환경

- python 3.5.3 버전
- GCP에 notebook 서버를 띄워서 작업.
 - <http://imma.jameskim.xyz:8080>
- ipynb 파일은 github에서 버전관리
 - <https://github.com/JamesKim2998/imma-term-project>

Problem 1. wordcloud 생성

Problem 1-1. html을 가공하기

arXiv.org > cs > arXiv:1811.06128

Computer Science > Machine Learning

Machine Learning for Combinatorial Optimization: a Methodological Tour d'Horizon

Yoshua Bengio, Andrea Lodi, Antoine Prouvost

(Submitted on 15 Nov 2018)

This paper surveys the recent attempts, both from the machine learning and operations research communities, at leveraging machine learning to solve combinatorial optimization problems. Given the hard nature of these problems, state-of-the-art methodologies involve algorithmic decisions that either require too much computing time or are not mathematically well defined. Thus, machine learning looks like a promising candidate to effectively deal with those decisions. We advocate for pushing further the integration of machine learning and combinatorial optimization and detail methodology to do so. A main point of the paper is seeing generic optimization problems as data points and inquiring what is the relevant distribution of problems to use for learning on a given task.

Subjects: **Machine Learning (cs.LG); Machine Learning (stat.ML)**

Report number: DS4DM-2018-008

Cite as: arXiv:1811.06128 [cs.LG]
(or arXiv:1811.06128v1 [cs.LG] for this version)

```
<h1 class="title mathjax">  
  <span class="descriptor">Title:</span>  
  "Machine Learning for Combinatorial Opti  
  Tour d'Horizon"  
</h1>
```

```
<div class="authors">  
  <span class="descriptor">Authors:</span>  
  <a href="https://arxiv.org/search/cs?  
  searchtype=author&query=Bengio%2C+Y">Yoshua Bengio</a>  
</div>
```

```
<div class="dateline">(Submitted on 15 Nov 2018)</div>
```

```
<blockquote class="abstract mathjax">  
  <span class="descriptor">Abstract:</span>  
  " This paper surveys the recent attempts, bo  
  learning and
```

```
<td class="tablecell subjects">  
  <span class="primary-subject">Machine Lea  
  </span>  
  "; Machine Learning (stat.ML)"  
</td>
```

<https://arxiv.org/abs/1811.06128>

Problem 1-1. html을 가공하기

- <http://imma.jameskim.xyz:8080/notebooks/Problem1.ipynb#Problem-1>

Problem 1-2. Abstract를 Pos Tagging하기

- nltk 패키지의 word_tokenize() 함수와 pos_tag() 함수를 사용.

```
1 from nltk.tokenize import word_tokenize
2 from nltk.tag import pos_tag
3 tokenized_words = word_tokenize(abstract, preserve_line=True)
4 tagged_list = pos_tag(tokenized_words, tagset=None, lang='eng')
5 print("Pos Tag Result: ", tagged_list)
```




```
Pos Tag Result: [('This', 'DT'), ('paper', 'NN'), ('surveys', 'VBZ'), ('the', 'DT'), ('recent', 'JJ'), ('attempts', 'NNS'), ('both', 'DT'), ('from', 'IN'), ('the', 'DT'), ('machine', 'NN'), ('learning', 'NN'), ('and', 'C'), ('operations', 'NNS'), ('research', 'NN'), ('communities', 'NNS'), ('at', 'IN'), ('leveraging', 'VBG'), ('machine', 'NN'), ('learning', 'VBG'), ('to', 'TO'), ('solve', 'VB'), ('combinatorial', 'JJ'), ('optimization', 'NN'), ('problems.', 'NN'), ('Given', 'NNP'), ('the', 'DT'), ('hard', 'JJ'), ('nature', 'NN'), ('of', 'IN'), ('these', 'DT'), ('problems', 'NNS'), ('state-of-the-art', 'JJ'), ('methodologies', 'NNS'), ('involve', 'VBP'), ('algorithmic', 'JJ'), ('decisions', 'NNS'), ('that', 'IN'), ('either', 'DT'), ('require', 'VB'), ('too', 'RB'), ('mu
```

Problem 1-3. 단어를 카운팅해서 소팅하기

1. 토큰화된 단어들을 numpy의 unique함수를 이용해서 카운팅을 함.
2. 카운팅 결과를 sorted 함수를 이용해서 정렬함.

```
1 import numpy as np
2 unique_word_list, word_count_list = np.unique(tokenized_words, return_counts=True)
3 token_count = dict(zip(unique_word_list, word_count_list))
4
5 token_sorted = sorted([(n,m) for n,m in token_count.items()], key=lambda x: -x[1])
6 print("Sorting Token by frequency: ", token_sorted)
```



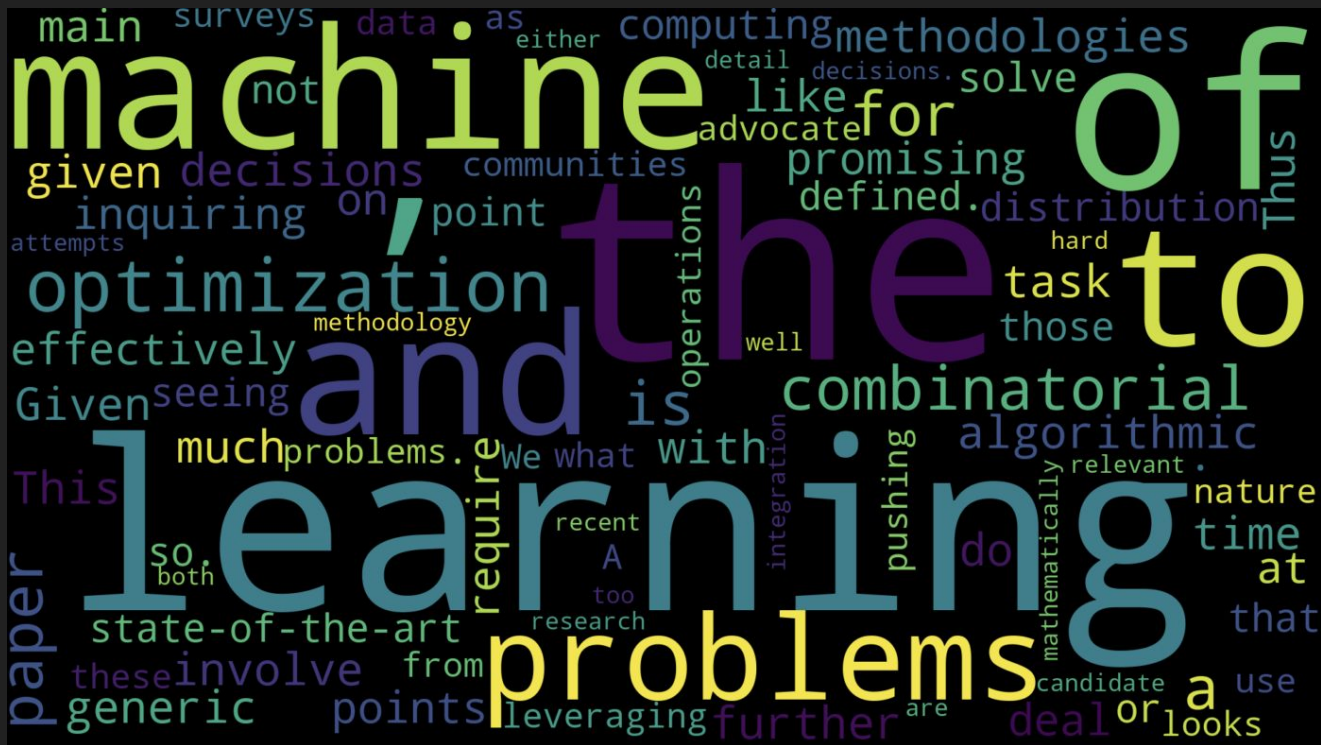
Sorting Token by frequency: [('the', 6), ('learning', 5), ('of', 4), ('', 4), ('machine', 4), ('and', 4), ('to', 4), ('problems', 3), ('optimization', 3), ('for', 2), ('combinatorial', 2), ('is', 2), ('a', 2), ('paper', 2), ('Give n', 1), ('much', 1), ('involve', 1), ('with', 1), ('decisions', 1), ('task', 1), ('time', 1), ('do', 1), ('given', 1), ('methodologies', 1), ('generic', 1), ('algorithmic', 1), ('This', 1), ('further', 1), ('promising', 1), ('inquiring', 1), ('points', 1), ('Thus', 1), ('on', 1), ('main', 1), ('like', 1), ('effectively', 1), ('at', 1), ('deal', 1), ('require', 1), ('computing', 1), ('solve', 1), ('state-of-the-art', 1), ('those', 1), ('or', 1), ('defined.', 1), ('that', 1), ('point', 1), ('so', 1), ('distribution', 1), ('not', 1), ('.', 1), ('seeing', 1), ('leveraging', 1)]

Problem 1-4. Wordcloud 생성하기

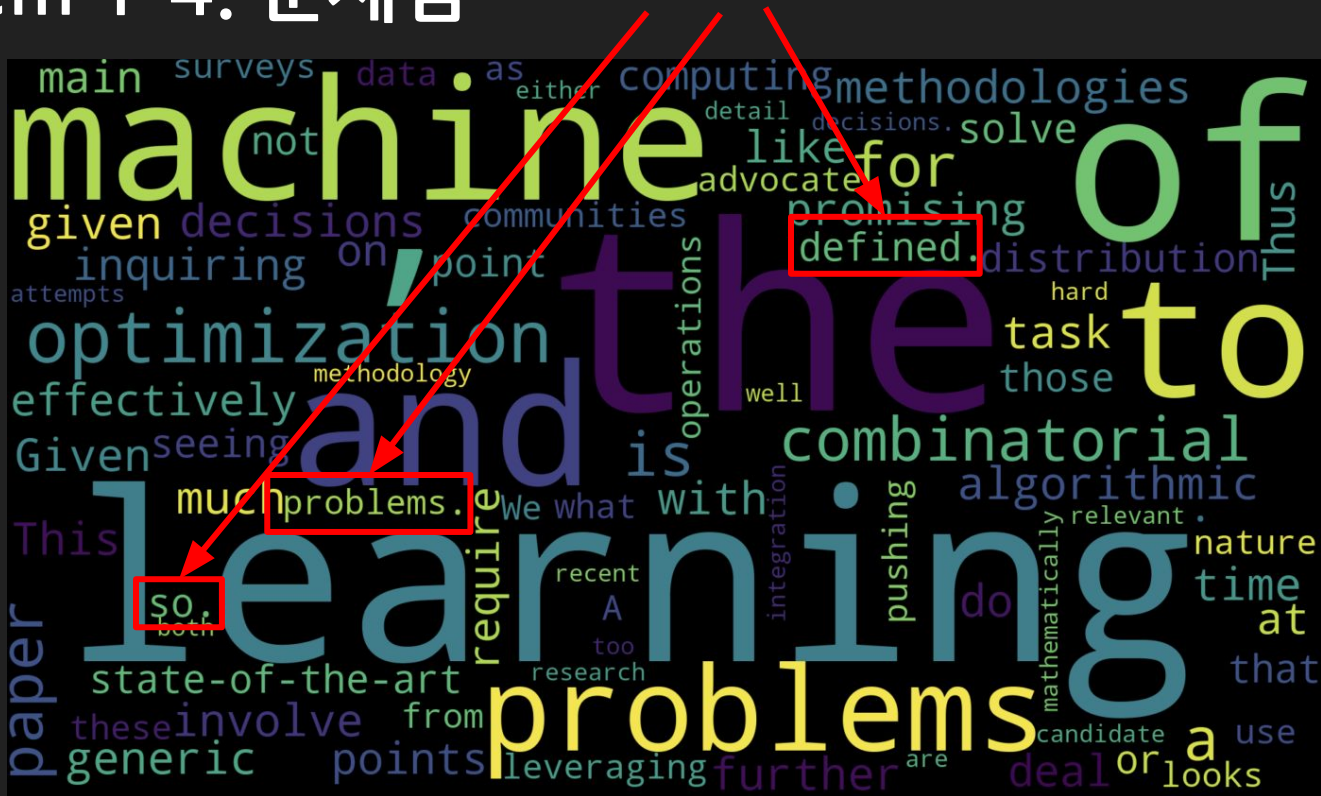
- 문제 1-3에서 얻은 단어의 등장횟수 dictionary를 이용해서 wordcloud를 생성함.

```
1 import matplotlib.pyplot as plt
2 from wordcloud import WordCloud
3
4 wordcloud = WordCloud().generate_from_frequencies(token_count_dict)
5 plt.imshow(wordcloud, interpolation='bilinear')
6 plt.axis("off")
7 plt.show()
```

Problem 1-4. Wordcloud 생성결과

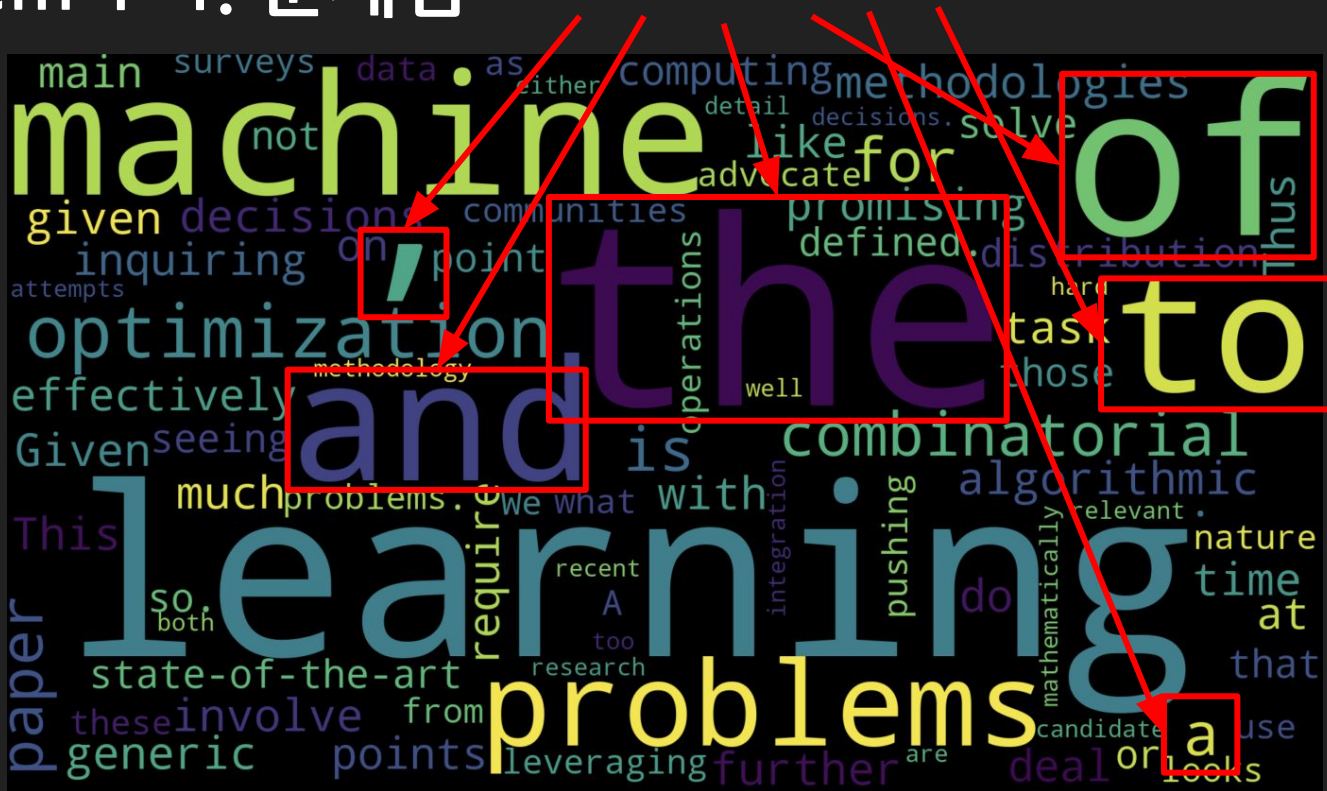


Problem 1-4. 문제점

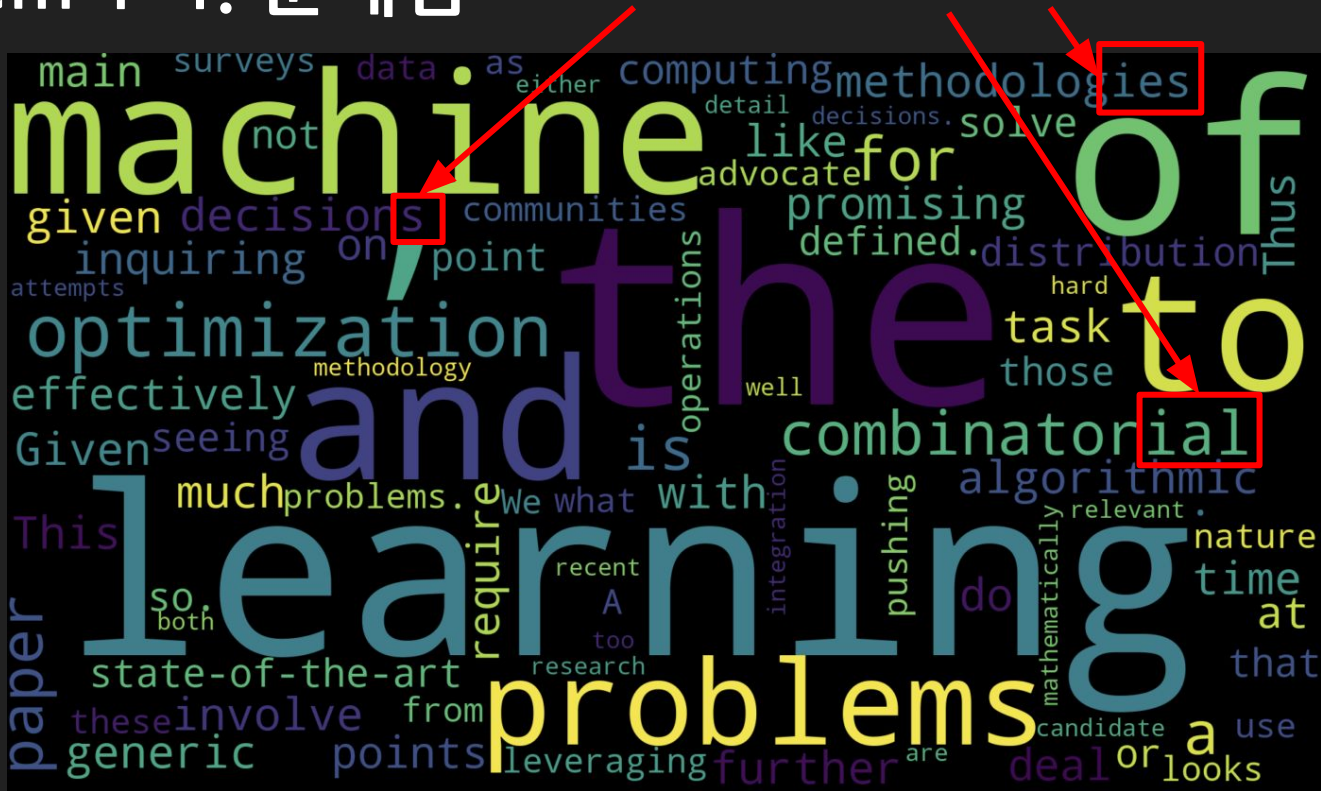


Problem 1-4. 문제점

문제점2. Stopword가 섞여있음

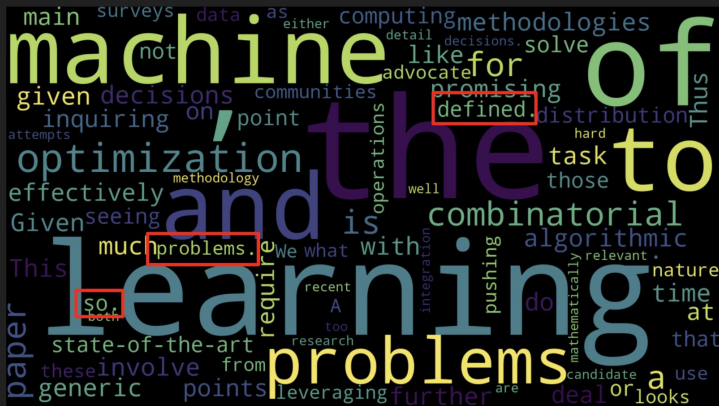


Problem 1-4. 문제점



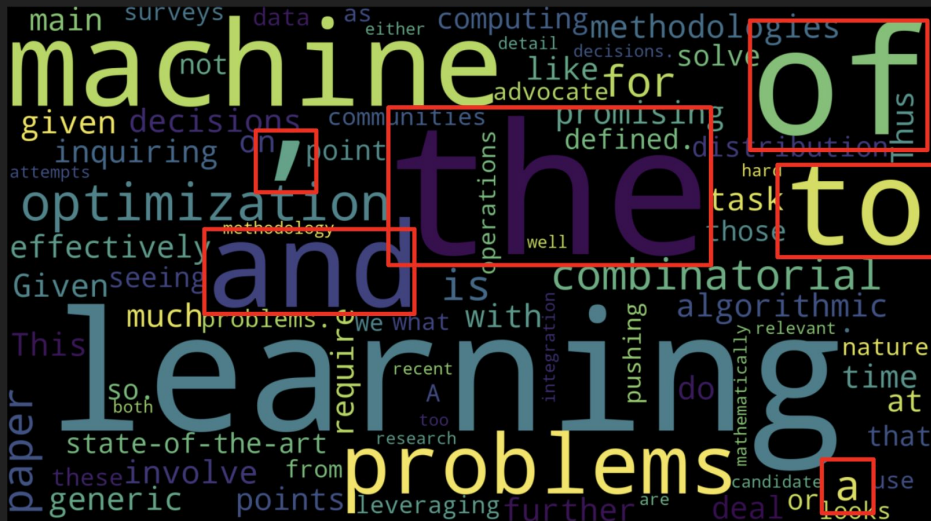
Problem 1-4. 개선방안

- 문제점 1. Punctuation이 토큰화된 단어에 붙어있는 경우가 종종 있음.
 - 예를 들어, defined 가 아니라 defined. 처럼 마지막에 ‘.’가 있음.
 - word_tokenize 함수에 preserve_line 옵션을 True로 넘긴게 원인.
 - 올바른 pos tagging을 위해서는 line이 끝난다는 정보를 보존해야함.
 - 하지만 Wordcloud를 생성하는 것과 같이 unique한 단어를 카운팅해야하는 경우, punctuation을 제거하는 처리를 추가로 해야함.
 - 따라서 word_tokenize 함수에 preseve_line 옵션을 False로 넘김.

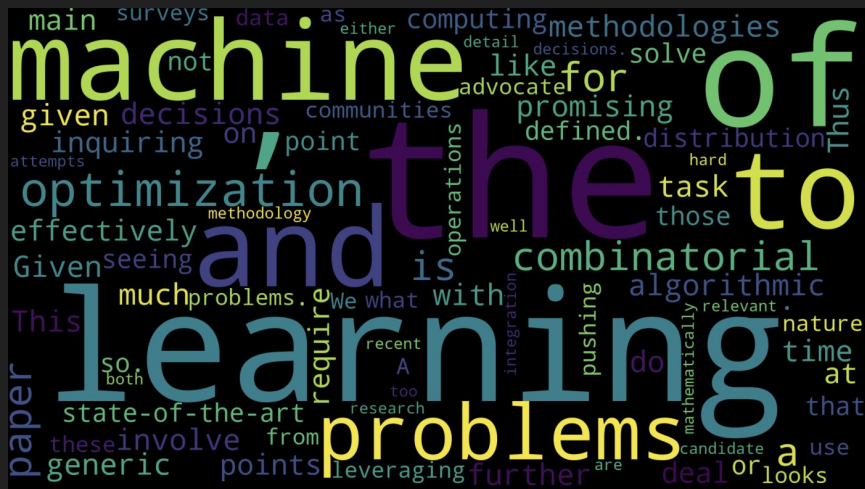


Problem 1-4. 개선방안

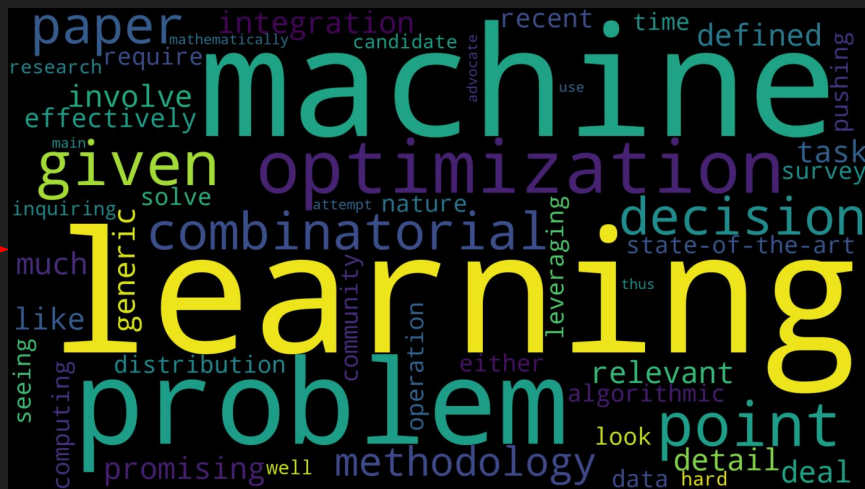
- 문제점 2. Stopword가 섞여있음
 - nltk.corpus 패키지의 stopwords 모듈을 이용하면 됨.
 - stopwords 모듈의 영어 stopwords를 가져와서 단어들에서 제외함.



Problem 1-4. Wordcloud 개선결과



Before



After

Problem 1-4. 최종결과

arXiv.org > cs > arXiv:1811.06128

Search or Arxiv:1811.06128v1 [cs.LG] (Help | Advanced Search)

Computer Science > Machine Learning

Machine Learning for Combinatorial Optimization: a Methodological Tour d'Horizon

Yoshua Bengio, Andrea Lodi, Antoine Prouvost

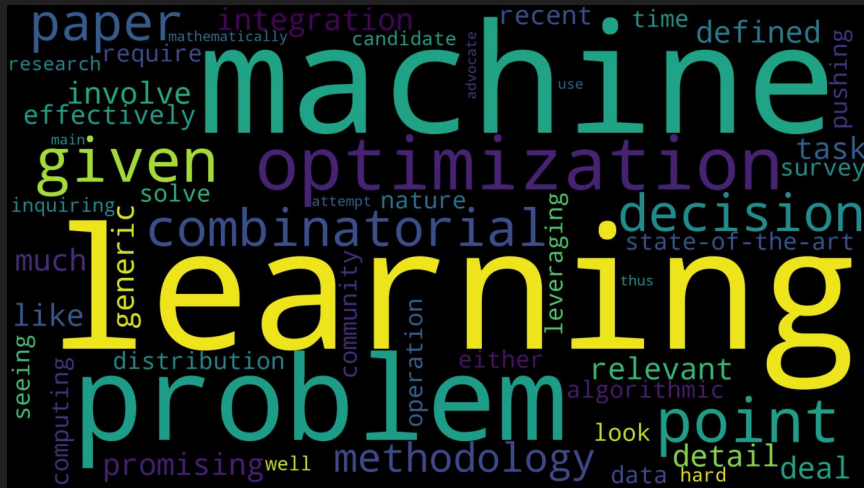
(Submitted on 15 Nov 2018)

This paper surveys the recent attempts, both from the machine learning and operations research communities, at leveraging machine learning to solve combinatorial optimization problems. Given the hard nature of these problems, state-of-the-art methodologies involve algorithmic decisions that either require too much computing time or are not mathematically well defined. Thus, machine learning looks like a promising candidate to effectively deal with those decisions. We advocate for pushing further the integration of machine learning and combinatorial optimization and detail methodology to do so. A main point of the paper is seeing generic optimization problems as data points and inquiring what is the relevant distribution of problems to use for learning on a given task.

Subjects: Machine Learning (cs.LG); Machine Learning (stat.ML)

Report number: DS4DM-2018-008

Cite as: arXiv:1811.06128 [cs.LG]
(or arXiv:1811.06128v1 [cs.LG] for this version)



Problem 2. 문자열 토큰 가공하기

Problem 2-1. 따옴표 제거

```
1  for (idx, token) in enumerate(tokens):
2      # '로 시작해서 '로 끝나는 단어
3      if token.startswith("'") and token.endswith("'"):
4          token = token.strip("'")
5          # 단어 도중에 '가 나오는 경우 뒤의 글자들 모두 삭제
6          location = token.find("'")
7          if location >= 0:
8              token = token[:location]
9          tokens[idx] = token
10 print(tokens)
```



```
['i', 'hello', 'hello', 'imlab', 'PH.D', 'I.B.M', 'snu.ac.kr', '127.0.0.1', 'galago.gif', 'ieee.803.99', 'naver.com', 'gigabyte.tw', 'pass..fail']
```

Problem 2-2. “.com” 토큰화하지 않기

- “.com”으로 끝나는 단어는 별도의 처리를 할 필요가 없음.

```
1  #.com으로 끝나는 단어는 토큰화하지 않는다는 것을 명시
2  for (idx, token) in enumerate(tokens):
3      if token.endswith(".com"):
4          tokens[idx] = token
5  print(tokens)
```



```
['i', 'hello', 'hello', 'imlab', 'PH.D', 'I.B.M', 'snu.ac.kr', '127.0.0.1', 'galago.gif', 'ieee.803.99', 'naver.com', 'gigabyte.tw', 'pass..fail']
```

Problem 2-3. 중간들어간 “.” 처리하기

```
1  for (idx, token) in enumerate(tokens):
2      #". "로 split
3      parts = token.split(".")
4      #모두 길이가 1인 경우에 한하여 "."을 지운다.
5      if all(len(part) == 1 for part in parts):
6          token = "".join(parts)
7      tokens[idx] = token
8  print(tokens)
```



```
['i', 'hello', 'hello', 'imlab', 'PH.D', 'IBM', 'snu.ac.kr', '127.0.0.1',  
'galago.gif', 'ieee.803.99', 'naver.com', 'gigabyte.tw', 'pass..fail']
```

Problem 3. Zipf's law

Problem 3-1. Unigram 플롯팅하기

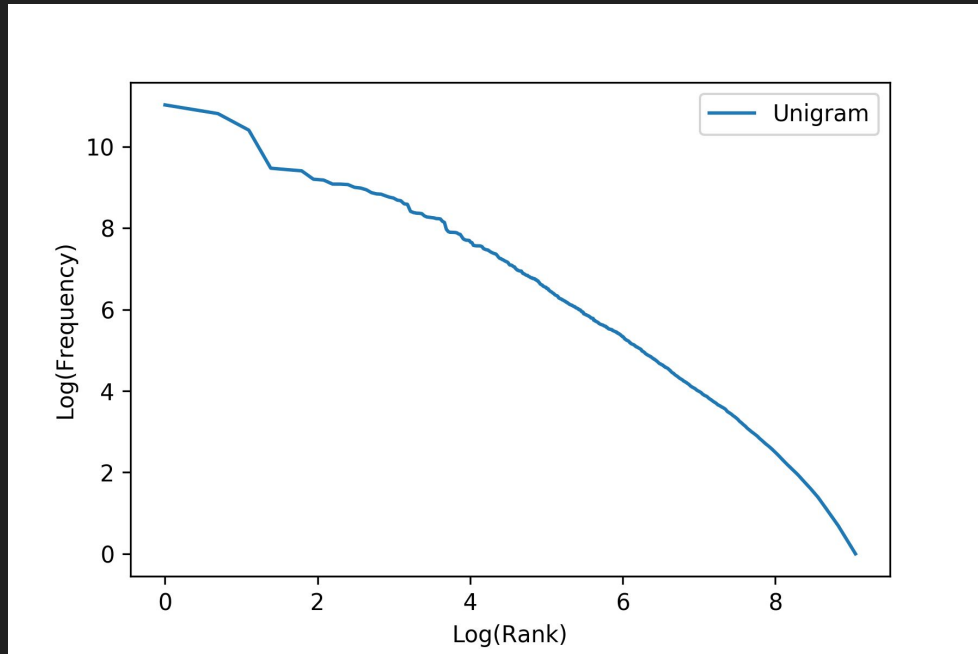
- zipf_law 함수는 아래와 같이 구성할 수 있음.
- count_words 함수는 word 리스트를 받아서 dict<word, freq>를 리턴함
- sort_words_by_frequency 함수는 word별 카운트를 받아서, 빈도가 많은 것에서 적은 순으로 정렬함.
- calculate_freq_and_rank_of_words는 freq와 rank를 리턴하는 함수.

```
1 def zipf_law(words: list) -> (list, list):  
2     word_count_dict = count_words(words)  
3     word_rank_list = sort_words_by_frequency(word_count_dict)  
4     return calculate_freq_and_rank_of_words(word_rank_list, word_count_dict)
```


Problem 3-1. Unigram 플롯팅하기

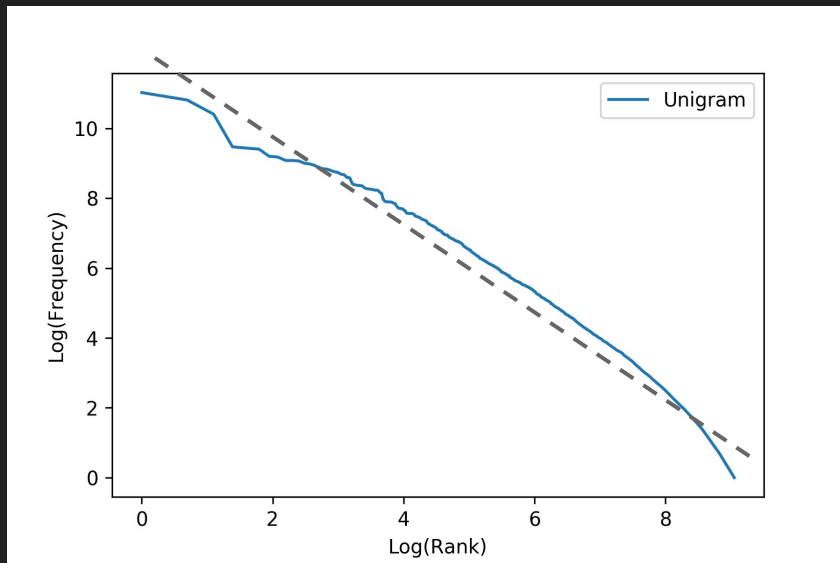
```
1 freq_list, rank_list = zipf_law(tokenized_words)
2 log_rank_list = np.log(rank_list)
3 log_freq_list = np.log(freq_list)
4 plt.plot(log_rank_list, log_freq_list, label='Unigram')
5
6 plt.xlabel('Log(Rank)')
7 plt.ylabel('Log(Frequency)')
8 plt.legend()
9 plt.show()
```

Problem 3-1. Unigram 플롯팅하기



Problem 3-1. Unigram 플롯팅하기

- Zipf's law는 빈도를 f , 순위를 r 이라고 하면 $r \times f = k$ 로 쓸 수 있음.
- 각 항에 로그를 취하면 $\log(r \times f) = \log(r) + \log(f) = \log(k) = k'$ 로 변형됨.
- Zipf's law에 따르면 빈도와 순위에 로그를 취해서 플롯팅을 하면 음의 기울기의 **선형함수**가 되어야함.
- Rank가 낮을 때와 높을 때 선형함수에서 벗어나는 것을 확인할 수 있음.



Problem 3-2. Bigram, Trigram 플롯팅하기

- nltk의 bigram과 trigram 함수를 사용해서 bigram, trigram 을 얻을 수 있음.
- 이때 리턴값이 tuple이므로 join을 해서 string으로 뭉쳐놔야함.

```
1 bigram_list = list(map(lambda x: ' '.join(x), nltk.bigrams(tokenized_words)))
2 trigram_list = list(map(lambda x: ' '.join(x), nltk.trigrams(tokenized_words)))
3 print(bigram_list[:20])
4 print(trigram_list[:20])
```



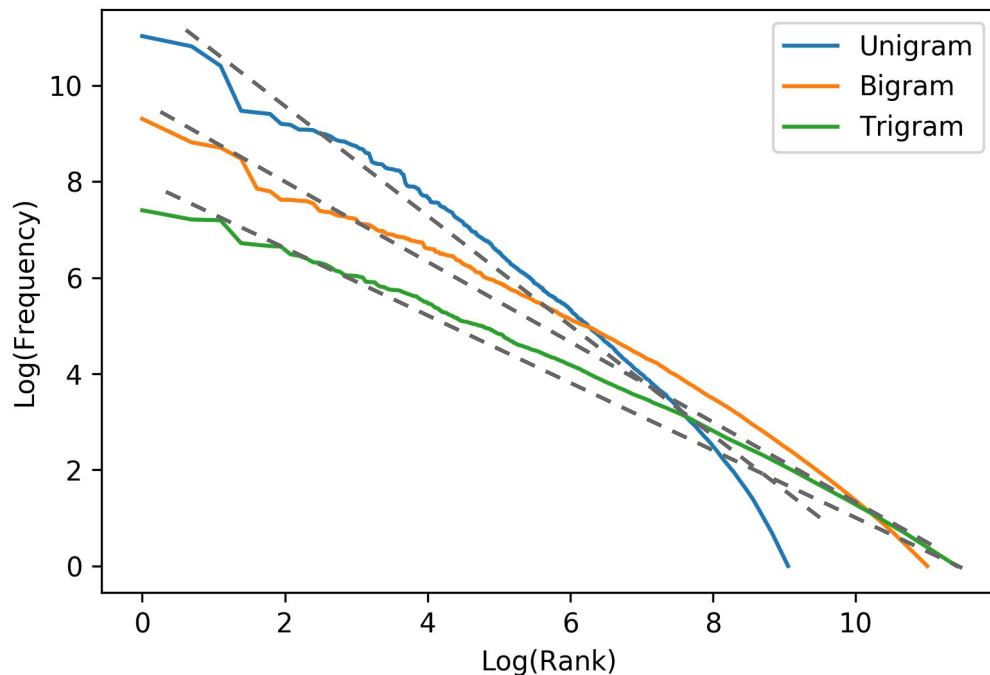
```
['in the', 'the beginning', 'beginning god', 'god created', 'created the', 'the heaven', 'heaven and', 'and the', 'the earth', 'earth and', 'and the', 'the earth', 'earth was', 'was without', 'without form', 'form and', 'and void', 'void and', 'and darkness', 'darkness was']
['in the beginning', 'the beginning god', 'beginning god created', 'god created the', 'created the heaven', 'the heaven and', 'heaven and the', 'and the earth', 'the earth and', 'earth and the', 'and the earth', 'the earth was', 'earth was without', 'was without form', 'without form and', 'form and void', 'and void and', 'void and darkness', 'and darkness was', 'darkness was upon']
```

Problem 3-2. Bigram, Trigram 플롯팅하기

- Unigram과 동일한 방식으로 bigram과 trigram도 플롯팅을 함.

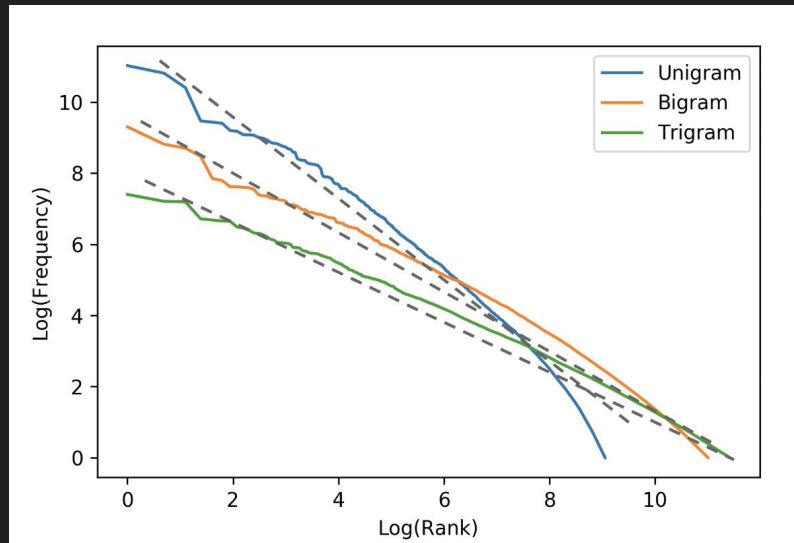
```
1 plt.plot(np.log(rank_list), np.log(freq_list), label='Unigram')
2 bigram_freq_list, bigram_rank_list = zipf_law(bigram_list)
3 plt.plot(np.log(bigram_rank_list), np.log(bigram_freq_list), label='Bigram')
4 trigram_freq_list, trigram_rank_list = zipf_law(trigram_list)
5 plt.plot(np.log(trigram_rank_list), np.log(trigram_freq_list), label='Trigram')
6
7 plt.xlabel('Log(Rank)')
8 plt.ylabel('Log(Frequency)')
9 plt.legend()
10 plt.show()
```

Problem 3-2. Bigram, Trigram 플롯팅하기



Problem 3-2. Bigram, Trigram 플롯팅하기

- Unigram \Rightarrow Bigram \Rightarrow Trigram 으로 갈수록 선형에 가까워지는 것을 확인할 수 있음.



감사합니다.