박홍준 연구 포트폴리오

포트폴리오 (웹사이트 버전)
https://hpark46.github.io/

TABLE OF CONTENTS

TABLE OF CONTENTS

01

Semantic Textual Similarity

http://velog.io/@howay96/Korean-Semantic-Textual-Similarity

1. 문장 유사도 추론 (STS

두 Benchmark 데이터를 Random Sampling 하여 튜닝한 모델을 API화 하였습니다.

- 모델: Klue/roberta-large
- Tokenizer: BertTokenizer
- Dataset
  - Klue-STS
  - Kor-STS
  - 문장1, 문장2 와 real-label/score 만 사용
- Preprocessing
  - 따옴표 + 모든 특수부호 제거
  - Labels = Labels / 5
- Train/Test 데이터 셋 9:1로 분배

```
train length: 10565
test length: 1103
TRAIN DISTRIBUTION
Semantic similarity pair with score 0 to 0.5 : 3200  (30.2886890676762878)
Semantic similarity pair with score 0.5 to 1.5 : 1229  (11.6327496450544258)
Semantic similarity pair with score 1.5 to 2.5 : 688  (6.5120681495504025%)
Semantic similarity pair with score 2.5 to 3.5 : 1343  (12.7117841930903918)
Semantic similarity pair with score 3.5 to 4.5 : 3559  (33.686701372456238)
Semantic similarity pair with score 4.5 to 5 : 504  (4.7704685281590158)
--------------------------------------------------------------------------
TEST DISTRIBUTION
Semantic similarity pair with score 0 to 0.5 : 334  (30.281051677243888)
Semantic similarity pair with score 0.5 to 1.5 : 129  (11.6953762466001828)
Semantic similarity pair with score 1.5 to 2.5 : 72  (6.5276515855675438)
Semantic similarity pair with score 2.5 to 3.5 : 140  (12.6926563916591148)
Semantic similarity pair with score 3.5 to 4.5 : 371  (33.635539437896658)
Semantic similarity pair with score 4.5 to 5 : 54  (4.8957388939256588)
```

△ Klue-STS 데이터 분포

```
Train Dataset Length: 5749
Train DISTRIBUTION
Semantic similarity pair with score 0 to 0.5 : 592  (10.2974430335710578)
Semantic similarity pair with score 0.5 to 1.5 : 833  (14.4894764306835998)
Semantic similarity pair with score 1.5 to 2.5 : 902  (15.6896851626369878)
Semantic similarity pair with score 2.5 to 3.5 : 1359  (23.6388937206470788)
Semantic similarity pair with score 3.5 to 4.5 : 1435  (24.9608627587406588)
Semantic similarity pair with score 4.5 to 5 : 362  (6.2967472603931128)
```

△ Kor-STS 데이터 분포

모델 / 파인 튜닝

Weight를 공유하는 Siamese BERT Network를 이용해, 각 sentence embedding에 mean-pooling operation을 추가로 진행해 Cosine Similarity 을 구하는 방식으로 설계되어 있습니다.

	V1	V2	V3	V4
Batch Size	8	16	8	16
Learning Rate	1.00E-05	1.00E-05	2.00E-05	3.00E-05
Warm up	0.1	0.1	0.2	0.6
Weight Decay	0.01	0	0	0.01
Epochs	4	4	5	5

Klue paper에 명시된 hyperparameter를 모두 시도하기에는 제약이 있어 4번에 random search로 진행하였습니다.

```python
class CustomPooling(nn.Module):
    def __init__(self):
        super(CustomPooling, self).__init__()

        self.robert = AutoModel.from_pretrained("klue/roberta-large")


        self.cos_score = nn.Sequential(
            nn.Identity()
        )

    def forward(self, senone, sentwo):
        output_one = self.robert(input_ids=senone['input_ids'], attention_mask=senone['attention_mask'],
                                 token_type_ids=senone['token_type_ids'])
        output_two = self.robert(input_ids=sentwo['input_ids'], attention_mask=sentwo['attention_mask'],
                                 token_type_ids=sentwo['token_type_ids'])

        pooled_one = mean_pooling_fn(output_one, senone['attention_mask'])
        pooled_two = mean_pooling_fn(output_two, sentwo['attention_mask'])


        cos_sim = torch.cosine_similarity(pooled_one, pooled_two)
        logit = self.cos_score(cos_sim)

        return logit
```

△ Model

```python
def mean_pooling_fn(output, attention_mask):
    embedding = output.last_hidden_state # (batch len, longest sentence length, 1
    att_msk = attention_mask             # (batch_len, 1024)
    mask = att_msk.unsqueeze(-1).expand(output.last_hidden_state.size()).float()
    masked_embedding = output.last_hidden_state * mask
    me_sum = torch.sum(masked_embedding, 1)       # (batch_len, 1024)
    ms_sum = torch.clamp(mask.sum(1), min=1e-9)  # (batch_len, 1024)
    mean_pool = me_sum/ms_sum                     # (batch_len, 1024)
    return mean_pool
```

△ Mean-pooling

Klue-STS Best Model

Klue-STS에만 4번 튜닝을 진행한 결과,
제일 좋은 성능을 보여주는 모델은 validation set에서

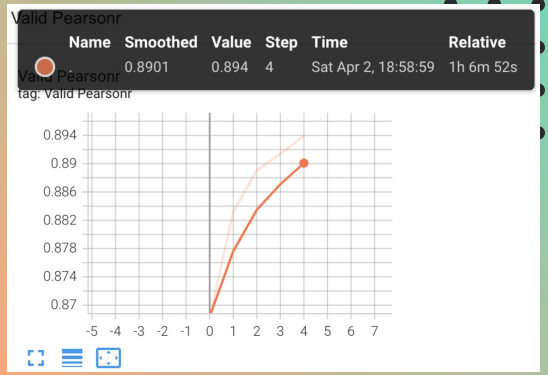
	My Model	Klue 최고 성능
●	Pearsonr: 89.4	Pearsonr: 93.35
●	F1 Score: 84.9	F1 Score: 86.63
●	Avg Loss: 0.594	

를 기록하였습니다. (성능 부족은 epoch의 차이로 보입니다)

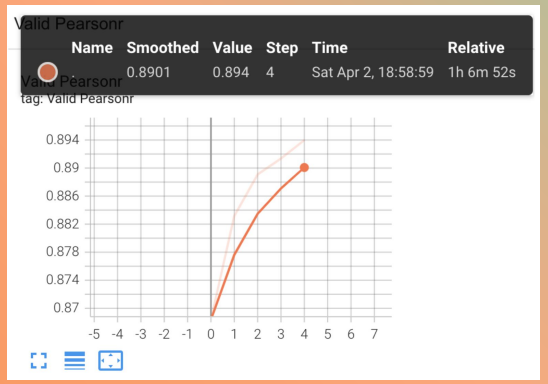
이 모델을 Kor-STS test set에 예측시켜본 결과

●	Spearman: 77.63	F1 Score: 77.93

으로 만족스럽지 않은 성능을 보여 두 데이터를 섞어 나온 결과가 궁금해 Random Sampling 두 데이터 셋에 대해 다시 fine-tuning 하였습니다.



△ Pearsonr



△ F1 Score

Final Model & API

다시 Fine-Tuning을 진행한 결과,
Klue-sts-dev
- Pearsonr: 88.45 (-0.95)
- F1 Score: 85.02 (-0.11)

Kor-STS-test
- Spearman: 83.55 (+5.92)
- F1 Score: 83.83 (+5.9)

의 성능을 보여주었습니다.

Metric
Pearson과 Spearman은 다른 모델들의 성능과 제 모델 성능을 비교하기 위해 사용하였습니다.

F1 Score: Predict 값과 label에 0-5 range 중 3에 Threshold를 부여해 계산하였습니다

API 기능

Get (1:1 점수 Predict)

```
requests.get("http://127.0.0.1:5000/predict",
        {'sentence_one': '한 남자가 밧줄을 타고 올라간다.',
         'sentence_two': '한 남자가 밧줄을 타고 올라가고 있다.'})
```

Similarity Score:
b'4.906567573547363'

Post (1:N 점수 Predict)

```
op = open('/content/drive/MyDrive/data/KorNLUDatasets/KorSTS/sts_sentence.csv', 'r')
post_example = requests.post("http://127.0.0.1:5000/predict",
                             params={'base_sen': '한 남자가 식당에서 밥을 먹고 있다.'},
                             files={'uploaded_file': op})
```

Top 3 유사 문장:
1. 한 남자가 음식을 먹고 있다.
2. 남자가 먹고 있다.
3. 한 남자와 여자가 식당의 테이블에 앉아 있다.

02

MatchSum

추출요약 구현

http://velog.io/@howay96/MatchSum

Overview

사용모델

- klue/roberta & Kobert
- 형태소 분석이 가능한 KorBERT를 사용하고 싶었으나 시간의 편의상 다른 bert 모델을 사용하였습니다.

사용한 데이터 셋

- 문서요약 텍스트 train_original_news (AI-Hub)
- Text / Extractive / Abstractive 3개의 Column만 사용하였습니다

Cleaning Data

- Abstractive 요약본이 없는 기사 (4개)
- Extractive Column에 None 포함 (8개)
- 는 제거하였습니다.

	id	category	media_type	media_sub_type	media_name	size	char_count	publish_date	title	text	annotator_id	document_quality_scores	extractive	abstractive
0	290741778	종합	online	지역지	광양신문	small	927	2018-01-05 18:54:55	논 타작물 재배, 2월 말까지 신청하세요	[{'index': 0, 'sentence': '농당 조사료 400만원...지원별 ...	11	{'readable': 4, 'accurate': 3, 'informative': ...	[2, 3, 10]	[전라남도가 쌀 과잉문제를 근본적으로 해결하기 위해 올해부터 벼를 심었던 논에 벼 ...
1	290741792	종합	online	지역지	광양신문	small	764	2018-01-05 18:53:20	여수시, 컬러빌리지 아우리...'색채와 빛'도시 완성	[{'index': 0, 'sentence': '벽이 투입, 고소천사벽화-자신마을...	12	{'readable': 4, 'accurate': 4, 'informative': ...	[2, 4, 11]	[여수시는 컬러빌리지 사업에 8억원을 투입하여 '색채와 빛' 도시를 완성하여 고소천...
2	290741793	스포츠	online	지역지	광양신문	medium	1066	2018-01-05 18:52:15	"새해 첫기 받고 올해는 반드시 일내자!"	[{'index': 0, 'sentence': '전남드래곤즈 해맞이 다짐...선수 영...	13	{'readable': 4, 'accurate': 4, 'informative': ...	[3, 5, 7]	[전남드래곤즈 임직원과 선수단이 4일 구봉산 정상에 올라 일출을 보며 2018년 구...
3	290741794	정치	online	지역지	광양신문	small	746	2018-01-05 18:50:17	농업인 역량 강화, 새해 실용교육 실시	[{'index': 0, 'sentence': '11~24일, 매실·감·참다래 등...	12	{'readable': 4, 'accurate': 4, 'informative': ...	[2, 3, 4]	[광양시는 농업인들의 경쟁력을 높이고, 소득안정을 위해 매실·감·참다래 등 지역특화...
4	290741797	종합	online	지역지	광양신문	small	978	2018-01-05 18:52:36	타이완 크루즈 관광객 4천여 명'전남'온다	[{'index': 0, 'sentence': '홍콩 크루즈선사'아쿠아리우스'4...	13	{'readable': 4, 'accurate': 4, 'informative': ...	[3, 7, 4]	[올해 4월과 6월 두 차례에 걸쳐 타이완의 크루즈 관광객 4000여명이 여수에 입...

△ 문서요약 텍스트 데이터셋

Preprocessing

기본 전처리

- 영문/한글/숫자 제외 모든 Special Characters는 제거 하였습니다
- 논문상에서 non-anonymized version을 사용하는 것을 보고 ㅁㅁㅁ 기자 와 같은 자주 등장하는 문장들은 따로 다루지 않았습니다

MatchSum Paper에서 명시된 3 전처리 Steps

- Candidates Pruning (BertExt)
- Candidate Combination
- Sorting based on Rouge (1+2+L)/3

Candidates Pruning
MatchSum paper 내에서 trigram blocking을 사용하지 않는 BertSum 모델을 사용해 원본 기사를 5-10줄로 Pruning 하였으나, 이를 대체할 방법으로 BertSum Paper에 명시된, Oracle Summary 제작에 사용되는 Greedy-Selection 알고리즘을 변형해 사용하였습니다.

Preprocessing

MatchSum paper 내에서 trigram blocking을 사용하지 않는 BertSum 모델로 문장당 점수를 부여해 원본 기사를 5-10줄로 Pruning 하였으나, 이를 대체할 방법으로 BertSum Paper에 명시된, Oracle Summary 제작에 사용되는 Greedy-Selection/Combination-Selection 알고리즘을 변형해 사용하였습니다.

- Extractive Column (사람이 추출한 3문장)을 포함해 Rouge 1&2 점수를 최대화하는 문장 2개를 추가로 선정하는 방법입니다.

5개의 문장으로 이루어진 'Extractive'로
5C2 + 5C3 = 20개의 Candidate Summary 제작 후, 본문과 비교하여 (Rouge 1 + 2 + L)/3 점수가 가장 높은 순서대로 정렬하였습니다.

	greedy	highest
0	[2, 3, 10, 1, 0]	[2, 3, 10, 5, 8]
1	[2, 4, 11, 1, 0]	[2, 4, 11, 9, 3]
2	[3, 5, 7, 2, 1]	[3, 5, 7, 2, 4]
3	[2, 3, 4, 0, 1]	[2, 3, 4, 0, 6]
4	[3, 7, 4, 2, 1]	[3, 7, 4, 2, 0]

△ Pruned Document Index (각 5문장)

*Greedy Algorithm 사용 시 가장 짧은 문장만 (index 0 &1) 추가로 선정하는 경향이 있어 Highest Column의 index 사용

Tokenization & Model

Document (본문), Candidate Summary, Gold Summary (사람 작성 요약본) 모두
[CLS] + Tokenized + [SEP] Format으로 토큰화 하였는데
Google Colab가 용량을 handle 하지 못해

- Candidate Summary 개수의 조정
- Candidate Summary max_length 조정 (Default: 180)
- Document max_length 조정 (Default: 512)
- Batch Size 조정

등을 필요로 하였습니다.

Model

- Siamese-Bert architecture (Document, Candidate & Gold summary의 Embedding 필요)
- [CLS] 토큰 벡터로 문서/요약본 Representation
- MarginRankingLoss 사용 (Margin-Based triplet loss)

```python
class MatchSum(nn.Module):
    def __init__(self):
        super(MatchSum, self).__init__()

        self.robert = AutoModel.from_pretrained("klue/roberta-large")

    def forward(self, text_id, candidate_id, summary_id):

        #document embedding
        doc = self.robert(input_ids=text_id['input_ids'], attention_mask=text_id['attention_mask'],
                          token_type_ids=text_id['token_type_ids'])
        doc_emb = doc[0][:,0] # batch_size, 1024

        #abst embedding
        abst = self.robert(input_ids=summary_id['input_ids'], attention_mask=summary_id['attention_mask'],
                          token_type_ids=summary_id['token_type_ids'])
        abst_emb = abst[0][:,0] # batch_size, 1024

        #summary score
        f_dcs = torch.cosine_similarity(abst_emb, doc_emb, dim=-1)

        #candidate embedding
        ids = []
        token_type = []
        attention = []
        for doc_tok in candidate_id:
            ids.append(doc_tok['input_ids'])
            token_type.append(doc_tok['token_type_ids'])
            attention.append(doc_tok['attention_mask'])
        ids = torch.stack(ids).view(-1, len(ids[0][0]))
        token_type = torch.stack(token_type).view(-1, len(token_type[0][0]))
        attention = torch.stack(attention).view(-1, len(attention[0][0]))


        cand = self.robert(input_ids=ids, attention_mask=attention,
                          token_type_ids=token_type)
        cand_emb = cand[0][:,0]


        #candidate score
        doc_emb = doc_emb.unsqueeze(1).expand_as(cand_emb)

        f_dcij = torch.cosine_similarity(cand_emb, doc_emb, dim=-1)


        return {'cand_score': f_dcij, 'abst_score': f_dcs}
```

△ MatchSum 모델

03

Hate Speech Detection

http://velog.io/@howay96/Korean-Hate-Speech-Detection

Overview

사용모델

- KcElectra-base
- 네이버 뉴스 댓글과 대 댓글을 수집해 tokenizer와 Electra 모델을 pretrain 한 모델로, 가장 적합한 모델이라고 판단하였습니다.

사용한 데이터 셋

- Korean-Hate-Speech-Detection (Kaggle)
- Comments, Hate Column만 사용하였습니다

Data Cleaning

- 한자 / special characters
- URL & HTML

```
none          3486
offensive     2499
hate          1911
Name: hate, dtype: int64

none: 0.44148936170212766 of the dataset
offensive: 0.31648936170212766 of the dataset
hate: 0.24202127659574468 of the dataset
```

△ Hate-Speech Dataset (Hate Column Label) 분포

Preprocessing

Multi-Class Classification Task 을 위해 Label을
- [1,0,0] : None (0)
- [0,1,0]: Offensive (1)
- [0,0,1]: Hate (2)

으로 변형하였습니다

```
[[0, 0, 1], [1, 0, 0], [0, 0, 1], [1, 0, 0], [0, 0, 1], [1, 0, 0], [0, 0, 1],
[' 현재 호텔주인 심정  아18 난 마른하늘에 날벼락맞고 호텔망하게생겼는데 누군 계속 추모받네     ', '
```

Model
Discriminator을 통해 나온 CLS 토큰 벡터를 3개의 Label에 대해 각각의 확률을 Return 하는 Classification Head로 통과시켜 Prediction을 진행합니다.
- Optimizer: AdamW
- Loss Function: CrossEntropyLoss

```python
class HateClassifier(nn.Module):
    def __init__(self, hidden_size, n_label):
        super(HateClassifier, self).__init__()

        dropout_rate = 0.5
        linear_layer_size = 515
        self.kcelectra = AutoModel.from_pretrained("beomi/KcELECTRA-base")
        self.classifier = nn.Sequential(
            nn.Linear(hidden_size, linear_layer_size),
            nn.ReLU(),
            nn.Dropout(dropout_rate),
            nn.Linear(linear_layer_size, n_label),
        )

    def forward(self, input_ids = None, attention_mask = None, token_type_ids = None):
        output = self.kcelectra(input_ids=input_ids, attention_mask=attention_mask
                                , token_type_ids=token_type_ids)
        cls = output[0][:,0]
        logit = self.classifier(cls)

        return logit
```

△ Hate-Speech Classifier 모델

Fine-Tuning/Test

4개의 다른 Hyperparameter Setting에서 Fine-Tuning을 진행하였습니다

- V3 : V2 Setting에서 Special Character 와 Punctuations를 제거하지 않은 데이터 사용
- V4: 사용 모델 KcElectra-base → kcelectra-base-v3-discriminator

모델 성능
V2 epoch 1 (Validation Set) 에서

- 74%의 Accuracy
- 0.633의 Loss

을 기록하였습니다

Kaggle Competition Score : 0.66421 (#2)

	V1	V2	V3	V4
Batch Size	32	32	32	32
Learning Rate	2.00E-05	2.00E-05	2.00E-05	2.00E-05
Dropout Rate	0.1	0.5	0.5	0.5
Eps	1.00E-08	1.00E-08	1.00E-08	1.00E-08
Epochs	4	4	4	4
Hidden Layer	768	515	515	515
Warm up	len*0.1	len*0.1	len*0.1	len*0.1

△ Fine-Tuning Details

	comments	label
0	ㅋㅋㅋㅋ 그래도 조아해주는 팬들 많아서 좋겠다 ㅠㅠ 니들은 온유가 안만져줌 ㅠㅠ	2
1	둘다 넘 좋다~행복하세요	0
2	근데 만원이하는 현금결제만 하라고 써놓은집 우리나라에 엄청 많은데	0
3	원곡생각하나도 안나고 러블리즈 신곡나온줄!!! 너무 예쁘게 잘봤어요	0
4	장현승 얘도 참 이젠 짠하다...	1

△ 예시 Prediction

04

Decision Tree Learning & Linear Classifiers

프로젝트 요약

Information Gain의 최대화 하는 Attribute 선정 방법을 포함한 Entropy based Decision tree를 구현하였습니다. 알고리즘은 Overfitting을 방지하기 위해 관련 없는 Node를 Prune 하며, 학습 시 tree의 정확도를 모니터하였습니다.

추가로, Perceptron Learning Rule과 Hard Threshold를 사용하는 Classifier, Logistic Regression을 바탕으로 분류하는 Classifier를 구현하여, Clean/Noisy 데이터와 Learning Rate가 학습에 미치는 영향을 학습할 수 있는 계기가 되었습니다.

Decision tree 학습용 예시 데이터로는 Iris 데이터 셋을 사용하였고, Classifier 학습을 위해 Earthquake 데이터 셋을 사용하였습니다.

05

Boyer-Moore 문자열 매칭

https://velog.io/@howay96/Boyer-Moore-%EB%AC%B8%EC%9E%90%EC%97%B4-%EB%

프로젝트 요약

Exact String Matching으로 자주 사용되는 Boyer-Moore 문자열 매칭 알고리즘을 구현하였습니다.

- Bad Character Heuristic
- Good Suffix Heuristic

두 가지로 이루어진 알고리즘으로 pattern의 포지션을 기준으로 가장 optimal 한 shift를 적용합니다.

Preprocessing

- Bad Character Heuristic
  - 패턴이 가진 character당 얼마만큼 shift 해야 하는지 matching 이전에 미리 계산 합니다
- Good Suffix Heuristic
  - Border와 Shift 두 array를 계산합니다
  - Shift[i] : i-1에서 mismatch가 일어날 시 shift할 값
  - Border: 각 패턴의 포지션에 대해 가장 넓은 border의 가장 빠른 index

```python
def preprocess_badchar(pattern):
    default = -1                    # default: jumping over mismatch
    badchar = [default for _ in range(256)] # all char

    i = 0
    for char in pattern:
        badchar[ord(char)] = i
        i += 1

    return badchar
```

△ Bad Character 전처리

```python
def preprocess_goodsuff(pattern, border, shift, pl):
    ione = pl
    itwo = pl + 1
    border[ione] = pl + 1   # border is outside of pattern

    # strong good suffix
    for _ in range(pl):
        while (itwo < pl + 1) and (pattern[ione-1] != pattern[itwo-1]): #th

            if shift[itwo] == 0:
                shift[itwo] = itwo - ione # amount to jump

            itwo = border[itwo] #


        ione, itwo = ione - 1, itwo - 1
        border[ione] = itwo


    # partial good suffix
    for index in range(len(border)):
        if shift[index] == 0:
            if index > border[0]:
                shift[index] = border[border[0]]

            else:
                shift[index] = border[0] # widest boarder of the pattern
```

△ Good Suffix 전처리

알고리즘 / 사용 예시

Input 예시

- Array 1: List of patterns to be matched
- Array 2: List of Text to be matched

```
P = ["aaa", "aaaabb", "aabbcc", "abb", "bcc", "bbcc",
T = ["aaaabbaabbccdd", "aabbccddcceeaabbaaaaa"]
```

Output 예시

- Mode 1
  - 각 패턴이 text에 존재하는지 Yes/No로 Return
- Mode 2
  - 각 Text에 대해 각 패턴의 첫 번째 appearance의 첫 번째 index를 return
- Mode 3
  - 각 Text에 대해 각 패턴의 모든 appearance의 첫 번째 index를 return

```python
def boyer_moore(pattern, text, mode):
    position = []

    pl, tl, index = len(pattern), len(text), 0 # pattern len

    border = [0 for _ in range(pl + 1)]
    shift = [0 for _ in range(pl + 1)]

    rb = preprocess_badchar(pattern)        # reference for
    preprocess_goodsuff(pattern, border, shift, pl) # prepro
    # print(rb[ord("a")])

    while (index < tl - pl + 1):
        # print(index)

        # check for mismatch index (on the text) (examine fr
        mismatch = -1

        for x in range(1, pl+1): # 1~8
            if text[index + pl - x] != pattern[pl - x]: #8 -
                mismatch = index + pl - x # index + 8 - (1~8
                break

        if mismatch == -1:  # matched
            position.append(index) # list.append(index) # to
            if mode != 3:
                return position
            else:
                index = index + shift[0]
        # determine what shift to take (bad_char/good_suff)
        else:
            gsi = good_suff_shift(shift, index, mismatch)
            bci = bad_char_shift(rb, text, index, mismatch)
            # print(gsi, bci)

            index = index + max(gsi, bci)

    return position #no match      (when finding all possibl
```

△ Main 알고리즘

06

Automated Reasoning

프로젝트 요약

명제 논리 추론 알고리즘 연습을 위해 Backus-Naur For
문법을 나타낼 수 있는 구조를 만들었고, 첫 번째 추론
방법은 진리표 enumeration 알고리즘을 구현하였습니다

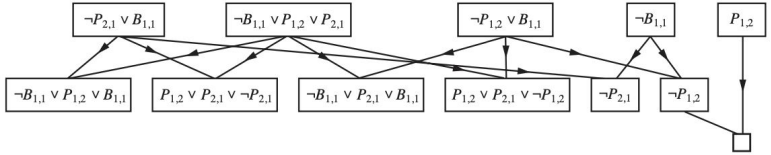
- 모든 Atomic Sentence에 True/False 값을 부여하
  지식베이스가 쿼리를 entail 하는지 판단합니다.

두번째 추론 방식으로는 지식베이스와 쿼리의 문법을
논리곱 정규형으로 변형해 대입하는 귀류법을 기반으로
한 Resolution 알고리즘을 구현하였습니다.

- 성공적으로 쿼리가 entail 될 수 있는지 리턴합니다

$$
\begin{aligned}
\textit{Sentence} &\rightarrow \textit{AtomicSentence} \mid \textit{ComplexSentence} \\
\textit{AtomicSentence} &\rightarrow \textit{True} \mid \textit{False} \mid P \mid Q \mid R \mid \ldots \\
\textit{ComplexSentence} &\rightarrow (\textit{Sentence}) \mid [\textit{Sentence}] \\
&\mid \neg \textit{Sentence} \\
&\mid \textit{Sentence} \wedge \textit{Sentence} \\
&\mid \textit{Sentence} \vee \textit{Sentence} \\
&\mid \textit{Sentence} \Rightarrow \textit{Sentence} \\
&\mid \textit{Sentence} \Leftrightarrow \textit{Sentence} \\
\text{OPERATOR PRECEDENCE} &: \neg, \wedge, \vee, \Rightarrow, \Leftrightarrow
\end{aligned}
$$

△ Backus-Naur Form



△ Resolution 알고리즘

07

Uncertain Inference

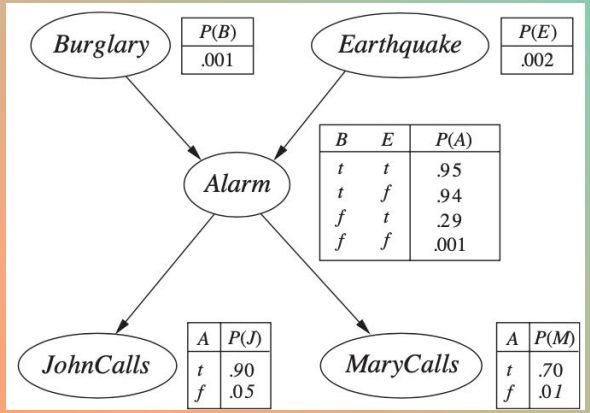
프로젝트 요약

Bayesian Network structure를 구현한 뒤, 세 가지 추론 알고리즘을 구현하였습니다.

Enumeration을 이용한 첫 번째 Exact Inference 알고리즘은
- XMLBIF 파일에서 주어진 Evidence에 따라 각 쿼리 variable의 확률을 리턴합니다.

그 다음으로 Approximate Inference를 위해서는
- Prior Distribution을 기반으로 모든 Event를 생성해 Evidence와 상반되는 Event를 제거하는 Rejection Sampling
- Evidence와 일정한 Event만 생성해 Rejection Sampling의 비효율성을 보완하는 Likelihood-weighting

두 알고리즘을 구현하였습니다



△ Bayesian Network 예시

08

모의 보험 데이터베이스

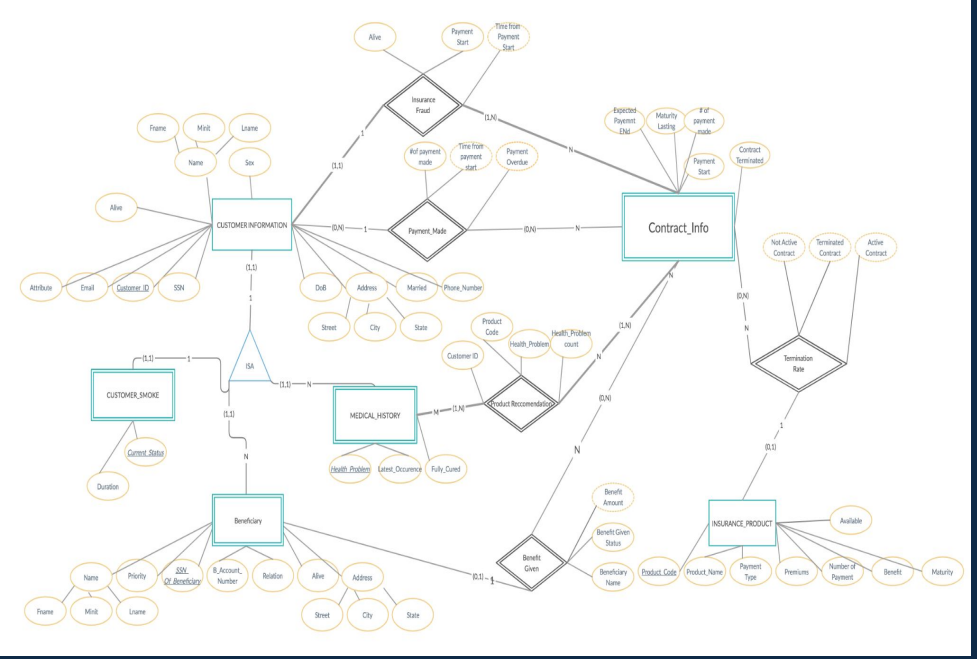
프로젝트 요약

HTML과 PHP를 사용해 로컬 서버에 생명보험 정보를 보고 편집할 수 있는 웹페이지를 만들었습니다.
데이터베이스는 MySQL로 구축되었으며, 데이터는 여섯 가지의 Entities와 다섯 가지의 Relationships로 이루어져 있습니다.

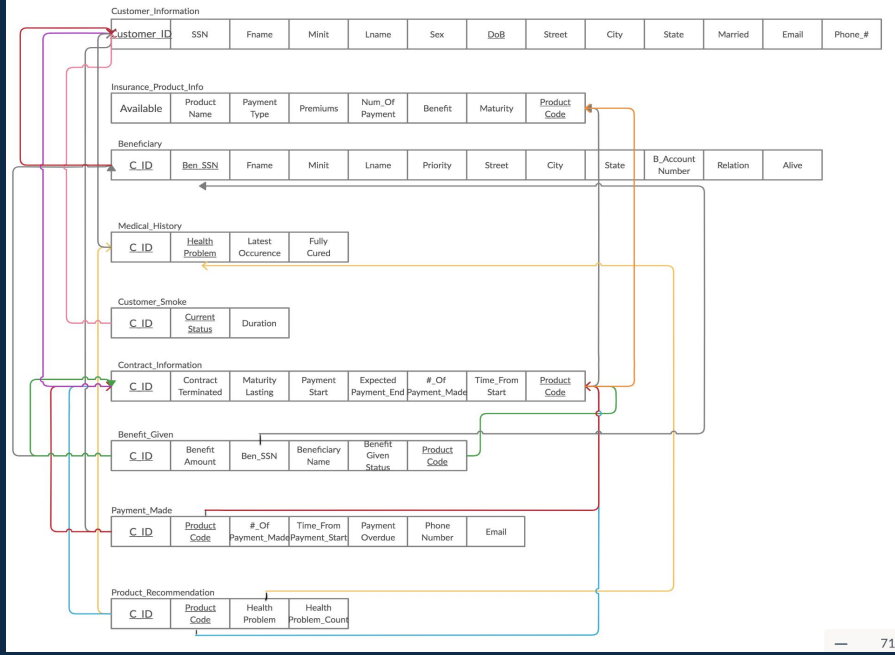
고객 계정으로 사이트에 접속하는 경우 non-key attributes를 수정할 수 있는 권한을 가지게 되며, 고객의 건강 기록을 바탕으로 추천하는 보험 상품들이 소개됩니다.

직원 계정은 더 많은 권한을 가지며 모든 데이터를 추가, 제거, 수정할 수 있는 권한을 가지게 됩니다.

- 프로젝트를 진행하면서 ER diagram, Relationship type mapping, 하고 데이터베이스의 시각화 연습할 기회가 되었습니다.



△ER Diagram



△Relation Diagram

프로젝트 Diagrams