빅콘테스트 프로젝트 결과보고서

뉴럴네트워크와 **TREE** 계열 **CLASSIFIER** 를 활용한 빅 테이블 학습, 하이퍼파라미터 최적화 과정

2016.08.02 ~ 09.30 이성현 (챌린지리그 개인참여)

1. 요약

전체 과정흐름

1. 데이터베이스 삽입
2. 가장 큰 테이블, 고정된 하이퍼파라미터로 **1** 차학습
3. 변수최적화 : 삭제 및 추가
4. 하이퍼파라미터 최적화
5. 모델 검증 및 제출

가장 높은 결과 (NN, ADABOOST, RANDOMFOREST, DECISIONTREE 중)

Claim 테이블을 중심으로 가능한 모든 변수를 추가해 라인당 85 개의 Variable 을 가진 상태에서

Adaboost classifier 사용, estimator 100, L-rate 0.9 에서 리콜-0.52 프리시전-0.73 F1-0.61 을 얻었습니다.

(Total 10 만 lines, test 2 만 lines, train 8 만 lines, random-nullfix 상태)

과정 일부가 제외되었습니다 : 3 번, 변수 최적화 섹션 전체.

최종제출 임박 시점에 test/train set 버그가 발견되어 변수 최적화 과정을 생략한 결과를 제출했습니다.

최초 빅 테이블로 하이퍼 파라미터 최적화만 적용한 결과이며,

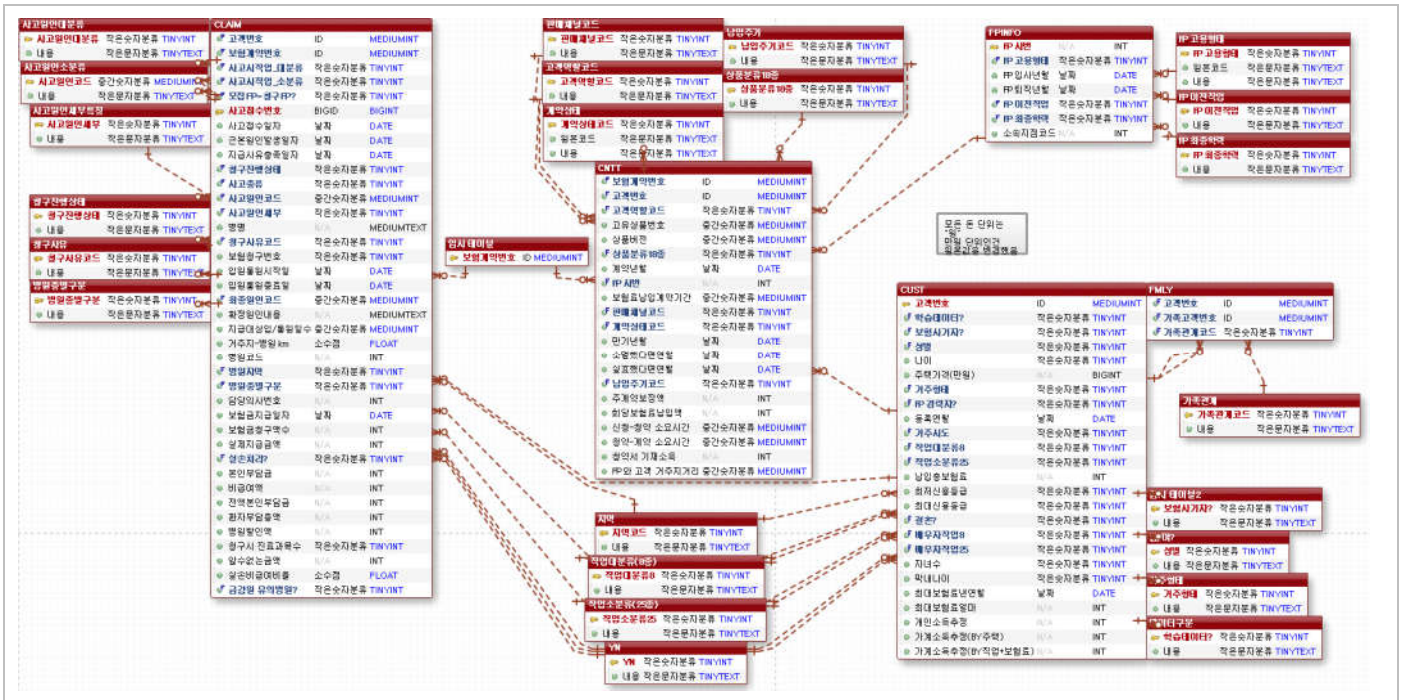
빠진 변수 최적화 부분은 원래 어떤식으로 접근하려 했는지 폐기한 내용을 활용해 컨셉만 설명했습니다.

2. 작업과정

데이터베이스 처리

이슈 1 : CSV 파일은 데이터 연결해 사용하기 힘듬

스키마 리버스 엔지니어링 후 파이썬 코드로 CSV 를 DB 화 함. Mysql 사용



스키마 : 테이블_실제이름.pdf, 테이블_한글이름.pdf

소스 : /dbinput/csv_input.py

이슈 2 : NULL 값의 처리 (머신러닝 툴 오류방지)

원본 컬럼의 평균과 분산을 유지하면서 Random 생성해 삽입 (원래 분포가 극단적일 경우 오차가 생깁니다)

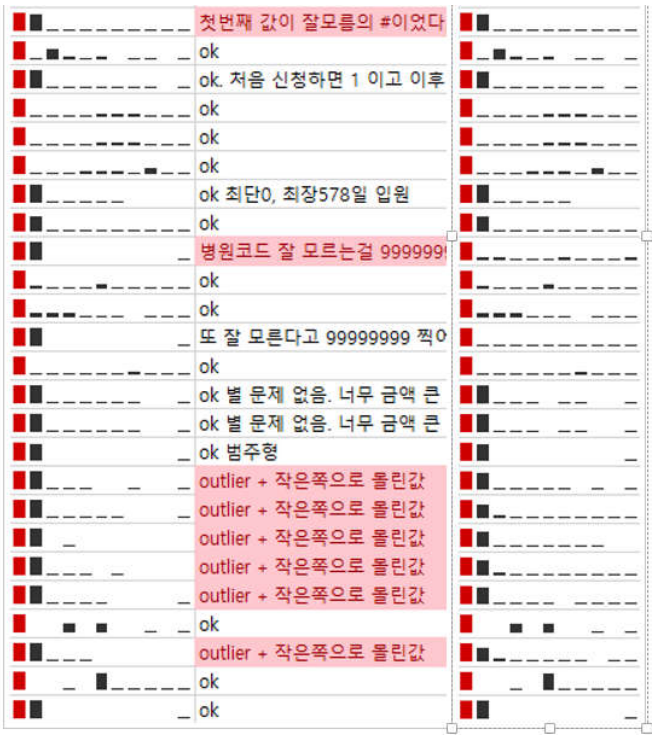
	original		rand 1		rand 2		rand 3	
행 레이블	합계 : 평균	합계 : 표준편차	합계 : 평균	합계 : 표준편차	합계 : 평균	합계 : 표준편차	합계 : 평균	합계 : 표준편차
ACCI_DVSN	2.3527	1	2.3527	1	2.3527	1	2.3527	1
ACCI_HOSP_ADDR	7.4442	1	7.4921	1	7.4955	1	7.4955	1
ACCI_OCCP_GRP1	4.3342	1	4.3395	1	4.3407	1	4.3398	1
ACCI_OCCP_GRP2	15.8022	1	15.8009	1	15.8	1	15.8026	1
AGE	44.7349	2	44.7349	2	44.7349	2	44.7349	2
BEFO_JOB	9.2498	1	9.2567	1	9.2571	1	9.2546	1
BRCH_CODE	1621301.124	3	1624370.644	3	1624370.644	3	1627543.032	3
CAUS_CODE	600.9616	1	600.9616	1	600.9616	1	600.9616	1
CAUS_CODE_DTAL	7.2516	1	7.2516	1	7.2516	1	7.2516	1
CHANG_FP_YN	0.4002	0	0.4002	0	0.4002	0	0.4002	0
CHLD_CNT	0.7095	0	0.7201	0	0.7203	0	0.7204	0
CHME_LICE_NO	1404355.858	6	1404355.858	6	1404355.858	6	1404355.858	6
CLLT_FP_PRNO	87468.6344	4	127238.3909	4	123699.0446	4	126353.7851	4
CNTT_RECP	1.8692	0	1.8728	0	1.8736	0	1.8731	0
CNTT_RECP_SQNO	2.01218E+12	2.00601E+12	2.01218E+12	2.00601E+12	2.01218E+12	2.00601E+12	2.01218E+12	2.00601E+12
CNTT_STAT_CODE	5.2389	1	5.2391	1	5.2396	1	5.2397	1
CNTT_YM	20062953.97	34790	20062953.97	34790	20062953.97	34790	20062953.97	34790
COUNT_TRMT_ITEM	1.0509	0	1.497	0	1.4989	0	1.5003	0
CRNT_PROG_DVSN	21.5678	11	21.5678	11	21.5678	11	21.5678	11
CTPR	7.2645	1	7.2861	1	7.2855	1	7.2919	1

랜덤분산을 사용하면 처음보는 값에 더 강인해질것으로 생각했기 때문. 동일값을 많이쓰면 overfittng 위험이 있다고 생각했습니다. 확인하진 않음. 소스 : /dbinput/csv_input_null_to_random.py

이슈 3 : 99999 값등 OUTLIER 가 보이는데 제거가 필요한가?

실험결과 영향이 미미했고 이후로 포함하지 않았습니다.

변수들의 분포를 체크해보니 undefined 값에 맥시멈을 할당해서 정규화시 값이 한쪽에 쏠리는 경우등이 확인되었고, 쏠림을 풀어주면 변수가 더 잘 학습될것이라 생각해 적당한 값으로 대체해 수정했습니다.



	outdel0			outdel1		
행 레이블	dummy0	dummy100	dummy300	dummy0	dummy100	dummy300
nullfix01						
5	**0.49696843**	**0.517620729**	**0.539840452**	**0.461329054**	**0.489676227**	**0.571455441**
0.4	0.469571227	0.482354215	0.539840452	0.412651419	0.454659625	0.571455441
0.6	0.494364975	0.49739056	0.531939873	0.45564566	0.444444437	0.545776609
0.9	0.49696843	0.517620729	0.498957242	0.461329054	0.489676227	0.475311862
10	**0.493735399**	**0.517680616**	**0.576566622**	**0.457086081**	**0.476110653**	**0.55114236**
0.3	0.448741328	0.479955088	0.508467303	0.409059902	0.448504981	0.503564102
0.5	0.472118502	0.49288311	0.53945453	0.450530467	0.452707115	0.55114236
0.7	0.493735399	0.517680616	0.576566622	0.457086081	0.476110653	0.530456851
20	**0.502533139**	**0.523862149**	**0.538779426**	**0.465132749**	**0.459627336**	**0.549600916**
0.4	0.45564962	0.494505498	0.538779426	0.464370107	0.45794276	0.514965737
0.6	0.488551394	0.504869504	0.533476609	0.464058671	0.446449274	0.549600916
0.9	0.502533139	0.523862149	0.515469132	0.465132749	0.459627336	0.499334812

Outdel 0->1 사이에서 뚜렷한 개선이 확인되지 않았고, 지나친 변수복잡성을 야기하므로 실험에 포함하지 않았습니다. (버그 제거한 버전으로 재실험한 내용)

이슈 4 : 큰 테이블로 JOIN 하기

Claim 을 기준으로 다른 테이블을 붙일 때 가장 크게 만들 수 있었습니다.

```
SELECT ( 생략: claim, cust, fpinfo, cntt 중 ID 등 의미없는 컬럼빼고 전체 )
From
    claim inner Join
        (Select cust.* From cust Where cust.SIU_CUST_YN = %s) as cust_part
      On cust_part.CUST_ID = claim.CUST_ID
    Left Join
    insurance_nullfix.cntt
      On insurance_nullfix.claim.POLY_NO = insurance_nullfix.cntt.POLY_NO And
      insurance_nullfix.cntt.CUST_ID = insurance_nullfix.claim.CUST_ID
    Left join
    insurance_nullfix.fpinfo
      On insurance_nullfix.cntt.CLLT_FP_PRNO = fpinfo.CLLT_FP_PRNO
```

내용중 %s 는 1 Order By rand() , 2 등으로 파이썬에서 동적으로 교체하기 위해서입니다.
rand() 로 CUST 를 미리 섞어서 가지고 오는건 SQL join 으로 데이터를 뽑을 때 중복 ID 문제를 해결하기 위한 임시 대책입니다.

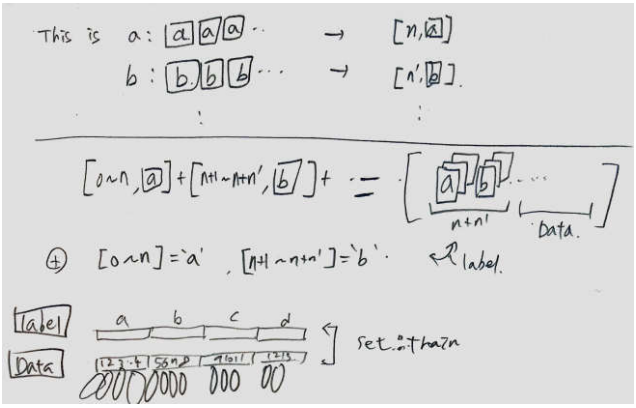
받아온 데이터를 머신러닝용 데이터로 정제

기본 작업 :

파이썬에서 라이브러리를 사용해 Mysql 로 쿼리를 보내고 데이터를 받는다.

Numpy 배열로 변환하고 NN 에서 학습편중을 막기위해 정규화 작업 (-0.5 ~ +0.5)

Test:20%, Train:80% 로 데이터를 나눠 .Pickle 로 만들고 섞는다.




```python
#picklelize
    f = open(pickle_name,'wb')
    save={
        'test_label' : test_label,
        'test_data' : test_data,
        'train_label' : train_label,
        'train_data' : train_data,
        'col_names' : get_cnames,
        'train_distri' : train_distri,
        'submit_data' : get_s,
        'submit_custid' : get_s_custid
        }
    pickle.dump(save,f,pickle.HIGHEST_PROTOCOL)
    f.close()
    print '₩npicklize finished.  filename :',pic
    pickletest(pickle_name)
```

소스 : /pickler/insurance_pickle_cufix.py

이슈 1 : 범주형 자료의 더미변수화

범주형 자료가 있을경우 NN 같은 linear classifier 에서는 범주가 아니라 연속데이터로 인식해서 문제가 됩니다.
수동으로 설정하지 않고, unique 항목수가 몇 개 이하인 컬럼을 자동으로 더미화 했습니다.
NN 에선 향상이 있었고, Tree 계열에서는 향상이 적어 NN 에만 적용했습니다.

Neural Network : dummy limit 가 0 -> 300 으로 갈수록 F1 이 증가합니다.

행 레이블	outdel0 dummy0	dummy100	dummy300	outdel1 dummy0	dummy100	dummy300
nullfix01						
5	**0.49696843**	**0.517620729**	**0.539840452**	**0.461329054**	**0.489676227**	**0.571455441**
0.4	0.469571227	0.482354215	0.539840452	0.412651419	0.454659625	0.571455441
0.6	0.494364975	0.49739056	0.531939873	0.45564566	0.444444437	0.545776609
0.9	0.49696843	0.517620729	0.498957242	0.461329054	0.489676227	0.475311862
10	**0.493735399**	**0.517680616**	**0.576566622**	**0.457086081**	**0.476110653**	**0.55114236**
0.3	0.448741328	0.479955088	0.508467303	0.409059902	0.448504981	0.503564102
0.5	0.472118502	0.49288311	0.53945453	0.450530467	0.452707115	0.55114236
0.7	0.493735399	0.517680616	0.576566622	0.457086081	0.476110653	0.530456851
20	**0.502533139**	**0.523862149**	**0.538779426**	**0.465132749**	**0.459627336**	**0.549600916**
0.4	0.45564962	0.494505498	0.538779426	0.464370107	0.45794276	0.514965737
0.6	0.488551394	0.504869504	0.533476609	0.464058671	0.446449274	0.549600916
0.9	0.502533139	0.523862149	0.515469132	0.465132749	0.459627336	0.499334812
50	**0.461426495**	**0.519970143**	**0.550164395**	**0.466920342**	**0.480611043**	**0.535912888**
0.4	0.456555145	0.496336214	0.52883542	0.466920342	0.480611043	0.532006913
0.6	0.461426495	0.514650184	0.550164395	0.464837663	0.46645828	0.535912888

소스파일 : pickler/insurance_pickle_cufix.py 모듈에 옵션으로 포함되어있습니다

Tree 류의 경우는 별로 의미가 없었습니다.

행 레이블	dummy0	dummy100	dummy300
adaboost	**0.582961862**	**0.610801187**	**0.591620515**
10	**0.548235578**	**0.556735079**	**0.563218391**
0.2	0	0	0.102771363
0.5	0.520944402	0.508551881	0.527182045
0.9	0.548235578	0.556735079	0.563218391
50	**0.573370555**	**0.599107251**	**0.591620515**
0.2	0.526825333	0.518568718	0.509938314
0.5	0.568113017	0.586045365	0.578316294
0.9	0.573370555	0.599107251	0.591620515
100	**0.582961862**	**0.610801187**	**0.586940391**
0.2	0.564863818	0.558357161	0.556036178
0.5	0.582961862	0.603851278	0.586940391
0.9	0.582559402	0.610801187	0.585513566

randomforest	**0.504678278**	**0.494571429**	**0.480117388**
10	**0.458496875**	**0.452120682**	**0.439602786**
-	0.458496875	0.452120682	0.439602786
50	**0.495942029**	**0.489661114**	**0.471480671**
-	0.495942029	0.489661114	0.471480671
100	**0.504678278**	**0.494571429**	**0.480117388**
-	0.504678278	0.494571429	0.480117388
tree	**0.44808506**	**0.46512605**	**0.471825851**
-	**0.44808506**	**0.46512605**	**0.471825851**
-	0.44808506	0.46512605	0.471825851

NN 학습

저사양 노트북에 리눅스 서버를 셋업하고 python 과 Tensorflow 를 사용했습니다.

결과적으로는 NN 보다 ADABOOST 가 F1 score 에서 0.10p 정도 뛰어났습니다. 더 깊은 NN 를 구성한다거나, 변수를 더 최적화 할 여지는 남아있습니다.

이슈 1 : NN 최적화 방법을 무엇까지 적용할것인가

레이어 수 외에도 학습속도 최적화, Dropout 기법 등 수많은 기법이 있습니다. 개발 기간을 고려해 2L 의 간단한 모델을 사용했습니다. Dropout 과 Learning rate decay 나 여러 학습기 등을 적용했지만 최적화에 시간이 많이 소요되어 제외했습니다.

이슈 2 : 하이퍼 파라미터 최적화에 드는 엄청난 시간

샘플 테스트 후 범위를 줄이고 파이썬으로 만든 쉘로 자동테스트를 하는식으로 진행했습니다.

결과는 엑셀파일로 출력되어 피벗테이블로 쉽게 분석할 수 있습니다.

적당한 범위의 구간을 잡았음에도 모두 테스트 한다면 엄청난 시간이 소요됩니다.

nullfix (0~4, original)
outlier-del (0,1)
SQLdataselect (feature engineering, cuclcntt,cuclaim, cucntt)
dummylize(0,1)
pickle (0,1,2,3,4)
NN h.layer (1,2,3)
nodes (10,100,1000)
L.rate (.1, .3, .5, .7, .9)
w init stddev(0.3,0.5,0.7)
test (0,1,2,3,4)
r.forest estimators (10,100,500)
adaboost estimators (10,100,500)
adaboost learningrate (0.5,0.7,0.9,1.1)
Feature engineering(컬럼하나씩 빼고 NN, forest 로 찾기,wb 분석, 상관분석,직관변수추가,변수제거)

6*2*4*2*5*(3*4*3*3*5+3)=260,640 회 test 필요. 실제로는 30,000 회 정도 테스트.
1 회 test 당 NN 은 10 초, forest 는 30 초 -> 컴퓨터 작동시간만 15days (코딩/학습시간 제외)

NN 학습 코어모델 소스 : /linux/Insurance_model.py
NN 학습 쉘소스 : /linux/shell_autotest.py

DEICISION-TREE, RANDOM-FOREST, ADABOOST 학습

세 트리 계열 학습기중 ADABOOST 가 압도적으로 좋았습니다. F1 의 최고값은 0.61 입니다.

학습기, 학습기별 파라미터 변경하며 테스트는 NN 과 마찬가지로 쉘 스크립트를 사용했습니다.

	A	B	C	D	E	F	G	H	I	J	K	L
1	picklename	outdel	dummy	tryno	trainer	estimators	L rate	test repeat	predict	correct predict	real	F1 score
2	nullfix02	outdel1	dummy100	0	adaboost	100	0.9	0	3513	2573	4912	0.610801187
3	nullfix02	outdel1	dummy100	0	adaboost	100	0.9	1	3513	2573	4912	0.610801187
4	nullfix02	outdel1	dummy100	0	adaboost	100	0.9	2	3513	2573	4912	0.610801187
5	nullfix02	outdel1	dummy100	0	adaboost	100	0.5	0	3345	2493	4912	0.603851275
6	nullfix02	outdel1	dummy100	0	adaboost	100	0.5	1	3345	2493	4912	0.603851275
7	nullfix02	outdel1	dummy100	0	adaboost	100	0.5	2	3345	2493	4912	0.603851275
8	nullfix02	outdel1	dummy100	0	adaboost	50	0.9	0	3377	2483	4912	0.599107251
9	nullfix02	outdel1	dummy100	0	adaboost	50	0.9	1	3377	2483	4912	0.599107251
10	nullfix02	outdel1	dummy100	0	adaboost	50	0.9	2	3377	2483	4912	0.599107251
11	nullfix03	outdel0	dummy300	2	adaboost	50	0.9	0	3394	2457	4912	0.591620515
12	nullfix03	outdel0	dummy300	2	adaboost	50	0.9	1	3394	2457	4912	0.591620515

Tree 모델 소스 : /auto/tree_trainers_forshell.py
Tree 모델 학습 쉘 소스 : /auto/thell_tree.py

결과 제출

이슈 1 : 예측이 의미를 가지는 기준은?

랜덤으로 예측했을 때 나오는 F1score 보다 높아야 의미가 있습니다.

어떤 테이블이 기준인지에 따라 경계값이 다른데 Claim-0.3094, Cust-0.1497, Cntt-0.1639 입니다.

제 모델은 Claim 을 기준으로 했고, 0.3094 를 넘었으니 의미가 있는 모델입니다.

cust 기준 사기:1806 일반:18801 합: 20607 테스트: 1793	8.8%가 보험사기자.
claim 기준 사기:24564 일반: 85209 합 :109773, 테스트: 15006	22.4% 가 보험사기자.
cntt 기준 사기:10241 일반:93773 합: 104014, 테스트:8996	9.8%가 보험사기자.

CLAIM 에 100 건의 데이터가 있고 랜덤으로 사기자를 판단할 시 테이블 보험사기자 비율에 따라서

0 이라고 찍은 50 중 22.4%인 11.2 명은 F 예측에 T	FT:11.2
0 이라고 찍은 50 중 77.6%%인 38.8 명은 F 예측에 F	FF:38.8
1 이라고 찍은 50 중 22.4%인 11.2 명은 T 예측에 T	TT:11.2
1 이라고 찍은 50 중 77.6%%인 38.8 명은 T 예측에 F	TF:38.8

precision: 11.2/50 = 0.224
recall: 11.2/22.4 = 0.5
F1 score : 2*(0.224*0.5)/(0.224+0.5)= **0.3094**
claim 기준으로 판단시 **0.3094** 까지는 랜덤으로 맞춰도 얻을 수 있는 정확도라는 의미.

Cust 기준	Cntt 기준
precision : 4.4/50=0.088 recall: 4.4/8.8 = 0.5 F1 score : 2*(0.088*0.5)/(0.088+0.5)= **0.1497**	precision : 4.9/50=0.098 recall: 4.9/9.8 = 0.5 F1 score : 2*(0.098*0.5)/(0.098+0.5)= **0.1639**

이슈 2 : CLAIM 기준 결과 9 천을 어떻게 제출 CUST 1 천 가량으로 줄일지

. Claim 테이블엔 cust_id 가 중복으로 있기 때문에 Claim 테이블 기준으로 예측하면 9 천줄 가량 결과가 나옵니다. 최상위 2 개 모델의 예측 결과에 동일하게 sin 이 표시되는 항목이거나,
한쪽 모델에서만 나왔더라도 그 ID 의 거의 모든행이 sin 표시일경우 채택했습니다.

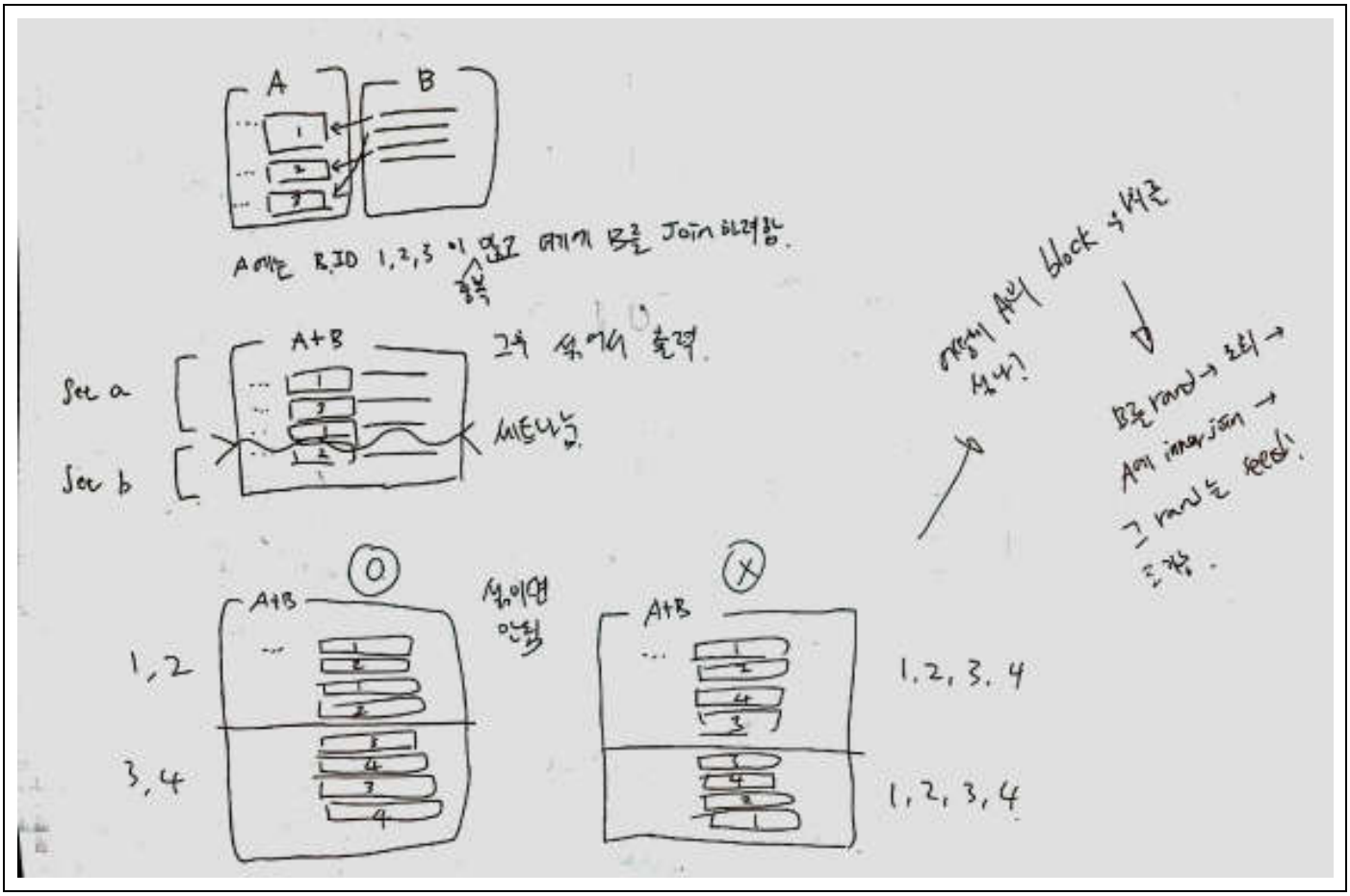
	A	B	C
1		모델1	모델2
146	307	0	0
147	307	0	0
148	307	0	0
149	318	0	0
150	318	0	0
151	321	0	1
152	321	0	1
153	321	0	0
154	321	0	1
155	321	0	1
156	321	0	1
157	321	0	1
158	321	0	0
159	321	0	0
160	321	0	0
161	321	0	0
162	321	0	1
163	321	0	1
164	321	0	1
165	321	0	0

E	F	G	H
	sin_모델1	sin_모델2	최종판단
297	3	2	1
307	0	0	0
318	0	0	0
321	0	9	1
344	0	0	0
353	0	0	0
369	0	0	0
380	0	0	0
397	0	0	0
401	0	0	0
410	0	0	0
417	0	0	0
433	0	0	0
436	1	1	1
451	0	0	0
453	0	0	0
466	1	1	1
487	13	13	1
488	0	0	0

3. 변수 최적화 (적용하지 못함)

버그 내용

데이터 순서를 섞으며 동일한 고객정보가 TEST, TRAIN 셋 양쪽에 포함됨.



Claim 에는 중복된 Cust_id 들이 있고, join 해 붙이면 Claim 의 고유정보는 다르지만 Cust_id 를 보고 가져온 고객 정보는 Cust_id 의 중복 수만큼 또 중복되게 됩니다.

그럼 claim 만 고려할게 아니라 claim 에 연결된 cust_id 들의 중복도 관리해야 했습니다.

분류기는 Train 에서 이미 데이터를 보았기 때문에 Test 에서도 높은 결과가 나왔습니다 (0.998 이상)

모든 변수분석결과를 버려야 했습니다. 아래는 버린 결과중 일부입니다

```
by  cuclaim

not_sin pricision :  0.985544
not_sin   recall  :  0.826775
not_sin F1 score  :  0.899205231763

sin pricision :  0.0720427
sin   recall  :  0.112426
sin F1 score  :  0.087814108769
```

```
rand02 outdel dummy0 not-normalized
predict = [ 17094.   4860.]
real = [17035  4919]
correct_predict = [17035  4860]
precision = 1.0
recall = 0.988005692214
f1_score = 0.993966663258
[Finished in 93.5s]
```

해당 버그는 데이터 전처리 과정해서 발생했기 때문에 이후 변수분석 등 모든 과정이 의미가 없어졌습니다.

변수최적화 개요

실험 이전에 데이터를 서로다른 전처리 방법들로 생성해 각각 테스트 해야합니다.

1. 범주형자료의 Dummy 화 여부.
2. Null 제거를 랜덤값으로 했으니 5 가지정도 null 제거 DB 를 만들어 일관성 테스트
   (테스트결과 모두 일관성 없음으로 나타나 심각한 버그를 발견할 수 있었습니다.)
   - 이번 공모전에서 모델의 신뢰도를 테스트하는 방법으로 주로 반복평가 일관성을 보았습니다.

NN 기반 변수최적화 : 학습된 W,B / 컬럼하나씩 제외하며 SCORE DROP 관찰하기

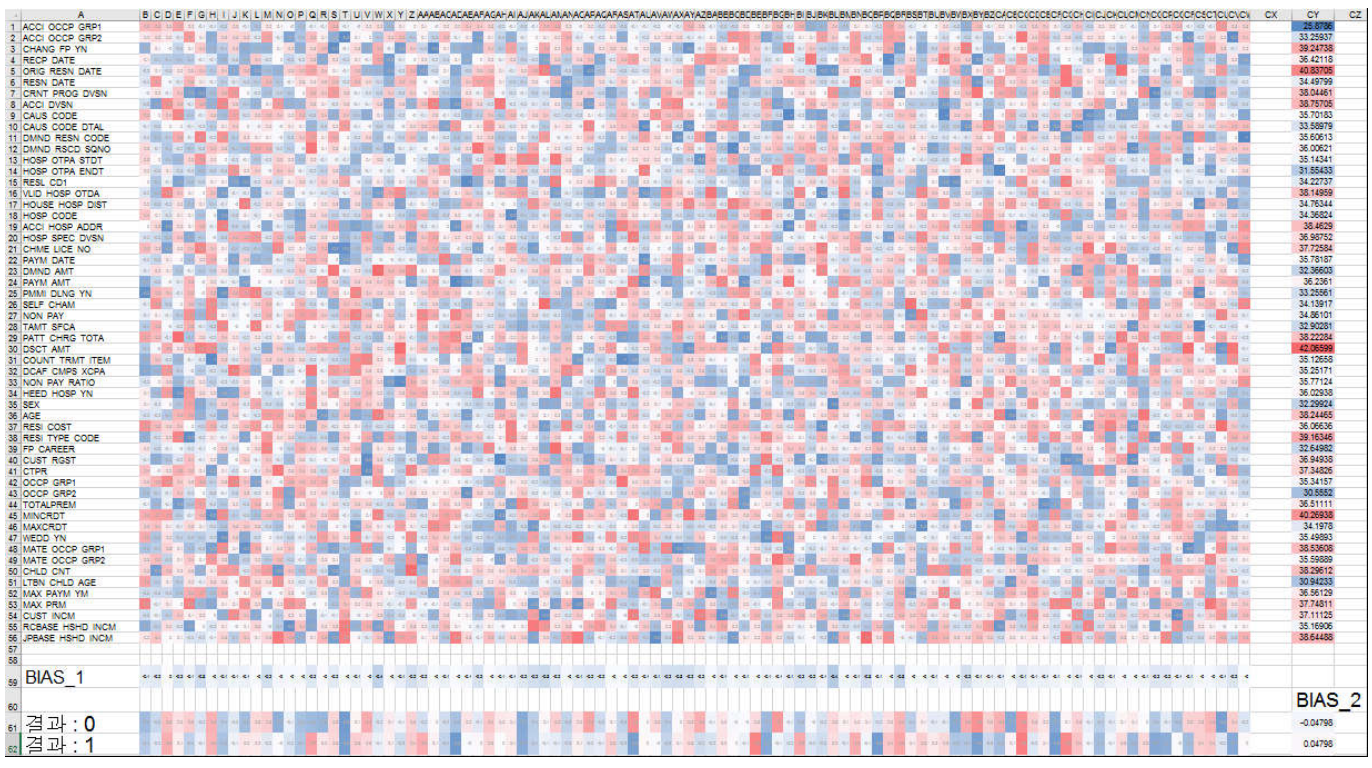
의미없는 변수 삭제를 위해 사용한 방법입니다. 주어진 변수를 모두 가지고 시작했기 때문에 먼저 변수 수를 줄인 후 추가변수를 넣으려고 했습니다. Tree 기반의 변수최적화와 비교한다면 의미가 있을것입니다.

(버그로 인해 일관성이 매우 낮게 나와 기각되었지만, 문제를 해결한 지금 시점에서는 시도해볼 가치가 있습니다)

1. 학습된 W, B 분석

NN 의 학습 node 를 단순화해서 반복실험하고 매 학습의 w, b 셋을 모아 자주 사용되는 변수를 찾습니다.
예를들어 결과 1(sin) 을 구분하기 위해 아래쪽 히든레이어 두줄에서 빨간색 칸을 많이 사용하고 있고,
각 히든레이어 한칸씩은 그 위의 큰 박스 형태로 변수를 사용하니 역추적할 수 있을 것입니다.

다만 레이어가 깊어지고 노드가 많아질 시 분석이 어려우므로, F1 하락이 적은 수준에서 노드와 레이어를 최소화 한 상태로 분석해야 사람이 찾을 수 있게됩니다.



소스 : Insurance_model.py 안에 옵션으로 학습값 저장기능이 있습니다.

2. 한 컬럼씩 제외하며 가장 F1 을 많이 떨어트리는 컬럼을 높은 가치로 평가

모든 컬럼이 100 개라면, 1 개를 제외한 99 개로 동일조건 학습을 해보고 , 다음시행에는 제외한걸 되돌리고 다른컬럼을 제외하고 학습하는 식으로 F1 을 측정합니다.

가장 F1-score 가 많이 떨어진 컬럼이 가장 중요하게 사용되는 컬럼일 것이라고 추측했습니다.

각 컬럼을 제외한 데이터셋 생성

뺀 컬럼 학습결과의 F1 score 를 정렬하면 아래처럼 중요한 컬럼을 얻을 수 있습니다.
아래 결과는 버그 포함이기 때문에 sin 인 cust 의 특징을 잘못 수집하고 있습니다.

claim.CHANG_FP_YN	0.408295175		
claim.DMND_RESN_CODE	0.392480996		
claim.RESL_CD1	0.384991413		
claim.PMMI_DLNG_YN	0.358270422		실손처리여부
cust.MINCRDT	0.349735239		최소신용등급
cust.CUST_INCM	0.340339832		소득추정
claim.CAUS_CODE	0.295211595		원인코드
claim.HOSP_SPEC_DVSN	0.240494297		병원종별구분
claim.HOSP_OTPA_STDT	0.164941339	가장 의미있음	입원/통원시작일자

소스 : /variable relation check/insurance_pickle_variableEffectChk.py

개별컬럼을 제외하며 측정하는 위 방식은 각 컬럼이 독립적으로 예측에 영향을 미친다는 가설에 기반합니다. 실제로는 변수들의 "조합"으로 학습이 이루어지기 때문에 1 번컬럼과 2 번컬럼이 중요하다고 나왔음에도 1,2 만 남겨 학습했을 때 같은 결과가 나오지 않을것이라 예상합니다.

이 부분은 feature combination 최적화에 대해 좀 더 생각해보아야 합니다.

TREE 기반 변수최적화 : FEATURE IMPORTANCE, TREE 학습결과 분석.

역시 의미없는 변수 삭제를 위해 사용한 방법입니다. 트리를 섞어 학습하는 RandomForest 와 ADABOOST 는 비주얼표현이 힘들지만 변수 중요도를 계산할 수 있고, Decision tree 는 트리가 하나이기 때문에 학습결과를 PDF 등으로 쉽게 시각화할 수 있습니다.

1. 변수 임팩트 분석.

ExtraTreeClassifier 로 변수의 상대 중요도를 추출할 수 있습니다. 내용만 알고 시도해보지는 못했습니다.

1.11.2.5. Feature importance evaluation ¶

The relative rank (i.e. depth) of a feature used as a decision node in a tree can be used to assess the relative importance of that feature with respect to the predictability of the target variable. Features used at the top of the tree contribute to the final prediction decision of a larger fraction o the input samples. The **expected fraction of the samples** they contribute to can thus be used as an estimate of the **relative importance of the features**.

By **averaging** those expected activity rates over several randomized trees one can **reduce the variance** of such an estimate and use it for feature selection.

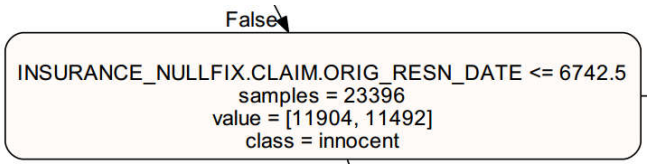
The following example shows a color-coded representation of the relative importances of each individual pixel for a face recognition task using a `ExtraTreesClassifier` model.

출처 : http://scikit-learn.org/stable/modules/ensemble.html#feature-importance-evaluation

2. DECISION-TREE 의 NODE 이용

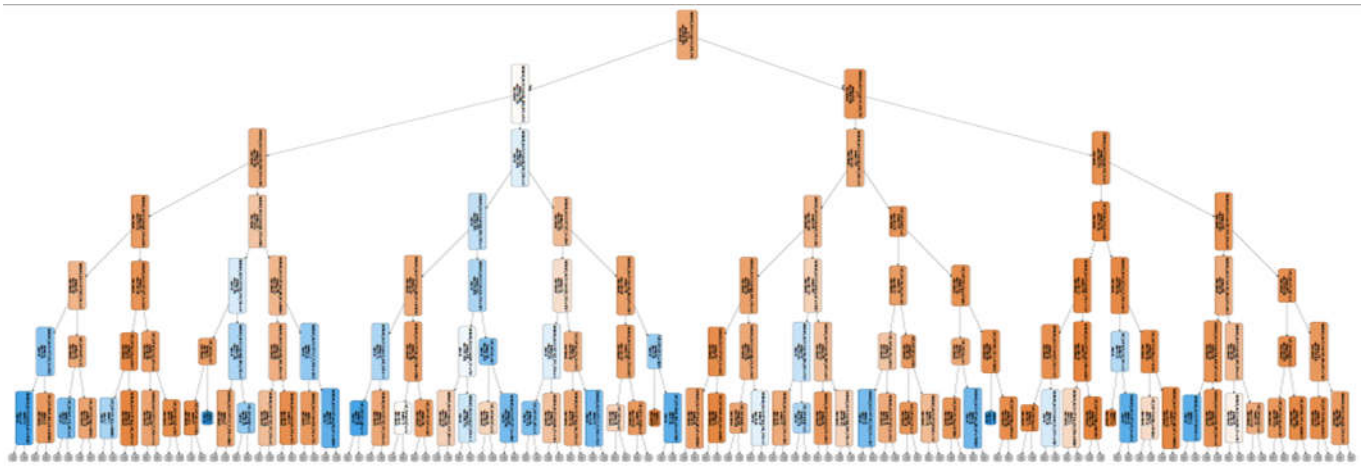
최종적으로 남은 변수 집합은 Adaboost 만큼은 아니지만 Decision-tree 에서도 괜찮은 예측결과가 나올것이라 생각했습니다. Decision-tree 에서 노드를 출력해보면 어떤 변수가 의미있게 쓰이고 있는지 짐작해볼 수 있습니다.

예를들어 아래 F1-score 0.41 수준의 Tree 분석 노드에서는 Sin 으로 가는 중요한 변수로 ORIG_RESN_DATE 가 나타나는데, 이 변수는 사고가 발생한 날짜를 의미합니다. 보험사기는 해가 갈수록 건수가 증가하며 과거에는 탐지기법이 발달하지 않았으므로, 특정 시점 이후에 사건이 몰려있는 시간대를 1 차적 기준으로 삼은것입니다.



<6742.5 는 2013 년 4 월 이후를 의미합니다.

(1995 년 1 월 1 일 + 6742.5days)



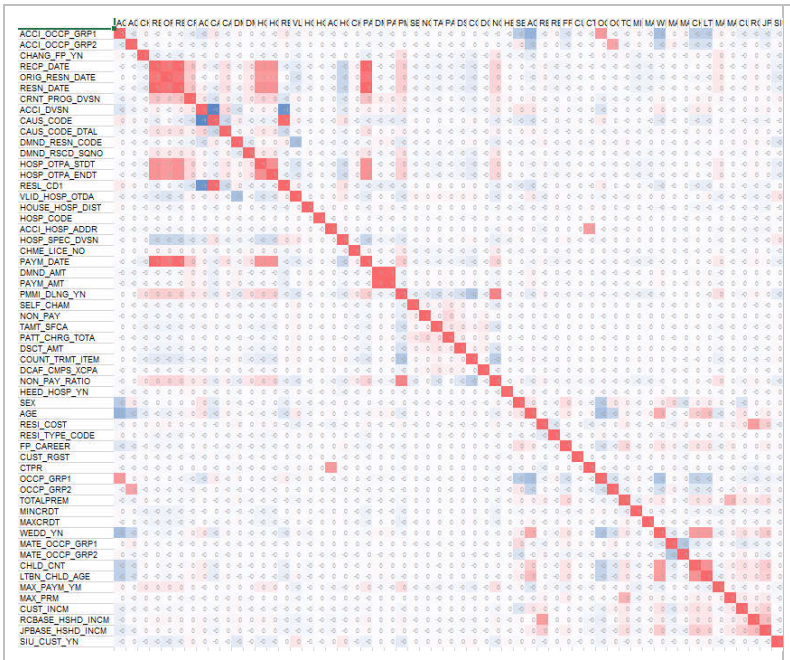
예제파일 : decisionTreetest.pdf

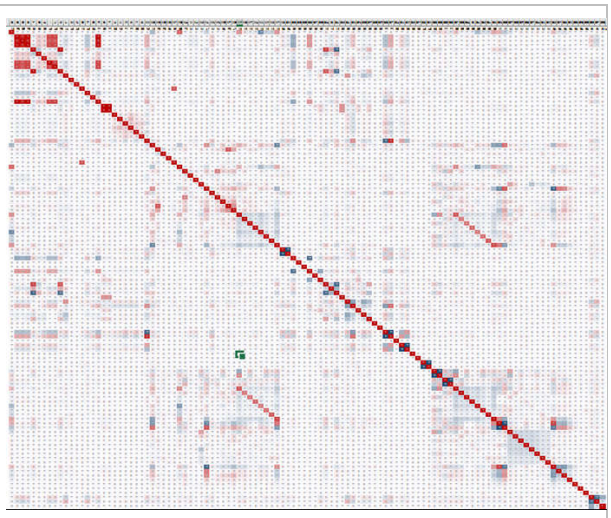
전통적 통계분석 기반 변수최적화

Sin 여부에 직접 관여하는 선명한 변수를 찾을 수는 없었습니다. 변수 상호간의 움직임을 파악해 신규 변수를 발굴할 아이디어를 얻으려 했습니다.

1. CORRELATION

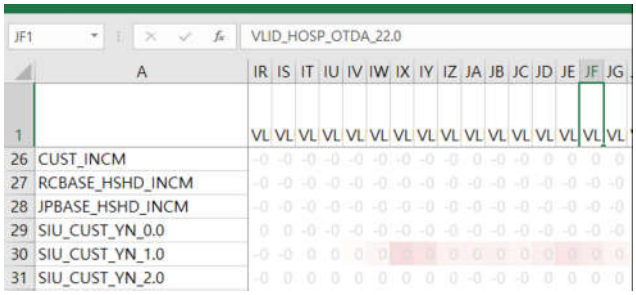
범주형 자료가 포함되어 Spearman correlation 을 사용했습니다. (실제로는 pearson 과 spearman 사이의 결과 차이가 거의없었습니다) 가장 아랫줄이 Sin_cust_YN 인데 직접 영향을 주는 변수가 없습니다.





범주형 변수 더미화 후 상관분석↑

←변수 직접 사용



하지만

좌측에서처럼 보험사기자는 HOSP_ODTA 의 일부 구간에 몰려있다는 식의 가설을 발굴할 수 있습니다.

범주형 데이터를 섹션별로 쪼개 새로운 변수로 만들어 테스트에 활용할 수 있습니다.

소스 : variable relation check/column_correlate_coeffitient.py

변수 추가

데이터에서 새롭게 알아낸 관계, 내 학습모델에서 사용못하는 정보의 추가, 사회학적 분석에 의한 변수를 추가하려 했습니다. 개발 시간 부족과 버그가 겹쳐 시도해보지 못했습니다.

1. 고객별 CLAIM 횟수

제 학습모델은 Claim 라인 한줄 한줄 각각에서 보험사기자인지 판단하므로 개인의 청구횟수 누적정보를 예측에 활용하지 못합니다. 따라서 이 정보를 사용하기 위해 청구횟수를 새로운 변수로 추가했습니다.

끝

상기 모든 코드는 numpy, sklearn, openpyxl, Tensorflow 라이브러리를 활용했으며, 라이브러리 외 코드와 쉘스크립트는 모두 직접 작성했습니다.

https://github.com/lavelee/2016bigcontest_insurance