Data Mining Term Project

최종보고서

목차

1. 최종 발표 후 추가 사항

최종 발표 이후 교수님 피드백을 반영하여 참고한 논문과 평가지표 비교를 실시하였다. 같은 교통사고 심각도 예측일지라도 다른 데이터 세트를 사용하였고 상해 정도 클래스를 이번 프로젝트 처럼 '상해없음" 클래스를 사용하지 않은 논문들이 많았다. 그래서 완벽한 비교가 되지는 못하지만 상해 정도 클래스를 줄인 논문은 많으나 3개로 줄이고 Undersampling 기법을 사용한 논문과 비교를 해보았다. "차대차 교통사고에 대한 상해 심각도 예측*(고창완, 김현민, 정영선, 김재희)" p.23*에서는 "치명적 상해", "경미한 상해", "심각한 상해"로 분류를 하였으나 "치명적 상해"를 사망으로 ""심각한 상해"를 중상과 경상으로, "경미한 상해"를 상해없음, 부상신고로 구성하여 이번 프로젝트와 차이가 존재한다. 완벽한 조건이 똑같지는 않지만 비교를 해보려고 한다.

이번 프로젝트와 위 논문에서 겹치는 모델만 비교하였고 평가지표를 이번 프로젝트에서는 F1-score와 Accuracy를 사용했지만 위 논문에서는 sensitivity와 specificity를 사용했으므로 이 지표들을 비교해 본다.

Model / Injury severity	NB		DT		RF	
	sen.	spec.	sen.	spec.	sen.	spec.
Minor injury	0.60 (0.035)	0.836 (0.015)	0.412 (0.071)	0.851 (0.035)	0.586 (0.036)	0.846 (0.018)
Intermediate injury	0.570 (0.03)	0.747 (0.021)	0.665 (0.05)	0.765 (0.033)	0.551 (0.034)	0.785 (0.02)
Serious Injury	0.726 (0.028)	0.864 (0.015)	0.799 (0.036)	0.836 (0.023)	0.794 (0.025)	0.835 (0.017)
Average sensitivity	0.632	0.816	0.625	0.817	0.644	0.822

논문에서 사용된 평가지표

본 프로젝트	NB		DT		RF	
	sen.	**spec.**	**sen.**	**spec.**	**sen.**	**spec.**
상해없음	**0.747**	**0.804**	**0.713**	**0.953**	**0.771**	**0.913**
경상	**0.540**	**0.710**	**0.609**	**0.745**	**0.517**	**0.806**
중상	**0.510**	**0.885**	**0.712**	**0.819**	**0.754**	**0.802**
평균	**0.599**	**0.800**	**0.678**	**0.839**	**0.681**	**0.840**

sensitivity와 specificity를 단순 비교 해봤을 때 NB에서는 평균이 떨어졌지만 DT와 RF에서는 조금 높은 점수를 보이고 있다. 하지만 데이터를 합치는 과정이 다르기 때문에 완벽한 비교라고 할수는 없다.

2. 주제 소개 및 연구배경

OECD 국가 중 교통사고로 인한 사망자수는 우리나라가 4위를 차지할 정도로 교통 사고 발생률이 높은 편에 속한다. 해가 거듭할 수록 우리나라의 교통 사고 건수와 사망자수 미세하게 감소하는 추세를 보이지만 사고 수 대비 부상자의 수가 증가한 경우를 볼 수 있다. 최근 3년간 인구밀도가 높은 서울특별시내의 교통사고 발생건수와 특징을 이용해 피해자 중심으로 사고 심각도를 예측한다. 이를 통해 어떤 요인이 사고의 심각도에 영향을 미치는 지와심각도 예측을 함으로서 교통사고에 대비할 수 있도록 유의미한 정보를 발견하고자 한다.

3. 데이터 전처리 및 EDA

3-1. 데이터 전처리

```
data_test = pd.read_csv('Final_Dataset.csv')
data_test.head()
```

✓ 0.3s Python

	사고일시	요일	시군구	사고내용	사망자수	중상자수	경상자수	부상신고자수	사고유형	법규위반	...	기상상태	도로형태	가해운전자 차종	가해운전자 성별	가해운전자 연령	가해운전자 상해정도	피해운전자 차종	피해운전자 성별	피해운전자 연령	피해운전자 상해정도
0	2020년 1월 1일 05시	수요일	서울특별시 강남구 도곡동	사망사고	1	0	0	0	차대사람 - 차도통행중	안전운전불이행	...	맑음	단일로 - 기타	승용	남	63세	상해없음	보행자	여	29세	사망
1	2020년 1월 1일 07시	수요일	서울특별시 강남구 논현동	경상사고	0	0	1	0	차대차 - 기타	안전운전불이행	...	맑음	단일로 - 기타	승용	남	29세	상해없음	승용	남	41세	경상
2	2020년 1월 1일 09시	수요일	서울특별시 강남구 역삼동	경상사고	0	0	1	0	차대차 - 기타	안전운전불이행	...	맑음	기타 - 기타	승용	남	32세	상해없음	승합	남	48세	경상
3	2020년 1월 1일 18시	수요일	서울특별시 강남구 삼성동	경상사고	0	0	1	0	차대사람 - 횡단중	보행자보호의무위반	...	맑음	단일로 - 기타	승용	남	44세	상해없음	보행자	남	42세	경상
4	2020년 1월 1일 18시	수요일	서울특별시 강남구 역삼동	경상사고	0	0	5	0	차대차 - 측면충돌	교차로운행방법위반	...	맑음	교차로 - 교차로안	승용	여	47세	상해없음	승용	남	34세	경상

5 rows × 21 columns

각자 2020~2022년 데이터를 7~8개의 서울특별시 “구”별 데이터를 모으고, 모두 취합한 후 “Final_Dataset.csv”로 저장을 해준 뒤 파이썬으로 불러와 준다.

Column	전처리 전	전처리 후
사고일시	2023년 01월 01일	‘계절’, ‘시간대’ 컬럼 생성 / 연도삭제
요일	월요일,화요일… (Text type)	-
시군구	서울특별시 00 구 00 동	‘구’, ‘동’ 컬럼 생성 / ‘시군구’삭제
사고내용		컬럼 삭제
사망,중상,경상,부상신고자수		컬럼 삭제
사고유형		사고원인 제거 및 ‘차량단독’ 제거
법규위반		‘큰위반’,’작은위반’,’안전운전불이행’으로 통합
노면상태	‘건조’, ‘젖음/습기’, ‘서리/결빙’	‘습기’, ‘결빙’ 통일
기상상태	‘눈’, ‘비’, ‘맑음’, ‘흐림’, ‘기타’	‘기타’ 삭제
도로형태	‘건조’, ‘젖음/습기’, ‘서리/결빙’	‘습기’, ‘결빙’ 통일

가해운전자 차종	화물, 자전거, 승용, 승합, 특수/건설기계, 농기계..	'특수', '이륜'으로 통합 및 기타불명, 농기계, 사륜오토바이 제거
가해운전자 성별	'남', '여', '성별 불명'	'성별 불명' 제거
가해운전자 연령	수치형	-
가해운전자 상해정도		컬럼 삭제
피해운전자 차종	화물, 자전거, 승용, 승합, 특수/건설기계, 농기계..	'특수', '이륜'으로 통합 및 기타불명, 농기계, 사륜오토바이 제거
피해운전자 성별	'남', '여', '성별 불명'	'성별 불명' 제거
피해운전자 연령	수치형	-
피해운전자 상해정도	'경상', '중상', '상해 없음' …	'기타 불명' 제거

이 표는 전체적인 전처리 과정을 보여준다. 자세한 전처리 과정은 바로 아래에 서술한다.

1. 사고일시

연도, 월, 일, 시간으로 구성된 사고일시를 연도를 삭제하고 계절과 시간대(새벽, 오전, 오후, 밤) 컬럼으로 변경하였다.

```
data_test['사고일시'] = pd.to_datetime(data_test['사고일시'], format="%Y년 %m월 %d일 %H시")

data_test['월'] = data_test['사고일시'].dt.strftime("%m-%d")

# 계절 컬럼 생성
season_map = {1: '겨울', 2: '겨울', 3: '봄', 4: '봄', 5: '봄', 6: '여름', 7: '여름', 8: '여름', 9: '가을', 10: '가을', 11: '가을',
data_test['계절'] = data_test['사고일시'].dt.month.map(season_map)

# 시간대 컬럼 생성
time_bins = [0, 6, 12, 18, 24]
time_labels = ['새벽', '오전', '오후', '밤']
data_test['시간대'] = pd.cut(data_test['사고일시'].dt.hour, bins=time_bins, labels=time_labels, include_lowest=True)
data_test = data_test.drop(columns=['월'])
```

2. 시군구

중복되는 서울특별시 삭제하고 row를 구분할 수 있는 구와 동으로 나뉘어 컬럼을 추가하였다.

```
# '시군구' 컬럼 분리하여 '시'와 '구동' 컬럼 생성
data_test[['시', '구동']] = data_test['시군구'].str.split(n=1, expand=True)

# '구동' 컬럼을 다시 '구'와 '동'으로 분리
data_test[['구', '동']] = data_test['구동'].str.split(n=1, expand=True)

# 불필요한 컬럼 제거
data_test = data_test.drop(columns=['시군구', '구동', '시'])
```

3. 사고유형

사고난 원인이 되는 텍스트를 삭제(차대차-정면추돌이면 정면추돌 텍스트 삭제)하고 차량단독등의 데이터를 제거하였다.

```
# 사고유형 컬럼 변환
data_test['사고유형'] = data_test['사고유형'].str.replace(r'^차대사람.*', '차대사람', regex=True)
data_test['사고유형'] = data_test['사고유형'].str.replace(r'^차대차.*', '차대차', regex=True)

data_test = data_test[(data_test['사고유형'] == '차대사람') | (data_test['사고유형'] == '차대차')]
```

4. 법규위반

공식적인 12대 중과실 교통사고 종류를 기반으로 큰 위반(과속, 보행자보호의무위반, 불법유턴, 신호위반, 중앙선침범), 작은 위반(교차로운방법위반, 안전거리미확보, 직진우회전
진행방해, 차로위반), 안전운전불이행으로 통합하였다.

```
data_test['법규위반'] = data_test['법규위반'].replace({
    '과속': '큰 위반',
    '보행자보호의무위반': '큰 위반',
    '불법유턴': '큰 위반',
    '신호위반': '큰 위반',
    '중앙선침범': '큰 위반',
    '교차로운행방법위반': '작은 위반',
    '안전거리미확보': '작은 위반',
    '직진우회전진행방해': '작은 위반',
    '차로위반': '작은 위반',
    '기타': None  # 기타는 삭제
})

data_test = data_test.dropna()
```

5. 노면상태

건조, 젖음/습기를 습기, 서리/결빙을 결빙으로 전처리 하였다.

```
data_test = data_test[~data_test['노면상태'].isin(['기타', '적설', '침수', '해빙'])]
# '젖음/습기'를 '습기'로, '서리결빙'를 '결빙'으로  바꾸기
data_test['노면상태'] = data_test['노면상태'].replace('젖음/습기', '습기')
data_test['노면상태'] = data_test['노면상태'].replace('서리/결빙', '결빙')
```

6. 기상상태

눈, 비, 맑음, 흐림, 안개로 구성하고 기타는 삭제하였다.

```
data_test = data_test[~data_test['기상상태'].isin(['기타'])]
```

7. 도로형태

교차로, 단일로, 주차장, 미분류로 컬럼이 구성 되어 있고 교차로 안, 부근, 지하차도 등 텍스트로 세분화 되어 있는 컬럼을 교차로, 단일로로 통합하였다.

```
# 도로형태 컬럼 변환
data_test['도로형태'] = data_test['도로형태'].str.replace(r'^단일로.*', '단일로', regex=True)
data_test['도로형태'] = data_test['도로형태'].str.replace(r'^교차로.*', '교차로', regex=True)
data_test = data_test[data_test['도로형태'] != '기타-기타']

data_test = data_test[(data_test['도로형태'] == '단일로') | (data_test['도로형태'] == '교차로')]
```

8. 가해/피해운전자 차종

화물, 자전거, 승용, 승합, 특수/건설기계를 특수로, 원동기/이륜/PM을 이륜으로, 기타불명, 농기계, 사륜오토바이를 제거하였다.

```
data_test['가해운전자 차종'] = data_test['가해운전자 차종'].replace({
    '특수': '특수',
    '건설기계': '특수',
    '원동기': '이륜',
    '이륜': '이륜',
    '개인형이동수단(PM)': '이륜',
    '기타불명': None,  # 기타불명은 삭제
    '농기계': None,    # 농기계는 삭제
    '사륜오토바이(ATV)': None  # ATV는 삭제
})

data_test = data_test.dropna()
```

```
data_test['피해운전자 차종'] = data_test['피해운전자 차종'].replace({
    '특수': '특수',
    '건설기계': '특수',
    '원동기': '이륜',
    '이륜': '이륜',
    '개인형이동수단(PM)': '이륜',
    '기타불명': None,  # 기타불명은 삭제
    '농기계': None,    # 농기계는 삭제
    '사륜오토바이(ATV)': None  # ATV는 삭제
})

data_test = data_test.dropna()
```

9. 가해/피해운전자 성별

성별 불명인 데이터를 제거하였다. 추가로 NaN값 제거하였다.

```
data_test = data_test[~data_test['피해운전자 성별'].isin(['기타불명'])]
data_test = data_test[~data_test['가해운전자 성별'].isin(['기타불명'])]

data_test = data_test.dropna()
```

10. 가해/피해운전자 나이

중간 피드백 이후 수치형을 그대로 사용하였다.

11. 가해/피해 운전자 상해정도

기타불명인 데이터를 제거하였다. 추가로 사망,중상 “중상”, 경상,부상신고 “경상” 으로 합쳤다.

```
data_test = data_test[~data_test['피해운전자 상해정도'].isin(['기타불명'])]
```

```
data_test = data_test.drop(columns=['가해운전자 상해정도'], axis =1)
```

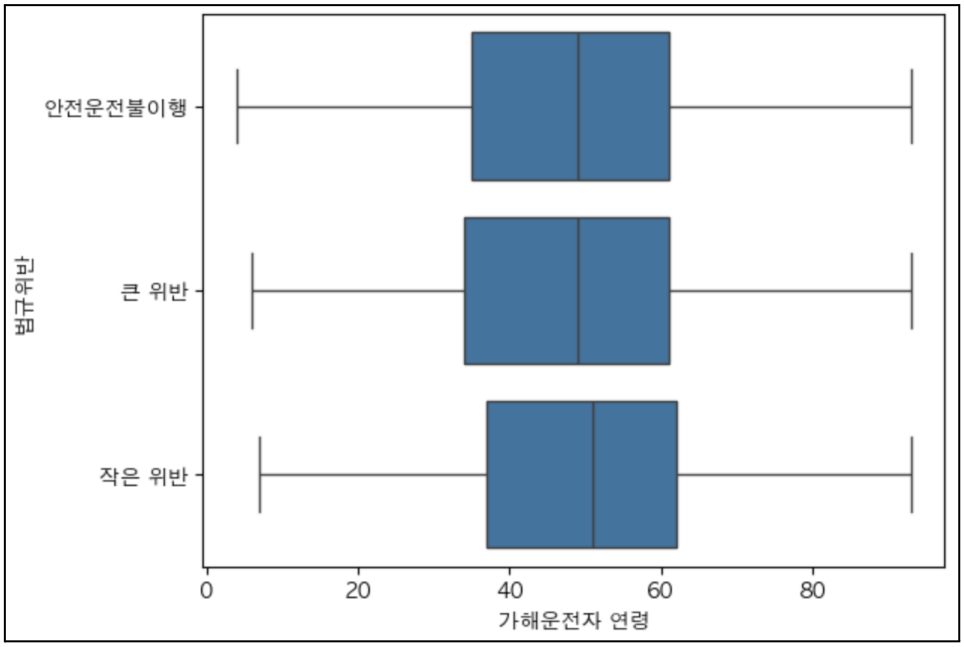
```
df['피해운전자 상해정도'] = df['피해운전자 상해정도'].replace({'부상신고': '경상', '사망': '중상'})
```

12. 추가내용

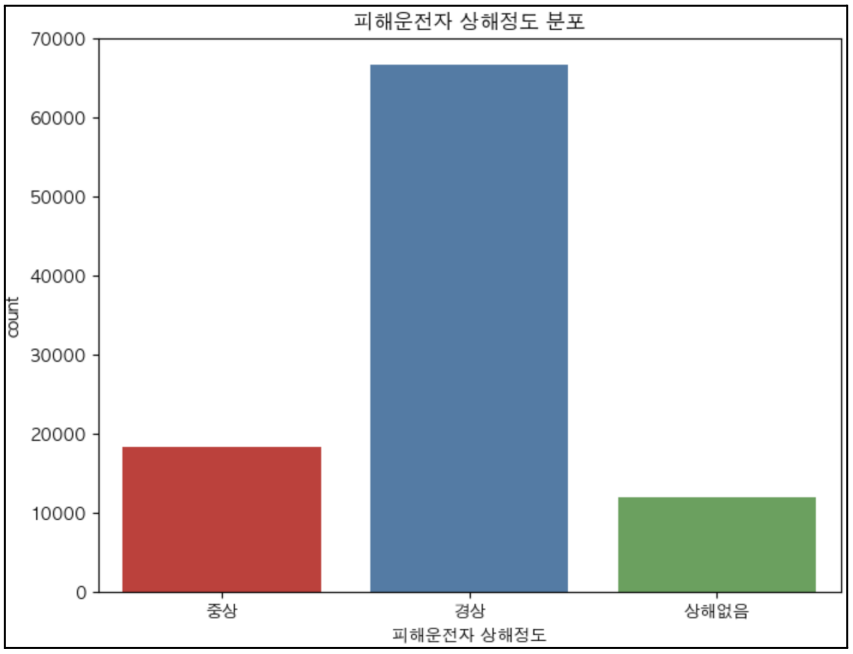
사고내용, 사망자수, 중상자수, 경상자수, 부상신고자수, 사고유형, 가해운전자 상해정도 위에 있는 컬럼들은 삭제하였다

```
data_test = data_test.drop(columns=['사고내용', '사망자수', '중상자수', '부상신고자수', '경상자수'])
```

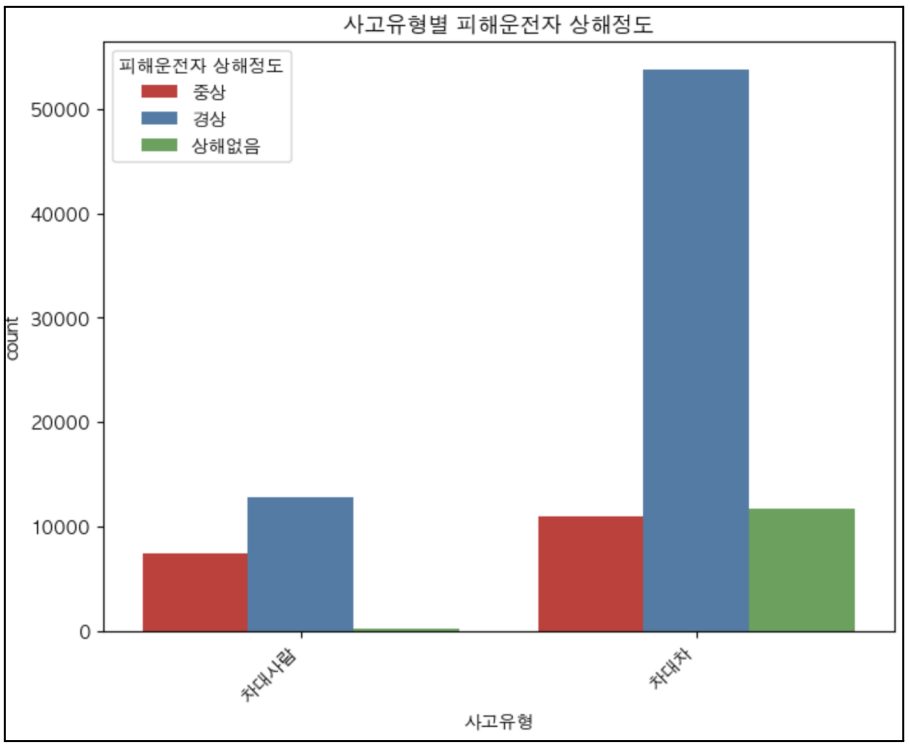
3-2. EDA



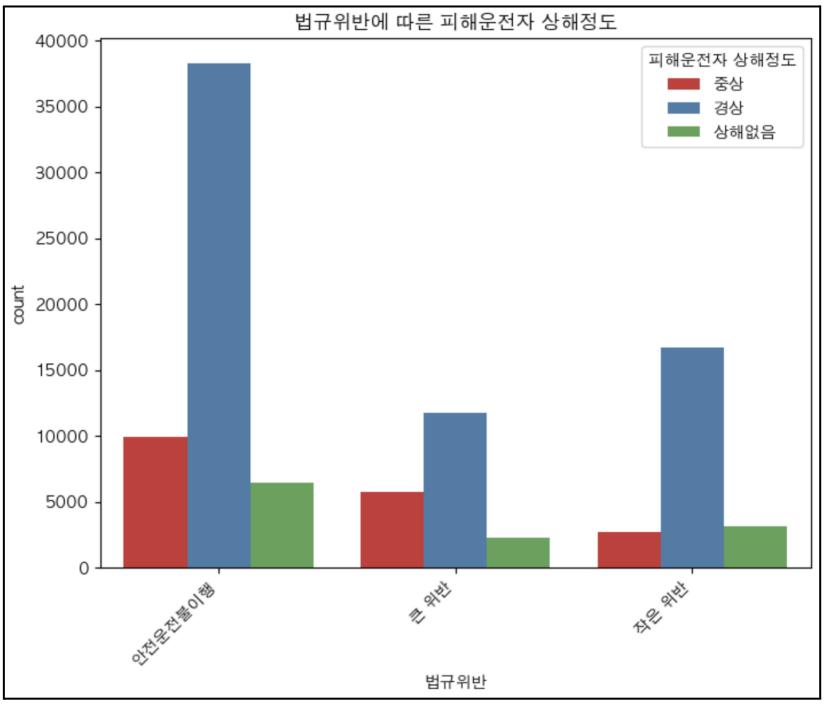
가해운전자 연령별 법규위반 정도를 박스 플랏으로 나타내었다. 큰 위반 부분에서 약간 다른 위반에 비해 나이대가 낮은 것을 알 수 있다.



중상, 경상, 상해없음 클래스별 수를 시각화 하였다. 경상이 제일 많고 그다음 중상, 상해없음 순이다.



사고 유형별 피해운전자 상해정도를 파악해보았다. 차대사람 부분에서는 “상해없음” 비율이 차대차에 비해 굉장히 낮은 것을 알 수 있었다.



법규위반별 피해운전자 상해정도를 파악해보았다. 큰 위반 부분에서 안전운전 불이행이나 작은 위반에 비해 중상의 비율이 확실히 더 높을 것을 알 수 있었다.

4. 데이터마이닝

데이터 전처리 후 의사결정나무, 랜덤포레스트, 나이브베이즈, XGBoost를 활용하여 데이터 마이닝을 수행한다. 먼저, 특성 변수와 타겟변수를 분리하고 범주형 변수는 더미 변수로 변환한다.
데이터를 Train Set과 Test Set로 나눠 준다. 이때 Train Set과 Test Set의 비율을 7:3으로 설정한다.
UnderSampling을 진행하지 않고 Train Set과 Test Set을 나누면 각각의 세트가 다음과 같은 개수를 가진다.

	가해운전자 연령	피해운전자 연령	요일_금요일	요일_목요일	요일_수요일	요일_월요일	요일_일요일	요일_토요일	요일_화요일	사고유형_차대사람	...	동_회현동1가	동_회현동2가	동_효자동	동_효제동	동_효창동	동_후암동	동_훈정동	동_휘경동	동_흑석동	동_흥인동
0	63.0	29.0	0	0	1	0	0	0	0	1	...	0	0	0	0	0	0	0	0	0	0
1	29.0	41.0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	44.0	42.0	0	0	1	0	0	0	0	1	...	0	0	0	0	0	0	0	0	0	0
3	47.0	34.0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	50.0	51.0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

	경상	중상	상해 없음
Train Set	46,664	12,869	8,259
Test Set	20,006	5,451	3,598

Target Class 중 '경상'이 66,670개, 중상'이 18,320개,'상해 없음'이 11,857개로 '상해 없음' Class가 적다. 이러한 Class의 개수 차이는 데이터의 불균형을 초래할 수 있기 때문에 UnderSampling을 진행하였다. 논문을 살펴본 결과 차대차 교통사고에 대한 상해 심각도 예측 연구(고창완, 김현민, 정영선, 김재희) 에서 '사망' Class가 제일 적은 것을 활용하여 UnderSampling 기법을 적용하여 예측을 연구하였고 다른 논문에서는 UpSampling을 활용한 논문도 있으나 '상해 없음' 이 11,857개 기준으로 11,857을 3개의 Class로 나누어도 약 35,000개로 데이터를 충분히 확보할 수 있다고 생각하였기에 UnderSampling을 사용하였다.
UnderSampling을 1대1대1비율로 진행하면 Train Set과 Test Set을 나누면 각각의 세트가 다음과 같은 개수를 가진다.

	경상	중상	상해 없음
Train Set	8,307	8,287	8,305
Test Set	3,550	3,570	3,552

뿐만 아니라, 추가로 Raw Data의 비율에 가까운 2:1:1 의 비율로 UnderSampling을 시행하였다. 이때의 Train Set과 Test Set의 각각의 개수는 다음과 같다. 2:1:1로 나눠본 이유는 결론에서 다시 언급이 되지만 1:1:1로 나누면 전체적인 Accuracy가 낮아지고 그냥 Raw data를 쓰게 되면 F1-score에서 낮아지게 된다. 그래서 이 중간 정도인 2:1:1로 나누어서 데이터 마이닝을 해보고자 한다.

	경상	중상	상해 없음
Train Set	16,576	8,371	8,252
Test Set	7,138	3,605	3,486

이러한 Train Set과 Test Set을 바탕으로 의사결정나무, 랜덤포레스트, 나이브베이즈, XGBoost를 진행하였다.

<의사결정 나무>

우선 UnderSampling을 하지 않은 Data Set으로 의사결정나무 모델을 사용해보았다. Data에 대한 최적화된 의사결정 나무 모델을 얻기 위해 Grid Search를 사용하여 최적의 파라미터 조합을 찾았다. Grid Search는 사용자가 지정한 여러 파라미터 값들의 조합을 시도하여 최적의 조합을 찾아 준다. 의사결정 나무 모델의 파라미터 중 Max_Depth, Min_Samples_Leaf, Min_Samples_Split, Random_

State를 조정하였다. Grid Search를 진행할 파라미터의 범위는 다음 그림과 같고 Grid Search를 통해 도출된 최적의 조합을 구하였다.

```
param_grid_DT ={
    'max_depth' : list(range(2,16)),
    'min_samples_leaf' : list(range(1,6)),
    'min_samples_split' : list(range(2,6)),
    'random_state' : [10]
}
```

```
grid_search_DT = GridSearchCV(model_DT, param_grid_DT, n_jobs=-1 ,cv=5, scoring='accuracy')
grid_search_DT.fit(X_train, y_train)
```

```
Best Hyperparameters: {'max_depth': 6, 'min_samples_leaf': 1, 'min_samples_split': 2, 'random_state': 1
0}
```

이를 바탕으로 의사결정 나무 모델을 구축하였다.

```
tree1 = DecisionTreeClassifier(max_depth=6, min_samples_leaf=1, min_samples_split=2, random_state=10)
tree1.fit(X_train, y_train)
```

의사결정 나무 모델에 대해 Train Set과 Test Set의 정확도 및 분류 리포트를 구하였다.

```
Training set의 정확도: 0.745
Test set의 정확도: 0.743
```

1:1:1의 비율로 UnderSampling한 Data Set으로 의사결정 나무 모델을 구축하였을 때 Grid Search 결과와 Grid Search에서 도출된 최적의 조합의 파라미터로 의사결정 나무 모델을 구축하였다.

```
Best Hyperparameters: {'max_depth': 7, 'min_samples_leaf': 5, 'min_samples_split': 2, 'random_state': 1
0}
```

```
tree2 = DecisionTreeClassifier(max_depth=7, min_samples_leaf=5, min_samples_split=2, random_state=10)
tree2.fit(X_resampled_train1, y_resampled_train1)
```

의사결정 나무 모델에 대해 Train Set과 Test Set의 정확도 및 분류 리포트를 구하였다.

```
Training set의 정확도: 0.689
Test set의 정확도: 0.678
```

```
Decision Tree 클래스별 리포트:
              precision    recall  f1-score   support

          경상       0.54      0.61      0.57      3550
        상해없음       0.88      0.71      0.79      3552
          중상       0.66      0.71      0.69      3570

    accuracy                           0.68     10672
   macro avg       0.70      0.68      0.68     10672
weighted avg       0.70      0.68      0.68     10672
```

2:1:1의 비율로 UnderSampling한 Data Set으로 의사결정 나무 모델을 구축하였을 때 Grid Search 결과와 Grid Search에서 도출된 최적의 조합의 파라미터로 의사결정 나무 모델을 구축하였다.

```
Classification Report:
              precision    recall  f1-score   support

          경상       0.57      0.52      0.54      3550
        상해없음       0.82      0.77      0.79      3552
          중상       0.66      0.75      0.70      3570

    accuracy                           0.68     10672
   macro avg       0.68      0.68      0.68     10672
weighted avg       0.68      0.68      0.68     10672
```

```
Best Hyperparameters: {'max_depth': 7, 'min_samples_leaf': 1, 'min_samples_split': 4, 'random_state': 10}
```

```
tree3 = DecisionTreeClassifier(max_depth=7, min_samples_leaf=1, min_samples_split=4, random_state=10)
tree3.fit(X_resampled_train2, y_resampled_train2)
```

의사결정 나무 모델에 대해 Train Set과 Test Set의 정확도 및 분류 리포트를 구하였다.

```
Training set의 정확도: 0.693
Test set의 정확도: 0.682
```

```
Decision Tree 클래스별 리포트:
              precision    recall  f1-score   support

          경상       0.67      0.78      0.72      7138
        상해없음       0.80      0.71      0.75      3605
          중상       0.59      0.46      0.51      3486

    accuracy                           0.68     14229
   macro avg       0.69      0.65      0.66     14229
weighted avg       0.68      0.68      0.68     14229
```

<랜덤포레스트>

UnderSampling을 하지 않은 Data Set으로 랜덤포레스트를 구축하였다. 랜덤 포레스트는 앙상블 기법의 일종으로 여러 개의 의사결정 나무 모델을 사용하여 결과들을 조합한 후 예측 모델을 만들게 된다. 랜덤 포레스트의 파라미터는 N_estimators=150, Max_Depth=50, Min_Samples_Split=30, Min_Sample_Leaf=4, Random_State=42로 설정하여 구축하였다.

```
rf_model1 = RandomForestClassifier(n_estimators=150, max_depth=50, min_samples_split=30, min_samples_leaf=4, random_state=42)
rf_model1.fit(X_train, y_train)
```

랜덤 포레스트에 대해 Train Set과 Test Set의 정확도 및 분류 리포트를 구하였다.

```
훈련 세트 정확도: 0.7661523483596885
테스트 세트 정확도: 0.7446911030803648
```

```
Classification Report:
              precision    recall  f1-score   support

          경상       0.74      0.96      0.84     20006
        상해없음       0.78      0.51      0.62      3598
          중상       0.68      0.10      0.17      5451

    accuracy                           0.74     29055
   macro avg       0.74      0.52      0.54     29055
weighted avg       0.74      0.74      0.69     29055
```

1:1:1의 비율로 UnderSampling한 Data Set으로 랜덤 포레스트를 구축하였다.

```
rf_model2 = RandomForestClassifier(n_estimators=150, max_depth=50, min_samples_split=30, min_samples_leaf=4, random_state=42)
rf_model2.fit(X_resampled_train1, y_resampled_train1)
```

랜덤 포레스트에 대해 Train Set과 Test Set의 정확도 및 분류 리포트를 구하였다.

```
훈련 세트 정확도: 0.7153700951845455
테스트 세트 정확도: 0.6806596701649176
```

2:1:1의 비율로 UnderSampling한 Data Set으로 랜덤 포레스트를 구축하였다.

```
rf_model3 = RandomForestClassifier(n_estimators=150, max_depth=50, min_samples_split=30, min_samples_leaf=4, random_state=42)
rf_model3.fit(X_resampled_train2, y_resampled_train2)
```

랜덤 포레스트에 대해 Train Set과 Test Set의 정확도 및 분류 리포트를 구하였다.

```
훈련 세트 정확도: 0.7267387571914816
테스트 세트 정확도: 0.6882423220184131
```

```
Classification Report:
              precision    recall  f1-score   support

          경상       0.66      0.82      0.73      7138
        상해없음       0.81      0.71      0.76      3605
          중상       0.63      0.39      0.48      3486

    accuracy                           0.69     14229
   macro avg       0.70      0.64      0.66     14229
weighted avg       0.69      0.69      0.68     14229
```

<나이브베이즈>

UnderSampling을 하지 않은 Data Set으로 나이브베이즈 분류기를 사용해보았다. Data에 대한 최적화된 나이브베이즈 분류기를 얻기 위해 Grid Search를 사용하여 최적의 파라미터 조합을 찾았다. 나이브베이즈 분류기의 파라미터 중 alpha를 조정 하였다. Grid Search를 진행할 파라미터의 범위는 다음 그림과 같고 Grid Search를 통해 도출된 최적의 조합을 구하였다.

```
param_grid_NB = {'alpha': [0.1, 0.5, 1.0, 2.0, 5.0, 10.0]}
```

```
grid_search_NB1 = GridSearchCV(model_NB, param_grid_NB, cv=5, scoring='accuracy')
grid_search_NB1.fit(X_train, y_train)
```

```
Best Hyperparameters: {'alpha': 10.0}
```

이를 바탕으로 의사결정 나무 모델을 구축하였다.

```
model_NB1 = MultinomialNB(alpha=10)
model_NB1.fit(X_train, y_train)
```

나이브베이즈 분류기에 대해 Train Set과 Test Set의 정확도 및 분류 리포트를 구하였다.

```
Training set의 정확도: 0.668
Test set의 정확도: 0.661
```

```
Classification Report:
              precision    recall  f1-score   support

          경상       0.76      0.76      0.76     20006
        상해없음       0.42      0.54      0.47      3598
          중상       0.45      0.37      0.40      5451

    accuracy                           0.66     29055
   macro avg       0.54      0.56      0.55     29055
weighted avg       0.66      0.66      0.66     29055
```

1:1:1의 비율로 UnderSampling한 Data Set으로 나이브베이즈 분류기를 구축하였을 때 Grid Search 결과와 Grid Search에서 도출된 최적의 조합의 파라미터로 나이브베이즈 분류기를 구축하였다.

```
Best Hyperparameters: {'alpha': 10.0}
```

```
model_NB2 = MultinomialNB(alpha=10)
model_NB2.fit(X_resampled_train1, y_resampled_train1)
```

나이브 베이즈 분류기에 대해 Train Set과 Test Set의 정확도 및 분류 리포트를 구하였다.

```
Training set의 정확도: 0.610
Test set의 정확도: 0.599
```

```
Classification Report:
              precision    recall  f1-score   support

          경상       0.48      0.54      0.51      3550
        상해없음       0.66      0.75      0.70      3552
          중상       0.69      0.51      0.59      3570

    accuracy                           0.60     10672
   macro avg       0.61      0.60      0.60     10672
weighted avg       0.61      0.60      0.60     10672
```

2:1:1의 비율로 UnderSampling한 Data Set으로 나이브베이즈 분류기를 구축하였을 때 Grid Search 결과와 Grid Search에서 도출된 최적의 조합의 파라미터로 나이브베이즈 분류기를 구축하였다.

```
Best Hyperparameters: {'alpha': 10.0}
```

```
model_NB3 = MultinomialNB(alpha=10)
model_NB3.fit(X_resampled_train2, y_resampled_train2)
```

나이브 베이즈 분류기에 대해 Train Set과 Test Set의 정확도 및 분류 리포트를 구하였다.

```
Training set의 정확도: 0.611
Test set의 정확도: 0.609
```

```
Classification Report:
              precision    recall  f1-score   support

          경상       0.63      0.67      0.65      7138
        상해없음       0.60      0.67      0.63      3605
          중상       0.56      0.42      0.48      3486

    accuracy                           0.61     14229
   macro avg       0.60      0.59      0.59     14229
weighted avg       0.61      0.61      0.60     14229
```

<XGBoost>

UnderSampling을 하지 않은 Data Set으로 XGBoost를 사용해보았다. XGBoost 모델을 사용하기 위해서는 범주형 레이블을 숫자로 변환해주는 과정이 필요하다. 이러한 변환을 통해 모델이 이해할 수 있는 형태의 입력 데이터로 전처리 과정을 진행해야 한다. 따라서 범주형 표현을 가진 '상해없음', '경상', '중상'을 [0,1,2]로 전환하였다.

```
class_mapping = {'상해없음': 0, '경상': 1, '중상': 2}
y_mapping = y.map(class_mapping)
y_mapping
```

```
0        2
1        1
2        1
3        1
4        1
        ..
96842    2
96843    2
96844    1
96845    1
96846    1
Name: 피해운전자 상해정도, Length: 96847, dtype: int64
```

Data에 대한 최적화된 XGBoost를 얻기 위해 Grid Search를 사용하여 최적의 파라미터 조합을 찾았다. XGBoost 파라미터 중 Learning_Rate, Max_Depth, Min_Child_Weight, N_Estimators를 조정 하였다. Grid Search를 진행할 파라미터의 범위는 다음 그림과 같고 Grid Search를 통해 도출된 최적의 조합을 구하였다.

```
param_grid_XG = {
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5],
    'min_child_weight':[1,3,5],
    'n_estimators': [50, 100, 200]
}
```

```
grid_search_XGB1 = GridSearchCV(model_XGB, param_grid_XG, cv=3, scoring='accuracy')
grid_search_XGB1.fit(X_trainX, y_trainX)
```

```
Best Hyperparameters: {'learning_rate': 0.2, 'max_depth': 5, 'min_child_weight': 3, 'n_estimators': 200}
```

이를 바탕으로 의사결정 나무 모델을 구축하였다.

```
model_XGB1 = XGBClassifier(learning_rate = 0.2, max_depth = 5, min_child_weight = 3 ,n_estimators = 200)
model_XGB1.fit(X_trainX, y_trainX)
```

XGBoost에 대해 Train Set과 Test Set의 정확도 및 분류 리포트를 구하였다.

```
Training set의 정확도: 0.771
Test set의 정확도: 0.749
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.61      0.66      3598
           1       0.77      0.92      0.84     20006
           2       0.57      0.23      0.33      5451

    accuracy                           0.75     29055
   macro avg       0.69      0.58      0.61     29055
weighted avg       0.73      0.75      0.72     29055
```

1:1:1의 비율로 UnderSampling한 Data Set으로 XGBoost를 구축하였을 때 Grid Search 결과와 Grid Search에서 도출된 최적의 조합의 파라미터로 XGBoost를 구축하였다.

```
Best Hyperparameters: {'learning_rate': 0.2, 'max_depth': 5, 'min_child_weight': 3, 'n_estimators': 50}
```

```
model_XGB2 = XGBClassifier(learning_rate = 0.2, max_depth = 5, min_child_weight = 3 ,n_estimators = 50)
model_XGB2.fit(X_resampled_trainX1, y_resampled_trainX1)
```

XGBoost에 대해 Train Set과 Test Set의 정확도 및 분류 리포트를 구하였다.

```
Training set의 정확도: 0.710
Test set의 정확도: 0.687
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.77      0.80      3552
           1       0.57      0.58      0.57      3550
           2       0.67      0.71      0.69      3570

    accuracy                           0.69     10672
   macro avg       0.69      0.69      0.69     10672
weighted avg       0.69      0.69      0.69     10672
```

2:1:1의 비율로 UnderSampling한 Data Set으로 XGBoost를 구축하였을 때 Grid Search 결과와 Grid Search에서 도출된 최적의 조합의 파라미터로 XGBoost를 구축하였다.

```
Best Hyperparameters: {'learning_rate': 0.1, 'max_depth': 5, 'min_child_weight': 1, 'n_estimators': 50}
```

```
model_XGB3 = XGBClassifier(learning_rate = 0.1, max_depth = 5, min_child_weight = 1 ,n_estimators = 50)
model_XGB3.fit(X_resampled_trainX2, y_resampled_trainX2)
```

XGBoost에 대해 Train Set과 Test Set의 정확도 및 분류 리포트를 구하였다.

```
Training set의 정확도: 0.698
Test set의 정확도: 0.692
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.71      0.76      3605
           1       0.67      0.80      0.73      7138
           2       0.60      0.47      0.53      3486

    accuracy                           0.69     14229
   macro avg       0.70      0.66      0.67     14229
weighted avg       0.69      0.69      0.69     14229
```

5. 결론

의사결정나무

Raw Data					Undersampling Data				
	Precision	Recall	f1-score	Accuracy		Precision	Recall	f1-score	Accuracy
중상	0.56	0.20	0.29		중상	0.66	0.71	0.69	
경상	0.76	0.91	0.83	0.743	경상	0.54	0.61	0.57	0.678
상해 없음	0.71	0.60	0.63		상해 없음	0.88	0.71	0.79	

랜덤포레스트

Raw Data					Undersampling Data				
	Precision	Recall	f1-score	Accuracy		Precision	Recall	f1-score	Accuracy
중상	0.68	0.10	0.17		중상	0.66	0.75	0.70	
경상	0.74	0.96	0.84	0.744	경상	0.57	0.52	0.54	0.680
상해 없음	0.78	0.51	0.62		상해 없음	0.82	0.77	0.79	

나이브 베이즈

Raw Data					Undersampling Data				
	Precision	Recall	f1-score	Accuracy		Precision	Recall	f1-score	Accuracy
중상	0.45	0.37	0.40		중상	0.69	0.51	0.59	
경상	0.76	0.76	0.76	0.661	경상	0.48	0.54	0.51	0.599
상해 없음	0.42	0.54	0.47		상해 없음	0.66	0.75	0.70	

XGBoost

Raw Data

	Precision	Recall	f1-score	Accuracy
중상	0.57	0.23	0.33	
경상	0.77	0.92	0.84	0.749
상해 없음	0.73	0.61	0.66	

Undersampling Data

	Precision	Recall	f1-score	Accuracy
중상	0.67	0.71	0.69	
경상	0.57	0.58	0.57	0.687
상해 없음	0.83	0.71	0.80	

전체적인 결과를 보면 Raw Data가 더 좋은 Accuracy를 보여주나 f1-score에서 상대적으로 클래스 수가 적은 중상과 상해없음에서 낮은 수치를 보여준다. 반대로 1:1:1 Undersampling 데이터를 사용하면 Accuracy는 낮아지나 f1-score에서 경상 부분은 낮아졌으나 나머지 2개의 클래스는 큰 폭으로 증가 하였기 때문에 더 좋은 결과가 도출되었다고 생각한다.

또한 4개의 분류 모델을 비교해보았을 때 Accuary와 f1-score를 봤을때 XGBoost가 가장 좋은 모델임을 확인할 수 있었다. 여기서 추가로 1:1:1 Undersampling과 Raw Data 사이의 2:1:1 Undersampling을 1:1:1 데이터와 비교 해보고자 한다.

1:1:1	Decision Tree	Random Forest	Naïve Bayes	XGBoost
중상	0.69	0.70	0.59	0.69
경상	0.57	0.54	0.51	0.57
상해 없음	0.79	0.79	0.70	0.80
Accuracy	**0.678**	**0.680**	**0.599**	**0.687**

2:1:1	Decision Tree	Random Forest	Naïve Bayes	XGBoost
중상	**0.51**	**0.48**	**0.48**	**0.53**
경상	0.72	0.76	0.65	0.73
상해 없음	0.75	0.76	0.63	0.76
Accuracy	**0.682**	**0.688**	**0.609**	**0.692**

1:1:1과 2:1:1 데이터들을 비교해 봤을 때 2:1:1 데이터에서 당연히 Accuracy가 높아졌고 f1-score가 “중상”, “상해없음” 클래스에서 낮아 진것을 볼 수 있었다. 하지만 Raw Data와 비교하면 Accuracy가 큰 폭으로 증가하지 않았고 f1-score가 “상해없음” 은 크게 낮아지지 않았으나 “중상”에 대해서 f1-score가 낮아진 것은 큰 사고에 대해서는 잘 맞추지 못한다고 생각하였고 “상해 없음”을 맞추는 것보다 “중상”을 맞추는 것이 더 중요하다고 생각하여 1:1:1 데이터를 선택한다.

특성 중요도(의사결정나무)

Raw Data		Undersampling Data	
1~5등	6~10등	1~5등	6~10등
가해운전자 차종_자전거	피해운전자 차종_화물	가해운전자 차종_승용	피해운전자 차종_자전거
가해운전자 차종_이륜	법규위반_큰 위반	사고유형_차대차	피해운전자 연령
피해운전자 차종_승용	가해운전자 차종_승용	가해운전자 차종_이륜	가해운전자 차종_승합
피해운전자 차종_승합	가해운전자 차종_화물	피해운전자 차종_화물	가해운전자 차종_특수
피해운전자 연령	법규위반_작은 위반	피해운전자 차종_승합	가해운전자 차종_자전거

특성 중요도(랜덤 포레스트)

Raw Data		Undersampling Data	
1~5등	6~10등	1~5등	6~10등
가해운전자 차종_이륜	피해운전자 차종_이륜	가해운전자 차종_이륜	피해운전자 차종_보행자
가해운전자 차종_자전거	피해운전자 차종_승합	가해운전자 차종_승용	피해운전자 차종_승합
피해운전자 연령	가해운전자 연령	피해운전자 차종_이륜	사고유형_차대사람
피해운전자 차종_승용	피해운전자 차종_보행자	피해운전자 차종_승용	피해운전자 연령
가해운전자 차종_승용	사고유형_차대사람	가해운전자 차종_자전거	사고유형_차대차

특성 중요도(**XGBoost**)

Raw Data		Undersampling Data	
1~5등	6~10등	1~5등	6~10등
피해운전자 연령	법규위반_안전불이행	피해운전자 연령	피해운전자 차종_승용
가해운전자 연령	도로형태_교차로	가해운전자 연령	피해운전자 차종_승합
가해운전자 차종_이륜	가해운전자 차종_자전거	가해운전자 차종_자전거	가해운전자 차종_승용
가해운전자 차종_승용	법규위반_큰 위반	가해운전자 차종_이륜	피해운전자 성별_남
피해운전자 차종_승용	피해운전 성별_남	법규위반_큰 위반	피해운전자 차종_이륜

특성 중요도를 보면 의사결정 나무와 랜덤 포레스트에서 Raw Data든 1:1:1 Undersampling 데이터든 둘 다 가해/피해 운전자 차종이 중요한 특성임을 알 수 있었다. XGBoost에서도 가해/피해 운전자 연령이 1,2등을 차지하긴 하지만 나머지 특성들 중에서도 가해/피해 운전자 차종이 많이 있음을 알 수 있었다.

6. 한계점

원래 교통사고 데이터에서는 “사망”, “중상”, “경상”, “부상신고”, “상해없음”으로 이뤄졌으나 데이터 불균형 해소를 위해 “중상’, “경상”, “상해없음”으로 나누었기 때문에 더 정확한 예측이 어려워졌다.

추가로 교통사고 피해자만 예측을 하였으므로 가해자에 대해서는 예측이 이뤄지지 않았고 교통사고의 피해 규모에 대해서는 정확하게 알지 못한다. 이에 대해서는 ECLO 수치를 이용하여 회귀분석 등을 이용하면 어느 정도 예측 할 수 있다고 생각한다. 하지만 ECLO 수치는 숫자 이므로 분류 문제가 아닌 수치 예측이 된다.

또한 지역에서도 서울특별시 데이터만을 썼으므로 다른 지역에 대해서는 예측력이 조금 떨어질 수도 있다.

7. 느낀점

이규민**:** 처음에 주제 선정부터 이것저것 해보고 싶었지만 생각보다 공개된 데이터가 많지 않거나 유료인 경우도 많았다. 그래서 고민도 많이 했는데 공공데이터를 보다가 교통사고를 뒤늦게 생각하게 되었다. 이걸 가지고 EDA나 데이터 마이닝을 하면서 당연한 인사인트나 “차대사람”에서 상해없음이 거의 없다는 인사이트도 신기했다. 그리고 좋은 데이터가 있어서 나름 높은 정확도가 나왔던거 같고 “Garbage in, Garbage out”라는 말처럼 모델의 맞는 전처리를 잘 해야하고 어떤 데이터가 정확도에 영향이 가는지도 잘 파악을 해야한다는 것을 알게 되었다.

하지만 이번 프로젝트에서 제일 크게 느낀것은 주제가 큰 역할을 한다는 것이다 데이터 전처리나 데이터 마이닝 등은 주어진 일을 하면서 논문을 읽고 다른 참고자료를 찾으면서 어떻게 해결을 하면 되었다. 연구 주제도 그렇고 이런 팀프로젝트에서도 좋은 주제가 절반을 차지한다는 말이 지금까지 들은 과목들과 이제 진행할 졸업 프로젝트에 대입해서 생각해 보면 정말 맞는 말 같다.

이 과목을 계기로 다양한 머신러닝 기법들을 익히고 여러가지 참고자료를 바탕으로 지식을 쌓았고 다른 과목에서도 도움이 많이 되었던거 같다. 또한 팀원들간 팀워크가 더 올라가서 다음학기에도 주제선정 부터 결과 도출까지 의미있는 프로젝트가 되었으면 좋겠다.

조민정: 처음 프로젝트를 기획했을 때는 데이터 마이닝이라는 원대한 목표를 갖고 사회적으로도 유의미하면서 관심이 있는 주제를 선정하고 싶었다. 하지만, 가장 중요한 데이터 수집에 있어서 많은 어려움이 있었고 데이터가 존재한다 해도 우리 프로젝트에 맞는 컬럼들로 구성되어있지 않은 데이터들이 많았다. 그래서 데이터 수집 과정부터 문제가 있었을 뿐더러 관심있는 주제를 선택하면, 유의미한 결론을 도출할 만한 아이디어가 부족했기에 간극을 메우는게 쉽지 않았다.

그러면서 많은 고민을 통해 잘 정제되고 수집된 공공데이터 중 교통사고라는 키워드를 주제로 생각을 확장하였고 실제 운전하는 입장에서 유의미한 결론 도출을 위해 목표를 설정했다. 이 과정에서 팀원들 각자의 경험들을 모아 생각한 덕분에 프로젝트의 목적성이 분명해진 것이라 생각한다. 또한 데이터 수집 후 전처리가 가장 중요하다는 것을 많이 느낀 프로젝트였다. 수많은 수치형 데이터에서 어떤게 분석에 필요한지, 범주형으로 바꿔야하는 이유가 있는지, 이른 고민들이 있었기에 분석하는데 많은 도움이 됐다고 생각한다. 그럼에도 해소가 안됐던 데이터 불균형문제 때문에 추가 공부가 필요했는데, 관련한 연구들을 찾아 공부했던게 프로젝트 뿐만아니라 개인적으로도 가장 큰 도움이 됐다. 우리가 학부수준에서 수업에서 배운정도로 프로젝트의 완성도를 높이는데는 경험적인 것이 부족하다 생각하였고 그 시점에서 경험적으로 할 수 있는 노력은 많은 논문을 읽으며 다양한 연구 케이스를 이론공부 외로 공부하는 것이었다.

따라서 여러가지 방법론을 우리데이터데 적용시켜보고 같은 방향성을 가진 논문을 따라해보며(예를들면, upsampling) 프로젝트의 완성도를 높이려는 시도를 한 것이 이 프로젝트에서 가장 많이 얻어간 부분이었다. 데이터 마이닝이라는 것은 이론으로만 배웠을땨 '그렇구나'하고 넘어갈 수 있는 부분이 많다 생각했는데 프로젝트를 통해 체득하면서 이론을 체화시키는데도 도움을 많이 받았다.

김민지: 처음 프로젝트를 시작하면서는 여러 종류의 주제가 떠올랐다. 근래 빅데이터가 많은 사람들의 관심을 받으면서 많은 정제되지 않은 정보들 사이에 유의미한 결과를 찾아낸다는 것이 흥미로워보였다. 처음에는 이러한 데이터들을 수집하는 것에 있어서 어려움이 적을 것이라 생각했다. 하지만 데이터를 얻는 것조차 힘들고 설령 얻었다고 하더라도 우리가 원하는 방식대로 완전히 정제된 형태가 아니였다. 또한 데이터의 신뢰성이 보장되어 있는 데이터를 수집하고 싶었다.

수업중에 나왔던 Data Processing 과정에서 70%의 노력이 소요된다는 말을 체감하게 되었다. 우리는 흥미로워했던 주제 중 공공사이트 포털에서 신뢰할 수 있는 데이터를 수집할 수 있는 주제를 발견하게 되었고 이를 진행하기 팀원들과 의견을 나눈 뒤 결정하였다.이렇게 선정된 주제가 교통사고 심각도 분석이었다. 이는 교통사고에 영향을 줄 수 있는 정보들을 이용해 피해자 중심으로 사고 심각도를 예측하였다. 직관적으로도 알 수 있듯이 차대사람이 사고를 당한다면 피해자의 사고의 심각도가 올라가고 법규위반에서 큰 위반을 했을 시에 작은 위반을 했을 때보다 심각도가 올라간다는 사실이 실험을 통해 명확히 알 수 있다는 것이 신기했다. 사람은 직관적으로 느낄 수 있지만 그 결과의 증거를 만들어 근거로 삼는 일은 어렵다고 생각하기 때문이다. 실제로 새벽에는 4개의 시간대 중에서 큰 위반이 가장 많이 발생한다는 등의 상관관계 또한 프로젝트를 진행하면서 알 수 있었다.

프로젝트를 진행하면서 아쉬웠던 점은 기법에 사용하는 파라미터를 보다 더 우리의 모델과 데이터에 맞게 선정하는 것에 조금 어려움이 있었다는 점이다. 의사결정나무, XGBoost 등에서는 최적의 파라미터를 구할 수 있었지만 랜덤포레스트에서는 후보로 올릴 파라미터의 범위를 넓게 하면 24시간이 지나도 Grid Search의 결과값이 나오지 않았다. 우리의 프로젝트에서 교통사고에 대한 요인을 더 확장시켜 예를 들어, 어린이 보호구역, 신호등의 유무, cctv의 유무, 그 도로의 속도제한, 교통섬 여부, 근방의 밀도 지역등 추가시킨다면 보다 더 유의미한 결과를 도출할 수 있지 않았을까 라는 생각이 프로젝트를 마무리하면서 들게 되었다. 나중에 기회가 있다면 시도해보고 싶다.

허강**:** 프로젝트를 진행하면서 주제 선정과 데이터 획득에 어려움을 겪었다. 초기에는 무작정 관심 있는 주제를 선택하기보다는 유의미한 결론을 도출할 수 있는 적절한 데이터를 찾는 것이 우선이라는 점이 그 이유였다. 이로 인해 관심 있는 주제를 선택하기에는 한계가 있었던 것이 아쉬웠다.

데이터 마이닝 기법과 모델에 적용하는 과정에서는 특히 전처리에 많은 시도와 실험이 있었는데, 이러한 노력 끝에 모델 적용 단계에서는 어려움 없이 진행할 수 있었다. 교수님께서 강조하신 대로, 이 프로세스에서 데이터 전처리가 분석의 핵심이라는 말씀을 몸소 이해할 수 있었다. 데이터의 품질과 정확성은 분석의 기반이 되며, 이를 위해 전처리 단계에서의 세심한 작업이 매우 중요하다는 것을 깨닫게 되었다.

팀원 간 서로 부족한 부분을 인지하고 지원하며 PM은 프로젝트를 원활히 진행할 수 있도록 조율하는 역할을 잘 수행해 효과적으로 협업할 수 있었다.

이번 프로젝트를 통해 가장 크게 얻은 점은 논문 참고와 읽기에 대한 경험이었다. 프로젝트를 진행하면서 많은 논문을 살펴보며 논문을 읽는 데에 익숙해지고, 이에 대한 두려움이 사라진 것을 느꼈다. 이를 통해 논문을 통한 지식 습득에 대한 자신감이 상승하였다. 앞으로의 다른 프로젝트에서도 이번 경험을 활용하여 논문 리서치를 효과적으로 수행할 수 있을 것으로 믿어진다.

8. 참고자료

- 강흥식, 노명규*(2022).* 앙상블 학습기법을 활용한 보행자 교통사고 심각도 분류*:* 대전시 사례 중심으로 - *https://doi.org/10.14400/JDC.2022.20.5.039*

- 홍성은, 이구연, 김화종*(2015).* 공공데이터를 활용한 교통사고 상해 심각도 예측 모델 연구 - *10.14801/jkiit.2015.13.5.109*

- 강영옥, 손세린, 조나혜*(2017).* 의사결정나무와 시공간 시각화를 통한 서울시 교통사고 심각도 요인 분석 - *https://doi.org/10.22640/lxsiri.2017.47.2.233*

- 이호준, 이수기*(2021).* 보행자 교통사고 심각도 예측을 위한 머신러닝 모형 비교 분석: 보행사 교통사고 자료의 불균형 보정 중심으로 - *https://dx.doi.org/10.7319/kogsis.2021.29.2.003*

- 손소용, 신형원*(1998).* 데이터 마이닝을 이용한 교통사고 심각도 분류 분석

- 고창완, 김현민, 정영선, 김재희*(2020).* 차대차 교통사고에 대한 상해 심각도 예측 *https://doi.org/10.12815/kits.2020.19.4.13*