6조 기계학습론 최종 보고서

로봇공학과 2015041694 신상혁

로봇공학과 2016007183 조정용

산업경영공학과 2017010146 이시현

로봇공학과 2018043490 함서연

<목차>

전반적으로 데이터프레임을 많이 줄이고 표로 정리하는 것이 필요해 보임

1. 서론 : 다양한 전처리 방식 및 모델 방식

성능을 향상시키기 위하여 다양한 전처리 방식 및 모델 방식을 적용하였습니다.

아래의 내용은 6조가 적용해 보았던 전처리 방식 및 모델 방식에 대한 설명 입니다.

1) 전처리 방식

◎ 결측치 처리

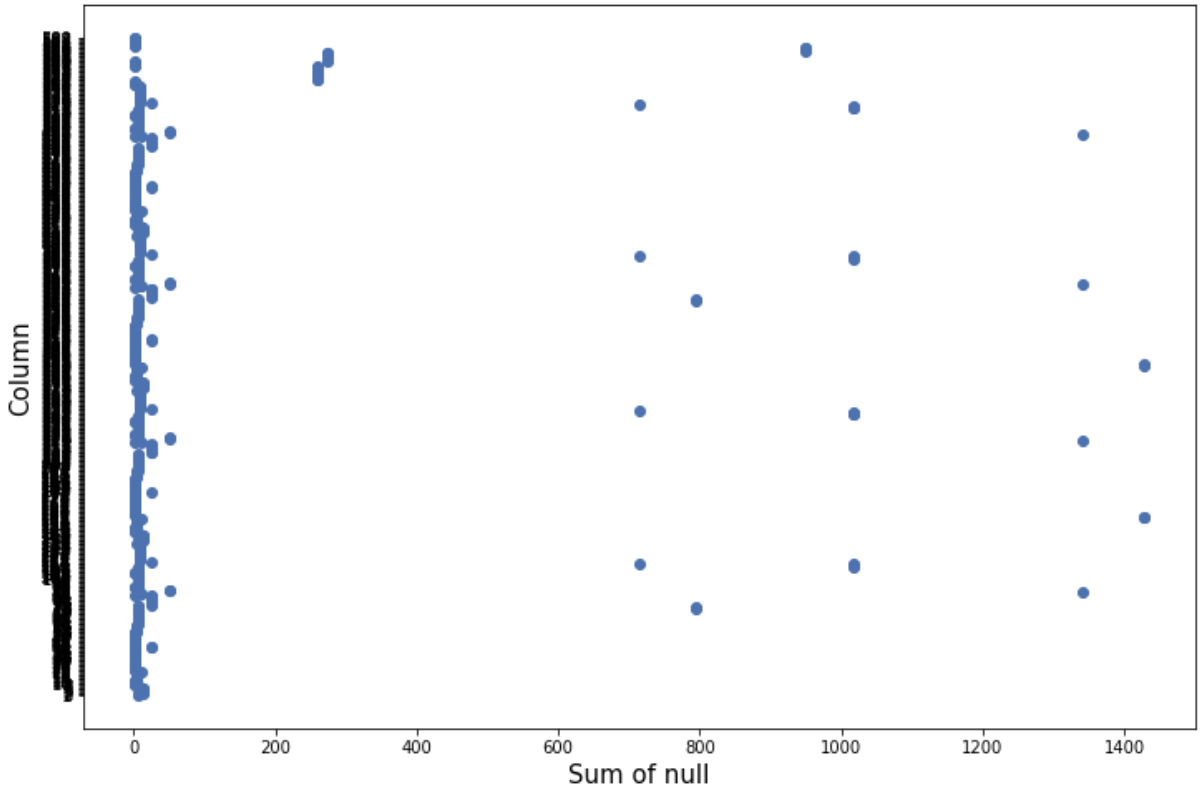
- 결측치 값 개수의 유일값의 전체 대비 % 확인

결측치 값 개수의 유일값	해당 feature	비율
0	20, 86, 87, 88, 113, 114, 115, 116, 117, 119, 120, 156, 221, 222, 223, 248, 249, 250, 251, 252, 254, 255, 291, 359, 360, 361, 386, 387, 388, 389, 390, 392, 393, 429, 493, 494, 495, 520, 521, 522, 523, 524, 526, 527, 570, 571, 572, 573, 574, 575, 576, 577, 590	0%
1	32, 33, 34, 35, 36, 37, 38, 39, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 83, 170, 171, 172, 173, 174, 175, 176, 177, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 218, 305, 306, 307, 308, 309, 310, 311, 312, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 356, 441, 442, 443, 444, 445, 446, 447, 448, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 490, 558, 559, 560, 561, 582, 583, 584, 585, 586, 587, 588, 589	0.06%
2	8, 9, 10, 11, 12, 21, 22, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 92, 93, 103, 104, 144, 145, 146, 147, 148, 159, 160 161, 162, 163, 164, 165, 167, 168, 169, 227, 228, 238, 239, 279, 280, 281, 282, 283, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 365, 366, 376, 377, 417, 418, 419, 420, 421, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 499, 500, 510, 511, 542, 543, 544, 545	0.13%
3	13, 14, 15, 16, 17, 18, 149, 150, 151, 152, 153, 154, 284, 285, 286, 287, 288, 289, 422, 423, 424, 425, 426, 427	0.19%
4	53, 54, 55, 56, 57, 58, 190, 191, 192, 193, 194, 195, 326, 327, 328, 329, 330, 331, 462, 463, 464, 465, 466, 467	0.26%
5	135, 270, 408	0.32%
6	0, 60, 61, 62, 66, 67, 68, 69, 70, 71, 74, 91, 94, 95, 96, 97, 98, 99, 100, 101, 102, 105, 106, 107, 108, 136, 197, 198, 199, 203, 204, 205, 206, 207, 208, 209, 226, 229, 230, 231, 232, 233, 234, 235, 236, 237, 240, 241, 242, 243, 271, 333, 334, 334, 339, 340, 341, 342, 343, 344, 347, 364, 367, 368, 369, 370, 371, 372, 373, 374, 375, 378, 379, 380, 381, 409, 469, 470, 471, 475, 476, 477, 478, 479, 480, 481, 498, 501, 502, 503, 504, 505, 506, 507, 508, 509, 512, 513, 514, 515	0.38%
7	1, 59, 63, 64, 65, 137, 196, 200, 201, 202, 272, 332, 336, 337, 338, 410, 468, 472, 473, 474	0.45%
8	132, 133, 134, 267, 268, 269, 405, 406, 407, 539, 540, 541	0.51%
9	7, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 143, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 278, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 416, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538	0.57%
10	19, 155, 290, 428	0.64%
12	84, 219, 357, 491	0.77%
14	2, 3, 4, 6, 138, 139, 140, 141, 142, 273, 274, 275, 276, 277, 411, 412, 413, 414, 415	0.89%
24	40, 41, 75, 76, 77, 78, 79, 80, 81, 82, 118, 178, 210, 211, 212, 213, 214, 215, 216, 217, 253, 313, 314, 348, 349, 350, 351, 352, 353, 354, 355, 391, 449, 450, 482, 483, 484, 485, 486, 487, 488, 489, 525	1.53%
51	89, 90, 224, 225, 362, 363, 496, 497	3.25%
260	546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557	16.59%
273	562, 563, 564, 565, 566, 567, 568, 569	17.42%
715	112, 247, 385, 519	45.63%
794	72, 73, 345, 346	50.67%
949	578, 579, 580, 581	60.56%
1018	109, 110, 111, 244, 245, 246, 382, 383, 384, 516, 517, 518	64.96%
1341	85, 220, 358, 492	85.58%
1429	157, 158, 292, 293	91.19%

https://dacon.io/competitions/open/235560/codeshare/518/ 참고 url

➤각 열에서 포함하고 잇는 **null**값 개수 확인

< 각 열별 null 값 분포 그래프> ③④⑤②①⑥⑧⑨⑩⑦



➤결측치 제거할 열 선정

- 전체 데이터가 총 1567개로 적은 데이터를 가지고 있다. 위의 분포에서 확인할 수 있듯이 700개 이상 결측치를 갖는 열이 있음을 알 수 있다. 이는 결측치가 전체 데이터의 40% 이상을 차지하므로 … 제거를 진행했다. 또한, 결측치가 20개 이상 600개 이하인 열 제거 전, 제거 후 성능을 비교해본 결과 제거 후 성능이 더 높음을 확인했고, 결측치 비율이 15% 이상 넘어가는 열도 제거했다. 결과적으로 결측치 비율이 1% 이하인 열만 남기고 나머지 열은 제거했다.

➤결측치 대체 값

null값을 대체하기 위해 0값, 중간값, 평균값, 최빈값을 넣어 성능을 비교해 보았다.

- 0값을 대체하였을 때는 이후 전처리를 안했을땐 좋은 성능을 보였지만, 전처리를 추가할수록 다른 대체값에 비해 성능이 떨어지는 경향을 보였다.
- 평균값으로 대체하였을 때는 준수한 경향을 보였으며, 특히 feature scaling 중 standardScaler에 최적의 성능을 보였다.
- 최빈값은 이후 전처리를 추가하거나 아님 전처리 전에서도 최저의 성능을 보였다.
- 중간값은 전처리 전에는 보통의 성능을 보였으나, 전처리를 할 수록 성능이 올라가는

것을 보여줬으며, 위의 3개의 값보다 최고의 성능을 보여줬다.

데이터 정규화

데이터를 모델링하기 전에 모든 변수가 동일한 정도의 스케일(중요도)로 반영되도록 해주기 위해 정규화를 진행했다.

Null 값 처리 후, 로지스틱 회귀 분류 모델을 베이스라인 모델로 설정하여 세 종류의 정규화 방법에 따른 성능을 비교해보았다. 성능은 random_state 1~100개의 값의 평균 값을 기준으로 한다.

Min-Max Normalization (최소-최대 정규화)

모든 Feature에 대해 각각의 최소값 0, 최대값 1로, 그리고 다른 값들은 0과 1사이의 값으로 변환하는 정규화 방법이다.

Standardization (Z-score Normalization, Z-점수 정규화)

데이터의 각 특성 값을 평균이 0, 표준편차는 1인 표준정규분포로 표준화 하는 방법이다.

RobustScaling

이상치, 특이값에 덜 민감한 정규화 방식으로 중앙값과 IQR을 이용해서 척도를 표준화하는 방법이다.

<정규화 방법 별 성능 비교 테이블>

정규화 방식	평균 roc_auc_score
Min-Max Normalization	0.6798342272062409
Standardization	0.6550446936453763
RobustScaling	0.6530131643100927

위의 표에서 볼 수 있듯이, Min-Max Normalization 정규화 방식이 가장 데이터 셋에 적합한 것을 확인하였다. 따라서 Min-Max Normalization 정규화 방식을 채택해 이후 데이터 전처리 및 모델링을 시행하였다.

① 변수 축소

⑴ 차원 축소

대부분의 머신러닝 모델은 입력 데이터의 차원이 클 경우, 차원의 저주와 학습 속도가 저하되는 문제를 갖고 있다. 이를 해결하기 위해 데이터에서 불필요한 Feature를 제거하는 작업을 진행하는데, 이를 차원 축소 혹은 특징 선택이라고 한다.

* 차원의 저주: 차원이 클수록 저차원일때보다 예측이 더 불안정해지는 문제

⑵ 특징 선택(Feature selection)과 특징 추출(Feature extraction)

데이터의 차원을 축소하기 위한 방법으로는 크게 특징 선택과 특징 추출이 있다. 특징 선택은 d 차원의 데이터 x를 구성하는 각 feature 중 어떠한 기준을 만족하는 p개의 feature를 선택한다. 반면에 특징 추출은 기존의 feature를 조합하여 새로운 feature를 생성하는 것이 특징이다.

이번 프로젝트에서는 특징 선택, 특정 추출 방법 모두 사용하였으며, 시도한 방법은 아래 표와 같다.

< 사용한 변수 축소 방법 >

<table>
<tr><th>시도한 특징 선택(Feature selection) 방법</th><th>시도한 특징 추출(Feature extraction) 방법</th></tr>
<tr><td>상관분석</td><td rowspan="2">주성분 분석 (PCA)</td></tr>
<tr><td>알고리즘 내재 방법
(Random Forest, Lasso, LGBM 사용)</td></tr>
</table>

래퍼 방법 (Recursive Feature Elimination, Sequential Feature Selection)	

⑶ 차원 축소 방법 설명

차원 축소 방법만의 성능 척도 변화를 살펴보기 위해 모델 및 전처리 방식의 베이스라인을 설정했다. 각각의 베이스라인은 다음과 같다.

전처리 베이스라인: Null값이 20개 이상인 열 제거 후, Null 값을 Median으로 변경

성능 평가 모델 베이스 라인: LogisticRegression(random_state=13,solver='liblinear',C=10.0)

⑶-1 상관 분석

변수(feature)간 상관관계가 큰 변수 쌍 중 하나를 제거하는 방식이다. 두 변수 중, label과 상관관계가 낮은 것 제거하였다. 성능평가 모델과 전처리는 설정한 베이스라인을 따라 진행하였다.

상관 분석 과정 및 결과

thereshold 값을 0.7로 설정하여 변수 간 상관관계가 thereshold 이상인 변수 추출

추출된 변수 쌍 중 label과의 상관관계가 낮은 변수를 제거

그 결과, 590개 변수 중 213개 변수 선택

➤ 상관분석 코드 (아래 코드는 직접 함수를 만들어서 작성하였습니다.)

```
def correlation_find(x, y, threshold):
    # Calculate the correlation matrix
    #<변수간의 상관관계 나타내기>
    corr_matrix = x.corr()
    iteration = range(len(corr_matrix.columns) - 1)
    drop_cols = []
    find_good = []

    # Iterate through the correlation matrix and compare correlations
    for i in iteration:
```

```
        for j in range(i+1):
            number = corr_matrix.iloc[j:(j+1), (i+1):(i+2)]
            column = number.columns
            row = number.index
            feature_value = abs(number.values)

            # If correlation exceeds the threshold
            if feature_value >= threshold:
                # Print the correlated features and the correlation value
                print(column.values[0], "and", row.values[0], "correlation value is
                ", round(feature_value[0][0], 2))
                drop_cols.append(column.values[0])
                find_good.append([column.values[0],row.values[0]])
    print(drop_cols)
    print(find_good)

    # Drop one of each pair of correlated columns
    drops = set(drop_cols)
    x = x.drop(columns=drops)

    # Calculate the correlation matrix
    #<feature와 label간의 상관관계>
    corr_matrix = y.corr()
    iteration = range(len(corr_matrix.columns) - 1)
    drop_cols = []

    # Iterate through the correlation matrix and compare correlations
    for j in iteration:
        number = corr_matrix.iloc[j:(j+1), 590:]#상관관계 높은 feature 두쌍중
     label(590)과 관계가 낮은 친구 제거
        column = number.columns
        row = number.index
        feature_value = abs(number.values)
        for fi in range(len(find_good)):
          if find_good[fi][0] == row.values:
            find_good[fi].append( ['Pass/fail',row.values[0],
                                   round(feature_value[0][0],2)] )
            print(column.values[0], "and", row.values[0], "correlation value is ",
                                   round(feature_value[0][0], 2))

          if find_good[fi][1] == row.values:
            find_good[fi].append( ['Pass/fail',row.values[0],
                                   round(feature_value[0][0],2)] )
            print(column.values[0], "and", row.values[0], "correlation value is",
                                   round(feature_value[0][0], 2))
        drop_cols.append(column.values[0])

    # Drop one of each pair of correlated columns
    drops = set(drop_cols)
    y = y.drop(columns=drops)
    trash_feature = []
    for trash in range(len(find_good)):
      if find_good[trash][2][2] > find_good[trash][3][2]:
        trash_feature.append(find_good[trash][3][1])
      else:
        trash_feature.append(find_good[trash][2][1])

    print(find_good)
    trash_feature = set(trash_feature)
    trash_feature = list(trash_feature)
    print(trash_feature)

    return trash_feature
```

```
semiconductor_passfail_correlation=correlation_find(semiconductor_final,semiconductor, 0.7)
```

➤ 상관 분석 결과 최종 선택된 데이터 셋

	0	1	2	3	5	6	7	8	9	10	11	12	13	14	15	18	20	21
0	3030.93	2564.00	2187.7333	1411.1265	100.0	97.6133	0.1242	1.5005	0.0162	-0.0034	0.9455	202.4396	0.0	7.9558	414.8710	192.3963	1.4026	-5419.00
1	3095.78	2465.14	2230.4222	1463.6606	100.0	102.3433	0.1247	1.4966	-0.0005	-0.0148	0.9627	200.5470	0.0	10.1548	414.7347	191.2872	1.3825	-5441.50
2	2932.61	2559.94	2186.4111	1698.0172	100.0	95.4878	0.1241	1.4436	0.0041	0.0013	0.9615	202.0179	0.0	9.5157	416.7075	192.7035	1.4123	-5447.75
3	2988.72	2479.90	2199.0333	909.7926	100.0	104.2367	0.1217	1.4882	-0.0124	-0.0033	0.9629	201.8482	0.0	9.6052	422.2894	192.1557	1.4011	-5468.25
4	3032.24	2502.87	2233.3667	1326.5200	100.0	100.3967	0.1235	1.5031	-0.0031	-0.0072	0.9569	201.9424	0.0	10.5661	420.5925	191.6037	1.3888	-5476.25
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1562	2899.41	2464.36	2179.7333	3085.3781	100.0	82.2467	0.1248	1.3424	-0.0045	-0.0057	0.9579	203.9867	0.0	11.7692	419.3404	193.7470	1.4072	-5418.75
1563	3052.31	2522.55	2198.5667	1124.6595	100.0	98.4689	0.1205	1.4333	-0.0061	-0.0093	0.9618	204.0173	0.0	9.1620	405.8178	193.7889	1.3949	-6408.75
1564	2978.81	2379.78	2206.3000	1110.4967	100.0	99.4122	0.1208	1.4616	-0.0013	0.0004	0.9658	199.5356	0.0	8.9670	412.2191	189.6642	1.4256	-5153.25
1565	2894.92	2532.01	2177.0333	1183.7287	100.0	98.7978	0.1213	1.4622	-0.0072	0.0032	0.9694	197.2448	0.0	9.7354	401.9153	187.3818	1.3868	-5271.75
1566	2944.92	2450.76	2195.4444	2914.1792	100.0	85.1011	0.1235	1.4616	-0.0013	0.0004	0.9658	199.5356	0.0	8.9670	412.2191	189.6642	1.4048	-5319.50

1567 rows × 271 columns

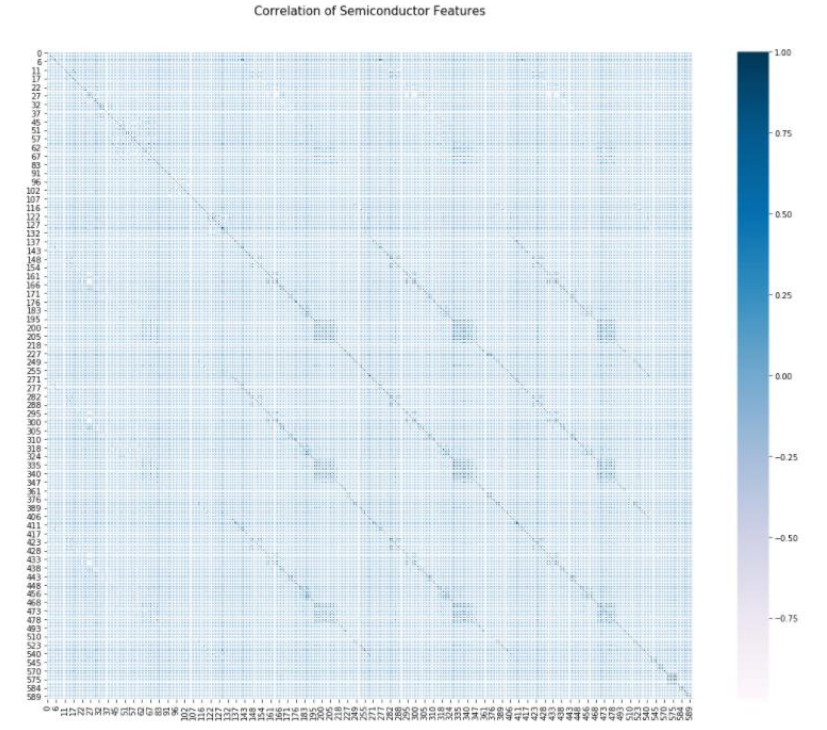
➤ 상관 분석 결과 시각화

상관관계 분석후 뽑힌 피처들중 일부를 선택하여 히트맵을 통한 데이터 시각화를 하여 보았다.

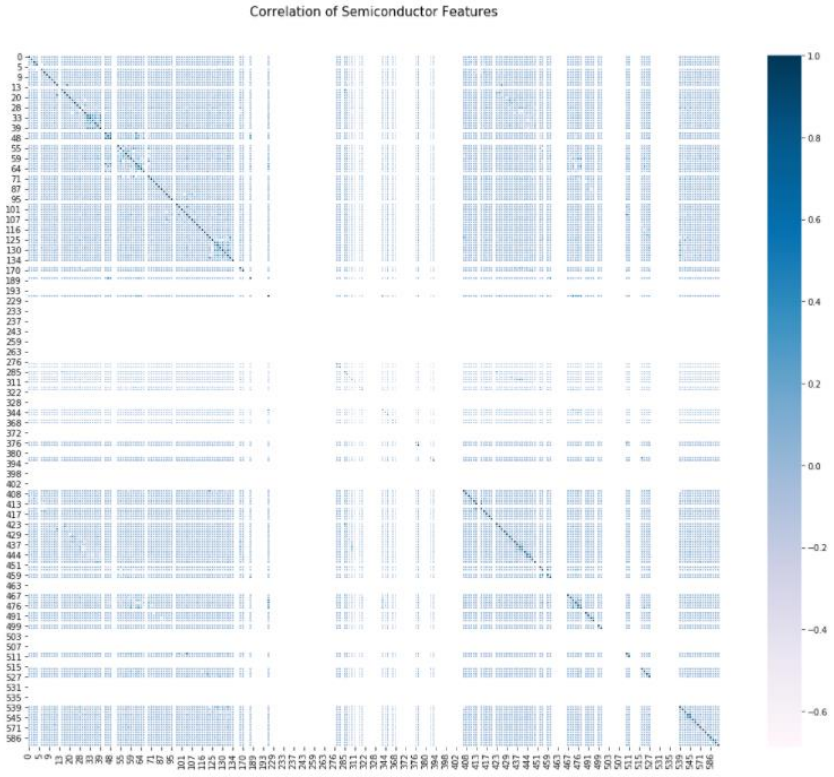
```
colormap = plt.cm.PuBu
plt.figure(figsize=(20, 15))
plt.title("Correlation of Semiconductor Features", y = 1.05, size = 15)
sns.heatmap(heatmap_data.astype(float).corr(), linewidths = 0.1, vmax = 1.0, square = True,
cmap = colormap, linecolor = "white", annot = True, annot_kws = {"size" : 16})
```

아래 히트맵을 통해 알 수 있듯이 상관 분석 전보다 상관 분석 후의 히트맵이 연해진 것을 알 수 있다. 이는 변수 간 상관관계가 줄어들었음을 의미한다.

< 상관 분석 전 데이터 셋의 히트맵>



< 상관 분석 후 데이터 셋의 히트맵>



➤ 성능 평가

Logistic regression과 KNN(K-nearest neighbor) 을 대표로 적용하여 성능 측정하였다. train_test_split 함수의 파라미터 randomstate는 13으로 고정하는 방법과 random_state 를 1~100까지로 설정 하여 평균 낸 값 모두 확인하였다. 또한, 다양한 스케일링 방법을 적용하여 최선의 결과를 확인하였다.

randomstate = 13으로 고정한 값

Data preprocessing based on data correlation ➜ Logistic regression: 0.7426764

- data preprocessing based on data correlation ➜standardscalar ➜logistic regression:0.7341592
- data preprocessing based on data correlation ➜minmaxscalar ➜logistic regression:0.750548
- data preprocessing based on data correlation ➜knn(k nearest neighborhood):0.656213
- data preprocessing based on data correlation ➜standardscalar ➜knn(k nearest neighborhood):0.6987353
- data preprocessing based on data correlation ➜minmaxscalar ➜knn(k nearest neighborhood):0.656213
- data preprocessing based on data correlation ➜standardscalar➜ stratified k fold ➜ knn(k nearest neighborhood):0.759414
- data preprocessing based on data correlation ➜minmaxscalar ➜stratified k fold ➜knn(k nearest neighborhood):0.775389

randomstate1~100까지 평균낸 값

- data preprocessing based on data correlation ➜standardscalar ➜logistic regression:0.6935712008394685
- data preprocessing based on data correlation ➜minmaxscalar ➜logistic regression:0.6900315719649156

Best ROC_AUC_SCORE

randomstate 14일 때의 값: 0.8054683998207082

Data preprocessing based on data correlation ➜ Minmaxscalar ➜ Logistic regression

Random_state=14 이고, 위의 설명과 같이 모델링을 진행했을때 가장 높은 roc_auc_score를 볼 수 있었다. 하지만 random_state를 1~100으로 평균을 냈을 때 성능이 하락했으므로,최종 변수 선택 방법으로는 사용하지 않았다.

⑶-2 알고리즘 내재 방법

트리 기반 모델(또는 선형 모델)들이 특성 중요도(Feature Importance)를 제공하는 것에 기반한 방법이다. 대표적으로 Decision Tree Classifier가 있다.

프로젝트에서 사용한 알고리즘 Random Forest, LGBM는 트리 기반의 알고리즘이다. Lasso에 대한 설명 추가. 코드 설명 추가. LGBM은 파라미터가 100개 이상으로 굉장히 많은 파라미터를 가지고 있다.

모든 가능한 분할점에 대해 정보 획득을 평가하기 위해 데이터 개체 전부를 스캔해야 했기 때문이다. 이는 당연하게도, 굉장히 시간 소모적이다.

(3)-3 래퍼 방법

지도학습을 미리 선정한 뒤, 해당 지도학습 모형에 대해 선택 가능한 모든 특징 집합을 비교하는 방법이다. Feature의 수가 n개일 경우, 2^n-1번이라는 비교가 실질적으로 쉽지 않다. 효율적으로 특징 집합을 탐색하는 방법이 필요한데, 그 방법에는 전방 선택(forward selection) 과정, 후방 제거(backward elimination) 과정이 있다. 전방 선택 방법으로 Sequential Feature Selection을 사용하였고, 후방 선택 방법으로는 Recursive Feature Elimination를 사용하였다.

Sequential Feature Selection (SFS)

그리디 알고리즘으로 빈 subset에서 변수를 하나씩 추가하는 방법으로 실행된다. 최후엔 원하는 피쳐만 남게 된다. 전처리는 설정한 베이스라인을 따라 진행하였다.

최대 검출 변수는 250으로, 방향은 forward로, 성능 평가는 'roc_auc'로, 교차 검증은 3겹으로 함수를 설정하였다. (최대 검출 피쳐를 250으로 한 이유는 500개로 설정하였을 때, 230부근에서 최댓값을 갖고 점점 성능이 떨어지는 경향을 보여서 250으로 설정함.)

SFS를 통한 변수 선택 과정 및 결과

SFS로 238개 변수 선택

LogisticRegression으로 훈련 후 성능 평가

➤ **SFS를 통한 변수 선택** 코드

```
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
log_reg = LogisticRegression(random_state=13, solver='liblinear')
sfs1 = SFS(log_reg,
           k_features=250,
           forward=True,
           floating=False,
           verbose=2,
           scoring='roc_auc',
           cv=3)
```

➤ **Best ROC_AUC_SCORE**

randomstate1~100까지 평균낸 값 : 0.7196587030716722

Recursive Feature Elimination(RFE)

RFE는 재귀적(반복적) 특성 선택 제거 방법으로, 모든 속성들을 포함하는 모델을 기준으로 삼고 시작한다. 하나씩 변수를 제거하면서 새 모델을 만들고, 새 모델이 기준 모델보다 좋다면 그 모델을 새로운 기준으로 삼는 방법이다. 종료 기준이 만족되면 종료한다.

전처리는 설정한 베이스라인을 따라 진행하였다.

(3)-4 PCA (Principle Component Analysis)

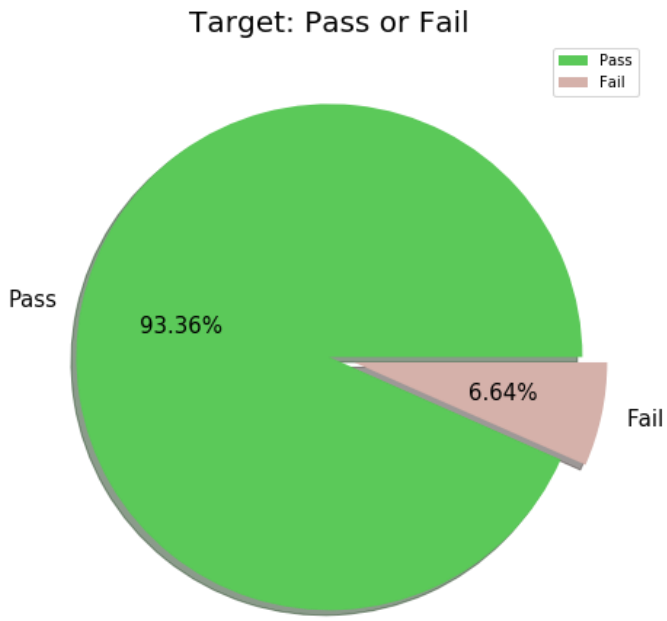
데이터의 분산은 최대한 보존하면서 서로 직교하는 평면을 찾아 고차원 공간의 표본들을 저차원 공간으로 변환하는 기법이다. PCA의 목적은 데이터의 분산을 최대한 보존하는 축을 찾는데 있다. 이번 프로젝트에서는 훈련 세트의 분산을 95%로 유지하는 평면을 찾아 PCA를 진행하였다.

성능 비교 표.

PCA 는 변수를 결합하여 주성분을 찾아주고, 차원 감소도 시키기 때문에 차원 축소 방법 중 인기 있는 방법이다. 하지만 어떤 변수가 합쳐져서 주성분이 만들어졌는지를 알 수 없는 한계가 존재했기 때문에 더 투명한 변수 선택 방법인 트리 기반 알고리즘을 활용한 알고리즘 내재 방법을 채택하였다.

③ 언더 샘플링, 오버 샘플링

1. 샘플링을 왜 시도했는지, 각 샘플링 방법별로 어떤 장점때문에 사용했는지 적기
2. 각 시도별로 성능이 어땠는지 비교하는 표 보여주기
3. 왜 이게 가장 좋은 성능이 나왔을지 분석
4. **smote친구랑 아다신 친구중 조장님께서 해본 친구들은 코드 같이 올려주시면 좋을것 같아용**

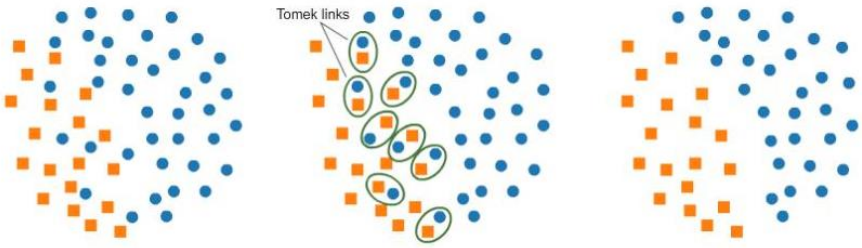


- **언더 샘플링**

  (1) Random Under Sampling

  - **majority class의 값들을 임의로 제거하는 방법이다.**
  - **단점 : 임의로 제거하는 방법이기 때문에 편향된 결과가 나올 수도 있고 모집단을 대표하지 못 할 수 있다.**

  (2) Tomek links

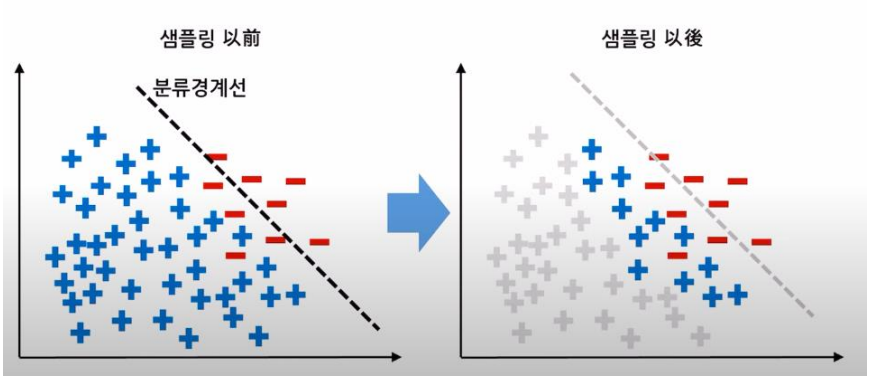


[그림 3] 토멕링크(Tomek Link) /출처: Rafael Alencar - Resampling strategies for imbalanced datasets

- **Tomek links란 쉽게 말해 가장 가까운 거리에 있는 minor class와 major class의 쌍을 말하는데, 이렇게 Tomek links를 형성한 후, major class에 속한 데이터를 지우는 방법이 Tomek links 언더 샘플링 기법이다.**
- **주로 경계선 근처에 있는 major class의 데이터가 삭제 된다.**

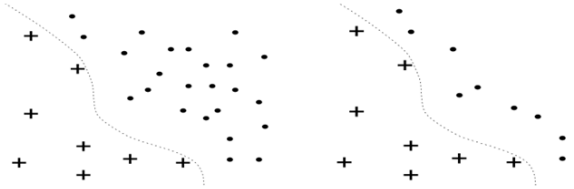
⑶ CNN (condensed nearest neighbor)

- **major class에 밀집된 데이터가 없을 때까지 데이터를 제거하여 데이터 분포에서 대표적인 데이터만 남도록 하는 방법이다.**



⑷ OSS (one-side selection)

- **Tomek links와 CNN기법을 합친 방법이다.**
- **각각의 기법에서 언더샘플링 된 부분을 모두 적용하여 서로의 단점을 보완한 방법이다.**



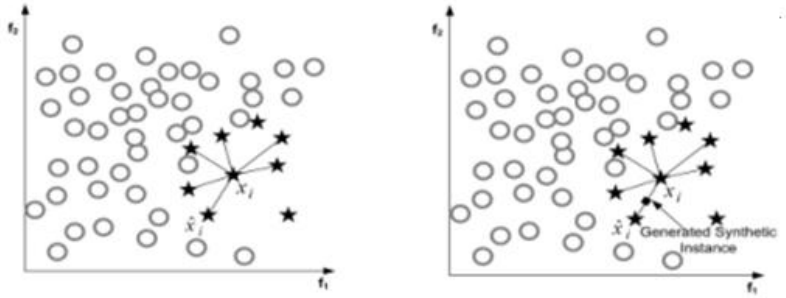
[그림 4] One Sided Selection / 출처: M Kubat, S Matwin - Addressing the curse of imbalanced training sets: one-sided selection

- **오버 샘플링**

  (1) Random Over Sampling

  - **이름 그대로 minor class의 instance를 임의로 복제하여 수를 늘리는 방법이다.**
  - **단점 : minor class에 과적합 될 가능성이 있다.**

  (2) SMOTE

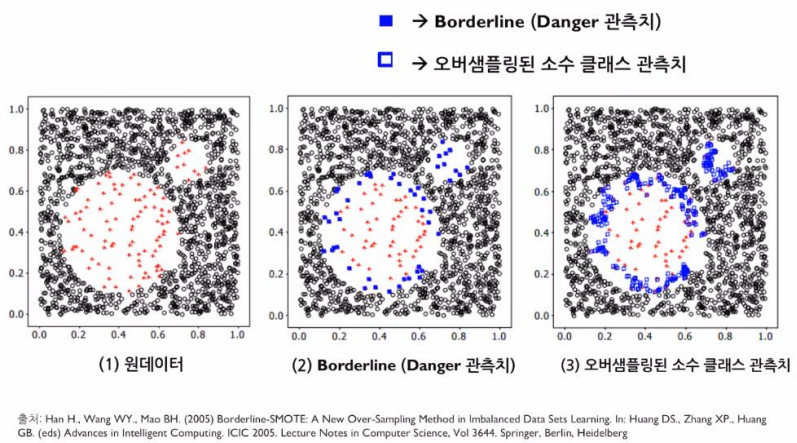


[그림 1] SMOTE / 출처: Haibo He, Edwardo A. Garcia - Learning from imbalanced data

  - **가장 가까운 이웃 k개중 랜덤하게 하나를 선택하여 그 두 점 사이에 가상의 데이터를 추가하는 방법이다.**

  (3) Borderline - SMOTE

  - **SMOTE의 단점을 개선한 방법으로, Borderline 부분만 SMOTE를 적용하여 샘플링하는 방법이다.**



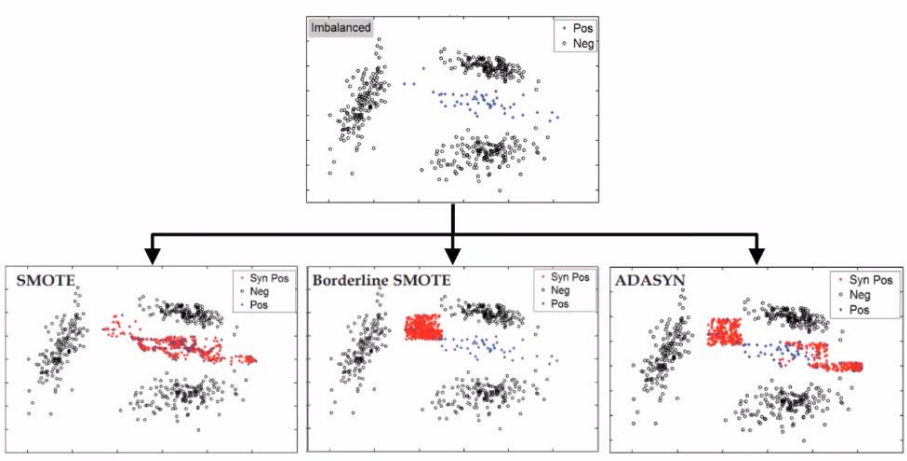
출처: Han H., Wang WY., Mao BH. (2005) Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning. In: Huang DS., Zhang XP., Huang GB. (eds) Advances in Intelligent Computing. ICIC 2005. Lecture Notes in Computer Science, Vol 3644. Springer, Berlin, Heidelberg

  (4) ADASYN

  - **minor클래스 데이터 주변의 k개중 major클래스 데이터가 얼마나 존재하는지**

에 따라 샘플링하는 개수를 다르게하는 방법이다.

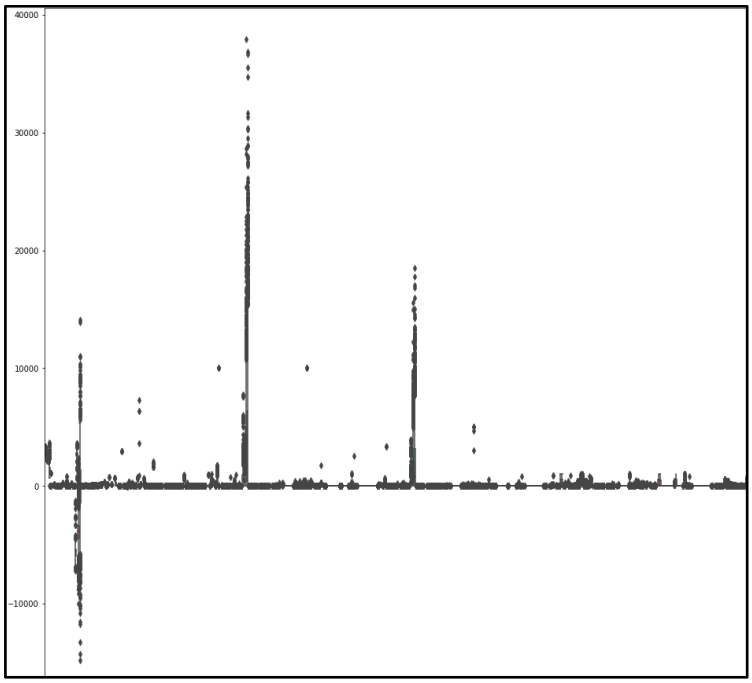
- SMOTE와 Borderlin-SMOTE의 장점을 모두 가지고 있다.



출처: He, H., Bai, Y., Garcia, E.A., & Li, C. (2008). ADASYN: Adaptive synthetic sampling approach for imbalanced learning. 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), 1322-1328.

④이상치(Outlier) 제거

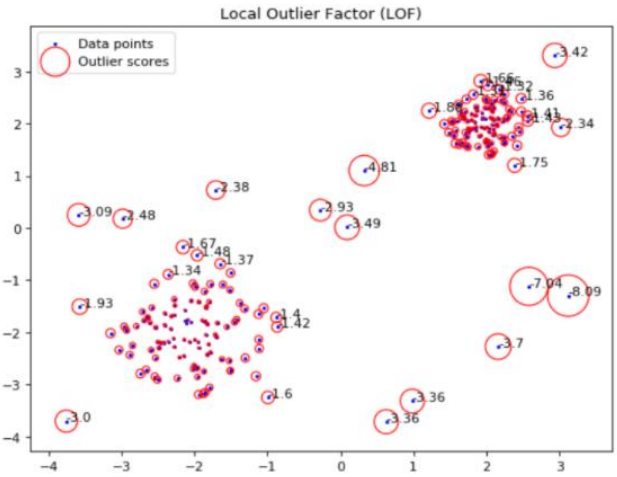
1) 데이터 셋에서의 열에 따른 데이터 분포



2) 이상치 제거 알고리즘

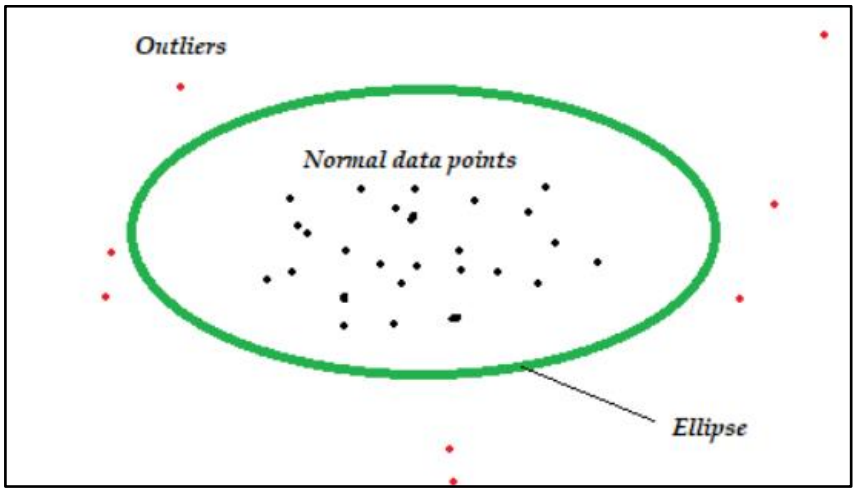
➤LocalOutlierFactor

- 데이터간의 거리와 밀도를 상대적으로 고려하여 이상치 판별.



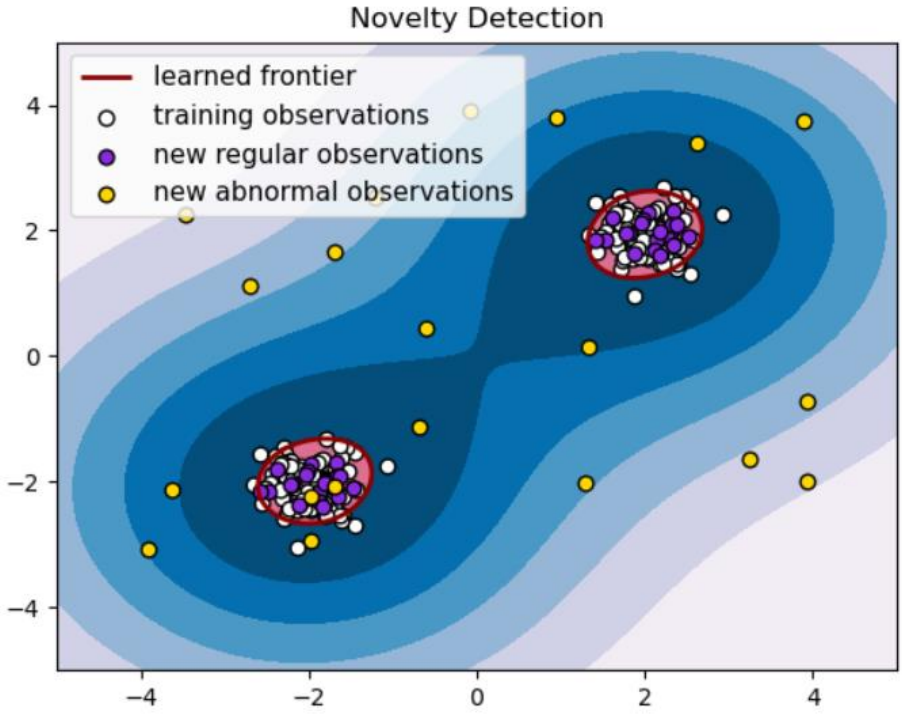
➤EllipticEnvelope

- 공분산 추정을 하고, 중심 데이터들을 타원형에 fit하여, central mode를 벗어난 데이터를 이상치로 판별.



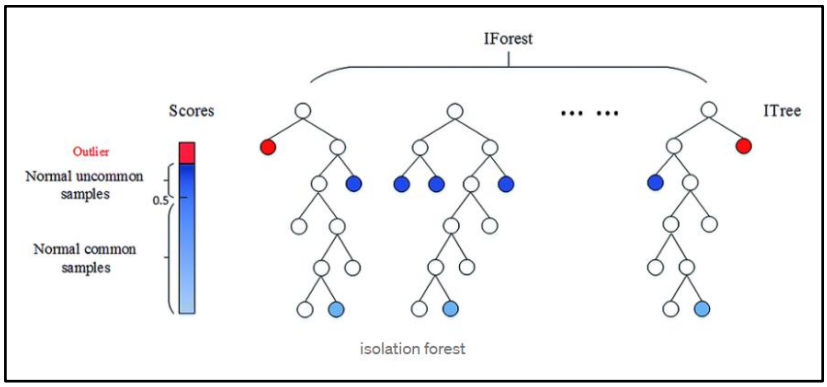
➤OneClassSVM

- 특정 범위를 설정할 수 있으며, 커널을 이용하여 이상치 판별.



➤IsolationForest

- 고차원 데이터에서 random forest를 사용하여 이상치 판별.



3) 이상치 제거 비율

알고리즘	제거된 행 개수(전체 대비 비율)
LocalOutlierFector	43(2.744%)
EllipticEnvelope	157(10.02%)
OneClassSVM	784(50.03%)
IsolationForest	2(0.128%)

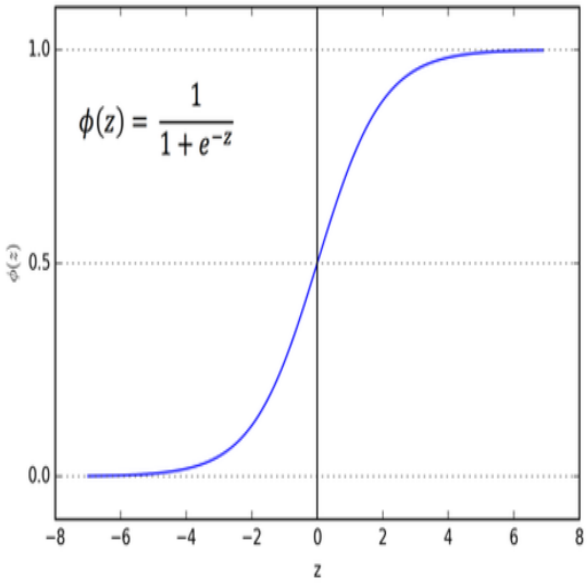
4) 알고리즘 선택

- EllipticEnvelope와 OneClassSVM은 전체 데이터중 10%이상을 제거하므로 정보 손실이 크기 때문에 부적합하다고 판단했다.
- 10%미만인 LocalOutlierFector와 IsolationForest에서는 여러 전처리와 모델들에 대해서 성능을 비교하여 선택하는 것으로 판단한다.

2) 모델 학습 방식

1. **모델 학습 방식 간단히 설명, 분류가 예측 모델에 목적이므로 ~한 장점을 가진 이진 분류 모델을 선택했다 등과 같은 모델 선택 이유 설명**

① Logistic regression

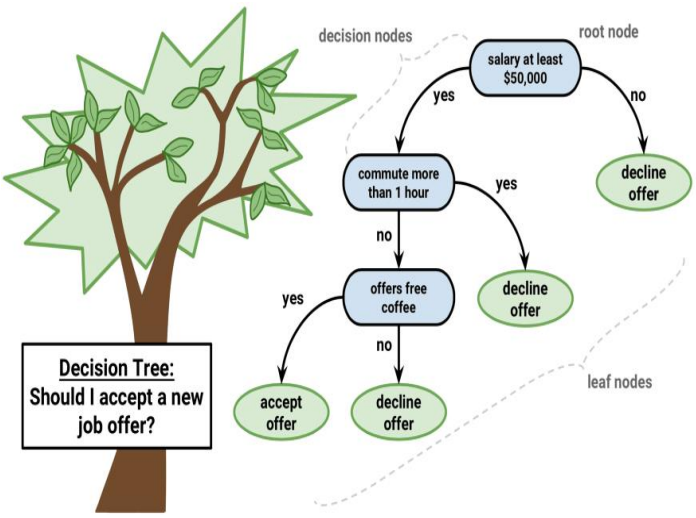


로지스틱 회귀의 목적은 종속변수와 독립변수 간의 관계를 구체적인 함수로 나타내어 향후 예측 모델에 사용하는 것이다. 독립변수의 선형 결합을 이용하여 사건의 발생가능성 (일어날 확률/일어나지 않을 확률)을 예측하는 통계적 기법을 활용한 것이다.결과적으로 로지스틱 회귀의 y값은 0~1 사이의 확률값이 된다. classification 기법이다.

➤ **LogisticRegression모델에 학습**

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score

random_state = []

for j in range(1,101):
    random_state.append(j)
ai = []
for rand in random_state:
  X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=
rand,stratify=y)
  log_reg = LogisticRegression(random_state=13,solver='liblinear',C=10.0)
  log_reg.fit(X_train, y_train)
  pred_proba_1 = log_reg.predict_proba(X_test)[:,1]
  auc_test = roc_auc_score(y_test, pred_proba_1)
  print('random_state=', rand,'  ' ,'auc_test =', auc_test)
  print('')
  ai.append(auc_test)
print("평균 = ", np.mean(ai))
```

②Decision tree



특정 기준(질문)에 따라 데이터를 구분하는 모델을 결정 트리 모델이라고 한다. 한번의 분기 때마다 변수 영역을 두 개로 구분한다. 결정 트리에서 질문이나 정답을 담은 네모 상자를 노드(Node)라고 한다. 맨 처음 분류 기준(즉, 첫 질문)을 Root Node라고 하고, 맨 마지막 노드를 Terminal Node 혹은 Leaf Node라고 한다.

③Light Gradient Boosting(LGBM)

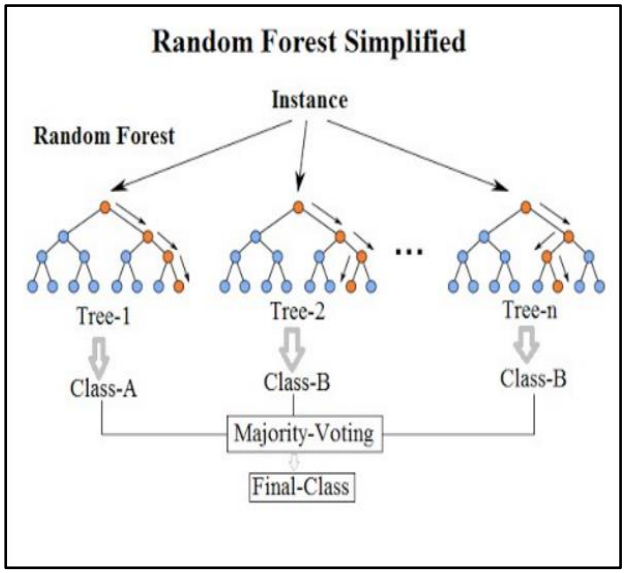
Light GBM은 Gradient Boosting 프레워크로 Tree 기반 학습 알고리즘이다. 기존의 다른 Tree 기반 알고리즘과 다른 확장 방식을 사용한다. Light GBM은 Tree가 수직적으로 확장되는 반면에 다른 알고리즘은 Tree가 수평적으로 확장된다. 즉 Light GBM은 leaf-wise 인 반면 다른 알고리즘은 level-wise 이다. 확장하기 위해서 max delta loss를 가진 leaf를 선택하게 된다. 동일한 leaf를 확장할 때, leaf-wise 알고리즘은 level-wise 알고리즘보다 더 많은 loss, 손실을 줄일 수 있는 장점이 있다.

④XGboost

트리 기반의 알고리즘의 앙상블 학습에서 각광받는 알고리즘 중 하나입니다. GBM에 기반하고 있지만, GBM의 단점인 느린 수행시간, 과적합 규제 등을 해결한 알고리즘입니다.

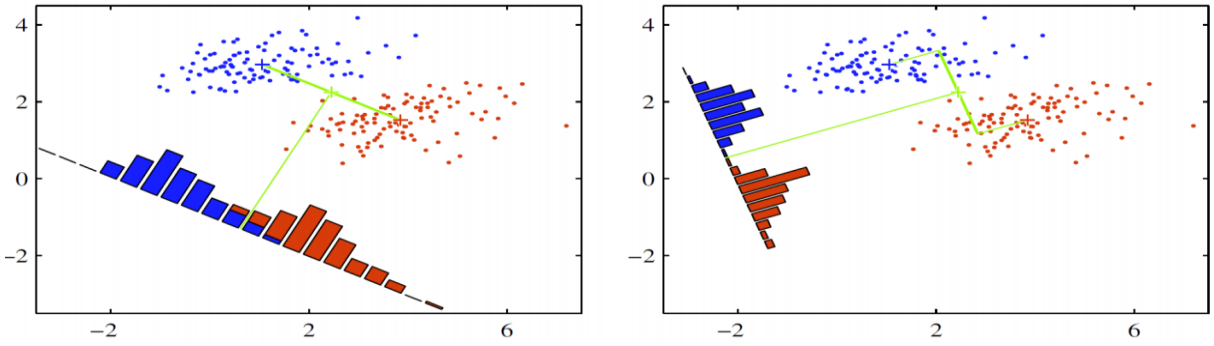
- 유연한 학습시스템으로 여러 파라미터를 조절해가면서 최적의 모델을 만들 수 있다.
- Overfitting(과적합)을 방지할 수 있다.
- 신경망에 비해 시각화가 쉽고, 직관적으로 이해할 수 있다.
- 자원(CPU, 메모리)이 많으면 많을수록 빠르게 학습하고 예측할 수 있다.
- Cross validation을 지원한다.

⑤Random forest



랜덤 포레스트의 포레스트는 숲(Forest)이다. 결정 트리는 트리는 나무(Tree)이다. 나무가 모여 숲을 이룹니다. 즉, 결정 트리(Decision Tree)가 모여 랜덤 포레스트(Random Forest)를 구성한다. 결정 트리 하나만으로도 머신러닝을 할 수 있지만 결정 트리의 단점은 훈련 데이터에 오버피팅이 되는 경향이 있다는 것이다. 따라서 여러 개의 결정 트리를 통해 랜덤 포레스트를 만들면 오버피팅 되는 단점을 해결할 수 있습니다.

⑥Linear Discriminant Analysis(LDA)



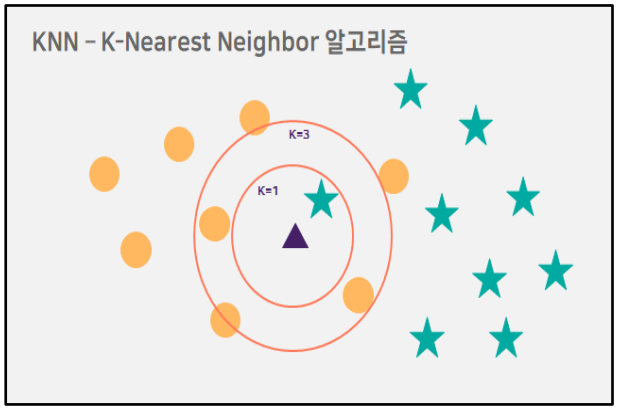
LDA는 데이터를 특정 한 축에 사영(projection)한 후에 두 범주를 잘 구분할 수 있는 직선을 찾는 걸 목표로 한다. 모델 이름에 linear라는 이름이 붙은 이유이다. 왼쪽 그림에 나타난 축은 중간에 빨간색과 파란색 점이 뒤섞여 있기 때문에 오른쪽이 더 나은 것 이다.두 범주를 잘 구분하기 위해서 직선은 사영 후 두 범주의 중심(평균)이 서로 멀리 있도록, 그 분산이 작도록 해야 한다.

⑦Quadratic Discriminant Analysis(QDA)

QDA는 LDA에서 크게 다르지 않다. QDA는 LDA에서 공통 공분산구조에 대한 가정을 제외

시킨 것이다. 각 확률변수들이 어떻게 퍼져있는지를 나내는 것이 공분산(Covariance)이다.

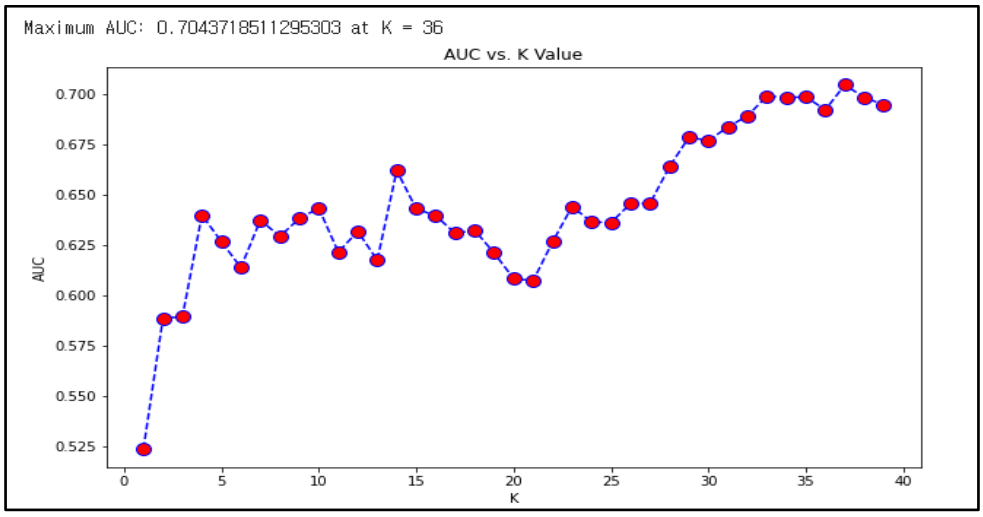
⑧K-Nearest Neighbor(KNN)



KNN 알고리은 새로운 데이터와 기존 데이터들 간거리를 측정하고 가까운 데이터들의 종류가 무엇인지 확인하여 새로운 데이터의 종류를 판별하는 알고리즘이다.K는 인접한 데이터의 갯수를 의미한다.

➤ 적합한 K value 찾기

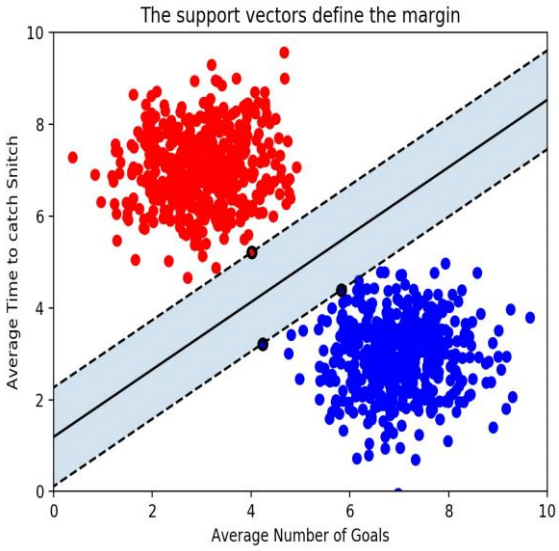
```
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt

auc = []
# Will take some time
from sklearn import metrics
for i in range(1,40):
    neigh = KNeighborsClassifier(n_neighbors = i).fit(X_train,y_train)
    pred_proba_1 = neigh.predict_proba(X_test)[:,1]
    auc.append(metrics.roc_auc_score(y_test, pred_proba_1))

plt.figure(figsize=(10,6))
plt.plot(range(1,40),auc,color = 'blue',linestyle='dashed',
         marker='o',markerfacecolor='red', markersize=10)
plt.title('AUC vs. K Value')
plt.xlabel('K')
plt.ylabel('AUC')
print("Maximum AUC:",max(auc),"at K =",auc.index(max(auc)))
```



➤ K value를 적용하여 KNN모델에 학습

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
random_state = []
for j in range(1,101):
    random_state.append(j)
ai = []

for rand in random_state:
  X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=
rand,stratify=y )
  KNN = KNeighborsClassifier(n_neighbors=36)
  KNN.fit(X_train, y_train)
  pred_proba_1 = KNN.predict_proba(X_test)[:,1]
  auc_test = roc_auc_score(y_test, pred_proba_1)
  print('random_state=', rand,'  ' ,'auc_test =', auc_test)
  print('')
  ai.append(float(auc_test))
print(ai)
print(sum(ai)/len(ai))
```

⑨Support Vector Machine(SVM)



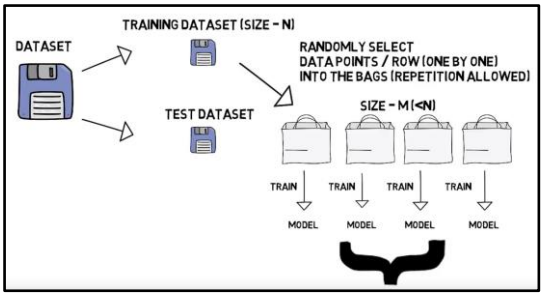
서포트 벡터 머신(SVM)은 결정 경계(Decision Boundary), 즉 분류를 위한 기준선을 정의하는 모델이다. 분류되지 않은 새로운 점이 나타나면 경계의 어느 쪽에 속하는지 확인해서 분류 과제를 수행할 수 있게 된다.. Support Vectors는 결정 경계와 가까이 있는 데이터 포인트들을 의미한다. 이 데이터들이 경계를 정의하는 결정적인 역할을 한다.

⑩ ExtraTreesClassifier

포레스트 모델의 변종으로 익스트림 랜덤 트리extremely randomized trees 혹은 엑스트라 트리ExtraTrees라 부르는 모델이 있다. 엑스트라 트리는 포레스트 트리의 각 후보 특성을 무작위로 분할하는 식으로 무작위성을 증가 시킨다.

⑪ BaggingClassifier

Bagging은 입력 데이터를 모델 수 만큼 나눈 뒤, 각각 학습시킨다. 이후, Test dataset 을 각 모델에 넣어 예측할 때, 출력되어 나온 예측 값들을 voting 하여, 보다 더 투표를 받은 예측 값이 최종 예측값이 된다.



2. 제안하는방법: ROC_AUC_SCORE가 평균적으로 높은 방식(이론설명)

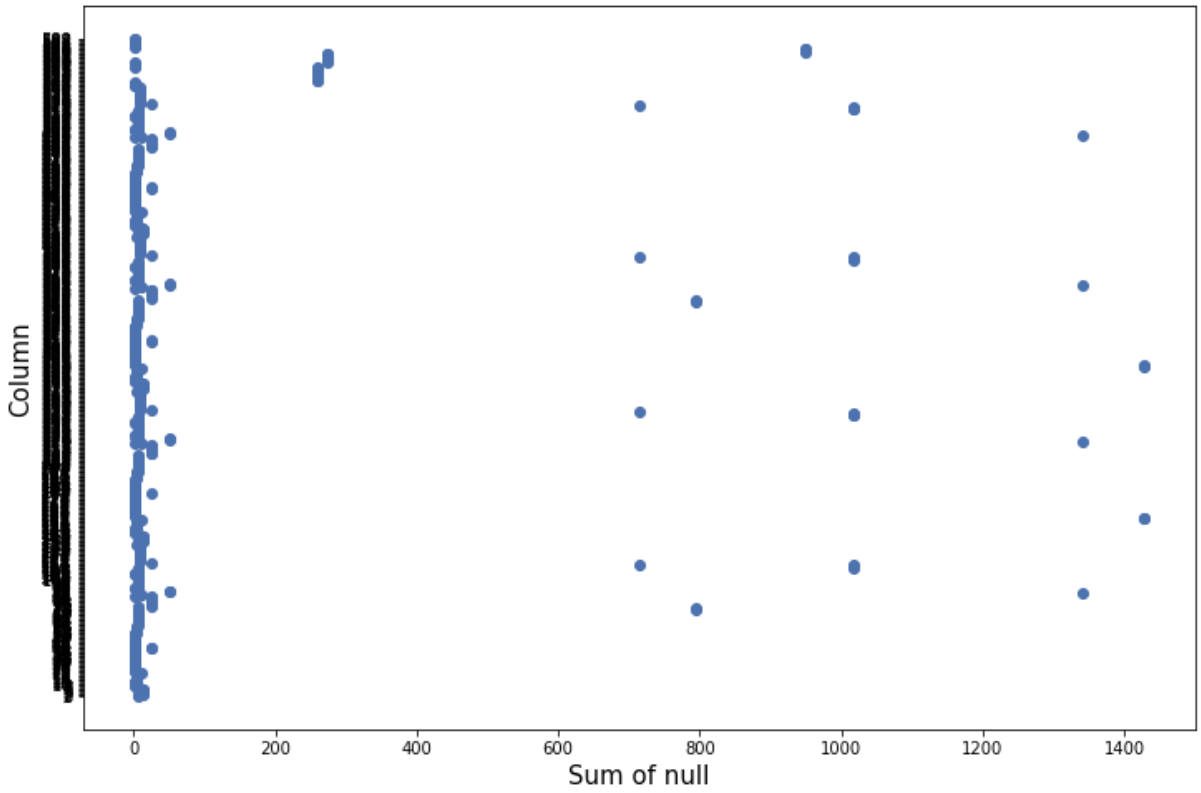
1. 결측치 값 처리

- 각 열의 결측치 값 개수의 유일값 확인

```
df[0].unique()
```

```
array([   6,    7,   14,    9,    2,    3,   10,    0,    1,   24,    4,
        794,   12, 1341,   51, 1018,  715,    8,    5, 1429,  260,  273,
        949])
```

2. 최종 선택된 결측치 처리 방식 설명



전체 데이터의 1% 이상의 결측치를 가진 열의 경우 데이터 수가 적어 대체의 의미가 없다고 판단하여 1% 이하의 결측치만 가진 열만 남기고 1% 이상의 결측치를 갖는 열은 삭제했다. 이후 남은 결측치는 가장 성능이 잘 나온 median 으로 대체했다.

3. 정규화

왜 minmax를 사용하는 것이 좋았는지에 대한 설명 덧붙임 혹은 각 모델별로 알맞는 전처리 방식이 있는지 확인 .. (어려운 부분임)

4. 변수 선택

1) 한 열의 모든 행의 값이 하나의 값으로 일정한 변수 제거

데이터에 따라 값이 바뀌지 않으므로 각 데이터의 정보가 담겨지지 않은 의미없는 변수라고 생각되어 제거 진행하였다. 모델의 성능에는 영향을 미치지 않았으나, 변수 선택 과정에서 훈련 시간에 따른 비용을 줄일 수 있었다.

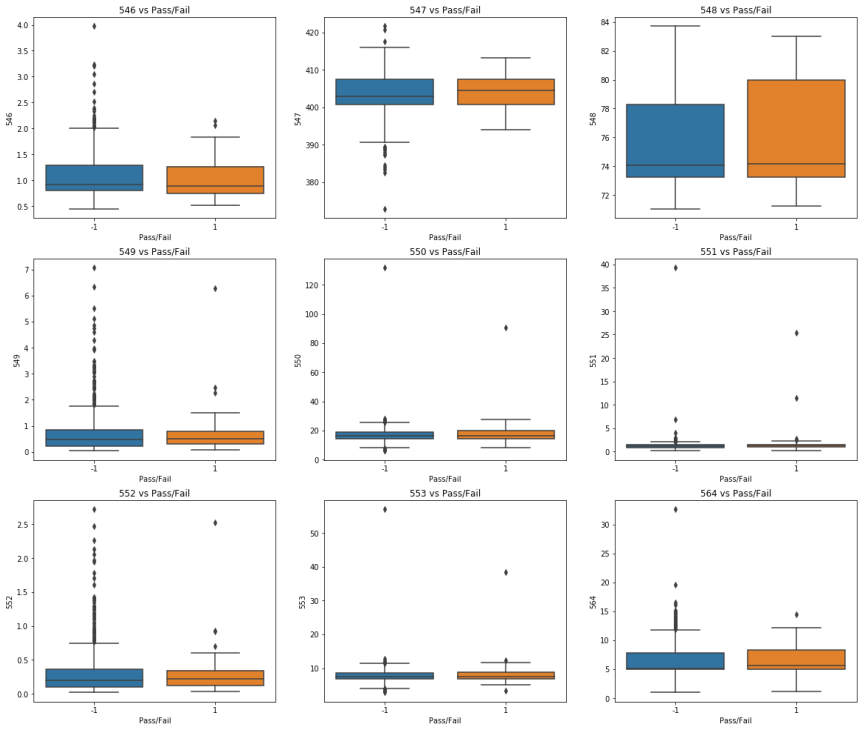
2) 알고리즘 내재 방법을 사용한 변수 선택

다양한 트리 기반 알고리즘을 사용하여 변수 선택을 진행하였다. 트리 기반의 알고리즘은 분기할 때, 분기의 기준으로 삼는 변수를 선택하기 때문에 변수 중요도가 명확하게 파악된다. 그만큼 투명하고 신뢰성 있는 변수 선택 방법이라 생각되어, 트리 기반의 모델인 Random Forest, LGBM를 사용하여 변수 선택을 진행하였다.

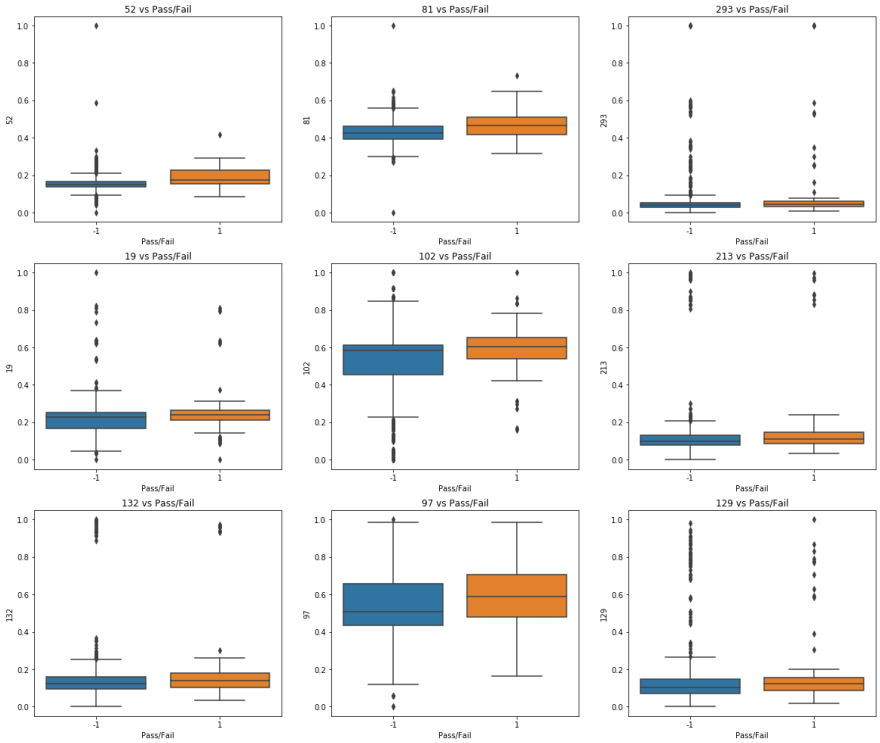
알고리즘 내재 변수 선택 방법	성능
Random Forest	0.748913
LGBM	0.778644

LGBM은 Random Forest 의 향상된 모델이기 때문에 변수 선택이 더 잘 되었다고 할 수 있다. 결과적으로, 높은 성능을 보인 LGBM을 통해 변수 선택 진행하였으며, 선택된 변수는 총 55개이다. 아래 그래프를 통해 선택한 변수의 분포를 확인하였다.

<선택이 안된 변수의 Box-Plot>



<변수 선택된 변수의 box-plot>



선택되지 못한 변수와 선택된 변수의 분포를 비교해보았다. 선택되지 못한 변수는 target 값에 따른 분포가 없음을 알 수 있다. 즉, target 변수에 대한 설명을 잘 못해주는 변수임을 의미한다. 하지만 선택받은 변수의 분포를 살펴보면 선택받지 못한 변수보다 target 값에 따른 분포가 다름을 알 수 있다. 하지만 target 변수와 독립변수와의 최대 상관관계도 0.156008로 높

지 않으므로 선택된 변수 또한 분포가 눈에 띄게 다르지 않음을 확인할 수 있다.

5. 이상치 처리

상관관계에 의한 변수 선택으로 나온 변수들에서 변수 내에 이상치를 탐색해보고, 이상치가 나온 것에 대해서 해당 이상치를 중간값으로 대체하는 방식을 사용하였다.

5. 모델링

본격적인 모델링을 실시하기 전에 기존에 고려했던 머신러닝 모델의 성능을 비교해 보았다. random_state 1~100까지 평균으로 모델 성능을 비교하였으며, 총 9 개의 모델을 활용하여 Baseline model 로 삼았다.

< 동일한 전처리 후 모델별 성능 비교>

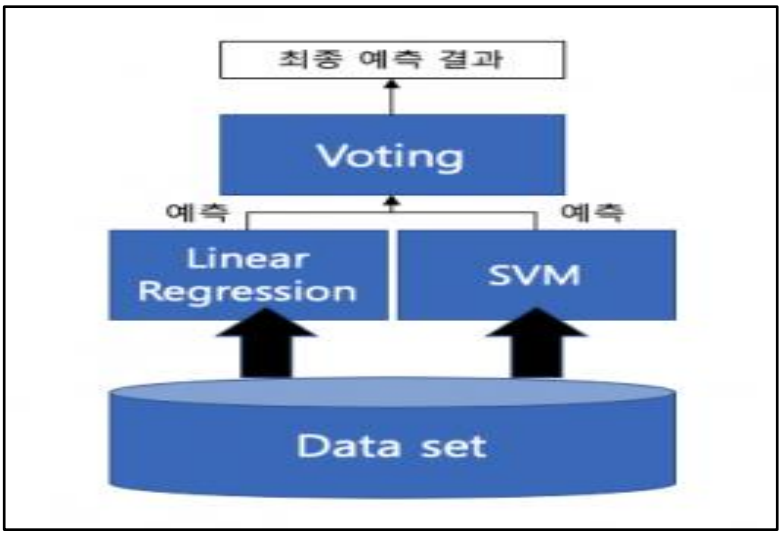
Logistic regression	**Decision tree**	**Light Gradient Boosting(LGBM)**	**XGboost**	**Random forest**	**Linear Discriminant Analysis(LDA)**	**Quadratic Discriminant Analysis(QDA)**	**K-Nearest Neighbor(KNN)**	**Support Vector Machine(SVM)**
0.755292	0.667081	0.793508	0.788470	0.779203	0.743055	0.536464	0.670333	0.689011
BaggingClassifier	**ExtraTreesClassifier**							
0.786089712335445 6	0.767942467089224 8							

앙상블 내용

3. 성능평가: 제안하는 방법에 따른 성능평가 및 방식 (코드)

보팅(Voting): 여러 개의 분류기가 투표를 통해 최종 예측 결과를 결정하는 방식으로 서로 다른 알고리즘을 여러 개 결합하여 사용한다.

- **하드 보팅(Hard Voting):** 다수의 분류기가 예측한 결과값을 최종 결과로 선정한다.
- **소프트 보팅(Soft Voting):** 모든 분류기가 예측한 레이블 값의 결정 확률 평균을 구한 뒤 가장 확률이 높은 레이블 값을 최종 결과로 선정한다.




```
y = semiconductor['Pass/Fail']
X = semiconductor.drop(['Pass/Fail'],axis=1)
```

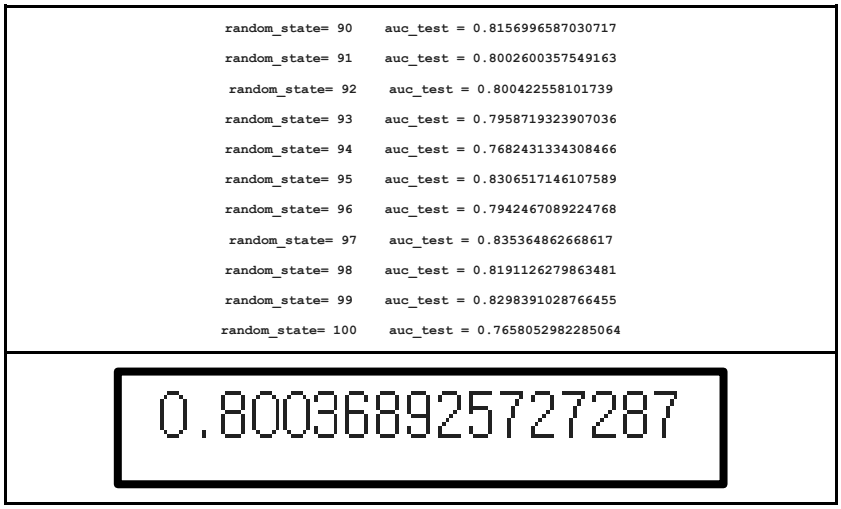
```
from sklearn.model_selection import train_test_split
from lightgbm import LGBMClassifier
from sklearn.metrics import roc_auc_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,
ExtraTreesClassifier, BaggingClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegressionCV, RidgeClassifier
from sklearn.model_selection import cross_validate
# VotingClassifier
from sklearn.ensemble import VotingClassifier

# ensemble 할 model 정의
models = [
    ('rfc', RandomForestClassifier(random_state=13, n_jobs=-1, n_estimators=1000)),
    ('lgbm', LGBMClassifier(n_estimators=2000, num_leaves=32, n_jobs=-
1,boost_from_average=False,C = 100.)),
    ('xgb', XGBClassifier(n_estimators=500, learning_rate = 0.1,C = 100)),
    ('etc',ExtraTreesClassifier(n_estimators=100, random_state=13)),
    ('bc', BaggingClassifier(base_estimator=XGBClassifier(),n_estimators=10,
```

```
random_state=13))
    ]

random_state = []
for j in range(1,101):
    random_state.append(j)
ai = []

for rand in random_state:
  X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=
rand,stratify=y)
  soft_vote  = VotingClassifier(models, voting='soft')
  soft_vote.fit(X_train, y_train)
  pred_proba_1 = soft_vote.predict_proba(X_test)[:,1]
  auc_test = roc_auc_score(y_test, pred_proba_1)
  print('random_state=', rand,'  ' ,'auc_test =', auc_test)
  print('')
  ai.append(float(auc_test))
print(ai)
print(sum(ai)/len(ai))
```

4. 결론: 제안하는 방법에 따른 성능 평가의 결과(최종 ROC_AUC_SCORE)

```
random_state= 1    auc_test = 0.8891597594669267
random_state= 2    auc_test = 0.8155371363562489
random_state= 3    auc_test = 0.7937591418820088
random_state= 4    auc_test = 0.7978222005525759
random_state= 5    auc_test = 0.7294002925402242
random_state= 6    auc_test = 0.7892085161709735
random_state= 7    auc_test = 0.8085486754428733
random_state= 8    auc_test = 0.7633674630261661
random_state= 9    auc_test = 0.7862831139281651
random_state= 10    auc_test = 0.7688932228181374
random_state= 11    auc_test = 0.8470664716398505
random_state= 12    auc_test = 0.8599057370388428
random_state= 13    auc_test = 0.8096863318706322
random_state= 14    auc_test = 0.8091987648301642
random_state= 15    auc_test = 0.8171623598244757
random_state= 16    auc_test = 0.8187875832927027
random_state= 17    auc_test = 0.7867706809686332
random_state= 18    auc_test = 0.8729075247846578
random_state= 19    auc_test = 0.8495043068421908
random_state= 20    auc_test = 0.8350398179749715
random_state= 21    auc_test = 0.7825450999512433
random_state= 22    auc_test = 0.8283764017552413
random_state= 23    auc_test = 0.8153746140094263
random_state= 24    auc_test = 0.7948967983097676
random_state= 25    auc_test = 0.7983097675930441
random_state= 26    auc_test = 0.8098488542174549
random_state= 27    auc_test = 0.7768568178124492
random_state= 28    auc_test = 0.7189988623435721
random_state= 29    auc_test = 0.7095725662278562
random_state= 30    auc_test = 0.8145620022753128
```

```
random_state= 31    auc_test = 0.8735576141719487
random_state= 32    auc_test = 0.7926214854542499
random_state= 33    auc_test = 0.7591418820087762
random_state= 34    auc_test = 0.7906712172923777
random_state= 35    auc_test = 0.7843328457662928
random_state= 36    auc_test = 0.7851454575004062
random_state= 37    auc_test = 0.7219242645863806
random_state= 38    auc_test = 0.8004225581017389
random_state= 39    auc_test = 0.7942467089224768
random_state= 40    auc_test = 0.7861205915813424
random_state= 41    auc_test = 0.7609296278238258
random_state= 42    auc_test = 0.8204128067609296
random_state= 43    auc_test = 0.8347147732813263
random_state= 44    auc_test = 0.8103364212579229
random_state= 45    auc_test = 0.8314643263448723
random_state= 46    auc_test = 0.7887209491305054
random_state= 47    auc_test = 0.8051357061595968
random_state= 48    auc_test = 0.8225255972696246
random_state= 49    auc_test = 0.775881683731513
random_state= 50    auc_test = 0.8215504631886884
random_state= 51    auc_test = 0.8126117341134406
random_state= 52    auc_test = 0.8355273850154397
random_state= 53    auc_test = 0.7004713148057857
random_state= 54    auc_test = 0.7146107589793597
random_state= 55    auc_test = 0.7544287339509183
random_state= 56    auc_test = 0.7554038680318544
random_state= 57    auc_test = 0.8204128067609295
random_state= 58    auc_test = 0.7708434909800097
random_state= 59    auc_test = 0.8132618235007313
random_state= 60    auc_test = 0.7757191613846903
random_state= 61    auc_test = 0.7862831139281652
random_state= 62    auc_test = 0.8122866894197952
random_state= 63    auc_test = 0.8548675442873395
random_state= 64    auc_test = 0.8265886559401917
random_state= 65    auc_test = 0.7830326669917114
random_state= 66    auc_test = 0.7934340971883633
random_state= 67    auc_test = 0.7893710385177962
random_state= 68    auc_test = 0.8130993011539086
random_state= 69    auc_test = 0.8166747927840078
random_state= 70    auc_test = 0.8018852592231431
random_state= 71    auc_test = 0.7588168373151308
random_state= 72    auc_test = 0.9019990248659191
random_state= 73    auc_test = 0.841540711847879
random_state= 74    auc_test = 0.8074110190151146
random_state= 75    auc_test = 0.8113115553388591
random_state= 76    auc_test = 0.8847716561027141
random_state= 77    auc_test = 0.7627173736388754
random_state= 78    auc_test = 0.740939379164635
random_state= 79    auc_test = 0.7697058345522508
random_state= 80    auc_test = 0.7716561027141231
random_state= 81    auc_test = 0.8039980497318381
random_state= 82    auc_test = 0.8543799772468713
random_state= 83    auc_test = 0.8618560052007151
random_state= 84    auc_test = 0.7856330245408744
random_state= 85    auc_test = 0.8451162034779782
random_state= 86    auc_test = 0.8126117341134406
random_state= 87    auc_test = 0.8072484966682919
random_state= 88    auc_test = 0.7636925077198115
random_state= 89    auc_test = 0.7079473427596294
```



5. 소감: 기계학습론 IC-PBL 활동(반도체 불량 여부 예측) 소감

① 문제점 및 해결방법

-오버샘플링 기법인 SMOTE를 처음에 적용하였을 때는 성능이 급격히 향상되는 현상이 발생하였다. 따라서 정확한 방식으로 SMOTE를 적용하기 위하여 시점에 관련된 정확한 조언이 필요했다. 교수님께 자문을 구한 결과 SMOTE를 포함한 다른 오버샘플링 기법 들은 트레인 테스트 스플릿 이후에 적용해야 한다는 것을 알 수 있었다. 조언을 토대로 적절한 시점에 적용하여 문제점을 해결할 수 있었다.

-Feature Scaling 및 Feature Selection과 같은 전처리 방식을 트레인 테스트 스플릿 이전과 이후에 적용함에 따라 성능 결과 값이 바뀌는 것을 볼 수 있었다. 이전과 이후중 어느 것이 맞는 방법인지에 대한 의문이 들었다. 이 문제점 또한 교수님께 자문을 구한 결과 두가지 경우 모두 적용 가능하지만 이번 IC-PBL에서는 성능 최적화를 위하여 전자인 트레인 테스트 스플릿 이전에 적용하기로 한 결과를 얻을 수 있었다. 따라서 이에 맞게 반도체 불량 여부 예측 데이터를 기반으로 전처리를 하였다.

② 느낀점

반도체 불량 여부 예측을 주어진 데이터를 기반으로 진행하면서 많은 시행착오를 겪으며 진행하였다. 특정한 전처리 및 모델 방식을 적용하였을때 팀원들이 생각하기에 더 높은 성능을 가져올것 같았지만 오히려 성능이 하락되는 현상이 자주 발생하였다. 예상치 못한 성능변화 전개에 당황한 적도 많았다. 하지만 문제점이 무엇인지 파악하고 이러한 문제점을 해결하기 위해 모든 팀원들과 함께 노력하였다. 비록 짧지도 길지도 않은 기간 동안 팀원들과 함께 문제를 해결하다 보니 각자 생각하지 못하였던 부분까지 채워줄 수 있어서 좋은 경험이 되었던

것 같다. 6조 모든 팀원들의 높은 참여도 덕분에 반도체 불량 여부 예측을 하는 모델을 구축하는데 좋은 밑거름이 되었다.

6. 참고자료 및 문헌: kaggle data(uci-secom.csv)

https://firework-ham.tistory.com/27

http://hleecaster.com/ml-svm-concept/

https://en.wikipedia.org/wiki/Random_forest

https://ratsgo.github.io/machine%20learning/2017/03/21/LDA/

https://medium.com/datadriveninvestor/decision-trees-lesson-101-f00dad6cba21

https://youtu.be/Vhwz228VrIk

https://joonable.tistory.com/27

https://www.analyticsvidhya.com/blog/2017/03/imbalanced-classification-problem/

https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets

https://wikidocs.net/87189

https://dailyheumsi.tistory.com/111

출처: Rafael Alencar - Resampling strategies for imbalanced datasets

출처: Haibo He, Edwardo A. Garcia - Learning from imbalanced data

출처: Han H., Wang WY., Mao BH. (2005) Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning. In: Huang DS., Zhang XP., Huang GB. (eds) Advances in Intelligent Computing. ICIC 2005. Lecture Notes in Computer Science, Vol 3644. Springer, Berlin, Heidelberg

출처: HE, H., Bai., Garcia, E.A., & Li, C. (2008). ADASYN: Adaptive synthetic sampling approach for imbalanced learning. 2008 IEEE International Joint Conference on Neural

Networks (IEEE World Congress on Computational Intelligence), 1322-1328

https://john-analyst.medium.com/isolation-forest%EB%A5%BC-%ED%86%B5%ED%95%9C-%EC%9D%B4%EC%83%81%ED%83%90%EC%A7%80-%EB%AA%A8%EB%8D%B8-9b10b43eb4ac