Classification

재무제표를 통한 2년 후 파산 여부 예측

4조 김민회 남승지 신예진 오태환 조민주 주일찬

목차

0

서론

조원 소개

남승지

데이터 전처리

모델링

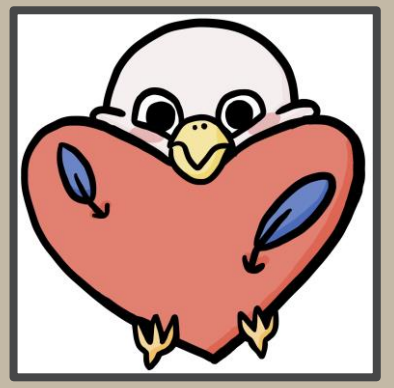
시각화

오태환

데이터 전처리

모델링

Git 튜터링



조민주

데이터 전처리

모델링

김민회

모델링

ppt 제작 및 발표

신예진

변수 생성

모델링

주일찬

데이터 전처리

모델링

1주차 진행 상황

상관관계 분석을 통한 변수 축소
NA Imputation
Outlier 제거
Skewness 조정
Scaling

PCA를 통해
변수 한 단계 더 축소

변수와 파산 여부 관계를
Plot을 그려 간단히 확인

1

EDA 마무리

1. 새 재무제표 변수 생성
2. NA Imputation – MICE / Median / KNN
3. Skewness / Outlier Handling & Scaling

1

EDA 마무리

1. **새 재무제표 변수 생성**
2. NA Imputation – MICE / Median / KNN
3. Skewness / Outlier Handling & Scaling

1. 새 재무제표 변수 생성

조합

6가지

실제 재무비율 중 하나이지만
생략된 비율

파산 여부 예측에 결정적일
것으로 보이는 지표

1. 새 재무제표 변수 생성 - 예시

$Attr1 \times Attr17$

▶

$\frac{Net\ Profit}{Total\ Liabilities} = ROI$

투자자본 수익률

Attr65

$\frac{Attr1}{Attr10}$

▶

$\frac{Net\ Profit}{Equity} = ROE$

자기자본 이익률

Attr67

$Attr21 = \frac{sales(n)}{sales(n-1)}$

▶

0 : 1년 이상 기업

1 : 1년 이내 기업 (신생기업)

Attr70 (Binary)

1

EDA 마무리

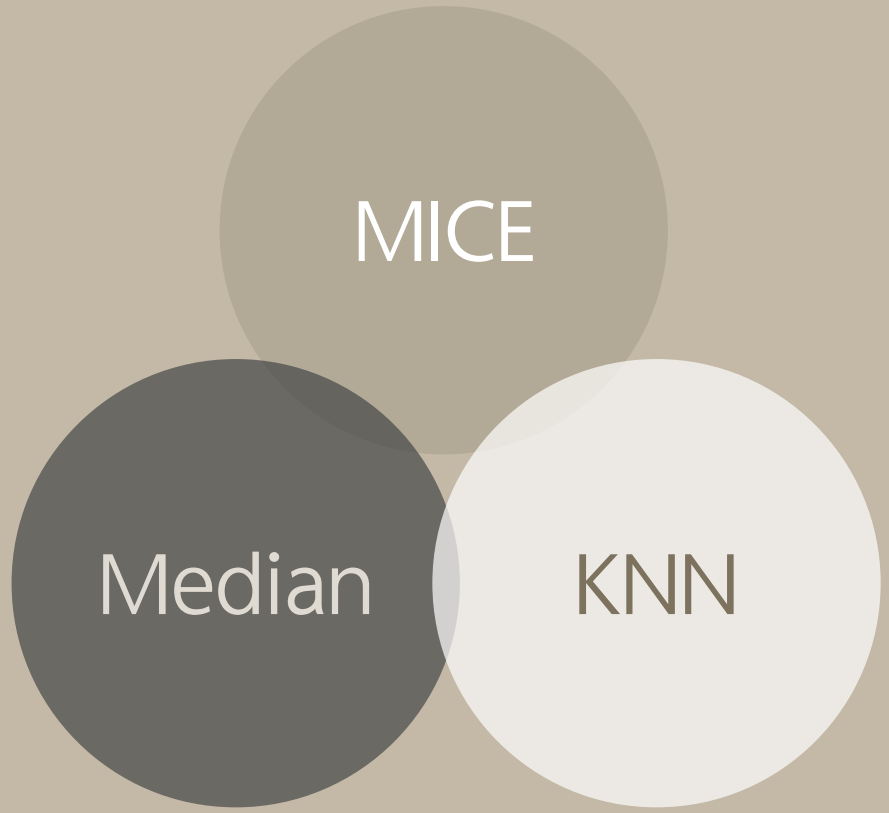
1. 새 재무제표 변수 생성
2. **NA Imputation – MICE / Median / KNN**
3. Skewness / Outlier Handling & Scaling

2. NA Imputation – 3가지 방법

MICE
PMM Method를 활용한 다중 대체
(Predictive Mean Matching)

Median
각 변수(column)의 median으로
NA 값 대체

KNN
K – Nearest Neighbors
근처의 값들을 통해 NA 예측하여 대체



2. NA Imputation – 3가지 방법

- NA 값에 대해 다른 모든 변수를 사용하여 예측한 값으로 대체
- 같은 방식으로 complete dataset을 여러 개 만듦
  default = 5개
- 분석 결과를 하나로 통합 혹은 한가지 결과를 배출
  complete() 함수

MICE

2. NA Imputation – 3가지 방법

- 각 변수 (column) 별로 median 계산
- 각 변수의 NA 값을 median으로 대체
- 같은 column에 속한 경우 동일한 값으로 채워진다

Median

2. NA Imputation – 3가지 방법

- K – Nearest Neighbor 알고리즘
- NA 값 근처 k개 데이터의 평균으로 대체
  default = 10

KNN

2. NA Imputation – Attr27 예시

Before Imputation

0.15979	0.44711	-0.16945
-0.21396	?	-0.96997
0.17054	-2.0477	0.38063
0.015581	?	0.18455
0.21363	5.2386	-0.16335
0.38355	71.08	41.562
0.34235	1.0301	0.052419
0.36072	4.4589	0.27942
-0.33034	-8.8784	5.1096
-0.29598	?	-0.20263

Median

같은 column은 같은 값으로 대체

0.15979	0.44711	-0.16945
-0.21396	1.0089	-0.96997
0.17054	-2.0477	0.38063
0.015581	1.0089	0.18455
0.21363	5.2386	-0.16335
0.38355	71.08	41.562
0.34235	1.0301	0.052419
0.36072	4.4589	0.27942
-0.33034	-8.8784	5.1096
-0.29598	1.0089	-0.20263

KNN

0.15979	0.44711	-0.16945
-0.21396	-0.92099	-0.96997
0.17054	-2.0477	0.38063
0.015581	-0.40553	0.18455
0.21363	5.2386	-0.16335
0.38355	71.08	41.562
0.34235	1.0301	0.052419
0.36072	4.4589	0.27942
-0.33034	-8.8784	5.1096
-0.29598	-23.0264	-0.20263

1

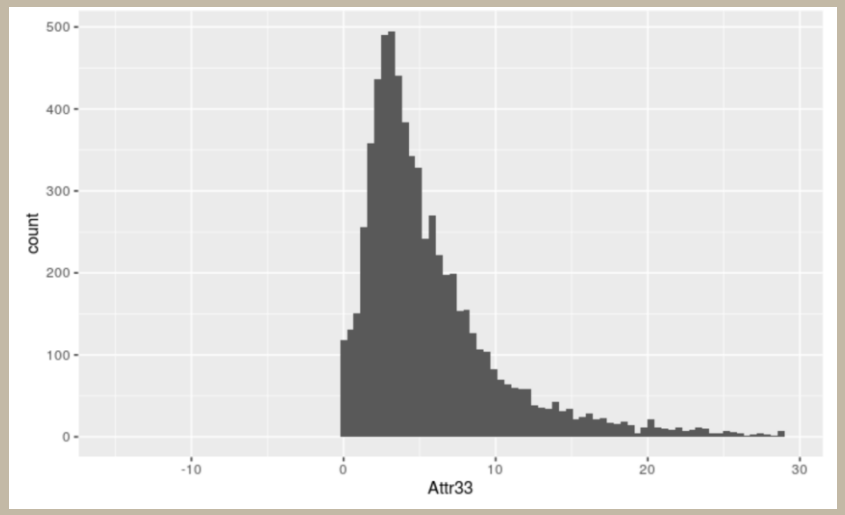
EDA 마무리

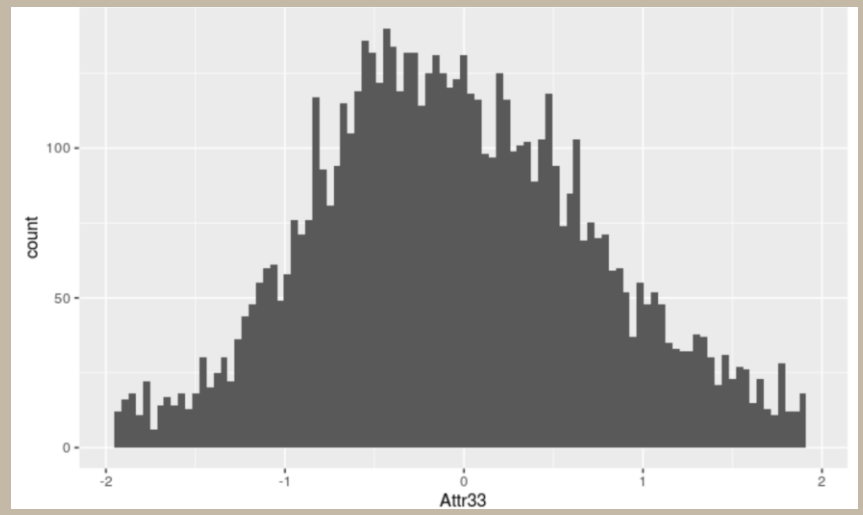
1. 새 재무제표 변수 생성
2. NA Imputation – MICE / Median / KNN
3. **Skewness / Outlier Handling & Scaling**

3. Skewness 조정 – log transformation

그림으로 skewness가 돋보이는 변수 확인

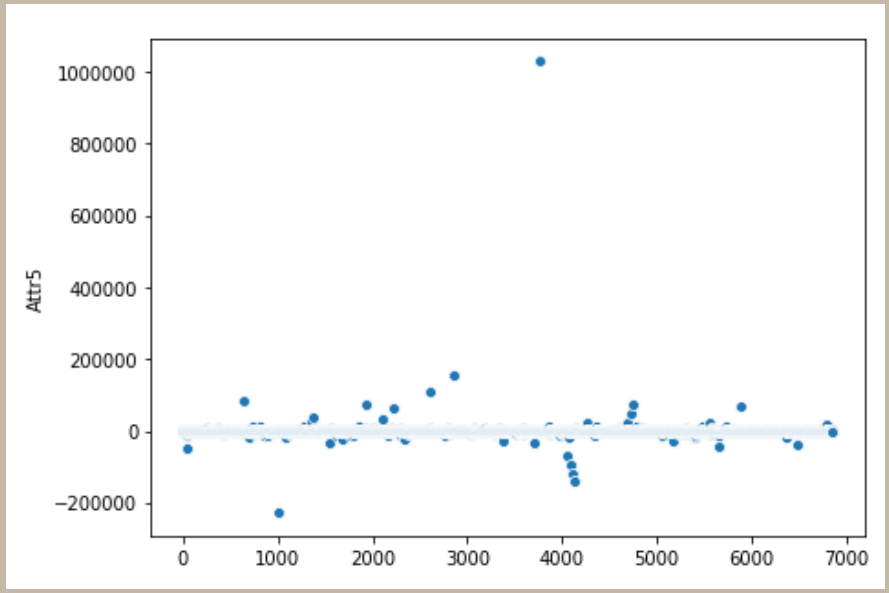
3. Skewness 조정 – log transformation

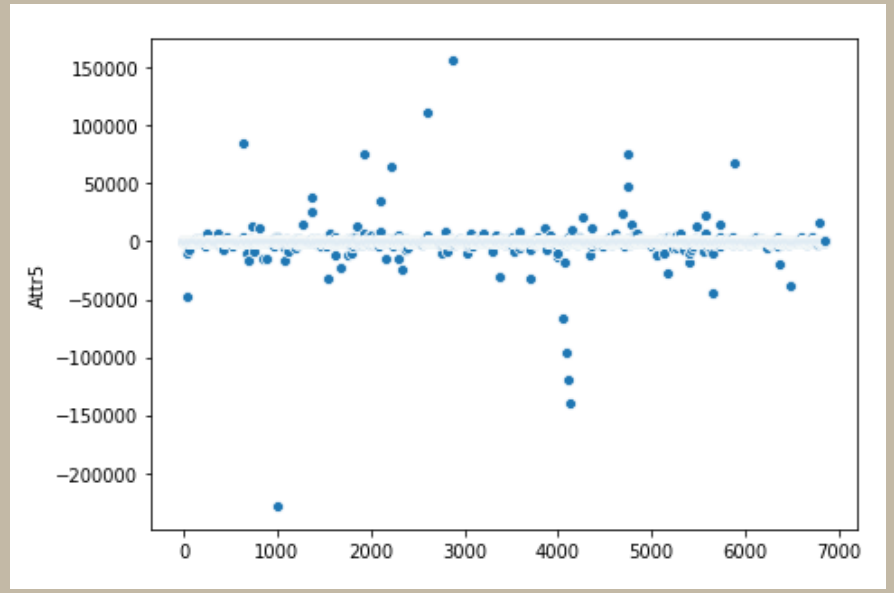




3. Outlier 제거

- 기준 : 1주차와 동일하게 각 변수에서 큰/작은 **3가지** 값들 (총 6개)
- 그 후, **3개 이상의 변수**에서 outlier로 관측되는 **회사(row) 삭제**





3. Scaling : Standard Scaler – knn 예시

```
In [58]: knnscale = dataknn.drop(["Attr70","class"], axis =1)
         knnscale = StandardScaler().fit_transform(knnscale)
         knnscale =pd.DataFrame(knnscale)
         knnscale.columns =dataknn.columns[:(dataknn.shape[1]-2)]
         dataknn = pd.DataFrame(pd.concat([knnscale,dataknn[['Attr70','class']]], axis =1))
         dataknn = dataknn.reset_index()

In [59]: dataknn.shape

Out[59]: (6805, 45)

In [62]: dataknn.drop("index", axis = 1, inplace = True)

In [63]: dataknn.head()
```

Out[63]:

	Attr1	Attr3	Attr4	Attr5	Attr6	Attr9	Attr10	Attr12	Attr13	Attr15	...	Attr57
0	-0.352842	-0.177767	-0.123483	0.009524	0.009448	-0.456467	0.301698	-0.031183	-0.001930	0.056725	...	-0.056065
1	2.918455	1.861965	1.097655	0.008598	-0.411517	-0.390186	0.725650	19.373965	0.416410	-0.041950	...	0.214041
2	-2.404133	0.017073	-0.330491	0.009558	0.019555	0.073659	0.099788	-0.035084	-0.172241	-0.049363	...	-0.378704
3	0.357261	1.028999	0.890857	0.025911	0.114729	-0.380344	0.476698	-0.026925	0.065757	-0.037176	...	0.017436
4	0.520052	-0.379922	-0.612011	-0.005114	0.019555	0.466834	0.109125	-0.029113	0.040011	-0.026674	...	0.073563

Scaling 후 correlation이 0.8보다 높은 변수들 추가로 제거

2

모델링

1. 각 imputed data에 모델들 적용 – CV & Grid Search

   (LDA, QDA, SVM, Decision Tree, Random Forest, Light GBM, Ada Boost, XG Boost)

2. 최적 모델 설정

   - F1 Score (+AUC)
   - Precision Recall Curve

1. 각각의 data에 모델 적용 – Cross Validation & Grid Search

교차 검증
StratifiedKFold

```
stf = StratifiedKFold(n_splits = 10, shuffle = True, random_state = 730)
```

LDA

```
lda = LinearDiscriminantAnalysis().fit(X_train, y_train)
```

```
score = cross_val_score(lda, X_train, y_train, cv = stf, scoring = 'f1_micro')
```

```
score.mean()
```

```
0.9553950334640315
```

```
score = cross_val_score(lda, X_train, y_train, scoring = "roc_auc", cv = stf)
```

```
score.mean()
```

```
0.7610154525386312
```

```
lda.fit(X_train,y_train)
pred_y = lda.predict(X_test)
```

```
print("F1 : %.3f" % f1_score(y_test, pred_y, average = 'micro'))
print("ROC AUC : %.3f" % roc_auc_score(y_test, pred_y))
```

```
F1 : 0.951
ROC AUC : 0.590
```

하이퍼 파라미터 튜닝
GridSearchCV

3-3) Grid Search

```
params = {'max_depth': [10, 15, 20, 25, 30],
          'min_child_samples': [20, 40, 60, 80, 100],
          'subsample': [0.8, 1]}

grid = GridSearchCV(lgbm, param_grid=params)
grid.fit(x_train, y_train, early_stopping_rounds=100, eval_metric='F1',
         eval_set=[(x_train, y_train), (x_test, y_test)])
```

```
[1]     valid_0's binary_logloss: 0.170336      valid_1's binary_logloss: 0.177498
Training until validation scores don't improve for 100 rounds
[2]     valid_0's binary_logloss: 0.154747      valid_1's binary_logloss: 0.168072
[3]     valid_0's binary_logloss: 0.142428      valid_1's binary_logloss: 0.161148
[4]     valid_0's binary_logloss: 0.13185       valid_1's binary_logloss: 0.156149
[5]     valid_0's binary_logloss: 0.123074      valid_1's binary_logloss: 0.151038
[6]     valid_0's binary_logloss: 0.116205      valid_1's binary_logloss: 0.147868
[7]     valid_0's binary_logloss: 0.109847      valid_1's binary_logloss: 0.144214
```

- parameter가 여러 개인 모델에 GridSearchCV() 함수 사용
- 단일 parameter인 모델에서는 교차 검증만 진행

1. 각각의 data에 모델 적용 – Cross Validation & Grid Search

예시 – SMOTE() + XGBoost Classifier (ROC AUC)

```
X_resampled, y_resampled = SMOTE().fit_sample(X_train,list(Y_train))
```

```
xgb_model = XGBClassifier()
```

```
grid_search2 = GridSearchCV(xgb_model, param_grid, scoring="roc_auc", cv=re_stf)
grid_result2 = grid_search2.fit(X_resampled, y_resampled)
```

```
print("Best: %f using %s" % (grid_result2.best_score_, grid_result2.best_params_))
means = grid_result2.cv_results_['mean_test_score']
stds = grid_result2.cv_results_['std_test_score']
params = grid_result2.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
# plot results
scores = np.array(means).reshape(len(learning_rate), len(n_estimators))
for i, value in enumerate(learning_rate):
    plt.plot(n_estimators, scores[i], label='learning_rate: ' + str(value))
plt.legend()
plt.xlabel('n_estimators')
plt.ylabel('ROC Score')
```

```
Best: 0.998097 using {'learning_rate': 0.2, 'n_estimators': 600}
0.900608 (0.008960) with: {'learning_rate': 0.01, 'n_estimators': 200}
0.920009 (0.007901) with: {'learning_rate': 0.01, 'n_estimators': 300}
0.931636 (0.007048) with: {'learning_rate': 0.01, 'n_estimators': 400}
0.940535 (0.006306) with: {'learning_rate': 0.01, 'n_estimators': 500}
0.947994 (0.005671) with: {'learning_rate': 0.01, 'n_estimators': 600}
```

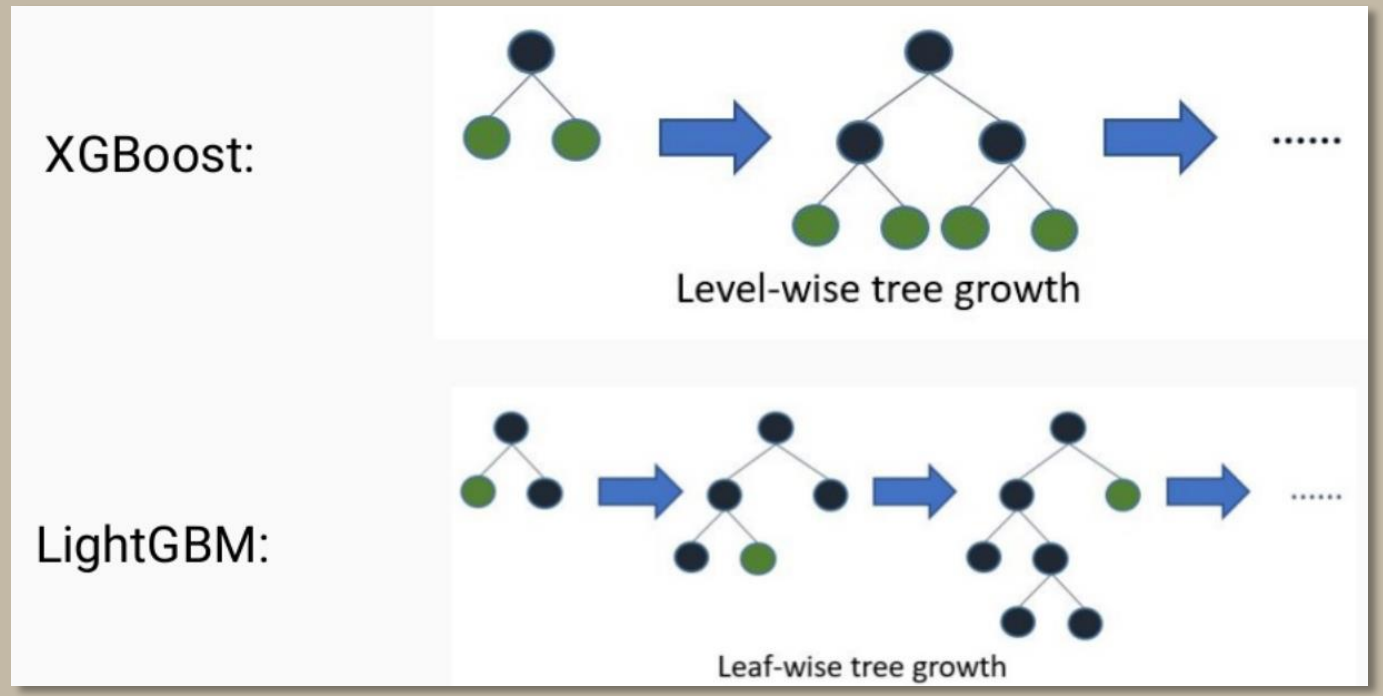
- **SMOTE(): 불균형한 data를 보정**
  oversampling 중 하나로, 수가 적은 target을 채워 넣어준다 (여기선 “class”==1 에 해당)

- n_estimators = [200, 300, 400, 500, 600]
- learning_rate = [0.01, 0.05, 0.1, 0.15, 0.2]

- 위의 변수들을 달리하여 총 25가지 모델에 대한 성능을 ROC Score로 나타낸다

1. 각각의 data에 모델 적용 – Cross Validation & Grid Search

Light GBM 이란?



- 기존의 부스팅 알고리즘의 확장 방식은 수평 방향

- 변수 별 가능한 모든 분할에 대해 평가 진행
  → 많은 시간이 소요

- Light GBM
  : **균형을 맞추지 않고, 리프를 분할해 나간다**

- **메모리 / 시간 소모가 적지만 결과는 비슷하다**

2

모델링

1. 각 imputed data에 모델들 적용

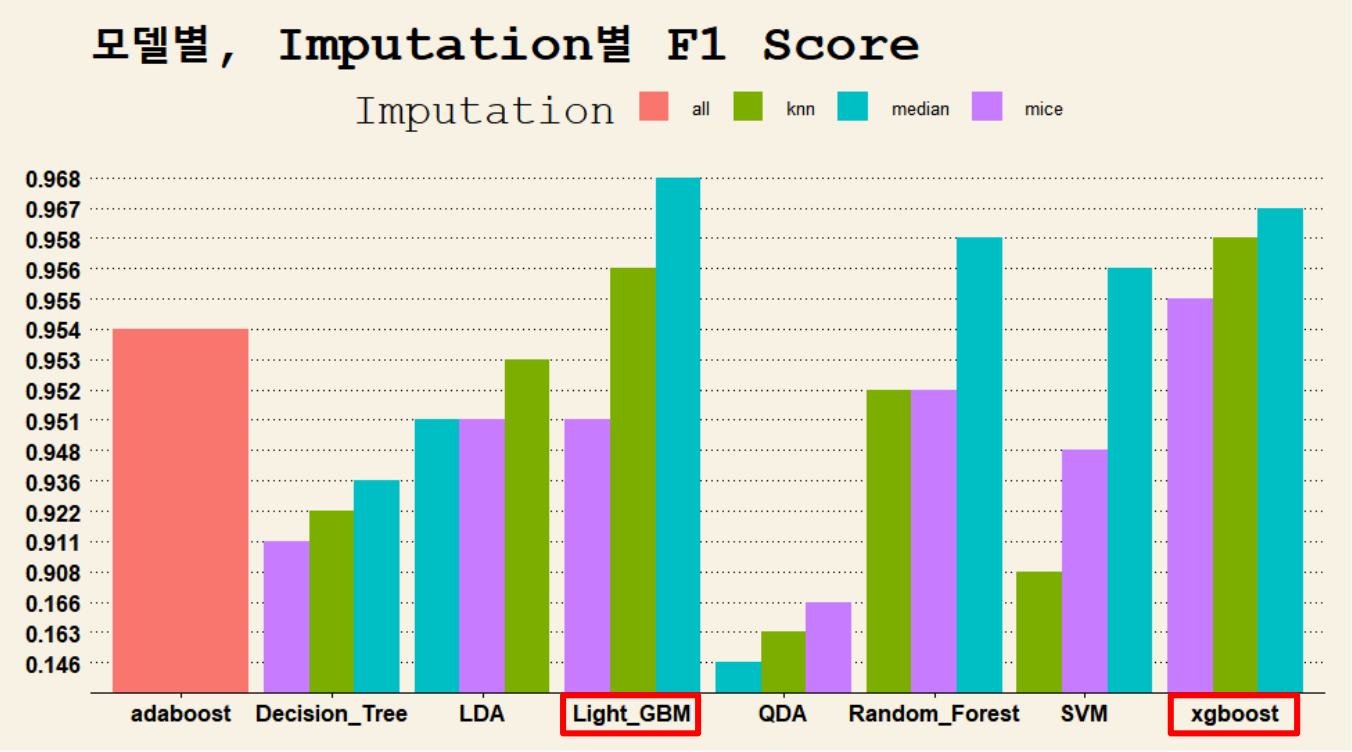
   (LDA, QDA, SVM, Decision Tree, Random Forest, Light GBM, Ada Boost, XG Boost)

2. **최적 모델 설정**

   - F1 Score (+AUC)
   - Precision Recall Curve

2. 최적 모델 설정

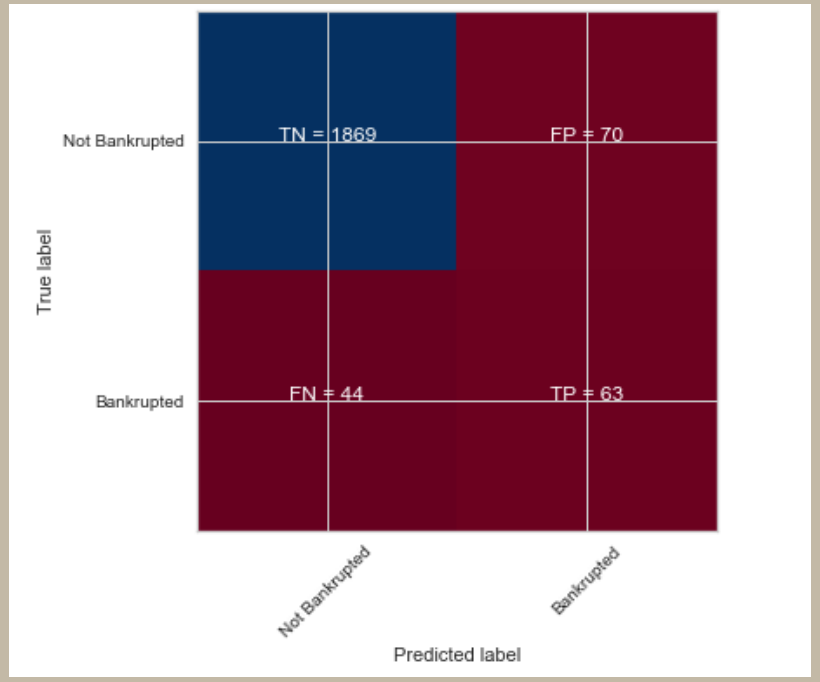
LDA, QDA, SVM, DecisionTree, RandomForest, LightGBM, AdaBoost, XGBoost



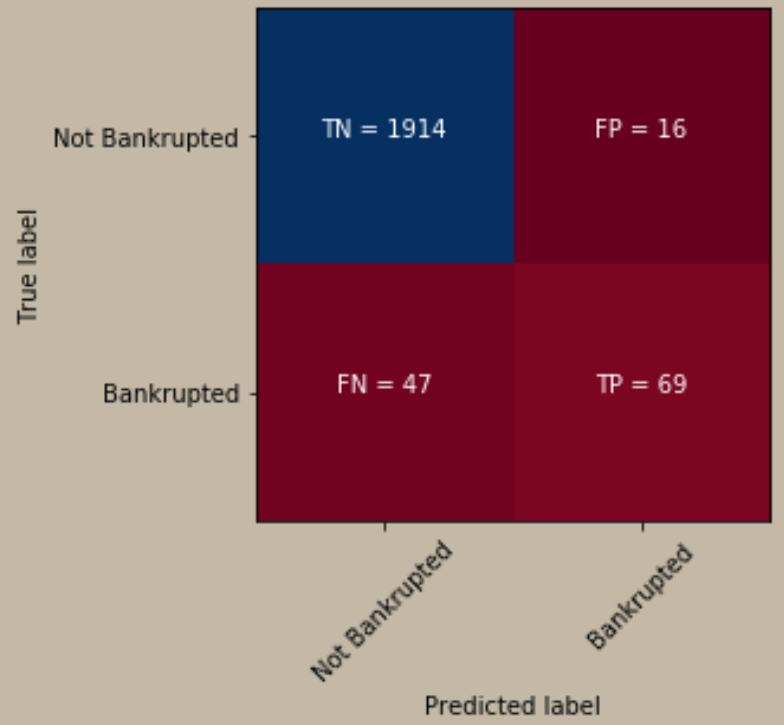
- 각 모델과 Imputation 방법 별 최적의 F1 Score를 시각화
- **Median** Imputation을 활용한 **LightGBM** 모델의 **F1 = 0.9676**
- **Median** Imputation을 활용한 **XG Boost** 모델의 **F1 = 0.967**
  learning_rate = 0.2 / n_est = 500

⇒ **XG Boost 모델 선정 !**

해석 – Confusion Matrix



Light GBM

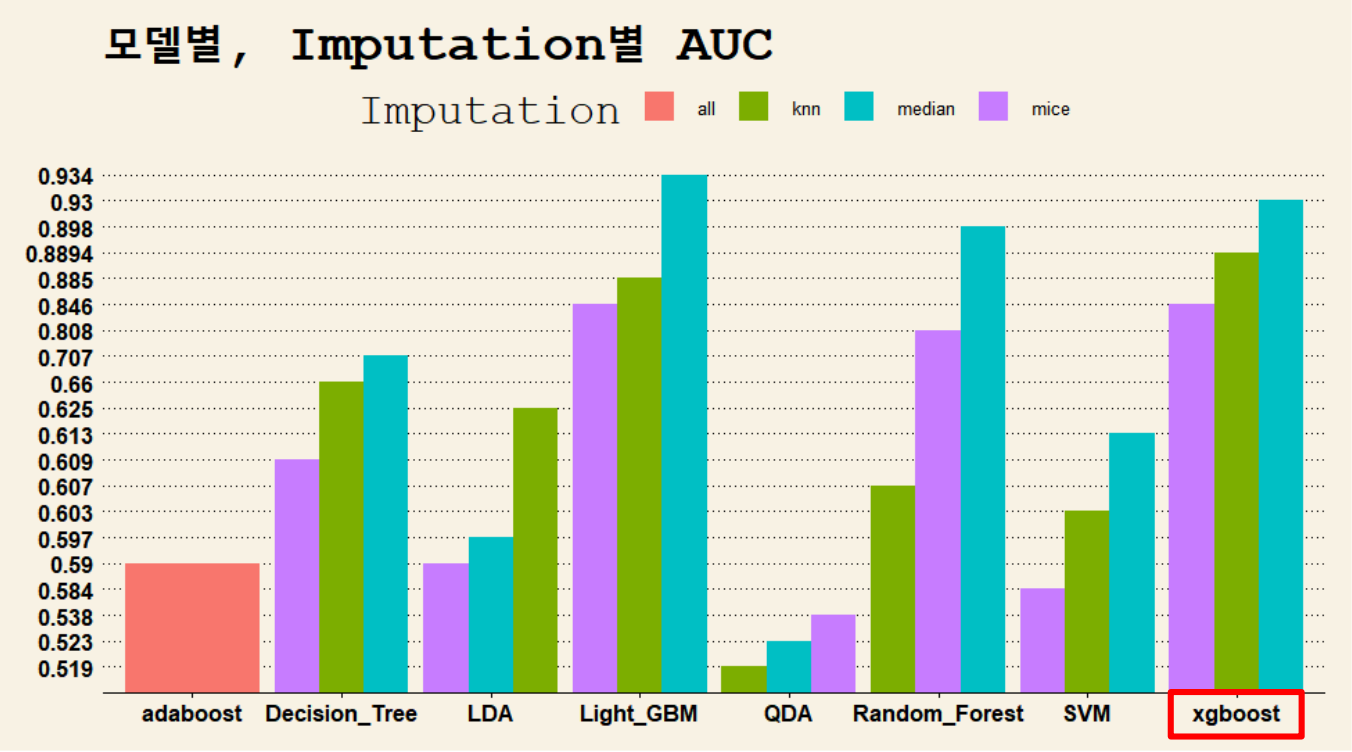


XG Boost

갖고 있는 train set을 Train-Test Split 했을 때,
Model Evaluation 비교

2. 최적 모델 설정

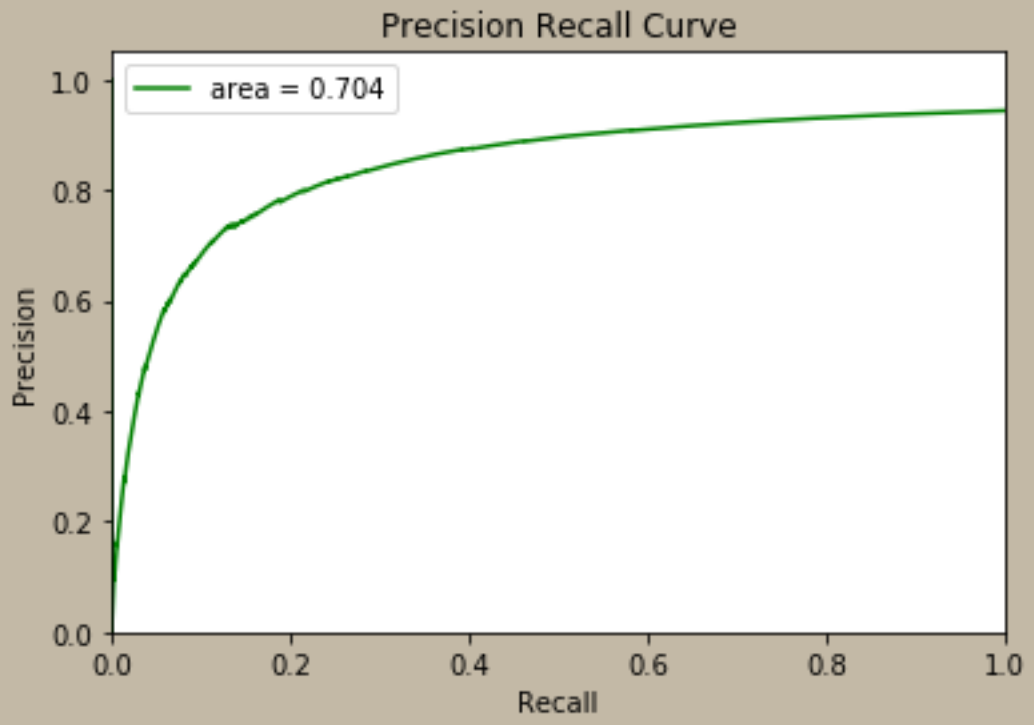
LDA, QDA, SVM, DecisionTree, RandomForest, LightGBM, AdaBoost, XGBoost



- AUC 시각화
- Median Imputation을 활용한 XG Boost 모델의 AUC = 0.93

2. 최적 모델 설정

LDA, QDA, SVM, DecisionTree, RandomForest, LightGBM, AdaBoost, XGBoost

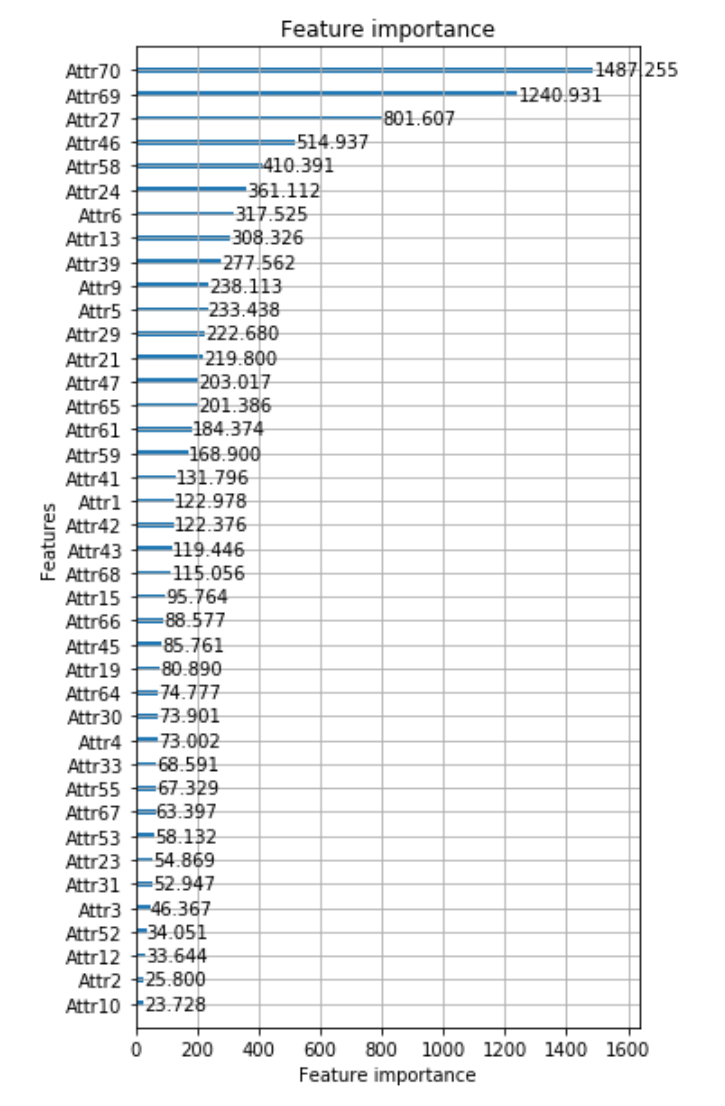


- 참고: Precision Recall Curve

3

해석

해석 – Feature Importance



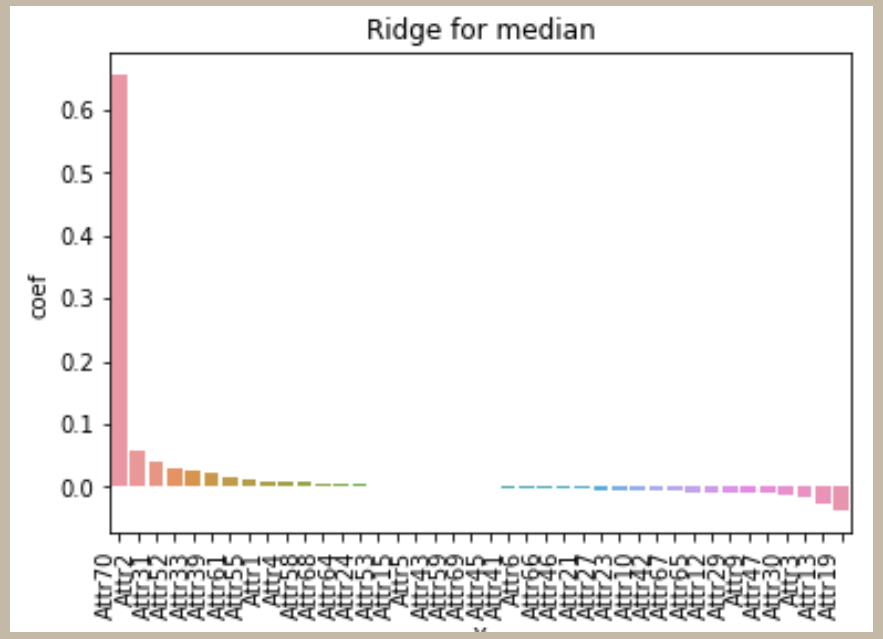
- Feature Importance 그래프 – gain 방식

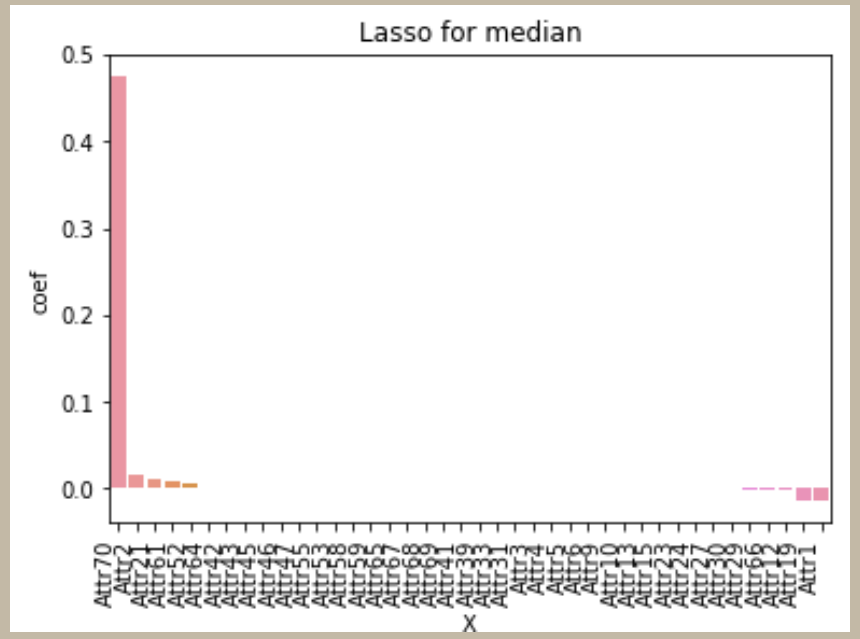
- Split 방식 : target 나누는 기준으로 얼마나 쓰였는가
- **Gain 방식** : 각 변수가 target을 얼마나 잘 나눴는가

- **Attr70** : 신생기업 여부
  “시작이 반이다!”
  1년만 버티면 6년도 버틴다!

- **Attr69** : 유동부채/유동자산 = **유동 비율** (안정성 지표)
  단기 부채에 대한 지급 능력
  능력 안 되면 파산 → 당연한 것

해석 – Ridge & Lasso





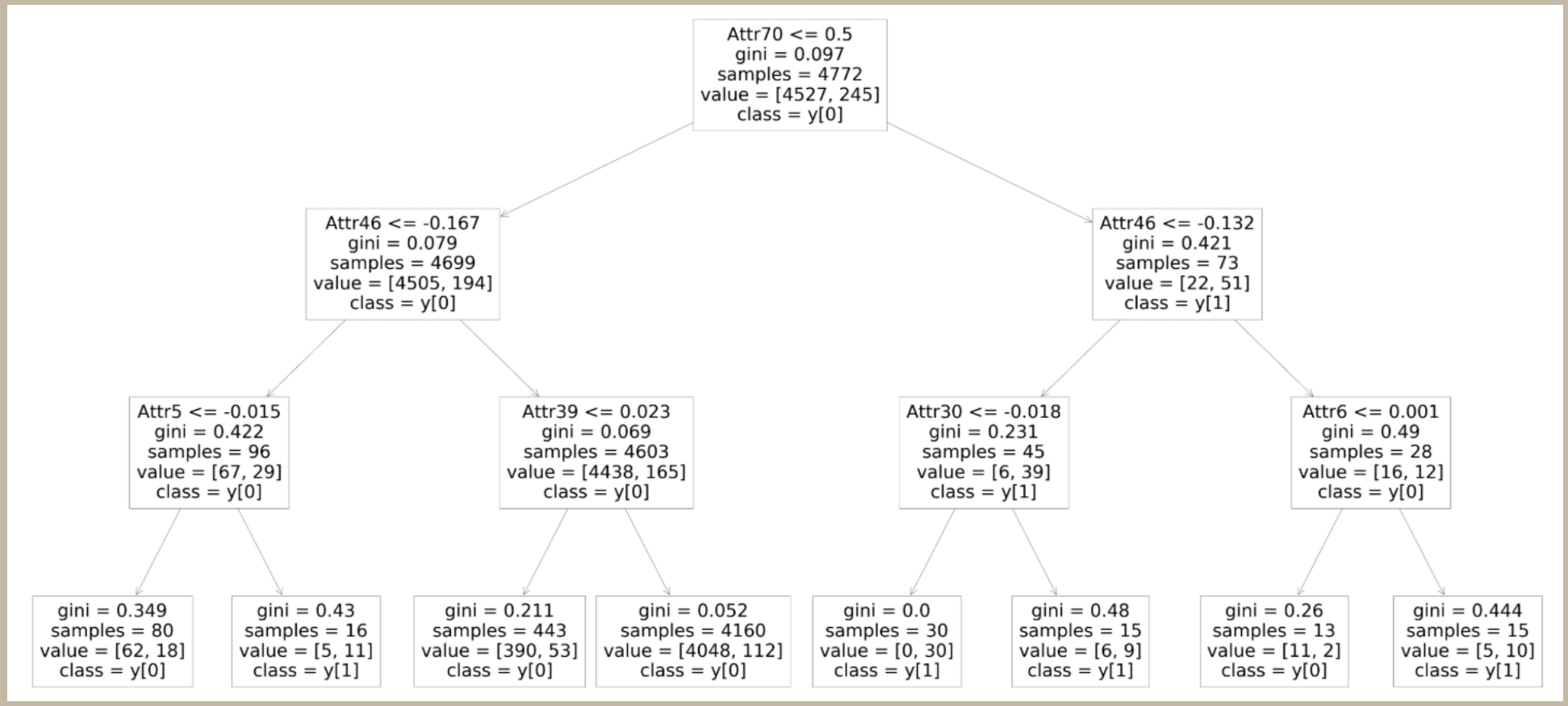
Attr70 : 신생기업 여부 – 압도적

Attr2 : 부채비율

Attr19, 52 : 매출 총 이익률 & 단기부채/매출원가 – 기업의 판매 상품에 대한 예측 가능

Attr21, 1, 62 : 기업의 자본 운용 효율성

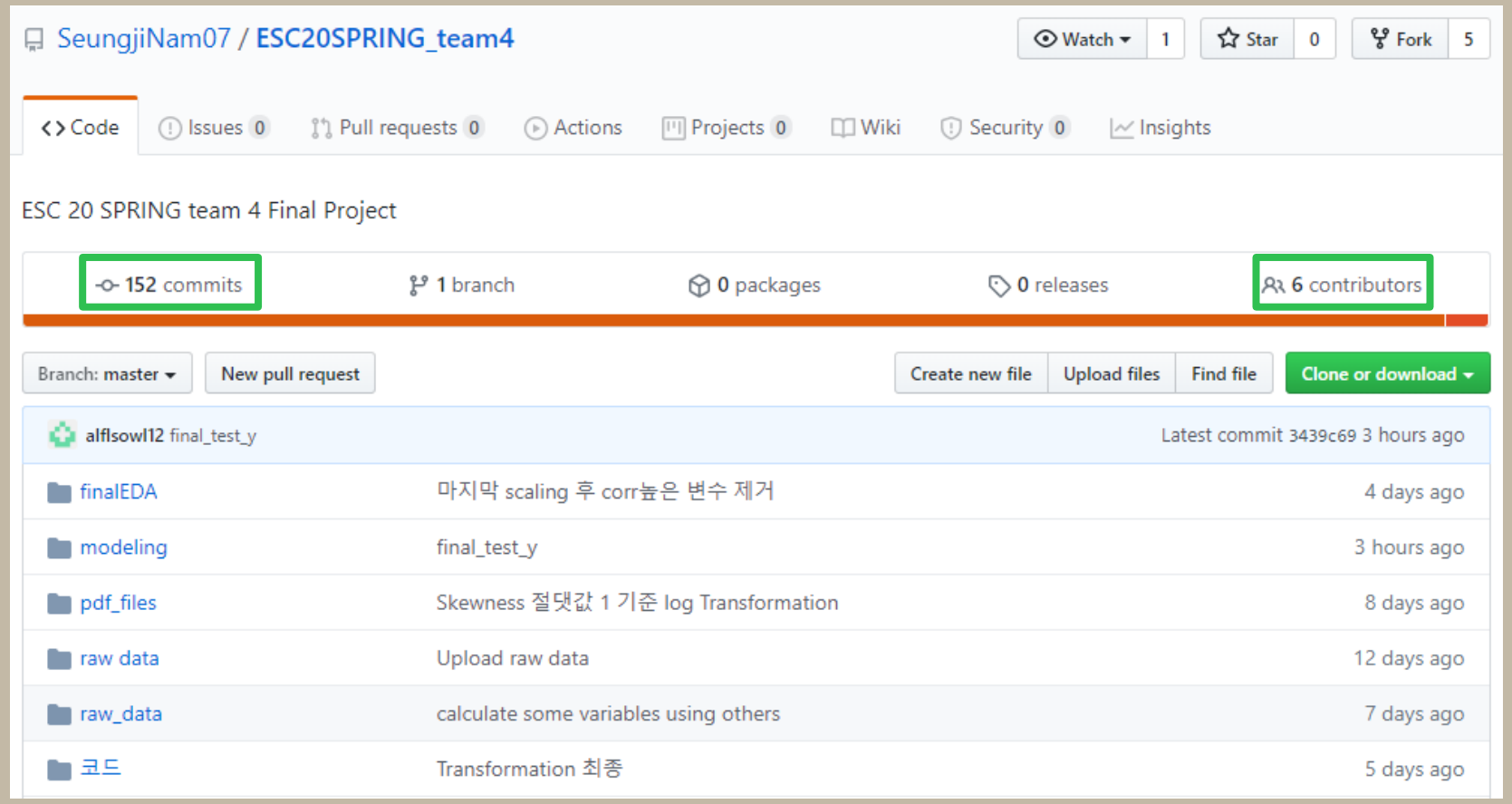
해석



Attr46 : (유동자산–재고)/단기부채

Attr5 : Gross Burn Rate

Git Hub Link



https://github.com/SeungjiNam07/ESC20SPRING_team4

Overall Interpretation

1. 생각보다 X70 (0: 1년이상 기업, 1: 신생기업) 변수의 영향이 크게 작용했다.

2. 재무비율 하나로 파산에 대해 평가할 수 있지 않다. 여러 지표를 통해 전반적으로 평가하는 것이 매우 중요하다.

3. 통상적으로 재무비율을 수익성 지표, 안정성 지표, 현금 상환능력지표, 활동성 지표로 grouping 하는데 중요하다고 여겼던 변수들로부터 골고루 나왔다.

참고

https://www.slideshare.net/GabrielCyprianoSaca/xgboost-lightgbm

https://aldente0630.github.io/data-science/2018/06/29/highly-efficient-gbdt.html

감사합니다 :)

THANK YOU