스타트업 성장 및 투자

5조 박동찬, 박채윤, 이가람

1. 서론 (Introduction)

스타트업의 기업 가치는 투자 판단과 전략 수립에 있어 매우 중요한 요소이다. 본 프로젝트는 머신러닝을 활용해 스타트업의 기업 가치를 예측하는 것을 목표로 하며, Kaggle에서 제공하는 Startup Growth and Investment 데이터셋을 기반으로 한다. 산업 분야, 펀딩 횟수, 투자 금액 등 다양한 변수를 분석함으로써 기업 가치(Valuation)에 영향을 미치는 주요 요인을 도출하고, 이를 바탕으로 예측 성능이 우수한 모델을 구축하고자 한다.

2. 데이터셋 설명 (Dataset Description)

2.1 데이터 개요

이 데이터는 Kaggle에서 수집된 것으로, 총 5,000개의 스타트업 정보를 포함하고 있다. 주요 변수로는 산업군, 국가, 펀딩 라운드 수, 투자 금액, 기업 가치(타겟 변수), 성장률 등이 있으며, 회귀(regression) 방식으로 기업 가치를 예측하는 것이 목적이다.

2.2 Feature 설명

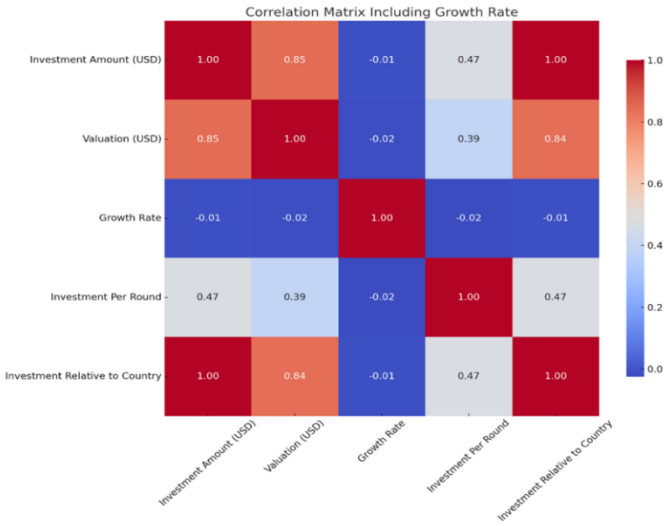
Column name	Data Type	Description
Startup Name	Categorical	스타트업의 이름
Industry	Categorical	산업 부문
Funding Rounds	Numerical	스타트업이 받은 펀딩 라운드 수
Investment Amout (USD)	Numerical	총 투자 금액
Valuation (USD)	Numerical	USD로 추정된 회사 가치
Number of Investors	Numerical	스타트업을 지원하는 투자자 총 수
Country	Categorical	스타트업이 위치한 국가
Year Founded	Numerical	스타트업이 설립된 연도
Growth Rate (%)	Numerical	연간 성장률 백분율

2.3 파생 Feature (Feature Engineering)

Investment Per Round	투자 금액 ÷ 펀딩 라운드 수
Funding Efficiency Ratio	투자 금액 ÷ 투자자 수
Funding Rounds Per Year	펀딩 횟수 ÷ (2025 - 설립 연도 + 1)
Growth Rate Relative to Industry	성장률 ÷ 해당 산업군의 중간 성장률
Investment Relative to Country	투자 금액 ÷ 해당 국가의 중간 투자 금액

2.4 데이터 분포 및 EDA

2.4.1 상관관계계수 히트맵



수치형 변수들 간의 상관관계를 시각화한 결과, Investment Amount와 Valuation 사이에서 높은 양의 상관관계가 확인되었다. 이 외에도 Investment Per Round, Funding Efficiency Ratio 등 주요 파생 변수들 또한 기업 가치와 일정 수준의 상관성을 보였다.

3. 실험 방법 (Methodology)

3.1 적용한 실험 방식

실험 조건	설명
A	Target 로그 변환 유무
B	범주형 인코딩(Label Encoding, One-Hot encoding)
C	이상치 처리 유무
D	Feature Engineering 적용 유무

Startup Growth & Investment Data에서 기업 가치(Valuation)을 target으로 예측하도록 한다. Startup Name은 Valuation과 관계가 없음을 미리 알 수 있기에 해당 feature는 제외했다. 위의 표와 같이 4가지의 실험 조건을 세팅했다. A 실험 조건은 Target인 Valuation과 Investment Amount에 로그 변환을 통해 $10^9 \sim 10^{10}$ 단위이던 값을 스케일링하여 분포 차이를 줄여준다. B 실험 조건은 Categorical Data에 대해 One-hot encoding과 label encoding을 각각 적용한다. C 실험 조건은 Valuation에서 상위 1% 값을 제거하여 분포 차이를 줄여줄 수 있다. D 실험 조건은 원본 데이터의 feature만으로는 복잡한 관계를 파악하기 어려울 수 있기에 도메인 지식 기반의 파생 feature를 추가로 학습시켜 모델이 중요한 패턴을 잘 학습하도록 한다. 자세한 파생 feature는 위 데이터셋 설명에 있다. (예: 투자 라운드당 평균 투자 금액, 기업 설립 후 경과 기간 등)

이 네 가지 실험 조건에 따라 어떻게 바뀌는지 알기 위해 표 4.1과 같이 Ablation study를 적용하여 실험 조건을 차례로 하나씩 제거하면서 아래 모델에 학습시켰다. 이 결과에서 큰 영향을 보여준 A와 C 실험 조건과 성능 상위 5개의 모델에 대하여 더 넓은 하이퍼파라미터 탐색과 Grid Search를 통해 실험했다. 그중 상위 3개 모델에 대해 Meta-ensemble하여 성능 개선이 이루어지는지 실험했다.

3.2 실험한 모델 리스트

- 선형 모델: Linear Regression, Ridge, Lasso, ElasticNet
- 서포트 벡터 머신: SVR
- 단일 트리: Decision Tree
- Bagging 기반 앙상블: Random Forest
- Boosting 기반 앙상블: Gradient Boosting, AdaBoost, XGBoost, CatBoost
- Meta-Ensemble: Voting Regressor, Stacking Regressor

3.3 각 모델에 적용한 하이퍼파라미터 탐색 그리드 표

3.3.1 첫 번째 실험(네 가지 실험 조건에 대한 Ablation study)

모델	하이퍼파라미터	탐색 값
LinearRegression	(none)	—
Lasso	lasso__alpha	[0.1, 1, 10]
	max_iter	50000
ElasticNet	elasticnet__alpha	[0.1, 1]
	elasticnet__l1_ratio	[0.1, 0.5, 0.9]
	max_iter	50000
Ridge	ridge__alpha	[0.1, 1, 10]
DecisionTreeRegressor	max_depth	[3, 5, 7, 10, None]
	min_samples_split	[2, 5, 10]
RandomForestRegressor	n_estimators	[30, 50, 100, 200]
	max_depth	[5, 10, None]
GradientBoostingRegressor	n_estimators	[30, 50, 100, 200]
	learning_rate	[0.01, 0.1]
	max_depth	[3, 5]

AdaBoostRegressor	n_estimators	[30, 50, 100, 200]
	learning_rate	[0.01, 0.1]
SVR	svr__C	[0.01, 0.1, 1, 10]
	svr__kernel	["linear", "rbf"]
XGBRegressor	n_estimators	[50, 100]
	learning_rate	[0.01, 0.1]
	max_depth	[3, 5]
CatBoostRegressor	iterations	[100, 200]
	depth	[3, 5]
	learning_rate	[0.01, 0.1]
KNeighborsRegressor	kneighborsregressor__n_neighbors	[3, 5, 7, 9]

3.3.2 두 번째 실험(상위 모델에 대한 추가 탐색)

모델	**하이퍼파라미터**	**탐색 값**
LinearRegression	fit_intercept	[True, False]
	positive	[False, True]
Ridge	alpha	[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
	fit_intercept	[True, False]
	solver	["auto"]
Lasso	alpha	[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
	fit_intercept	[True, False]
	selection	["cyclic", "random"]
ElasticNet	alpha	[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
	l1_ratio	[0.1, 0.3, 0.5, 0.7, 0.9]
	fit_intercept	[True, False]
RandomForestRegressor	n_estimators	[100, 200, 300, 500]
	max_depth	[None, 5, 10, 20]
	min_samples_split	[2, 5, 10]
	min_samples_leaf	[1, 2, 4]
	max_features	["auto", "sqrt", "log2"]
GradientBoostingRegressor	n_estimators	[30, 50, 60, 70, 80, 90, 100, 110, 120]
	learning_rate	[0.04, 0.05, 0.06, 0.07, 0.08, 0.1]
	max_depth	[2, 3, 4, 5, 6]

3.4 측정 방법에 대한 설명

Train set과 test set을 8:2로 나누고 랜덤 시드를 42로 고정하였으며 5-fold CV를 진행하였다. 모델의 성능을 다각도에서 평가하기 위해 MAE(Mean Ablsolute Error), MSE(Mean Squared Error), R^2(Coefficient of Determination)를 측정했다. Numerical feature는 Standard Scaler로 표준화하여 사용하였다. (Tree, Boosting 계열은 스케일링 영향을 크게 받지 않으므로 Standard Scaler를 사용하지 않았다.)

4. 실험 결과 (Results)

4.1. 전처리 조건별 GridSearch 성능 비교

dataset 이름	log	Encoding (default = label encoding)	이상치 처리	Feature engineering
E-full	O	O	O	O
E-log	X	O	O	O
E-enc	O	X	O	O
E-out	O	O	X	O
E-feat	O	O	O	X

표 4.1 전처리 조건 별 dataset 구성

index	Train MSE	Train R²	Test MSE	Test R²	Best Params
Linear Regression	0.113	0.8945	0.1112	0.902	{}
Ridge	0.1131	0.8944	0.1113	0.9019	{'ridge__alpha': 10}
Gradient Boosting	0.1054	0.9015	0.1118	0.9015	{'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 50}
ElasticNet	0.1164	0.8913	0.1122	0.9011	{'elasticnet__alpha': 0.1, 'elasticnet__l1_ratio': 0.1}
Random Forest	0.1055	0.9014	0.1132	0.9002	{'max_depth': 5, 'n_estimators': 200}
SVR	0.1158	0.8918	0.1137	0.8998	{'svr__C': 0.1, 'svr__kernel': 'linear'}
CatBoost	0.1123	0.8952	0.1138	0.8997	{'depth': 3, 'iterations': 100, 'learning_rate': 0.1}
AdaBoost	0.1142	0.8933	0.1153	0.8983	{'learning_rate': 0.1, 'n_estimators': 200}
XGBoost	0.1089	0.8983	0.117	0.8969	{'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 50}
Decision Tree	0.1099	0.8974	0.1179	0.8961	{'max_depth': 5, 'min_samples_split': 5}
Lasso	0.1237	0.8844	0.1185	0.8955	{'lasso__alpha': 0.1}
KNN	0.2478	0.7686	0.375	0.6695	{'kneighborsregressor__n_neighbors': 5}

그림 4.1.1 E-full

index	Train MSE	Train R²	Test MSE	Test R²	Best Params
Ridge	7.867317183353338e+18	0.7149	8.384425583362911e+18	0.7155	{'ridge__alpha': 1}
Linear Regression	7.865180021038893e+18	0.715	8.390073264745755e+18	0.7153	{}
Lasso	7.865180021039981e+18	0.715	8.390072914015414e+18	0.7153	{'lasso__alpha': 10}
ElasticNet	7.887385017576789e+18	0.7142	8.396804080623091e+18	0.7151	{'elasticnet__alpha': 0.1, 'elasticnet__l1_ratio': 0.9}
Gradient Boosting	7.516694535132474e+18	0.7276	8.564330963229933e+18	0.7094	{'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 30}
AdaBoost	7.774032931652848e+18	0.7183	8.563578037979939e+18	0.7094	{'learning_rate': 0.1, 'n_estimators': 100}
XGBoost	7.208290102606679e+18	0.7388	8.607293652831118e+18	0.708	{'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 50}
Random Forest	7.182938782127831e+18	0.7397	8.632255946724602e+18	0.7071	{'max_depth': 5, 'n_estimators': 200}
CatBoost	7.437692980937712e+18	0.7305	8.642882791182029e+18	0.7068	{'depth': 3, 'iterations': 100, 'learning_rate': 0.1}
Decision Tree	8.067063284973462e+18	0.7077	8.845220126701473e+18	0.6999	{'max_depth': 3, 'min_samples_split': 2}
KNN	9.447400066827487e+18	0.6576	1.3115727292085252e+19	0.555	{'kneighborsregressor__n_neighbors': 7}
SVR	2.831793785746464e+19	-0.0262	3.0210621140012167e+19	-0.025	{'svr__C': 10, 'svr__kernel': 'linear'}

그림 4.1.2 E-log

index	Train MSE	Train R²	Test MSE	Test R²	Best Params
Linear Regression	0.1136	0.8939	0.111	0.9022	{}
Ridge	0.1136	0.8939	0.111	0.9022	{'ridge__alpha': 0.1}
Gradient Boosting	0.1055	0.9015	0.112	0.9013	{'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 50}
ElasticNet	0.1166	0.8911	0.1123	0.901	{'elasticnet__alpha': 0.1, 'elasticnet__l1_ratio': 0.1}
SVR	0.1163	0.8914	0.1128	0.9006	{'svr__C': 0.1, 'svr__kernel': 'linear'}
Random Forest	0.1055	0.9014	0.1131	0.9003	{'max_depth': 5, 'n_estimators': 200}
CatBoost	0.1128	0.8946	0.1135	0.8999	{'depth': 3, 'iterations': 100, 'learning_rate': 0.1}
AdaBoost	0.1136	0.8939	0.1143	0.8993	{'learning_rate': 0.1, 'n_estimators': 100}
XGBoost	0.109	0.8981	0.1172	0.8967	{'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 50}
Decision Tree	0.1094	0.8978	0.1175	0.8965	{'max_depth': 5, 'min_samples_split': 2}
Lasso	0.1237	0.8844	0.1185	0.8955	{'lasso__alpha': 0.1}
KNN	0.1443	0.8652	0.167	0.8528	{'kneighborsregressor__n_neighbors': 9}

그림 4.1.3 E-enc

index	Train MSE	Train R²	Test MSE	Test R²	Best Params
Ridge	0.1139	0.8966	0.1126	0.8939	{'ridge__alpha': 0.1}
Linear Regression	0.1138	0.8966	0.1126	0.8938	{}
ElasticNet	0.1173	0.8934	0.1138	0.8927	{'elasticnet__alpha': 0.1, 'elasticnet__l1_ratio': 0.1}
Gradient Boosting	0.1062	0.9036	0.116	0.8906	{'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 50}
SVR	0.1167	0.894	0.1163	0.8903	{'svr__C': 0.01, 'svr__kernel': 'linear'}
Random Forest	0.1067	0.9031	0.1169	0.8898	{'max_depth': 5, 'n_estimators': 50}
Decision Tree	0.1099	0.9002	0.1187	0.8881	{'max_depth': 5, 'min_samples_split': 2}
CatBoost	0.1122	0.8981	0.1194	0.8874	{'depth': 3, 'iterations': 100, 'learning_rate': 0.1}
AdaBoost	0.1164	0.8943	0.1201	0.8868	{'learning_rate': 0.1, 'n_estimators': 100}
Lasso	0.1246	0.8868	0.1203	0.8866	{'lasso__alpha': 0.1}
XGBoost	0.1095	0.9006	0.121	0.8859	{'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 50}
KNN	0.2543	0.769	0.3814	0.6404	{'kneighborsregressor__n_neighbors': 5}

그림 4.1.4 E-out

index	Train MSE	Train R^2	Test MSE	Test R^2	Best Params
Linear Regression	0.1131	0.8944	0.1111	0.9021	{}
Ridge	0.1131	0.8944	0.1111	0.9021	{'ridge__alpha': 1}
Gradient Boosting	0.1067	0.9003	0.1127	0.9007	{'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 50}
Random Forest	0.1067	0.9003	0.1135	0.9	{'max_depth': 5, 'n_estimators': 200}
SVR	0.1159	0.8917	0.1137	0.8998	{'svr__C': 10, 'svr__kernel': 'linear'}
CatBoost	0.1127	0.8948	0.1141	0.8995	{'depth': 3, 'iterations': 100, 'learning_rate': 0.1}
AdaBoost	0.1145	0.8931	0.1155	0.8982	{'learning_rate': 0.1, 'n_estimators': 100}
ElasticNet	0.1216	0.8864	0.1166	0.8972	{'elasticnet__alpha': 0.1, 'elasticnet__l1_ratio': 0.1}
XGBoost	0.1103	0.897	0.1168	0.8971	{'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 50}
Decision Tree	0.1099	0.8973	0.1169	0.897	{'max_depth': 5, 'min_samples_split': 5}
Lasso	0.1237	0.8844	0.1185	0.8955	{'lasso__alpha': 0.1}
KNN	0.2043	0.8092	0.4269	0.6237	{'kneighborsregressor__n_neighbors': 3}

그림 4.1.5 E-feat

4.2 log 변환의 효과

	full_Test R^2	log_Test R^2	ΔR^2 (full − log)
Linear Regression	0.902	0.7153	0.1867
Ridge	0.9019	0.7155	0.1864
Gradient Boosting	0.9015	0.7094	0.1921
ElasticNet	0.9011	0.7151	0.186
Random Forest	0.9002	0.7071	0.1931
SVR	0.8998	-0.025	0.9248
CatBoost	0.8997	0.7068	0.1929
AdaBoost	0.8983	0.7094	0.1889
XGBoost	0.8969	0.708	0.1889
Decision Tree	0.8961	0.6999	0.1962
Lasso	0.8955	0.7153	0.1802
KNN	0.6695	0.555	0.1145
Average $\lvert\Delta R^2\rvert$	None	None	0.244225

그림 4.2.1 E-full의 R^2과 E-log의 R^2 비교

모든 전처리를 적용한 E-full 데이터셋과 log 변환만 제거한 E-log 데이터셋의 비교 결과, 로그 변환은 예측 성능에 매우 중요한 영향을 미치는 전처리 기법임을 확인했다. 평균 R^2의 차이가 0.2442로, 전체 실험 중 가장 큰 성능 변화폭을 보였다. 특히, SVR의 모델의 경우, log 변환을 하지 않았을 경우, R^2가 0.8998에서 -0.025로 급격히 감소하며 예측력이 완전히 무너졌다. 그 외 대다수의 모델에서도 0.18~0.20 수준의 R^2감소가 관찰되었다.

4.3 one-hot encoding의 효과

	full_Test R^2	enc_Test R^2	ΔR^2 (full − enc)
Linear Regression	0.902	0.9022	-0.0002
Ridge	0.9019	0.9022	-0.0003
Gradient Boosting	0.9015	0.9013	0.0002
ElasticNet	0.9011	0.901	0.0001
Random Forest	0.9002	0.9003	-0.0001
SVR	0.8998	0.9006	-0.0008
CatBoost	0.8997	0.8999	-0.0002
AdaBoost	0.8983	0.8993	-0.001
XGBoost	0.8969	0.8967	0.0002
Decision Tree	0.8961	0.8965	-0.0004
Lasso	0.8955	0.8955	0.0
KNN	0.6695	0.8528	-0.1833
Average $\lvert\Delta R^2\rvert$	None	None	0.015567

그림 4.3.1 E-full의 R^2과 E-enc의 R^2 비교

모든 전처리를 적용한 E-full 데이터셋과 one-hot encoding 변환만을 제거한 E-enc (label encoding 적용) 데이터셋을 비교한 결과, 인코딩 유무는 대부분의 모델에 큰 영향을 미치지 않는 전처리 항목임을 확인했다. 전체 모델의 R^2차이의 평균은 0.0156이며, KNN을 제외한 모델에서 R^2 값의 변화가 ±0.001 이내로 매우 작았다. 선형 모델이나 트리 기반 모델은 one-hot encoding, label encoding 방식의 관계없이 비슷한 예측 성능을 보였다. 다만, KNN 모델에서는 R^2가 0.6695에서 0.8528로 크게 상승하였다.

4.4 outlier(상위 1%) 제거의 효과

	full_Test R^2	out_Test R^2	ΔR^2 (full − out)
Linear Regression	0.902	0.8938	0.0082
Ridge	0.9019	0.8939	0.008
Gradient Boosting	0.9015	0.8906	0.0109
ElasticNet	0.9011	0.8927	0.0084
Random Forest	0.9002	0.8898	0.0104
SVR	0.8998	0.8903	0.0095
CatBoost	0.8997	0.8874	0.0123
AdaBoost	0.8983	0.8868	0.0115
XGBoost	0.8969	0.8859	0.011
Decision Tree	0.8961	0.8881	0.008
Lasso	0.8955	0.8866	0.0089
KNN	0.6695	0.6404	0.0291
Average $\lvert\Delta R^2\rvert$	None	None	0.01135

그림 4.4.1 E-full의 R^2과 E-out의 R^2 비교

모든 전처리를 적용한 E-full 데이터셋과 outlier 제거를 적용하지 않은 E-out 데이터셋을 비교한 결과, 이상치 제거는 전체 모델에서 성능 향상이 이뤄졌다. 전체 모델의 R^2차이의 평균은 0.01135로, 로그 변환 유무 실험에 비해서는 작았지만, 일관된 성능 향상이 관찰되었다.
대부분의 모델에서 이상치를 제거하지 않았을 때, 예측 성능이 소폭 하락했으며, 특히 KNN 모델은 R^2이 0.6695에서 0.6404로 감소하여 이상치에 대한 민감성이 가장 뚜렷하게 나타났다.

4.5 feature engineering의 효과

	full_Test R^2	feat_Test R^2	ΔR^2 (full − feat)
Linear Regression	0.902	0.9021	-0.0001
Ridge	0.9019	0.9021	-0.0002
Gradient Boosting	0.9015	0.9007	0.0008
ElasticNet	0.9011	0.8972	0.0039
Random Forest	0.9002	0.9	0.0002
SVR	0.8998	0.8998	0.0
CatBoost	0.8997	0.8995	0.0002
AdaBoost	0.8983	0.8982	0.0001
XGBoost	0.8969	0.8971	-0.0002
Decision Tree	0.8961	0.897	-0.0009
Lasso	0.8955	0.8955	0.0
KNN	0.6695	0.6237	0.0458
Average $\lvert\Delta R^2\rvert$	None	None	0.004367

그림 4.5.1 E-full의 R^2과 E-feat의 R^2 비교

모든 전처리를 적용한 E-full 데이터셋과 파생 변수 추가만 생략한 E-feat 데이터셋을 비교한 결과, Feature Engineering은 대부분의 모델에서 예측 성능에 거의 영향을 미치지 않는 전처리 항목으로 나타났다. 전체 모델의 R^2차이의 평균은 0.004367로, 실험된 전처리 중 가장 작은 수준이었다. 대부분의 모델에서 R^2값의 차이가 ±0.001 이내로 미미했으며, SVR과 Lasso 모델은 변화가 거의 없었다. KNN 모델에서는 R^2가 0.6695에서 0.6237로 감소하며, 상대적으로 큰 영향을 받은 것으로 나타났다.

4.6 세부 모델 선택 후 grid search

네 번의 실험을 통해, Feature Engineering과 One-hot Encoding은 대부분의 모델에서 성능을 다소 향상시키는 효과를 보였지만, 그 향상 폭은 미미했다. 또한, 두 전처리는 메모리 사용량이 증가한다는 단점이 있어, 이에 따라 로그 변환과 이상치 제거만을 적용한 전처리 방식으로 추가적인 Grid Search 실험을 진행하기로 했다.

또한, 기존 다섯 가지 데이터셋 실험에서 일관되게 우수한 성능을 보였던 모델들인 Linear Regression, Ridge, Gradient Boosting, ElasticNet, Random Forest를 실험 대상으로 선정하였고, Ridge 및 ElasticNet과 동일 계열의 모델인 Lasso를 추가하여, 총 6개의 모델에 대해 성능을 비교하였다.

4.6.1. linear/logistic + regularization

```
      Variant   Model    MAE_train    MSE_train  R2_train     MAE_test     MSE_test  R2_test                                                                  Best Params
log 0 | out 0   Lasso 2.879012e-01 1.138309e-01  0.893680 2.847229e-01 1.104295e-01 0.902671    {'est__alpha': 0.01, 'est__fit_intercept': True, 'est__selection': 'cyclic'}
log 0 | out 0 Elastic 2.879005e-01 1.138303e-01  0.893681 2.847234e-01 1.104298e-01 0.902670      {'est__alpha': 0.01, 'est__fit_intercept': True, 'est__l1_ratio': 0.9}
log 0 | out 0   Ridge 2.877321e-01 1.136694e-01  0.893831 2.847267e-01 1.105658e-01 0.902550   {'est__alpha': 0.0001, 'est__fit_intercept': True, 'est__solver': 'auto'}
log 0 | out 0  Linear 2.877865e-01 1.137208e-01  0.893783 2.850833e-01 1.107023e-01 0.902430                        {'est__fit_intercept': True, 'est__positive': True}
```

그림 4.6.1 label encoding

```
      Variant   Model    MAE_train    MSE_train  R2_train     MAE_test     MSE_test  R2_test                                                                  Best Params
log 0 | out 0 Elastic 2.876173e-01 1.136028e-01  0.893893 2.845051e-01 1.103493e-01 0.902741      {'est__alpha': 0.01, 'est__fit_intercept': True, 'est__l1_ratio': 0.7}
log 0 | out 0   Lasso 2.877249e-01 1.136834e-01  0.893818 2.845457e-01 1.103473e-01 0.902743    {'est__alpha': 0.01, 'est__fit_intercept': True, 'est__selection': 'cyclic'}
log 0 | out 0   Ridge 2.868359e-01 1.131045e-01  0.894359 2.854073e-01 1.110559e-01 0.902119      {'est__alpha': 1.0, 'est__fit_intercept': True, 'est__solver': 'auto'}
log 0 | out 0  Linear 2.868482e-01 1.131465e-01  0.894319 2.854911e-01 1.110782e-01 0.902099                        {'est__fit_intercept': True, 'est__positive': True}
```

그림 4.6.2 one-hot encoding

Linear/logistic + regularization 실험의 경우, one-hot encoding을 적용한 추가적인 grid search 실험도 함께 수행하였다. 그 결과, one-hot encoding을 적용했을 때, 최고 성능이 소폭 더 우수한 것으로 나타났다.

4.6.2. gradient boosting

log	outlier	encoding	feature_eng	train mae	train mse	train r^2	test mae	test mse	test r^2	best parameter
true	true	false	false	0.280548321996739	0.10835947809918932	0.898790570546160 5	0.28595755805402406	0.11244111624327859	0.9008976685936383	{'learning_rate': 0.07, 'max_depth': 2, 'n_estimators': 100}
true	true	false	false	0.281492213005332	0.10949275245523139	0.897732074759712 6	0.28579543429869053	0.11228104807228115	0.9010387480266766	{'learning_rate': 0.07, 'max_depth': 2, 'min_samples_leaf': 4, 'min_samples_split': 3, 'n_estimators': 85}

Gradient boosting의 경우 max_depth까지 지정하여 실험한 경우와, min_samples_leaf와 min_samples_split까지 추가적으로 지정하여 실험한 경우의 최고 성능이 다르게 나왔다. min_samples_leaf와 min_samples_split까지 추가적으로 지정한 경우가 성능이 조금 더 우수했다.
Log와 이상치 처리를 한 데이터셋에서 {'learning_rate': 0.07, 'max_depth': 2, 'min_samples_leaf': 4, 'min_samples_split': 3, 'n_estimators': 85} parameter 조합일 때, gradient boosting 성능이 가장 높게 나왔다.

4.6.3. Random forest

log	outlier	train mae	train mse	train r^2	test mae	test mse	test r^2	best parameter
TRUE	TRUE	0.278656	0.107065	0.9	0.287451	0.113374	0.900075	{'max_depth': 5, 'max_features': None, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 500}

Log와 이상치 처리를 한 데이터셋에서 {'max_depth' : 5, 'max_features' : None, 'min_samples_leaf' : 2, 'min_samples_split' :5, n_estimators' : 500} parameter 조합일 때, Random Forest 성능이 가장 높게 나왔다.

4.7. feature importance

4.8. ensemble

4.6 실험에서 높은 성능을 낸 3가지 조합에 대해 각각 Softvoting, Stacking을 진행하였다.

```
log O, enc X, out O

 Soft Voting 결과
MAE : 0.2847
MSE : 0.1104
R²  : 0.9027

 Stacking with Ridge 결과
MAE : 0.2855
MSE : 0.1109
R²  : 0.9023
 Best Params (Stacking): {'final_estimator__alpha': 0.001}
```

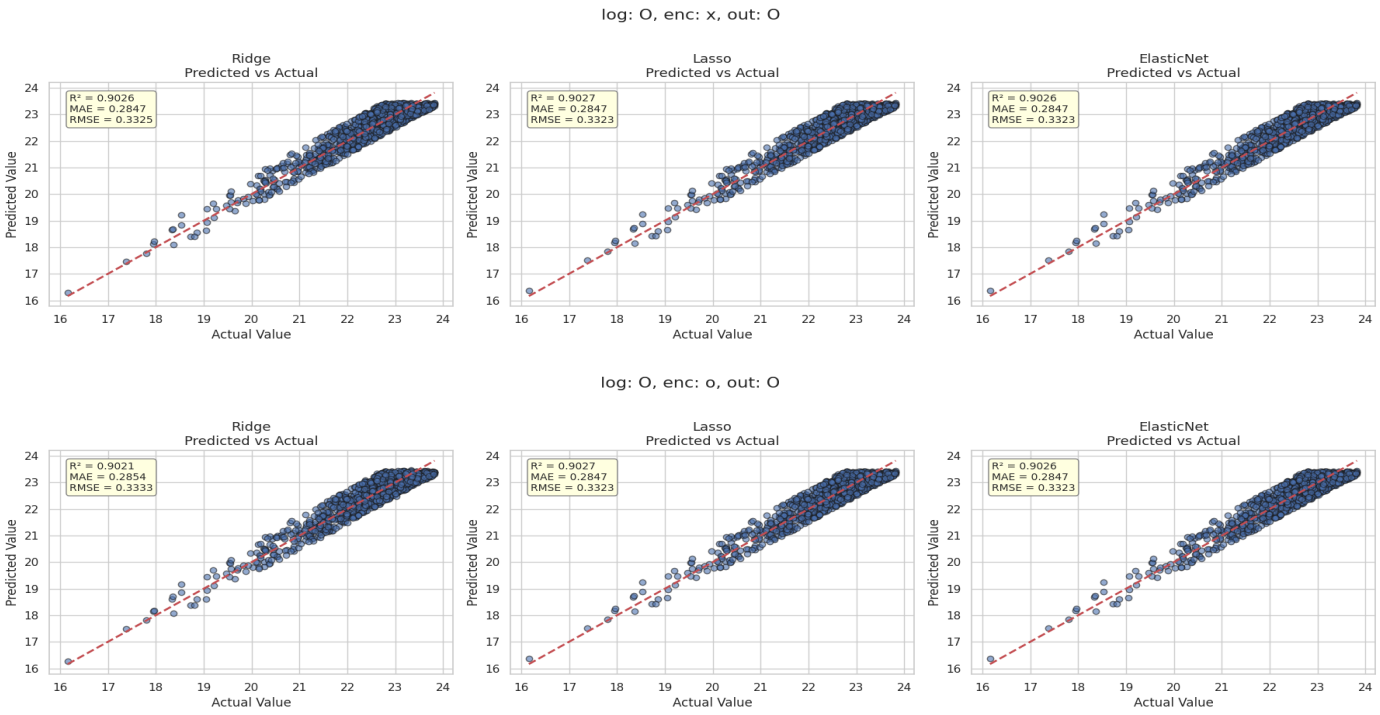
```
log O, enc O, out O

 Soft Voting 결과
MAE : 0.2847
MSE : 0.1105
R²  : 0.9026

 Stacking (with Ridge α tuning) 결과
MAE : 0.2849
MSE : 0.1106
R²  : 0.9025
 Stacking Best Params: {'final_estimator__alpha': 10.0}
```

Soft Voting한 결과 Stacking 결과보다 우수하게 나왔다.

4.9 예측 vs 실제 산점도



5. 결론 및 고찰 (Discussion)

5.1 실험 결과에 대한 상세 분석

실험 전반적으로 선형 모델이 트리 기반 앙상블 모델들보다 더 좋은 성능을 보였다. Target 변수에 로그 변환을 취하면 Feature와 Target 간의 관계가 선형 관계에 가깝게 변환되어 이러한 선형 관계를 파악하는데 선형 모델이 조금 더 유리했을 것이라 보인다(이는 5.4절에서 언급할 데이터의 근본적인 특징과 부합한다). 금융 데이터에서 로그 스케일에서의 기업 가치와 주요 feature는 실제 관계가 복잡한 비선형성 보다는 선형적인 증가, 감소 패턴을 주로 따를 수 있다. 또한, 트리 기반 모델은 feature 간의 복잡한 상호작용을 감지하는데 강점이 있는데 이 데이터셋에는 복잡한 상호작용 효과가 크지 않아 이러한 강점이 부각되지 않았을 것이라 판단한다. Ridge, Lasso, ElasticNet 같은 선형 모델은 feature 간의 상관관계에 의한 다중공선성 문제를 규제의 효과로 해결함으로써 더 높은 성능을 보였다.

첫 번째 실험에서 A를 제거한 경우는 모든 모델에서 성능이 크게 하락했는데 이는 target인 Valuation이 매우 편향된 분포를 가지고 있음을 나타낸다. 금액과 관련된 데이터는 일반적으로 right-skewed 형태가 많은데 로그 변환을 통해 이러한 분포를 정규분포에 가깝게 만들어 많은 알고리즘의 예측 성능을 크게 향상시킨다. B를 제거한 경우는 baseline과의 성능 차이가 미미하였다. Label Encoding으로 인해 인위적인 순서성이 부여될 수도 있지만 이것의 의한 영향은 그리 크지 않음을 알 수 있다. 다만, 두 번째 실험에서 선형 모델의 경우, One-Hot encoding이 미세하게 성능이 높았던 점을 고려하면 모델에 따라 최적의 인코딩 방식은 달라질 수 있다. C를 제거한 경우는 전체적인 모델의 성능이 소폭 감소하는 경향을 보였다. 이는 데이터 내에 이상치가 존재하며 이를 적절히 처리하는 것이 모델 성능에 기여한다는 것을 알 수 있다. 금융 관련 데이터에서는 이상치가 빈번하게 발생하므로 이상치 제거는 모델의 일반화 성능과 안정성에 중요한 역할을 한다. D 제거 실험 결과와 baseline을 비교했을 때 성능 차이가 미미하였다. Feature Engineering 기법이 성능에 결정적인 영향을 주지 않고 다른 조건들의 영향이 더 크다는 것을 알 수 있다.

첫 번째 실험에서 실험 조건 A와 C가 성능에 큰 영향을 미치는 것을 확인했다. 두 번째 실험에서는 이 두 조건을 각각 적용/미적용하고 성능이 좋은 모델(선형 모델, Random Forest, Gradient Boosting)에 대해 더 넓은 하이퍼파라미터 탐색을 진행했다. 모든 실험 결과에서 로그 변환을 미적용한 경우는 매우 낮은 R^2와 매우 높은 MSE를 보였다. 이는 첫 번째 실험의 결과와 같이 로그 변환이 필수적이라는 것을 의미한다. 이상치 처리는 적용한 것이 적용하지 않은 결과보다 일관되게 더 좋은 성능을 보였다. 따라서 이러한 금융 데이터에서 로그 변환과 이상치 처리는 성능 개선에 효과적임을 알 수 있다. 더 넓은 하이퍼파라미터 탐색을 통해 첫 번째 실험보다 더 좋은 성능을 보였다.

이전 실험에서 우수한 성능을 보인 선형 모델들(Ridge, Lasso, Elastic Net)을 기반으로 Meta-ensemble(Soft-voting, Stacking)을 진행했다. Soft Voting과 Stacking 한 결과, 모두 MSE와 R^2 값이 거의 동일한 수준을 보였으나 MAE 값이

개별 모델보다 개선되었다. 이는 앙상블 모델이 개별 모델보다 더 안정적인 성능을 보인 것을 알 수 있다. 모두 선형 모델을 앙상블 했기에 서로 유사성을 띄어 추가적인 성능 향상의 폭이 제한적이었다.

5.2 하이퍼파라미터에 대한 분석

우선 가장 좋은 성능을 보였던 모델 세가지는 다음과 같다.

1. Lasso (α = 0.01, selection = 'cyclic')

Lasso는 ℓ_1 규제를 통해 불필요한 변수의 계수를 0으로 만들어 모델을 단순화하며, α = 0.01로 설정된 것은 데이터가 로그 변환과 이상치 제거로 이미 안정화되어 강한 규제가 필요하지 않음을 나타낸다. 작은 α 값은 필수 피처를 유지하면서 잔차를 최소화하며, selection='cyclic'은 순차적 계수 갱신으로 안정적 수렴과 희소화를 달성해 테스트 MSE 0.110347을 기록했다. 이는 세 모델 중 가장 낮은 오류로, 최소한의 규제로도 뛰어난 일반화 성능을 보여준다.

2. ElasticNet (α = 0.01, l1_ratio = 0.7)

ElasticNet은 ℓ_1(70%)과 ℓ_2(30%) 규제를 조합해 희소화와 안정성을 동시에 추구한다. α = 0.01은 전처리로 잡음이 줄어든 데이터에서 과도한 규제가 불필요함을 보여주며, l1_ratio = 0.7은 변수 선택과 계수 안정화의 균형을 맞춘 선택이다. 이를 통해 테스트 MSE 0.110349를 달성했으며, Lasso와 거의 동등한 성능을 보이며 안정적 수렴을 확보했다.

3. Ridge (α = 1.0, solver = 'auto')

Ridge는 ℓ_2 규제로 모든 계수를 축소해 과적합을 방지하며, α = 1.0은 데이터가 전처리로 정제되어 강한 규제가 필요하지 않음을 반영한다. solver='auto'는 데이터 특성에 맞는 최적 알고리즘을 선택해 계산 효율을 높였다. 테스트 MSE 0.111056은 Lasso와 ElasticNet보다 약간 높지만, 여전히 안정적인 성능을 보여준다.

가장 좋은 성능을 보인 하이퍼파라미터는 "Lasso (α = 0.01, selection = 'cyclic')"로, 테스트 MSE 0.110347을 달성했다. 이 하이퍼파라미터가 뛰어난 이유는 예측컨데 다음과 같다:

1. α = 0.01의 작은 값: 데이터가 로그 변환과 이상치 제거로 상당히 정제되어 잡음과 다중공선성이 줄어든 상태였다. 이는 모델 복잡도가 낮아져 강한 규제가 필요하지 않음을 보여준다. 작은 α 값은 필요한 피처의 계수를 유지하면서 불필요한 변수를 효과적으로 제거해 과소적합 없이 최적의 예측력을 확보했다.

2. selection = 'cyclic': 이 설정은 좌표 하강법에서 계수를 순차적으로 갱신하며 안정적으로 수렴하도록 도왔다. 데이터의 피처 수가 많거나 상관관계가 있을 가능성이 있는 상황에서, 'cyclic' 방식은 희소화 과정을 반복적으로 최적화해 낮은 테스트 오류를 달성하는 데 기여했다.

ElasticNet (α = 0.01, l1_ratio = 0.7)도 거의 동일한 MSE(0.110349)를 기록했으나, ℓ_1과 ℓ_2 규제의 조합은 Lasso의 순수 ℓ_1 규제보다 약간 더 복잡한 모델을 생성한다. 따라서 단순성과 성능 면에서 Lasso가 약간 더 우수하다고 판단된다. Ridge의 α = 1.0은 적당한 ℓ_2 규제를 제공했지만, 변수 선택이 없는 특성상 희소화가 필요한 데이터에서 Lasso나 ElasticNet에 비해 약간의 성능 손실(MSE 0.111056)이 발생했다.

결론적으로, "Lasso (α = 0.01, selection = 'cyclic')"는 전처리로 정제된 데이터에서 최소한의 규제로 최대한의 예측력을 끌어내며, 불필요한 변수를 효과적으로 제거해 가장 낮은 테스트 MSE를 달성했다. 이는 데이터가 이미 안정화되어 강한 규제 없이도 일반화 성능을 확보할 수 있었음을 보여준다.

5.3 Feature importance에 대한 해석

모든 모델에서 Investment Amount (USD)가 가장 중요한 변수로 나타나며, 기업 평가액에 가장 직접적인 영향을 미치는 핵심 feature임을 나타낸다. 투자자들이 성장 가능성이 높은 기업에 더 많은 자금을 투자하는 경향이 있다. 그로 인해 해당 기업의 향후 평가되는 Valuation (USD) 또한 높게 평가되었을 가능성이 크다고 해석할 수 있다.

One-hot encoding을 적용함으로써, 각 범주형 변수의 개별 범주가 독립된 feature로 변환되어, 모델의 범주별 영향력을 보다 명확하게 볼 수 있다. 이를 통해, Industry, Country 범주형 변수들의 개별 feature가 예측 결과에 어떻게 기여하는지 구체적으로 분석할 수 있었다.

[Random Forest / Gradient Boosting]

Investment Amount, Growth Rate, Number of Investors와 같은 연속형 변수들이 주요 변수로 나타났으며, 이는 기업의 성장성과 투자 활동이 기업 가치에 중요한 영향을 미친다는 일반적인 해석과 일치한다.

다만, 트리 기반 모델의 feature importance는 불순도 감소량 기준으로 계산되며, 이는 연속형 변수나 범주 수가 많은 변수의 중요도를 과대평가하는 경향이 있음을 여러 연구에서 보여줬다. Zhou & Hooker (2020)의 연구(Unbiased Measurement of Feature Importance in Tree-Based Methods)[1]는, 연속형 변수는 더 많은 분할 지점을 제공해 자주 선택되며, 이로 인해 중요도가 과도하게 상승하는 편향이 발생함을 실험적으로 입증하였다. 또한, 트리 기반 모델은 비선형 관계와 변수 간 상호작용을 효과적으로 포착하는 것으로 잘 알려져 있다. Chen et al. (2024)의 연구(Nature Scientific Reports)[2]에서도 랜덤포레스트가 복잡한 비선형적 환경 변수 관계를 잘 모델링함을 실증적으로 보여주었다.

따라서 본 실험에서 Industry, Country와 같은 범주형 변수와 Investment Amount, Growth Rate 등 연속형 변수가 모두 중요하게 나타난 것은, 트리 모델의 구조적 특성과 비선형 학습 능력을 반영한 결과로 해석할 수 있다. 특히 Investment Amount의 중요도가 매우 높게 나온 것은 편향성에 따른 결과일 수 있다.

[Linear Regression]

선형 회귀 모델에서는 Investment Amount (USD)를 제외하고 대부분의 중요한 변수들이 Industry_ 또는 Country_로 시작하는 범주형 변수로 나타났다. 이는 one-hot encoding이 각 카테고리의 독립적인 영향을 계수로 명확히 반영하기 때문이다. 반면, label encoding은 인위적인 순서 관계를 도입하여 feature importance를 왜곡할 수 있다.[3] 이는 데이터 패턴(예: "Industry_Edtech"나 "Country_USA"의 영향력)이 모델에 정확히 반영되도록 one-hot encoding이 더 적합함을 시사한다.

[Ridge/Lasso/ElasticNet]

규제 기반 선형 모델에서는 모든 모델에서 공통적으로 Investment Amount 변수가 가장 중요한 변수로 나타났다. Ridge 모델은 상대적으로 범주형 변수의 중요도가 높게 나타났다. 이는 Ridge가 L2 규제를 사용면서 모든 변수의 계수를 일정 수준으로 유지하려는 특성으로 인해 다양한 변수의 영향을 분산하여 반영하기 때문인 것으로 보인다.
Lasso와 ElasticNet 모델에서는 연속형 변수들의 중요도가 높게 나타났다. Lasso는 L1 정규화를 통해 중요하지 않은 변수의 계수를 0으로 만드는 성질이 있어, 상대적으로 영향력이 작은 범주형 변수를 제거하고, 연속형 변수 위주로 학습하는 경향을 보였다. ElasticNet은 Lasso의 희소성과 Ridge의 변수 보존 특성을 결합하지만, L1 패널티의 영향으로 연속형 변수에 더 큰 가중치를 부여했다.

5.4 데이터에서 얻어낸 인사이트

스타트업의 기업 가치는 소수의 매우 높은 가치를 가진 기업으로 인해 분포가 크게 왜곡되어 있다. 이는 투자 세계의 일반적인 멱함수 분포와 같은 특징을 반영하고 있다. 수치형 데이터(펀딩 라운드 수, 총 투자 금액, 투자자 수, 성장률)와 범주형 데이터(산업 부문, 국가) 모두 기업 가치 예측에 중요한 정보를 포함하고 있고 기업 가치와 대체로 선형적인 관계를 이루고 있다. 또한, 이상치 실험을 통해 시장의 변동성 또는 특정 기업의 독보적인 성장을 보여주고 있다. 본 데이터셋은 공개 보고서를 바탕으로 실제 스타트업 투자에서 나타난 추세를 재현하기 위해 합성적으로 만들어졌다. 이로 인해 복잡한 비선형성이 의도적으로 배제되었을 가능성이 높고 샘플 수가 많지 않아 선형 모델이 일반화에 유리함을 알 수 있다.

---

[1] Zhou, Z., & Hooker, G. (2020). *Unbiased Measurement of Feature Importance in Tree-Based Methods*. ACM Transactions on Knowledge Discovery from Data, 15(2), Article 26.

[2] Chen, C., Wang, J., Li, D., Sun, X., Zhang, J., Yang, C., & Zhang, B. (2024). *Unraveling nonlinear effects of environment features on green view index using multiple data sources and explainable machine learning*. Scientific Reports, 14, Article 30189.

[3] Poslavskaya, E., & Korolev, A. (2023). Encoding categorical data: Is there yet anything 'hotter' than one-hot encoding? arXiv preprint arXiv:2312.16930. https://doi.org/10.48550/arXiv.2312.16930