

# 상호명 기반 표준산업분류(KSIC) 코드 예측 모델 개발

# 1. Data Preparation

## 입력 데이터 (Feature)

- 상호명 (텍스트 데이터)
- 사업자등록번호 중 관할 세무서 코드
- 사업자유형 (파생변수)
- 소재지-시도 (파생변수)
- 중소기업여부 (파생변수)
- 통신판매사업자 취급품목 (파생변수)

## 목표 변수 (Label)

- 표준산업분류 코드 (KSIC)
- 예: C2611 (전자집적회로 제조업)
- 예: G4711 (슈퍼마켓 운영업)
- 예: J6209 (기타 정보기술 서비스업)

본 모델은 상호명이라는 텍스트 데이터를 주요 특성으로 활용하며, 사업자등록번호에서 파생된 사업자유형, 소재지, 중소기업여부 등의 정보를 보조 특성으로 사용합니다. 이러한 입력 데이터를 바탕으로 해당 기업의 KSIC 코드를 예측하는 것이 목표입니다.

# 1. Data Preparation

사업자등록번호의 구성  
ABC-DE-FGHIJ

Possible Case :  $10 * 10 * 10 * 10 * 10 * 10 * 10 * 10 * 10 * 10 = 10^{10} = 100$ 억개

*마지막 자리수를 제외한 각 자리수에 1.3.7.1.3.7.1.3.5 곱한 후, 각각 더한 다음.  
마지막에서두전째 숫자에 5를 곱하고 10으로 나누어 나온 값의 몫을 더합니다.*

*그리고나서 10에서 아까 더한 수에 10으로 나눈 나머지를 뺍니다.*

*이 숫자가 사업자등록번호의 마지막 자리의 수와 일치하면 검증된 사업자번호입니다.*

Real Possible Case : 약 8억 9천만개(사업자등록번호는 101부터 시작하므로)

# 1. Data Preparation

1	1010100009
2	1010100014
3	1010100028
4	1010100033
5	1010100047
6	1010100052
7	1010100066
8	1010100071
9	1010100085
10	1010100090
11	1010100106
12	1010100111
13	1010100125
14	1010100130
15	1010100144

### 사업자등록 상태조회 API

POST

/status

^

🔒

사업자 상태조회 정보 제공.  
1회 호출 시 최대 100개 에 해당하는 사업자 상태정보 요청 가능.

기본정보			
데이터명	국세청_사업자등록정보 진위확인 및 상태조회 서비스		상세설명
서비스유형	REST	심의여부	자동승인
일 호출 제한	1000000		

# 1. Data Preparation

b\_stt\_cd

**string**

*example: 01*

*xml: OrderedMap { "name": "BSttCd" }*

납세자상태(코드):

01: 계속사업자,

02: 휴업자,

03: 폐업자

*xml:*

*name: BSttCd*

tax\_type

> [...]

tax\_type\_cd

**string**

*example: 01*

*xml: OrderedMap { "name": "TaxTypeCd" }*

과세유형메세지(코드):

01: 부가가치세 일반과세자,

02: 부가가치세 간이과세자,

03: 부가가치세 과세특례자,

04: 부가가치세 면세사업자,

05: 수익사업을 영위하지 않는 비영리법인이거나 고유번호가 부여된 단체, 국가기관 등,

06: 고유번호가 부여된 단체,

07: 부가가치세 간이과세자(세금계산서 발급사업자)

# 1. Data Preparation

약 4천만개의 유효한  
사업자등록번호 및 상태값 획득

	bizno	nts_tpyr_sts_cd	nts_ttn_tcd
1	1010100009	00	
2	1010100014	00	
3	1010100028	00	
4	1010100033	00	
5	1010100047	00	
6	1010100052	00	
7	1010100066	00	
8	1010100071	00	
9	1010100085	00	
10	1010100090	00	
11	1010100106	03	03
12	1010100111	00	
13	1010100125	00	
14	1010100130	00	
15	1010100144	00	
16	1010100159	00	
17	1010100163	00	

00	858724680
01	10612393
02	55530
03	29597396

# 1. Data Preparation

전체 (19)

메뉴 (5)

공지사항 · 자료실 (8)

국세청 홈페이지 (6)

적용된 검색어 '국세환급금' / 메뉴 (5건)

환급

 국세환급 > 국세 환급금찾기

나의 국세환급금 조회

납세자구분	<input checked="" type="radio"/> 내국인 <input type="radio"/> 외국인
* 주민등록번호(사업자등록번호)	<input type="text" value="....."/> <small>! 하이픈(-) 없이 숫자만 입력</small>
* 성명(상호)	<input type="text" value="가"/> <small>! 상호는 일부 단어로 입력도 가능</small>

조회

# 1. Data Preparation

나의 국세환급금 조회

납세자구분

☒ 내국인 ☐ 외국인

\* 주민등록번호(사업자등록번호)

.....

! 하이픈(-) 없이 숫자만 입력

\* 성명(상호)

나이스평가정보

! 상호는 일부 단어로 입력도 가능

조회



귀하께서 찾아가지 않은 국세환급금이 존재하지 않습니다.

1	1010100275	개인택시
2	1010100464	가평상회
3	1010100616	진흥상회
4	1010100765	삼청빵화탕
5	1010100784	삼청여관
6	1010102671	오복집
7	1010103457	부여상회
8	1010104010	삼성사
9	1010104102	개인택시
10	1010104418	중앙교육원구내매점
11	1010104887	충남미용실
12	1010105552	개인택시
13	1010105829	삼청루
14	1010105867	삼보부동산
15	1010105872	풍미식품
16	1010106283	명성사
17	1010106298	골목집
18	1010106304	오티터
19	1010106492	삼청설농탕



# 1. Data Preparation

	🔍 bizno ▼	📄 sido ▼	📄 idscd ▼
1	1010106624	서울	H49231
2	1010107840	서울	G47842
3	1010107977	서울	I56111
4	1010108210	서울	I56111
5	1010108772	서울	I56111
6	1010108807	서울	H52941
7	1010108883	서울	I56111
8	1010109015	서울	G47121
9	1010109034	서울	I56219
10	1010109048	서울	I56111
11	1010109067	서울	I56111
12	1010109086	서울	I56111
13	1010109091	서울	I56111

모 카드사의 가맹점 데이터  
사업자등록번호 | 소재지 | 산업분류코드

근무하고 있는 나이스평가정보의 데이터를 사용하고  
싶었지만 내부 규정으로 개인의 학술 연구 목적으로는  
데이터 활용이 불가능

카드사의 가맹점 데이터이므로 카드사에서 가맹점에  
부여한 업종코드를 기반으로 산업분류코드 매칭  
(실제 해당 업체가 영위하는 산업분류와 상이할 수 있음)

비공개 데이터이므로 전체 데이터 셋에서 주소의  
도/광역시 정보와 산업분류코드만 추출하여 사용함

# 1. Data Preparation

<https://sminfo.mss.go.kr/>

<div> 중소기업현황정보시스템 <b>SMINFO</b></div> <div>홈</div>		
중소기업현황	중소기업 범위	중소기업확인서 발급신청
기업정보	중소기업기준	발급절차 안내
기업통계	관계기업이란	온라인 자료제출
기업현황 등록	소상공인유예	제출자료 조회
	중소기업 유예	신청서 작성
	관련 법령	진행상황 확인
	중소기업여부 자가진단	확인서 출력/수정
	<u>확인서발급정보공개</u>	발급안내 문의처

# 1. Data Preparation

## 확인서발급정보공개


※ 확인서 발급정보공개 내역 조회는 최대 15개월 이내만 가능합니다.  
(현재 유효한 확인서 발급정보만 조회가능)


검색

사업자번호 ▾

1010112752

기간조회

2024-03-07 

2025-06-07 

검색

초기화

업체명	사업자번호	발급번호	유효기간	기업규모
토탈싸인대광부식	101-01-12752	0010-2025-339225	2025-04-01 ~ 2026-03-31	소기업(소상공인)

1	1010112752	소기업(소상공인)
2	1010115841	소기업(소상공인)
3	1010116082	소기업(소상공인)
4	1010116587	소기업(소상공인)
5	1010116651	소기업(소상공인)
6	1010116684	소기업(소상공인)
7	1010119129	소기업(소상공인)
8	1010121358	소기업(소상공인)
9	1010122885	소기업(소상공인)
10	1010129646	소기업(소상공인)
11	1010144633	소기업(소상공인)
12	1010145888	소기업(소상공인)
13	1010146591	소기업(소상공인)
14	1010154956	소기업(소상공인)
15	1010159441	소기업(소상공인)
16	1010170505	소기업(소상공인)
17	1010180643	소기업(소상공인)
18	1010195879	소기업(소상공인)
19	1010211051	소기업(소상공인)

# 1. Data Preparation

<https://www.ftc.go.kr/www/selectBizCommOpenList.do?key=255>

홈 > 정보공개 > 사업자정보공개 > 통신판매사업자

## 자료 다운로드



- 여기에 제공되는 데이터는 통신판매사업자 전체데이터이며 **매주 일요일 오전** 갱신됩니다.
- 데이터의 크기상 지역별로 구분하여 다운로드 받을 수 있습니다.
- **실시간 데이터의 조회**를 원하시면 '정보공개>사업자정보공개>통신판매사업자>등록현황' 메뉴에서 사업자별로 조회 가능합니다.
- 공공데이터포털에서 제공하는 오픈API를 활용하면 다양한 서비스 및 프로그램에 사용할 수 있습니다.
- 공공데이터포털([www.data.go.kr](http://www.data.go.kr)) 접속 후 '공정거래위원회'로 검색

서울특별시



전체



자료 다운로드



# 1. Data Preparation

```
select
    substring(A.bizno, 1, 3) as '관할세무서코드',
    case when substring(A.bizno, 4, 2) in ('81','82','83','84','85','86','87') then '법인' else '개인' end as '개인법인구분',
    nts_ttn_tcd as '과세유형코드',
    A.conm_nm as '상호명',
    nvl(entr_size, '미확인') as '기업규모',
    sido as '시도명',
    (select replace(replace(취급품목, ' ', ''), '/', ' ') from telesales where Z.bizno = bizno limit 1) as '취급품목',
    idscd as '산업분류코드'
from i_em501 Z,
     i_em503 A,
     sme_bizno B,
     em301_new C
where A.bizno = B.bizno
     and A.bizno = C.bizno
     and A.bizno = Z.bizno
     and Z.nts_tpyr_sts_cd != '00';
```

3,016,783건

# 1. Data Preparation

	사업자등록번호	관할세무서코드	개인법인가분	과세유형코드	상호명	기업규모	시도명	취급품목	산업분류코드
0	1010106624	101	개인	2	개인택시	미확인	서울	NaN	H49231
1	1010108807	101	개인	2	코리아 콜밴	미확인	서울	NaN	H52941
2	1010109048	101	개인	1	부동산임대(김은숙)	미확인	서울	NaN	I56111
3	1010109223	101	개인	1	한국황토건축연구소	미확인	서울	NaN	I56111
4	1010109257	101	개인	1	PENIEL	미확인	서울	NaN	I56111
...	...	...	...	...	...	...	...	...	...
3016778	8999901437	899	개인	4	기탄사고력교실 명지중흥점	미확인	부산	NaN	P85709
3016779	8999901456	899	개인	4	금강흑염소농장	소기업(소상공인)	충북	NaN	G47212
3016780	8999901475	899	개인	4	나디아킴 음악교습소	미확인	경기	NaN	P85709
3016781	9078401016	907	법인	4	한독상공회의소	미확인	서울	NaN	S94110
3016782	9088400169	908	법인	4	조지아센추럴대학교서울사무소	미확인	서울	NaN	N75290
3016783 rows × 9 columns									

## 2. XGBoost / LightGBM/ RandomForest

```
# 6. GridSearchCV for XGBoost
param_grid_xgb = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5, 7],
    'tree_method': ['gpu_hist'],
    'predictor': ['gpu_predictor']
}
xgb_model = xgb.XGBClassifier(use_label_encoder=False, eval_metric='mlogloss', verbosity=0)
start_xgb = time.time()
gs_xgb = GridSearchCV(xgb_model, param_grid_xgb, scoring='f1_macro', cv=cv_splitter, verbose=1, n_jobs=-1)
gs_xgb.fit(X_train, y_train)
time_xgb = time.time() - start_xgb
acc_xgb, f1_xgb = evaluate_model("XGBoost", gs_xgb.best_estimator_, X_test, y_test)
results.append(["XGBoost_GridSearchCV", acc_xgb, f1_xgb, time_xgb])

# XGBoost GridSearchCV 객체 저장
joblib.dump(gs_xgb, "/content/drive/MyDrive/cloud/gs_xgb_gridsearchcv.pkl")
print("gs_xgb 객체가 gs_xgb_gridsearchcv.pkl 파일로 저장되었습니다.")
# 51min
```

## 2. XGBoost / LightGBM/ RandomForest

```
# 7. GridSearchCV for LightGBM
param_grid_lgb = {
    'n_estimators': [100, 200],
    'max_depth': [5, 7, 9]
}
lgb_model = lgb.LGBMClassifier(verbose=-1)
start_lgb = time.time()
gs_lgb = GridSearchCV(lgb_model, param_grid_lgb, scoring='f1_macro', cv=cv_splitter, verbose=1, n_jobs=-1)
gs_lgb.fit(X_train, y_train)
time_lgb = time.time() - start_lgb
acc_lgb, f1_lgb = evaluate_model("LightGBM", gs_lgb.best_estimator_, X_test, y_test)
results.append(["LightGBM_GridSearchCV", acc_lgb, f1_lgb, time_lgb])

# LightGBM GridSearchCV 객체 저장
joblib.dump(gs_lgb, "/content/drive/MyDrive/cloud/gs_lgb_gridsearchcv.pkl")
print("gs_lgb 객체가 gs_lgb_gridsearchcv.pkl 파일로 저장되었습니다.")
# 300min
```



## 2. XGBoost / LightGBM/ RandomForest

```
# 8. GridSearchCV for RandomForest
#param_grid_rf = {
#    'n_estimators': [100, 200],
#    'max_depth': [10, 20, None]
#}
# System Crash를 방지하기 위해 HyperParam 축소
param_grid_rf = {
    'n_estimators': [100],
    'max_depth': [10, 20]
}
rf_model = RandomForestClassifier()
start_rf = time.time()
#gs_rf = GridSearchCV(rf_model, param_grid_rf, scoring='f1_macro', cv=cv_splitter, verbose=1, n_jobs=-1)
# System Crash를 방지하기 위해 n_jobs를 1로 고정
gs_rf = GridSearchCV(rf_model, param_grid_rf, scoring='f1_macro', cv=cv_splitter, verbose=1, n_jobs=1)
gs_rf.fit(X_train, y_train)
time_rf = time.time() - start_rf
acc_rf, f1_rf = evaluate_model("RandomForest", gs_rf.best_estimator_, X_test, y_test)
results.append(["RandomForest_GridSearchCV", acc_rf, f1_rf, time_rf])

# RandomForest GridSearchCV 객체 저장
joblib.dump(gs_rf, "/content/drive/MyDrive/cloud/gs_rf_gridsearchcv.pkl")
print("gs_rf 객체가 gs_rf_gridsearchcv.pkl 파일로 저장되었습니다.")
# 13min
```

## 2. XGBoost / LightGBM/ RandomForest

```
# 각 모델의 테스트셋 성능 측정
y_pred_xgb = gs_xgb_loaded.best_estimator_.predict(X_test)
acc_xgb = accuracy_score(y_test, y_pred_xgb)
f1_xgb = f1_score(y_test, y_pred_xgb, average='macro')

y_pred_lgb = gs_lgb_loaded.best_estimator_.predict(X_test)
acc_lgb = accuracy_score(y_test, y_pred_lgb)
f1_lgb = f1_score(y_test, y_pred_lgb, average='macro')

y_pred_rf = gs_rf_loaded.best_estimator_.predict(X_test)
acc_rf = accuracy_score(y_test, y_pred_rf)
f1_rf = f1_score(y_test, y_pred_rf, average='macro')

# 요약 테이블 생성
summary_df = pd.DataFrame({
    "Model": ["XGBoost_GridSearchCV", "LightGBM_GridSearchCV", "RandomForest_GridSearchCV"],
    "Accuracy": [acc_xgb, acc_lgb, acc_rf],
    "F1_Score": [f1_xgb, f1_lgb, f1_rf]
})

print("=" * 50)
print("저장된 pkl 파일을 활용한 모델별 성능 요약:")
print(summary_df.sort_values(by="Accuracy", ascending=False))

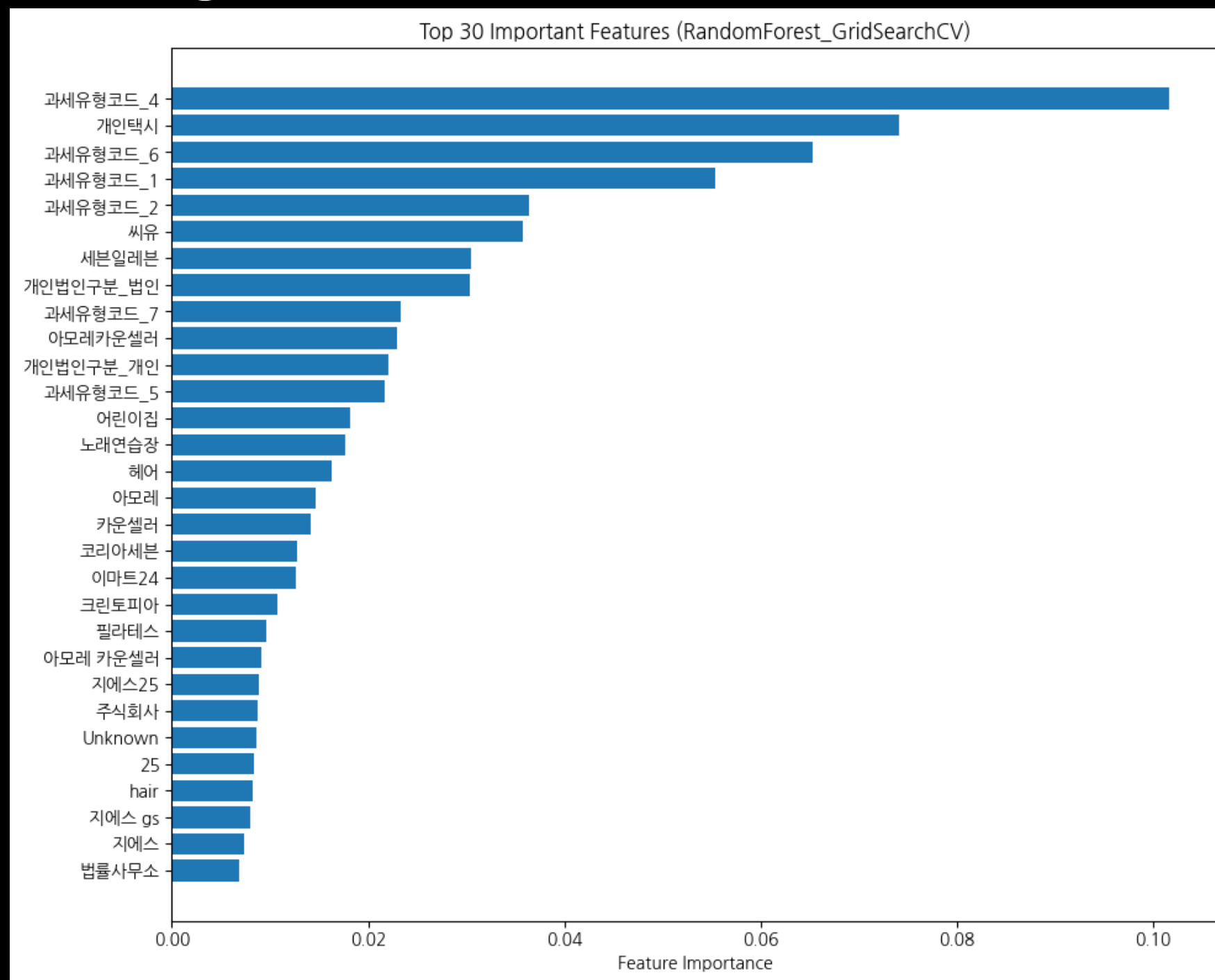
# 가장 정확도가 높은 모델 찾기
best_idx = summary_df["Accuracy"].idxmax()
best_row = summary_df.loc[best_idx]
print("\n가장 성능이 좋은 모델(정확도 기준):")
print(best_row)
```

```
=====
저장된 pickle 파일을 활용한 모델별 성능 요약:

              Model  Accuracy  F1_Score
2  RandomForest_GridSearchCV  0.221889  0.037752
1          LightGBM_GridSearchCV  0.127676  0.004027
0          XGBoost_GridSearchCV  0.025940  0.000660

가장 성능이 좋은 모델(정확도 기준):
Model          RandomForest_GridSearchCV
Accuracy              0.221889
F1_Score              0.037752
Name: 2, dtype: object
```

## 2. XGBoost / LightGBM/ RandomForest



## 2. XGBoost / LightGBM/ RandomForest

산업분류코드를 세세분류까지에서  
중분류까지로 낮춰보면 어떨까?

수준	명칭	코드 자릿수	예시	설명
대분류	Section	1자리 (영문자)	C	제조업
중분류	Division	2자리 (숫자)	26	전자부품, 컴퓨터, 영상, 음향 및 통신장비 제조업
소분류	Group	3자리	261	전자부품 제조업
세분류	Class	4자리	2611	전자집적회로 제조업
세세분류	Subclass	5~6자리	26111	논리집적회로 제조업

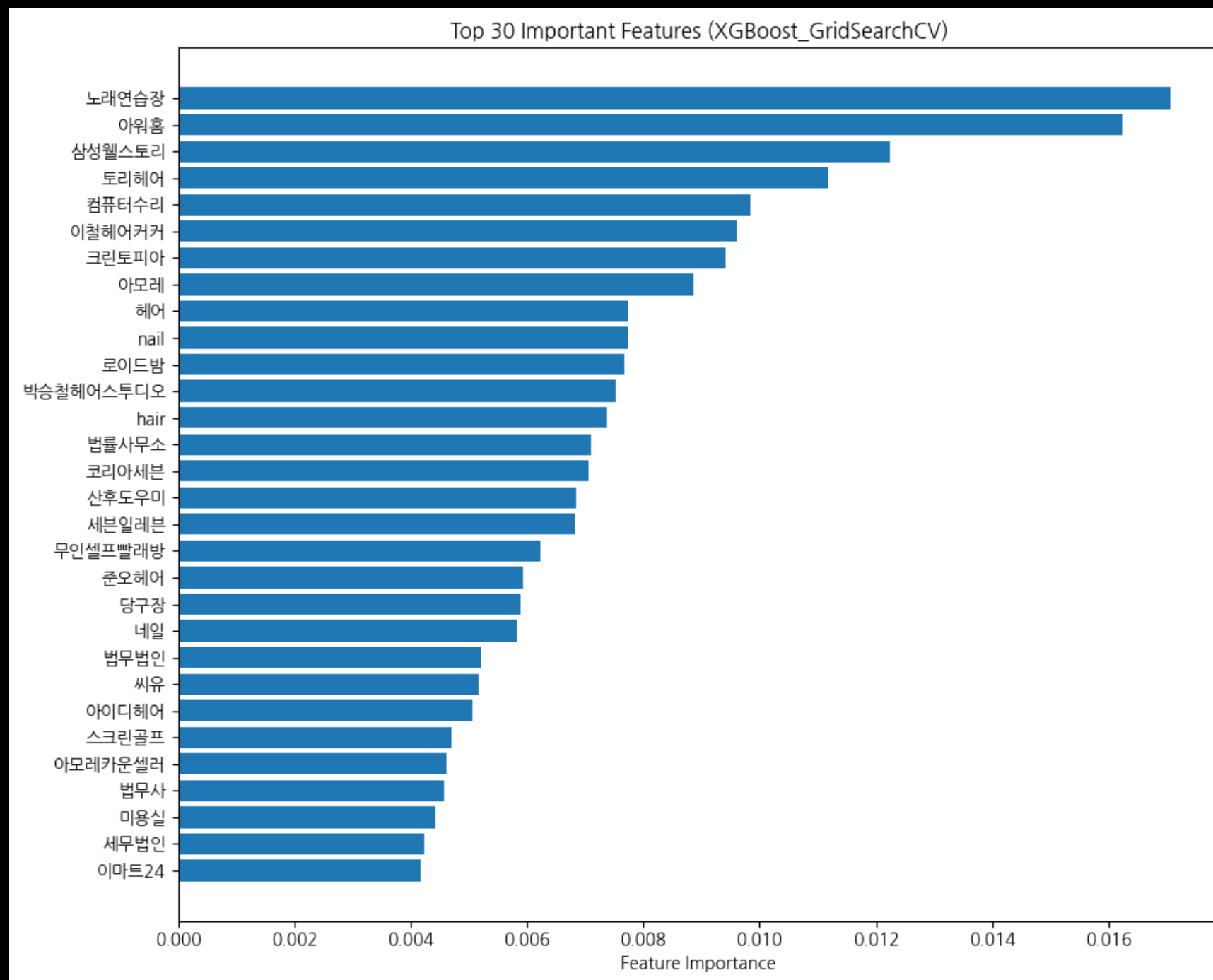
```
=====
저장된 pkl 파일을 활용한 모델별 성능 요약:
      Model  Accuracy  F1_Score
2  RandomForest_GridSearchCV  0.221889  0.037752
1    LightGBM_GridSearchCV  0.127676  0.004027
0    XGBoost_GridSearchCV  0.025940  0.000660

가장 성능이 좋은 모델(정확도 기준):
Model      RandomForest_GridSearchCV
Accuracy           0.221889
F1_Score           0.037752
Name: 2, dtype: object
```

```
=====
저장된 pkl 파일을 활용한 모델별 성능 요약:
      Model  Accuracy  F1_Score
0    XGBoost_GridSearchCV  0.493946  0.215355
2  RandomForest_GridSearchCV  0.375123  0.068387
1    LightGBM_GridSearchCV  0.045973  0.011457

가장 성능이 좋은 모델(정확도 기준):
Model      XGBoost_GridSearchCV
Accuracy           0.493946
F1_Score           0.215355
Name: 0, dtype: object
```

## 2. XGBoost / LightGBM/ RandomForest



# 3. MLP(using PyTorch)

상호명 + 취급품목 → TF-IDF임베딩 (3000개 특징)  
범주형 피쳐 5개 → OneHotEncoder  
희소행렬 합치기 (scipy.sparse.hstack → TensorDataset  
변환)  
StratifiedKFold 5-fold로 평가 분리(각 폴드에 클래스가 균형 있게  
분포되도록)  
레이블 인코딩으로 산업분류코드 정수화

TF (Term Frequency, 단어 빈도)  
특정 문서에서 단어가 얼마나 자주 등장했는지

IDF (Inverse Document Frequency, 역문서 빈도)  
자주 등장하는 단어일수록 중요도가 낮다

```
# Label encoding
y_labels = df[target_col].astype(str)
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y_labels)

# 통합 텍스트 (상호명 + 취급품목)
df['통합텍스트'] = df['상호명'] + ' ' + df['취급품목']

# TF-IDF 벡터 저장 및 재사용
vectorizer_path = "tfidf_vectorizer.pkl"
if os.path.exists(vectorizer_path):
    with open(vectorizer_path, "rb") as f:
        vectorizer = pickle.load(f)
    X_text = vectorizer.transform(df['통합텍스트'])
else:
    vectorizer = TfidfVectorizer(ngram_range=(1, 2), max_features=3000)
    X_text = vectorizer.fit_transform(df['통합텍스트'])
    with open(vectorizer_path, "wb") as f:
        pickle.dump(vectorizer, f)
```

### 3. MLP(using PyTorch)

활성화 함수 : ReLU

3 Layer : Input – Hidden – Output

nn.CrossEntropyLoss()를 사용할 예정이므로 **softmax** 생략

```
# Model
class MLPClassifier(nn.Module):
    def __init__(self, input_dim, output_dim):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(input_dim, 512),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Linear(256, output_dim)
        )

    def forward(self, x):
        return self.model(x)
```

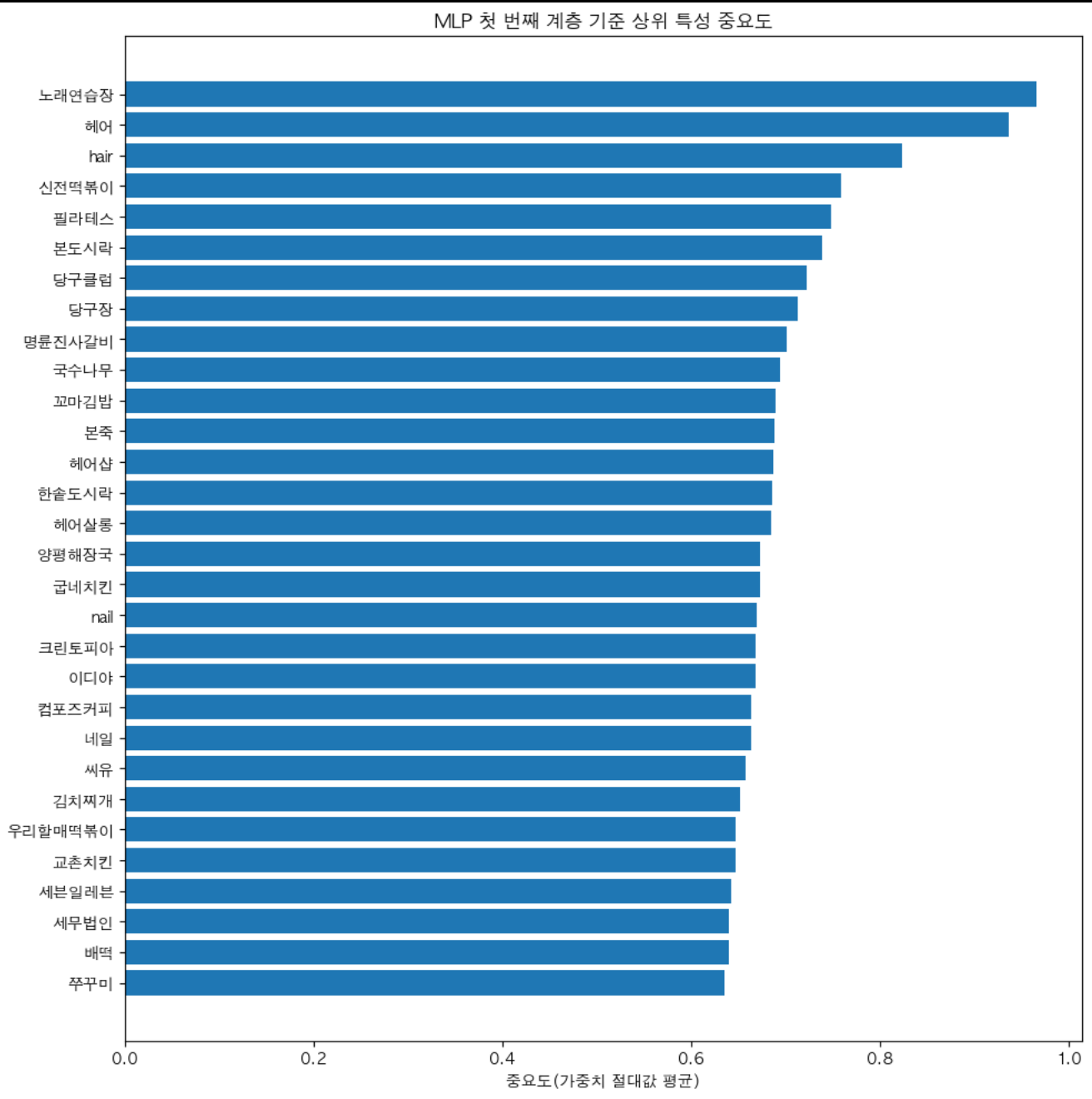
### 3. MLP(using PyTorch)

**StratifiedKFold**: 각 fold마다 y 클래스 비율을 유지  
5개의 폴드에서 각각 모델을 새로 학습 → 일반화 성능  
평가에 유리

```
# Cross Validation
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
fold accuracies = []
fold_f1s = []
```



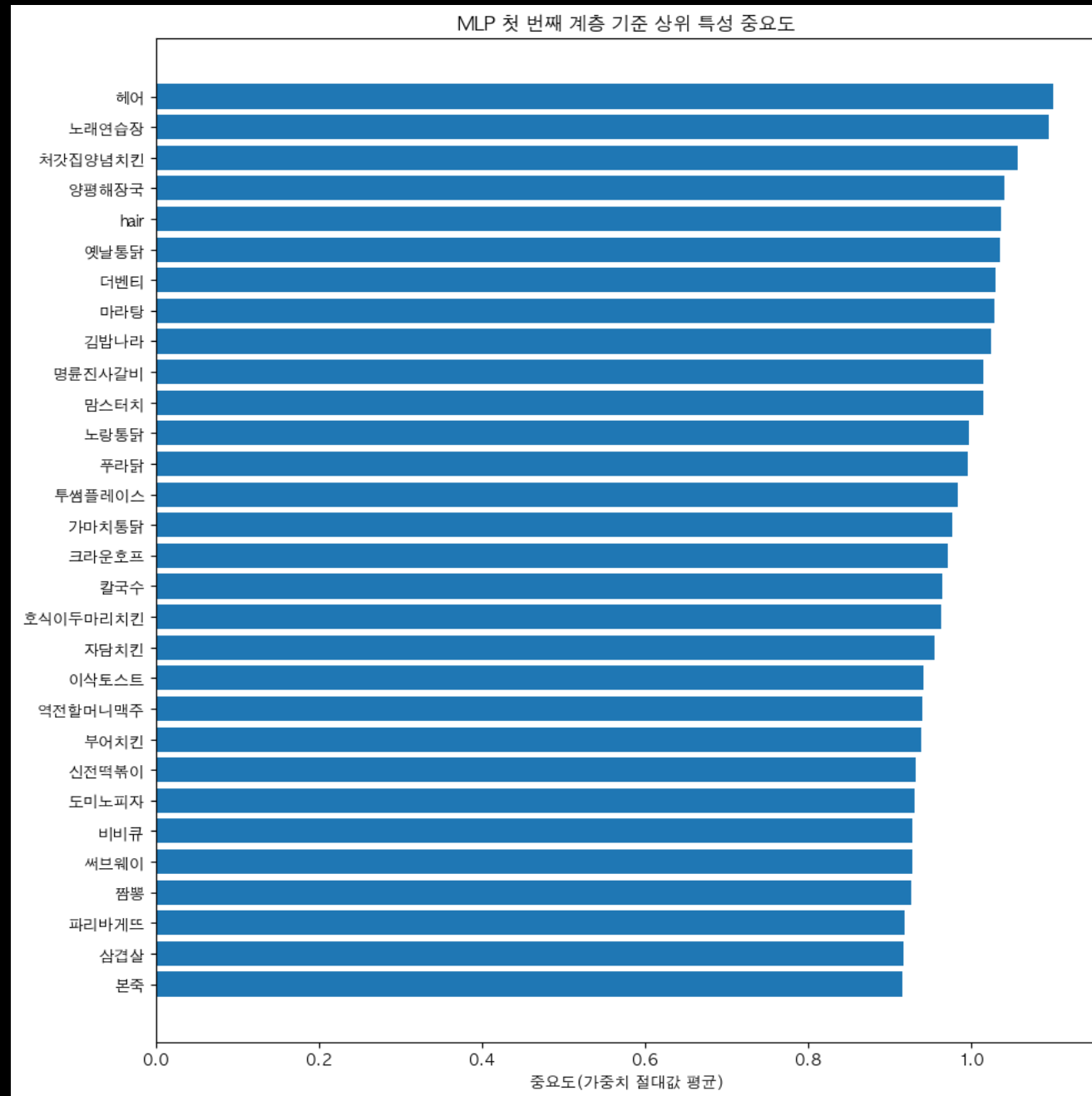
# 3. MLP(using PyTorch)



MLP에서는 Feature Importance를 제공하지 않으므로  
첫번째 Linear 계층의 가중치 절대값 평균 사용

```
==== Cross Validation Results ====  
Avg Accuracy: 0.3124  
Avg F1(Macro): 0.1082
```

### 3. MLP(using PyTorch)



산업분류코드를 세세분류까지에서  
중분류까지로 낮춰보면 어떨까?

==== Cross Validation Results ====

Avg Accuracy: 0.3124

Avg F1(Macro): 0.1082

==== Cross Validation Results ====

Avg Accuracy: 0.5046

Avg F1(Macro): 0.1969

## 4. Post-Mortem

MLP가 전통적인 ML방식보다 빠르고 높은 결과를 얻을 수 있었음  
RandomForest 사용시 높은 메모리 사용으로 인해 HyperParam 튜닝이 어려움  
GPU 활용이 가능하다면 GPU활용이 가능한 알고리즘을 적극 활용해보자

산업분류코드를 타 카드사의 데이터를 활용한 한계를 극복  
(회사 내부에서 재학습 필요 & 학습 데이터 보충)