0️⃣

Introduction to Feature Selection

캐글 노트 https://www.kaggle.com/willkoehrsen/introduction-to-feature-selection

Introduction : Feature Selection

- 이전 커널들에서 피처엔지니어링 진행한 데이터 이용
    - Part1
    - Part2

- feature selection에 사용할 방법
    - 다중공선성 있는 피처 제거
    - 널값이 특정 퍼센트 이상으로 많은 피처 제거
    - 모델로부터 feature importance 이용해 중요한 피처만 사용

  → 이렇게 줄인 피처들의 gradient boosting machine(여기서는 LGBM 사용)에서의 성능 측정

```
# pandas and numpy for data manipulation
import pandas as pd
import numpy as np

# featuretools for automated feature engineering
import featuretools as ft

# matplotlit and seaborn for visualizations
import matplotlib.pyplot as plt
plt.rcParams['font.size'] = 22
import seaborn as sns

# Suppress warnings from pandas
import warnings
warnings.filterwarnings('ignore')

# modeling
import lightgbm as lgb

# utilities
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import LabelEncoder

# memory management
import gc
```

- Feature engineering에서 만든 피처들 확인
    - `bureau`, `previous`에 각각 혹은 둘 다 있는 칼럼들 확인

      → set() 이용
    - `bureau` / `previous` / `orignial_features`란 이름으로 칼럼들 각각 저장

```
# Bureau only features
bureau_features = list(set(bureau_columns) - set(previous_columns))
# Previous only features
previous_features = list(set(previous_columns) - set(bureau_columns))
# Original features will be in both datasets
original_features = list(set(previous_columns) & set(bureau_columns))
```

- 중복되는 row들 없이 데이터프레임들 합쳐 train, test data 생성
  - `train_previous` 에서 `previous features` 만 가져와 `train_bureu` 와 합침
  - test data도 마찬가지

```
train_labels = train_bureau['TARGET']
previous_features.append('SK_ID_CURR')
train_ids = train_bureau['SK_ID_CURR']
test_ids = test_bureau['SK_ID_CURR']

# Merge the dataframes avoiding duplicating columns by subsetting train_previous
train = train_bureau.merge(train_previous[previous_features], on = 'SK_ID_CURR')
test = test_bureau.merge(test_previous[previous_features], on = 'SK_ID_CURR')
```

- 피처들 one-hot encoding 실행
  - 이후 train, test data 칼럼들을 순서에 맞게 정렬
    - align() 함수 이용

      → train, test data 간 칼럼들 동일한지 알 수 있음

```
# One hot encoding
train = pd.get_dummies(train)
test = pd.get_dummies(test)

# Match the columns in the dataframes
train, test = train.align(test, join = 'inner', axis = 1)
print('Training shape: ', train.shape)
print('Testing shape: ', test.shape)
```

- 모델링에 필요없는 피처들 제거
  - train, test data에 모델링에 사용하지 않을 client id 피처들이 있음 (`SK_ID_` prefix)

    → 확인 후 drop

```
cols_with_id = [x for x in train.columns if 'SK_ID_CURR' in x]
cols_with_bureau_id = [x for x in train.columns if 'SK_ID_BUREAU' in x]
cols_with_previous_id = [x for x in train.columns if 'SK_ID_PREV' in x]

train = train.drop(columns = cols_with_id)
test = test.drop(columns = cols_with_id)
```

결과적으로 총 **1416**개의 칼럼들을 갖고있음

1. 다중공선성 제거

- 다중공선성
  - 모델의 훈련 성능 감소
  - 모델의 해석 가능성 감소
  - test data에 대한 일반화 수행도 감소

- Pearson correlation coefficient가 0.9 이상인 피처들 확인

```
# Threshold for removing correlated variables
threshold = 0.9

# Absolute value correlation matrix
corr_matrix = train.corr().abs()
corr_matrix.head()
```

```
# Upper triangle of correlations
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))

# Select columns with correlations above threshold
to_drop = [column for column in upper.columns if any(upper[column] > threshold)]
```

- 다중공선성을 가진 피처들 제거

```
train = train.drop(columns = to_drop)
test = test.drop(columns = to_drop)
```

총 538개의 칼럼들 제거 → 814개의 칼럼들 남음

다중공선성 제거하기 이전의 train, test data 불러와 다음 과정 실행

2. 결측값 제거

- 임의로 정한 threshold 이상의 결측치 퍼센트지를 갖는 칼럼들 제거
  → 여기서는 75% 이상의 결측치 퍼센트지가 존재할 경우 제거

```
# Train missing values (in percent)
train_missing = (train.isnull().sum() / len(train)).sort_values(ascending = False)

# Test missing values (in percent)
test_missing = (test.isnull().sum() / len(test)).sort_values(ascending = False)


# Identify missing values above threshold
train_missing = train_missing.index[train_missing > 0.75]
test_missing = test_missing.index[test_missing > 0.75]

all_missing = list(set(set(train_missing) | set(test_missing)))
print('There are %d columns with more than 75%% missing values' % len(all_missing))
```

→ 17개의 칼럼들 제거

- 칼럼들 drop하고 one-hot encoding & align 진행

```
# Nalign하는 와중에 제거되기 때문에 미리 저장
train_labels = train["TARGET"]
train_ids = train['SK_ID_CURR']
test_ids = test['SK_ID_CURR']

train = pd.get_dummies(train.drop(columns = all_missing))
test = pd.get_dummies(test.drop(columns = all_missing))

train, test = train.align(test, join = 'inner', axis = 1)

# 모델링에 사용하지 않을 client id 피처 제거
train = train.drop(columns = ['SK_ID_CURR'])
test = test.drop(columns = ['SK_ID_CURR'])
```

총 845개의 칼럼들 존재

---

3. Feature Selection through Feature Importances

- 자동으로 시행하는 방법
    - Recursive Feature Elimination method

- 낮은 feature importance를 가진 피처들을 직접 제거
    - feature importance가 0인 피처들 확인하는 함수
        - LightGBM library 이용
        - 오버피팅 피하기 위해 2번 피팅 진행
        - 코드

```
def identify_zero_importance_features(train, train_labels, iterations = 2):

    # Initialize an empty array to hold feature importances
    feature_importances = np.zeros(train.shape[1])

    # Create the model with several hyperparameters
    model = lgb.LGBMClassifier(objective='binary', boosting_type = 'goss', n_estimators = 10000, class_weight = 'balanced')

    # Fit the model multiple times to avoid overfitting
    for i in range(iterations):

        # Split into training and validation set
        train_features, valid_features, train_y, valid_y = train_test_split(train, train_labels, test_size = 0.25, random_sta

        # Train using early stopping
        model.fit(train_features, train_y, early_stopping_rounds=100, eval_set = [(valid_features, valid_y)],
                  eval_metric = 'auc', verbose = 200)

        # Record the feature importances
        feature_importances += model.feature_importances_ / iterations

    feature_importances = pd.DataFrame({'feature': list(train.columns), 'importance': feature_importances}).sort_values('impo

    # Find the features with zero importance
    zero_features = list(feature_importances[feature_importances['importance'] == 0.0]['feature'])
    print('\nThere are %d features with 0.0 importance' % len(zero_features))

    return zero_features, feature_importances
```

        - 함수 실행 결과

          → zero_features : 중요도가 0인 피처들

          → features_importances : 피처들의 중요도

```
zero_features,feature_importances=identify_zero_importance_features(train, train_labels, iterations = 2)
```

    - feature importance 평균을 구한 후 중요도가 0인 피처 확인

```
# Make sure to average feature importances!
feature_importances = feature_importances / 2
feature_importances = pd.DataFrame({'feature': list(train.columns), 'importance': feature_importances}).sort_values('importan

# Find the features with zero importance
zero_features = list(feature_importances[feature_importances['importance'] == 0.0]['feature'])
print('There are %d features with 0.0 importance' % len(zero_features))
```

→ 총 271개의 피처들의 중요도가 0

→ 이때 모델링할 때 gradient boosting machine에서 중요도가 0인 피처들은 자동으로 사라짐 (지금은 아님!! 수동으로 하는 중이니까)

- 결과 plotting 함수
    - feature importance 정규화 실행
    - feature importance의 top15 피처들 막대 그래프
    - feature importance의 누적합이 threshold에 도달하기 위해 필요한 피처들 수
    - 코드

```
def plot_feature_importances(df, threshold = 0.9):

    plt.rcParams['font.size'] = 18

    # Sort features according to importance
    df = df.sort_values('importance', ascending = False).reset_index()

    # Normalize the feature importances to add up to one
    df['importance_normalized'] = df['importance'] / df['importance'].sum()
    df['cumulative_importance'] = np.cumsum(df['importance_normalized'])

    # Make a horizontal bar chart of feature importances
    plt.figure(figsize = (10, 6))
    ax = plt.subplot()

    # Need to reverse the index to plot most important on top
    ax.barh(list(reversed(list(df.index[:15]))),
            df['importance_normalized'].head(15),
            align = 'center', edgecolor = 'k')

    # Set the yticks and labels
    ax.set_yticks(list(reversed(list(df.index[:15]))))
    ax.set_yticklabels(df['feature'].head(15))

    # Plot labeling
    plt.xlabel('Normalized Importance'); plt.title('Feature Importances')
    plt.show()

    # Cumulative importance plot
    plt.figure(figsize = (8, 6))
    plt.plot(list(range(len(df))), df['cumulative_importance'], 'r-')
    plt.xlabel('Number of Features'); plt.ylabel('Cumulative Importance');
    plt.title('Cumulative Feature Importance');
    plt.show();

    importance_index = np.min(np.where(df['cumulative_importance'] > threshold))
    print('%d features required for %0.2f of cumulative importance' % (importance_index + 1, threshold))

    return df
```

    - 함수 결과

```
norm_feature_importances = plot_feature_importances(feature_importances)
```

→ 누적 importance가 0.9 이상이 되려면 288개의 칼럼들 필요

- 중요도가 0인 피처들 제거

```
train = train.drop(columns = zero_features)
test = test.drop(columns = zero_features)
```

→ 총 573개의 칼럼들 남음

- 재확인을 위해 다시 한번 함수 실행

```
second_round_zero_features, feature_importances = identify_zero_importance_features(train, train_labels)
```

→ 중요도가 0인 피처들 없다는 것 확인

- 누적 importance가 0.95인 피처들 확인
    - 앞서 나온 plotting 함수 사용
    - 코드

```
norm_feature_importances = plot_feature_importances(feature_importances, threshold = 0.95)
```

→ 총 360개의 피처가 필요함

- 누적 importance가 0.95 이상이 되도록 하는 피처들만 뽑아 저장
    - 혹시 모를 훈련도 손상을 막기 위해 원본 데이터셋은 건드리지 않음

```
# Threshold for cumulative importance
threshold = 0.95

# Extract the features to keep
features_to_keep = list(norm_feature_importances[norm_feature_importances['cumulative_importance'] < threshold]['feature'])

# Create new datasets with smaller features
train_small = train[features_to_keep]
test_small = test[features_to_keep]

# train, test data 저장
train_small['TARGET'] = train_labels
train_small['SK_ID_CURR'] = train_ids
test_small['SK_ID_CURR'] = test_ids

train_small.to_csv('m_train_small.csv', index = False)
test_small.to_csv('m_test_small.csv', index = False)
```

---

4. Test New Feature sets

- 전체 데이터셋에 대한 피처 중요도 산출 함수
    - five-fold cross validation 이용
    - 코드

```
def model(features, test_features, encoding = 'ohe', n_folds = 5):

    # Extract the ids
    train_ids = features['SK_ID_CURR']
    test_ids = test_features['SK_ID_CURR']

    # Extract the labels for training
    labels = features['TARGET']

    # Remove the ids and target
    features = features.drop(columns = ['SK_ID_CURR', 'TARGET'])
    test_features = test_features.drop(columns = ['SK_ID_CURR'])


    # One Hot Encoding
    if encoding == 'ohe':
        features = pd.get_dummies(features)
        test_features = pd.get_dummies(test_features)
```

```
        # Align the dataframes by the columns
        features, test_features = features.align(test_features, join = 'inner', axis = 1)

        # No categorical indices to record
        cat_indices = 'auto'

    # Integer label encoding
    elif encoding == 'le':

        # Create a label encoder
        label_encoder = LabelEncoder()

        # List for storing categorical indices
        cat_indices = []

        # Iterate through each column
        for i, col in enumerate(features):
            if features[col].dtype == 'object':
                # Map the categorical features to integers
                features[col] = label_encoder.fit_transform(np.array(features[col].astype(str)).reshape((-1,)))
                test_features[col] = label_encoder.transform(np.array(test_features[col].astype(str)).reshape((-1,)))

                # Record the categorical indices
                cat_indices.append(i)

    # Catch error if label encoding scheme is not valid
    else:
        raise ValueError("Encoding must be either 'ohe' or 'le'")

    print('Training Data Shape: ', features.shape)
    print('Testing Data Shape: ', test_features.shape)

    # Extract feature names
    feature_names = list(features.columns)

    # Convert to np arrays
    features = np.array(features)
    test_features = np.array(test_features)

    # Create the kfold object
    k_fold = KFold(n_splits = n_folds, shuffle = False, random_state = 50)

    # Empty array for feature importances
    feature_importance_values = np.zeros(len(feature_names))

    # Empty array for test predictions
    test_predictions = np.zeros(test_features.shape[0])

    # Empty array for out of fold validation predictions
    out_of_fold = np.zeros(features.shape[0])

    # Lists for recording validation and training scores
    valid_scores = []
    train_scores = []

    # Iterate through each fold
    for train_indices, valid_indices in k_fold.split(features):

        # Training data for the fold
        train_features, train_labels = features[train_indices], labels[train_indices]
        # Validation data for the fold
        valid_features, valid_labels = features[valid_indices], labels[valid_indices]

        # Create the model
        model = lgb.LGBMClassifier(n_estimators=10000, objective = 'binary', boosting_type='goss',
                                   class_weight = 'balanced', learning_rate = 0.05,
                                   reg_alpha = 0.1, reg_lambda = 0.1, n_jobs = -1, random_state = 50)

        # Train the model
        model.fit(train_features, train_labels, eval_metric = 'auc',
                  eval_set = [(valid_features, valid_labels), (train_features, train_labels)],
                  eval_names = ['valid', 'train'], categorical_feature = cat_indices,
                  early_stopping_rounds = 100, verbose = 200)

        # Record the best iteration
        best_iteration = model.best_iteration_

        # Record the feature importances
        feature_importance_values += model.feature_importances_ / k_fold.n_splits

        # Make predictions
        test_predictions += model.predict_proba(test_features, num_iteration = best_iteration)[:, 1] / k_fold.n_splits

        # Record the out of fold predictions
        out_of_fold[valid_indices] = model.predict_proba(valid_features, num_iteration = best_iteration)[:, 1]
```

```
        # Record the best score
        valid_score = model.best_score_['valid']['auc']
        train_score = model.best_score_['train']['auc']

        valid_scores.append(valid_score)
        train_scores.append(train_score)

        # Clean up memory
        gc.enable()
        del model, train_features, valid_features
        gc.collect()

    # Make the submission dataframe
    submission = pd.DataFrame({'SK_ID_CURR': test_ids, 'TARGET': test_predictions})

    # Make the feature importance dataframe
    feature_importances = pd.DataFrame({'feature': feature_names, 'importance': feature_importance_values})

    # Overall validation score
    valid_auc = roc_auc_score(labels, out_of_fold)

    # Add the overall scores to the metrics
    valid_scores.append(valid_auc)
    train_scores.append(np.mean(train_scores))

    # Needed for creating dataframe of validation scores
    fold_names = list(range(n_folds))
    fold_names.append('overall')

    # Dataframe of validation scores
    metrics = pd.DataFrame({'fold': fold_names,
                            'train': train_scores,
                            'valid': valid_scores})

    return submission, feature_importances, metrics
```

- 위 feature selection 과정을 거친 train,test data로 실행
    - 다중공선성이 0.9 이상인 피처 drop
    - 결측치가 80% 이상인 피처 drop
    - 중요도가 0인 피처 drop

```
train['TARGET'] = train_labels
train['SK_ID_CURR'] = train_ids
test['SK_ID_CURR'] = test_ids

submission, feature_importances, metrics = model(train, test)
```

→ 중요도가 0.783

- 누적 중요도가 95% 되도록 하는 피처들 data로 실행
    - 앞서 만든 m_train_small, m_test_small data 사용

```
submission_small, feature_importances_small, metrics_small = model(train_small, test_small)
```

→ 중요도가 0.782

---

5. Other Options for Dimensionality Reduction

- 차원축소 방법
    - PCA (Principle Components Analysis)
        - model interpretability에서 신경쓰지 않는 피처들 개수 줄임

        - 데이터가 가우시안 분포라고 가정
    - ICA (Independent Components Analysis)
        - 변수들의 physical meaning 없앰
        - 데이터의 가장 독립적인 차원들 보존
    - Maniford learning
        - non-linear dimensionality reduction
        - 차원축소보다는 T-SNE나 LLE처럼 저차원 시각화에 사용

---

6. Conclusions

- 노트북에서 사용한 feature selection 방법들
    - 다중공선성이 0.9 이상인 변수들 제거
    - 결측치가 0.75% 이상인 변수들 제거
    - gradient boosting machine에 의해 중요도가 0이라고 판단된 변수들 제거
      → 536개 변수들 + AUC ROC score 0.7838
    - (옵션) 누적 중요도 95%를 차지하는 변수들만 가져옴
      → 342개 변수들 + AUC ROC score 0.7482