Feedback

- 새로 추가한 데이터 컬럼에 대해 상관관계 분석
- Date, time 분포를 찍어서 새로 추가한 이유 넣기

인공지능 기술 기반의 이상금융거래 탐지

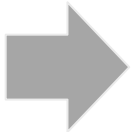
팀장 신여명

목차

01. 아이디어

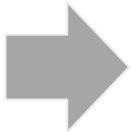
편리한 신용카드
및 결제 환경

도난, 오용,
보이스피싱 피해



Fraud Detection System
이상금융거래탐지시스템

전자금융거래 데이터
금융사기 패턴 탐지, 예방



다양한 기술에 의한
사기, 보안에 취약

01. 개발환경


python


NumPy


matplotlib


pandas


GitHub


scikit
learn


K
Keras


TensorFlow

02. 주제선정

LG히다찌, 하나은행 의심거래보고 고도화위한 머신러닝 시스템 구축

2019.11.18 14:21:14 / 이상일 2401@ddaily.co.kr

농협카드 신인식 사장 '금융사고 제로' 특명…이상거래 탐지 3중 시스템 박차

서상혁 기자 입력 2020.09.11 16:38

(단독)카카오뱅크·케이뱅크 보안 구멍...사기이용계좌 1년새 2배 급증

편의성 등이 범죄악용 키워...은행측 "범죄동조탓" 주장

입력 : 2020-09-23 06:00:00 | 수정 : 2020-09-23 06:00:00

1. 머신러닝 딥러닝 기반의 이상 거래 검출 방법
2. 이상 거래 검출에 효과적인 알고리즘 연구

02. 데이터 수집

https://www.kaggle.com/c/ieee-fraud-detection

고객 거래 이상징후 감지 대회

- **Vesta**사의 실제 전자상거래 데이터를 이용

03. 기본 데이터

- 참조 날짜/시간
- 결제 금액(USD)
- 제품 코드
- 결제 카드 정보
- 주소 / 거리
- 구매자 / 수신자 e-메일 도메인
- 집계 컬럼(결제 카드와 관련된 주소 수)
- 날짜/시간 컬럼(거래 사이의 시간)
- 카드정보 일치 여부
- Vesta 에서 제공한 다른 특성들

- ID 정보
  - 네트워크 연결 정보 및 결제와 관련된 디지털 서명
  - 사용 전자기기

```python
# 컬럼 추가 (애플 제품이면 1 아니면 0)
apple = ['iOS Device','iPhone','MacOS']
all_data["DeviceInfo"] = all_data["DeviceInfo"].astype('str')
all_data['IsApple'] = all_data["DeviceInfo"].map(lambda x: 1 if any(a in x for a in apple) else 0)
print(all_data['IsApple'].value_counts())

# 컬럼 추가 (삼성 제품이면 1 아니면 0)
samsung = ['SAMSUNG', 'SM', "GT-"]
all_data["DeviceInfo"] = all_data["DeviceInfo"].astype('str')
all_data['IsSamsung'] = all_data["DeviceInfo"].map(lambda x: 1 if any(s in x for s in samsung) else 0)
print(all_data['IsSamsung'].value_counts())

# 컬럼 추가 (거래가 일어난 요일)
all_data['ts_day'] = np.floor((test['TransactionDT'] / (3600 * 24) - 1) % 7)
print(all_data['ts_day'].value_counts().sort_index())

# 컬럼 추가 (거래가 일어난 시간)
all_data['ts_hour'] = np.floor(test['TransactionDT'] / 3600) % 24
print(all_data['ts_hour'].value_counts().sort_index())
```

```
6.0    70846
Name: ts_day, dtype: int64
```

```
23.0    33655
Name: ts_hour, dtype: int64
```

03.

```
# 컬럼 추가 (이메일 도메인 카테고리 & 서비스 국가 카테고리)
emails = {'gmail': 'google', 'att.net': 'att', 'twc.com': 'spectrum',
          'scranton.edu': 'other', 'optonline.net': 'other', 'hotmail.co.uk': 'microsoft',
```

```
# 컬럼 추가 (브라우저 카테고리)
all_data['Browser'] = np.NaN

all_data.loc[all_data['id_31'].str.contains('chrome', na=False), 'Browser'] = 'Chrome'
all_data.loc[all_data['id_31'].str.contains('firefox', na=False), 'Browser'] = 'Firefox'
all_data.loc[all_data['id_31'].str.contains('safari', na=False), 'Browser'] = 'Safari'
all_data.loc[all_data['id_31'].str.contains('edge', na=False), 'Browser'] = 'Edge'
all_data.loc[all_data['id_31'].str.contains('ie', na=False), 'Browser'] = 'IE'
all_data.loc[all_data['id_31'].str.contains('samsung', na=False), 'Browser'] = 'Samsung'
all_data.loc[all_data['id_31'].str.contains('opera', na=False), 'Browser'] = 'Opera'
all_data['Browser'].fillna("others", inplace=True)
print(all_data['Browser'].value_counts())
```

```
    all_data[c[0]+'_email_country'] = all_data[c].map(lambda x: str(x).split('.')[-1])
    all_data[c[0]+'_email_country'] = all_data[c[0]+'_email_country'].map(lambda x: x if str(x) not in us_emails else 'us')
```

03. 데이터 전처리

1. 결측치 (NaN) 처리
   - 완전 제거법
   - 결측치가 80%에 가까운 경우 그 변수 자체를 제거
   - 결측치를 지우면서 데이터 자체에 편향(bias)가 생길 수 있음

```
#결측치가 80%에 가까운 경우 그 변수 자체를 제거
col_drop_list = []
for i in all_data.columns:
    missing_percent = all_data[i].isnull().sum() / len(all_data[i]) * 100
    if missing_percent > 80:
        col_drop_list.append(i)
```

```
data drop col # :  74
[before drop] all_data.shape:  (1097231, 441)
[after drop] all_data.shape:  (1097231, 367)
```

03. 데이터 전처리

2. 카테고리형 데이터 Label Encoding

- 사이킷런은 문자열 값을 입력 값으로 처리 하지 않기 때문에 숫자형으로 변환
- One Hot Encoding 사용시 열이 너무 많아져 Label Encoding 선택
- Label Encoding시 모델이 카테고리끼리 선형적 관계가 있다고 착각할 수도 있음

  => 트리모델 사용
- 결측치 자동 처리

```
# 범주형 데이터 label encoding
le = LabelEncoder()
for i in cat_cols:
    all_data[i] = le.fit_transform(list(all_data[i]))
```

03. 데이터 전처리

3. 수치형 칼럼 Minmax Scaling

```
# MinMaxScaler 수치형 칼럼
for i in num_cols:
    all_data[i] = (minmax_scale(all_data[i], feature_range=(0,1)))
```

- 칼럼 간에 범위 차이가 큼
- 모델이 범위가 큰 특정 칼럼을 더 중요하다고 인식할 수 있기 때문에 스케일링
- 각 열 기준으로 Minmax Scale
- Standard Scaler는 평균과 표준편차를 사용하기 때문에 데이터 분산 보존을 위해 Minmax Scaler 사용

03. 데이터 전처리

4. 수치형 칼럼 나머지 결측치 처리
   - 결측치를 drop 하지 않고 보존
   - 결측치 -1로 대체

```
# 결측치 처리 수치형만 -1
for i in num_cols:
    all_data[i].fillna(-1, inplace=True)
```

03. 데이터 전처리

5. Vxxx 칼럼 PCA 적용
   - 총 292개의 칼럼
   - 누적된 분산의 비율이 95%, 99%가 되는 주성분 축 / 차원을 선택

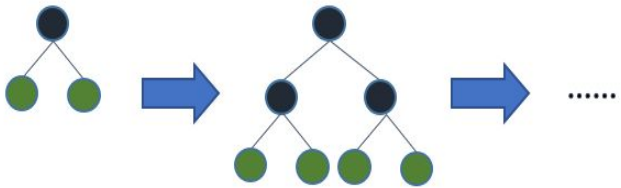
```python
# Vxxx 컬럼 PCA 적용
def PCA_change(df, cols, n_components, prefix='PCA_'):
    pca = PCA(n_components=n_components, random_state=77)
    pc = pca.fit_transform(df[cols])
    print('원래 차원(픽셀) 수 :', len(cols)) #292 컬럼
    print('선택한 차원(픽셀) 수 :', pca.n_components_) #0.95: 4 컬럼 / 0.99: 8컬럼

    pc_df = pd.DataFrame(pc, index=df.index)
    pc_df.rename(columns=lambda x: str(prefix)+str(x), inplace=True)

    df = df.drop(cols, axis=1)
    new_df = pd.concat([df,pc_df], axis=1)
    return new_df
```

```python
# PCA 95%
all_data2 = PCA_change(all_data, v_cols, n_components=0.95, prefix='PCA_V_')
# PCA 99%
all_data3 = PCA_change(all_data, v_cols, n_components=0.99, prefix='PCA_V_')
```

```
원래 차원(픽셀) 수 : 292
선택한 차원(픽셀) 수 : 4
원래 데이터 shape :  (1097231, 367)
PCA(0.95) 데이터 shape :  (1097231, 79)
```

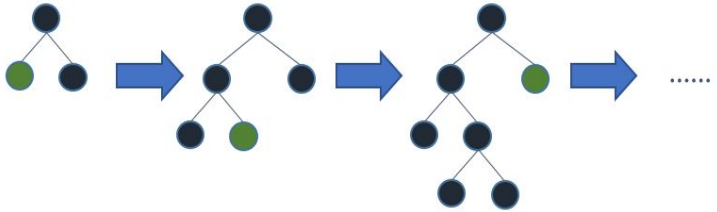
```
원래 차원(픽셀) 수 : 292
선택한 차원(픽셀) 수 : 8
원래 데이터 shape :  (1097231, 367)
PCA(0.99) 데이터 shape :  (1097231, 83)
```

04. 개념 설명



Level-wise tree growth

균형 트리 분할



Leaf-wise tree growth

리프 중심 트리분할

Gradient Boosting Algorithm (GBM)

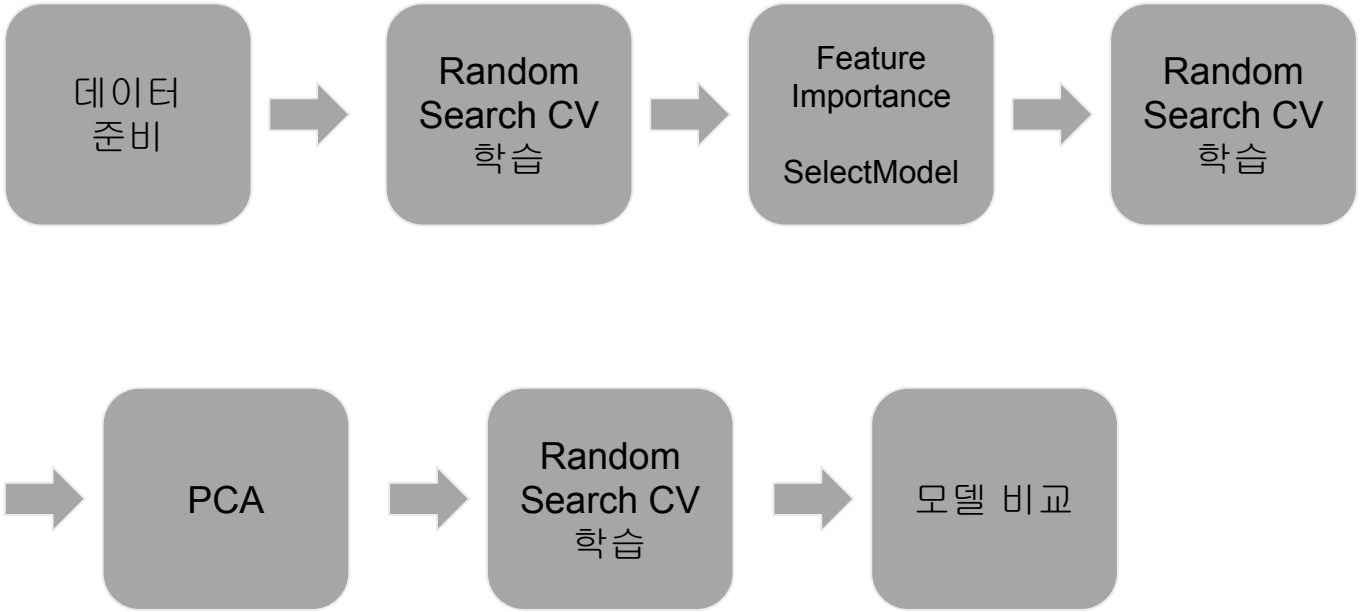
- 여러개의 Decision Tree 조합해서 결과
- 순차적으로 학습하면서 그 전 정보 (예측이 틀린 데이터) 를 바탕으로 다음 분류기를 만듬
- 약한 분류기를 모두 결합하여 강한 분류기

XGBoost vs LightGBM

- GBM의 단점 보완 (느림, 과적합 규제 부재)
- 더 빠른 학습시간 (최신)
- 트리 균형을 맞추지 않는 비대칭적 트리

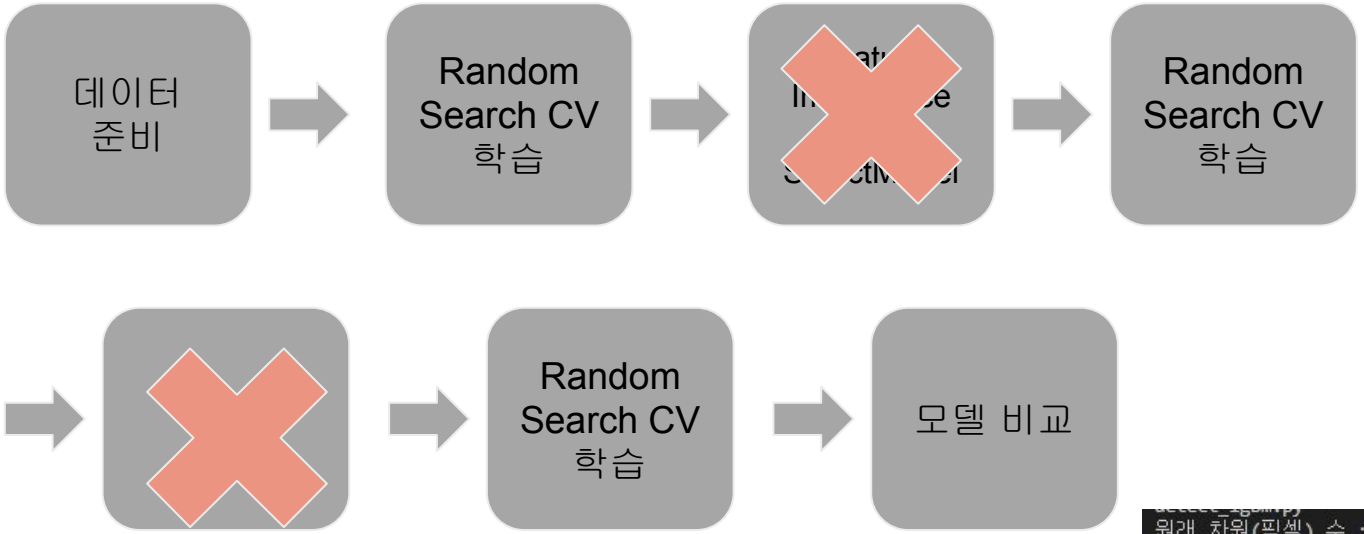
04. 모델링

- 약 40% 랜덤하게 데이터 뽑아서 모델 훈련



04. 모델링

- **SelectFromModel**를 사용해 **F.I** 순서대로 학습시켜본 결과 모든 피처를 다 사용하는게 가장 성능이 좋음
- **PCA (95%, 99%)** 적용했을 때 피처 수가 **1**로 줄어들면서 성능이 급격히 낮아짐




```
원래 차원(픽셀) 수 : 367
선택한 차원(픽셀) 수(0.95) : 1
```

```
최적 하이퍼 파라미터 :  {'subsample': 0.9, 'num_leaves': 50, 'n_estimators': 800, 'min_child_samples': 20, 'max_depth ': 15, 'learning_rate': 0.01, 'eval_metric': 'auc', 'colsample_bytree ': 0.7}
최고 AUC : 0.7319
acc :  0.9650033867690223
AUC : 73.3345%
```

05. 코드 설명

XGBoost

L1 : 불필요한 Feature에 대응하는 Weight를 정확히 0

L2 : 불필요한 Feature(이상치)에 대응하는 Weight를 0에 가깝게 만들 뿐, 0으로 만들지는 않음

```
params = {
    "n_estimators":[500, 800, 1000, 1200],
    "learning_rate":[0.01, 0.05, 0.001],
    "max_depth":range(3,10,3),
    "colsample_bytree":[0.5,0.6,0.7],
    "colsample_bylevel":[0.7,0.8,0.9],
    'min_child_weight':range(1,6,2),
    'subsample' :  [0.8,0.9],
    'objective' : ['binary:logistic'],
    }
scoring = {
    'AUC': 'roc_auc',
}

kfold = KFold(n_splits=5, shuffle=True, random_state=SEED)
xgb = xgboost.XGBClassifier(tree_method='gpu_hist',
                            predictor='gpu_predictor',
                            reg_alpha=0.15, #default 1,
                            reg_lamdba=0.85, #default 0,
                            random_state=SEED
                            )

model = RandomizedSearchCV(xgb, params, n_jobs=-1, cv=kfold, scoring=scoring,
                           n_iter=5, verbose=1, refit='AUC', return_train_score=True, random_state=SEED)

model.fit(x_train,y_train,
          eval_set=[(x_val, y_val)],
          eval_metric="auc",
          early_stopping_rounds=100,
          )

#cv 별 결과 출력
df = pd.DataFrame(model.cv_results_)
print("cv result : \n", df.iloc[:,[18,12]])
```

파라미터 세팅

평가방식 : auc로 최적의 파라미터 찾기

L1 Regularization (alpha)
L2 Regularization (lambda)
과적합 막기 위한 용도

Validation data 로 평가하면서 학습

조기 종료 제공

05. 코드 설명

XGBoost

```
result = model.predict(x_test)
acc = accuracy_score(y_test,result)
print("Accuracy : ", acc)

result2 = model.predict_proba(x_test)[:,1]
roc = roc_auc_score(y_test, result2)
print("AUC : %.4f%%"%(roc*100))

#roc curve 그리기
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, result2)
roc_auc = auc(false_positive_rate, true_positive_rate)
print("roc_auc :", roc_auc)
plt.figure(figsize=(10,10))
plt.title('Receiver Operating Characteristic')
plt.plot(false_positive_rate,true_positive_rate, color='red',label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],linestyle='--')
plt.axis('tight')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')


# top 20 feature importance by feature importance 값
def plot_feature_importances(model):
    plt.figure(figsize=(10,10))
    plt.title('Model Feature Importances')
    feature_names = index
    sorted_idx = model.feature_importances_.argsort()[::-1]
    plt.barh(feature_names[sorted_idx][:20][::-1], model.feature_importances_[sorted_idx][:20][::-1], align='center')
    plt.xlabel("Feature Imortances", size=15)
    plt.ylabel("Feautres", size=15)

plot_feature_importances(model)
plt.show()
```

Accuracy 계산

AUC 계산

ROC 커브 그리기AUC 계산

모델 변수 중요도 (top 20) 그래프 그리기

05. 코드 설명

XGBoost

```
thresholds = np.sort(model.feature_importances_)
save_score = 0
best_thresh = 0
for thresh in thresholds:
    selection = SelectFromModel(model, threshold=thresh, prefit=True)

    select_x_train = selection.transform(x_train)
    select_test = selection.transform(x_test)

    selection_model = XGBClassifier(n_jobs=-1)
    selection_model.fit(select_x_train,y_train)

    y_predict = selection_model.predict_proba(select_test)[:,1]
    roc = roc_auc_score(y_test, result2)

    # print("Thresh=%.4f, n=%d, roc: %.4f%%" %(thresh, select_x_train.shape[1], roc))

    if roc > save_score:
        save_score = roc
        best_thresh = thresh
    # print("best_thresh, save_score: ", best_thresh, save_score)

# print("====================================")
# print("best_thresh, save_score: ", best_thresh, save_score)

selection = SelectFromModel(model, threshold=best_thresh, prefit=True)
x_train = selection.transform(x_train)
x_test = selection.transform(x_test)

model = RandomizedSearchCV(xgb, params, n_jobs=-1, cv=kfold, verbose=1, scoring=scoring, n_iter=3, refit='AUC', return_train_score=True, random_state=SEED)
model.fit(x_train,y_train)
```

Model Selection

Model Selection 후
RandomizedSearchCV

05. 코드 설명

LGBM

```
params = {
    "n_estimators":[500, 800, 1000], #반복 수행하는 트리의 개수
    "learning_rate":[0.01, 0.05, 0.001],
    "num_leaves":[50], #하나의 트리가 가질 수 있는 최대 리프의 개수
    "max_depth ":[6, 10, 15, 20], # 트리의 최대 깊이,
    'min_child_samples':[20, 40, 60], #Leaf Node가 되기 위해서 최소한으로 필요한 데이터 개체의 수
    'subsample' :  [0.8,0.9], #데이터를 샘플링하는 비율
    'colsample_bytree ' : [0.5,0.7,1], #개별 트리를 학습할 때마다 무작위로 선택하는 피쳐의 비율을 제어
    'eval_metric' : ['auc']
    }
scoring = {
    'AUC': 'roc_auc',
}

kfold = KFold(n_splits=5, shuffle=True, random_state=SEED)
lgbm = lightgbm.LGBMClassifier(
                        tree_method='gpu_hist',
                        predictor='gpu_predictor',
                        random_state=SEED
                        )

model = RandomizedSearchCV(lgbm, params, n_jobs=-1, cv=kfold, scoring=scoring, n_iter=10, verbose=1, refit='AUC', return_train_score=True, random_state=SEED)

model.fit(x_train,y_train,
        eval_set=[(x_val, y_val)],
        eval_metric="auc",
        early_stopping_rounds=100,
        )

df = pd.DataFrame(model.cv_results_)
print("cv result : \n", df.iloc[:,[18,12]])
```

파라미터 세팅

평가방식 : auc로 최적의 파라미터 찾기

Validation data 로 평가하면서 학습

조기 종료 제공

05. 코드 설명

DNN

```
#load data
x_train = np.load('./data/project1/x_train.npy',allow_pickle=True)
y_train = np.load('./data/project1/y_train.npy',allow_pickle=True)
test = np.load('./data/project1/x_test.npy',allow_pickle=True)
index = np.load('./data/project1/index_no_s.npy',allow_pickle=True)


#모델 테스트를 위해 부분 데이터 잘라서 사용 (데이터 양이 너무 많음) - random으로 40%
x_train, x_test, y_train, y_test = train_test_split(x_train, y_train, train_size=0.7, random_state=SEED, stratify=y_train)
x_train, x_temp, y_train, y_temp = train_test_split(x_train, y_train, train_size=0.4, random_state=SEED, stratify=y_train)
x_test, x_temp, y_test, y_temp = train_test_split(x_test, y_test, train_size=0.4, random_state=SEED, stratify=y_test)

x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, train_size=0.8, random_state=SEED, stratify=y_train)

#random 40% 데이터
print("x train shape : ", x_train.shape) #(132280, 367)
print("y train shape : ", y_train.shape) #(132280,)
print("x test shape : ", x_test.shape) #(70864, 367)
print("y test shape : ", y_test.shape) #(70864,)


#모델링
model = Sequential()
model.add(Dense(1024, activation='relu',input_shape=(367,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(256, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

#컴파일, 훈련
model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["acc",tf.keras.metrics.AUC()])

es = EarlyStopping(monitor='loss',patience=30,mode='auto')
model.fit(x_train,y_train,epochs=500,batch_size=32,verbose=2,callbacks=[es],validation_data=(x_val,y_val))


#4. 평가
loss,acc,auc = model.evaluate(x_test,y_test,batch_size=32)
print("loss : ",loss)
print("acc : ",acc)
print("auc : ",auc)
```

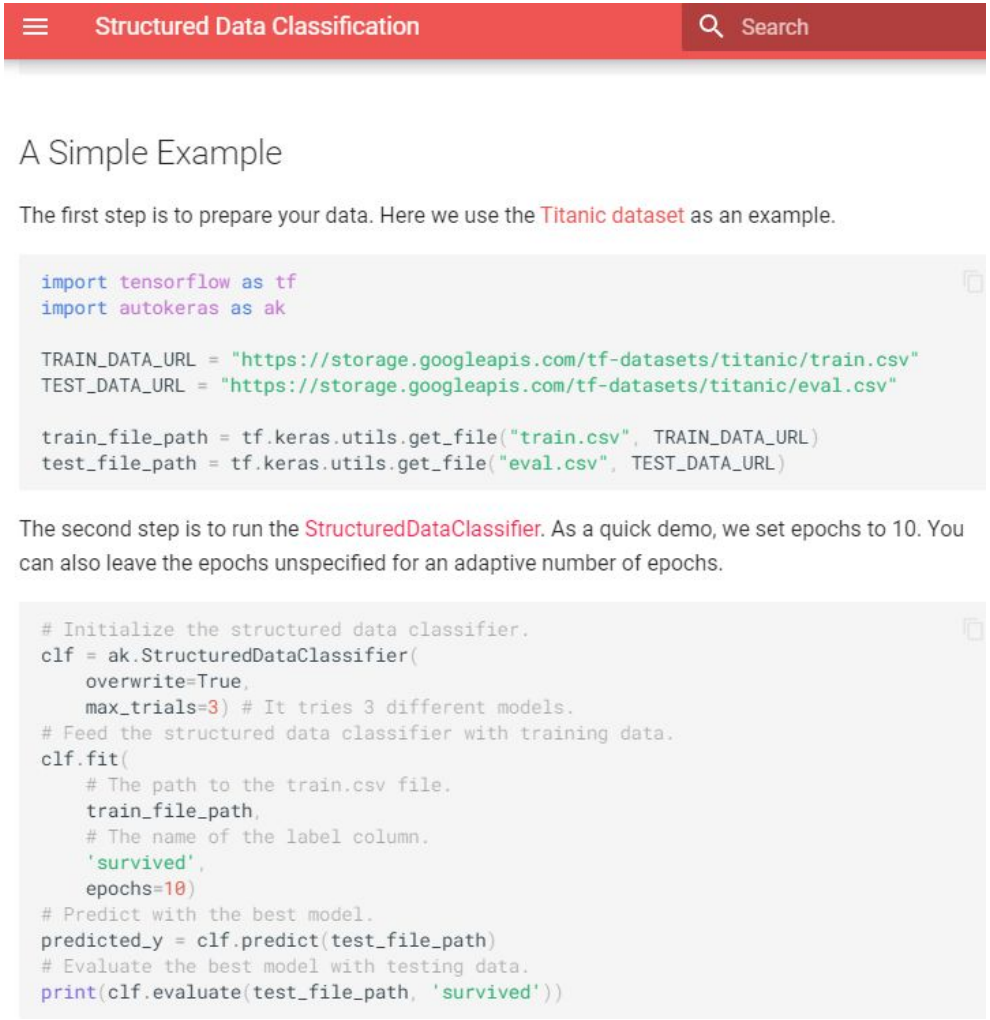
DNN 모델링

평가방식 : auc 추가

05. 코드 설명

AutoKeras

https://autokeras.com/tutorial/structured_data_classification/



Structured Data Classification

Search

A Simple Example

The first step is to prepare your data. Here we use the Titanic dataset as an example.

```
import tensorflow as tf
import autokeras as ak

TRAIN_DATA_URL = "https://storage.googleapis.com/tf-datasets/titanic/train.csv"
TEST_DATA_URL = "https://storage.googleapis.com/tf-datasets/titanic/eval.csv"

train_file_path = tf.keras.utils.get_file("train.csv", TRAIN_DATA_URL)
test_file_path = tf.keras.utils.get_file("eval.csv", TEST_DATA_URL)
```

The second step is to run the StructuredDataClassifier. As a quick demo, we set epochs to 10. You can also leave the epochs unspecified for an adaptive number of epochs.

```
# Initialize the structured data classifier.
clf = ak.StructuredDataClassifier(
    overwrite=True,
    max_trials=3) # It tries 3 different models.
# Feed the structured data classifier with training data.
clf.fit(
    # The path to the train.csv file.
    train_file_path,
    # The name of the label column.
    'survived',
    epochs=10)
# Predict with the best model.
predicted_y = clf.predict(test_file_path)
# Evaluate the best model with testing data.
print(clf.evaluate(test_file_path, 'survived'))
```

05. 코드 설명

AutoKeras

```
#모델 테스트를 위해 부분 데이터 잘라서 사용 (데이터 양이 너무 많음) - random으로 40%
x_train, x_test, y_train, y_test = train_test_split(x_train, y_train, train_size=0.7, random_state=SEED, stratify=y_train)
x_train, x_temp, y_train, y_temp = train_test_split(x_train, y_train, train_size=0.4, random_state=SEED, stratify=y_train)
x_test, x_temp, y_test, y_temp = train_test_split(x_test, y_test, train_size=0.4, random_state=SEED, stratify=y_test)

x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, train_size=0.8, random_state=SEED, stratify=y_train)

#random 40% 데이터
print("x train shape : ", x_train.shape) #(132280, 367)
print("y train shape : ", y_train.shape) #(132280,)
print("x test shape : ", x_test.shape) #(70864, 367)
print("y test shape : ", y_test.shape) #(70864,)


# Initialize the structed data classifier
clf = ak.StructuredDataClassifier(
    overwrite=True,
    max_trials=3
)
# Feed the image classifier with training data
clf.fit(x_train, y_train, validation_split=0.2, epochs=50)

# Predict with the best model
predicted_y = clf.predict(x_test)
print(predicted_y)

# Evaluate the best model with testing data
print(clf.evaluate(x_test, y_test))
print("====================")
```

```
#컴파일, 훈련
model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["acc",tf.keras.metrics.AUC()])
```

05. 코드 설명

AutoKeras

```
Search: Running Trial #1

Hyperparameter    |Value           |Best Value So Far
structured_data...|True            |?
structured_data...|2               |?
structured_data...|False           |?
structured_data...|0               |?
structured_data...|32              |?
structured_data...|32              |?
classification_...|0               |?
optimizer         |adam            |?
learning_rate     |0.001           |?
```

```
Trial 1 Complete [00h 41m 02s]
val_accuracy: 0.9650696516036987

Best val_accuracy So Far: 0.9650696516036987
Total elapsed time: 00h 41m 02s
```

```
Search: Running Trial #2

Hyperparameter    |Value           |Best Value So Far
structured_data...|True            |True
structured_data...|False           |False
structured_data...|2               |2
structured_data...|32              |32
structured_data...|0               |0
structured_data...|32              |32
classification_...|0.5             |0
optimizer         |adam            |adam
learning_rate     |0.001           |0.001
```

```
Trial 2 Complete [00h 40m 18s]
val_accuracy: 0.9650696516036987

Best val_accuracy So Far: 0.9650696516036987
Total elapsed time: 01h 21m 20s
```

```
#컴파일, 훈련
model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["acc",tf.keras.metrics.AUC()])
```

05. 코드 설명

AutoKeras

```
Search: Running Trial #3

Hyperparameter    |Value           |Best Value So Far
structured_data...|True            |True
structured_data...|False           |False
structured_data...|2               |2
structured_data...|32              |32
structured_data...|0               |0
structured_data...|32              |32
classification_...|0               |0
optimizer         |adam            |adam
learning_rate     |0.001           |0.001
```

```
Trial 1 Complete [00h 41m 02s]
val_accuracy: 0.9650696516036987

Best val_accuracy So Far: 0.96506965160369
Total elapsed time: 00h 41m 02s
```

```
Trial 2 Complete [00h 40m 18s]
val_accuracy: 0.9650696516036987

Best val_accuracy So Far: 0.9650696516036987
Total elapsed time: 01h 21m 20s
```

```
Trial 3 Complete [00h 41m 11s]
val_accuracy: 0.9650696516036987

Best val_accuracy So Far: 0.9650696516036987
Total elapsed time: 02h 02m 32s
```

06. 평가 지표

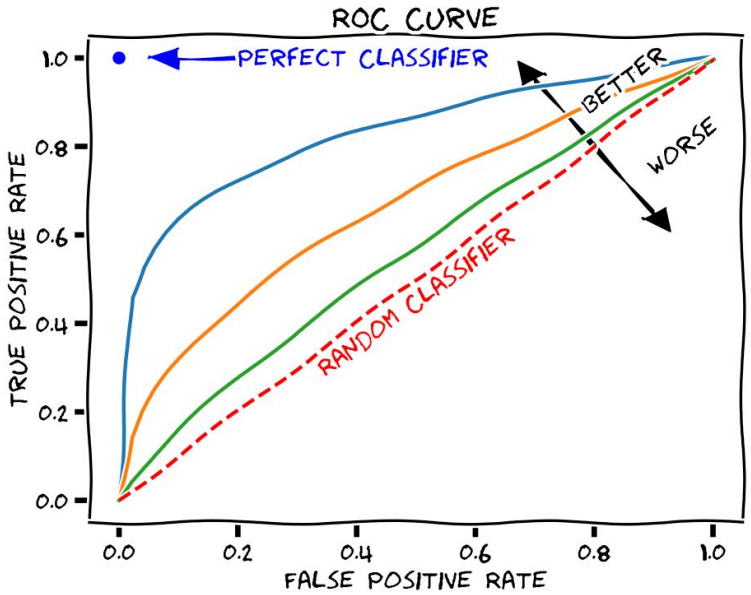
ROC 곡선 (Receiver Operating Characteristic)

- 이진 분류 모델 성능 평가 기법
- AUC는 'ROC 곡선 아래 영역'을 의미
- 곡선이 굽어지면 굽어질수록 AUC가 넓어지므로, 더욱 정확한 모델

1000	정상판정	암판정
정상환자	988 TN	2 FP
암환자	1 FN	9 TP

정상환자의 정확도는 99.8%지만

암환자의 정확도는 90%



06. 최종 결과

Accuracy 정확도 결과 모델별 비교

	데이터	데이터 Vxxx 컬럼 PCA 95%	데이터 Vxxx 컬럼 PCA 99%
XGBoost	97.80%	98.00%	98.01%
LGBM	98.13%	98.10%	98.12%
DNN	97.07%		
AutoKeras	96.50%		

06. 최종 결과

AUC 수신자 조작 특성 결과 모델별 비교

	데이터	데이터 Vxxx 컬럼 PCA 95%	데이터 Vxxx 컬럼 PCA 99%
XGBoost	92.42%	93.58%	93.57%
LGBM	94.38%	94.00%	93.80%

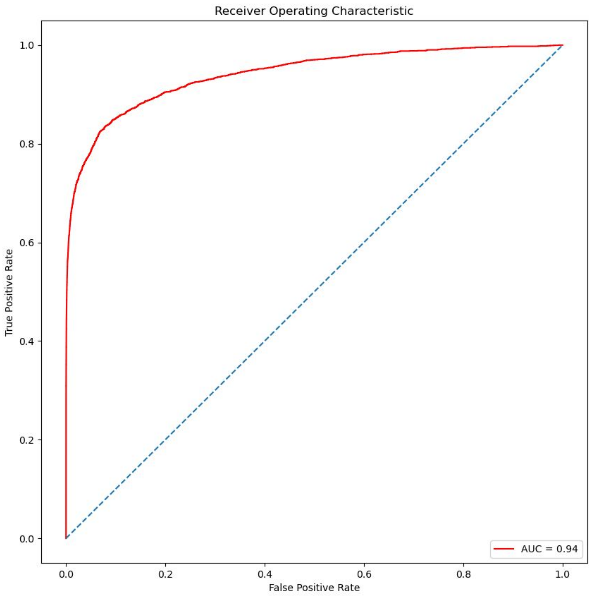
06. 최종 결과

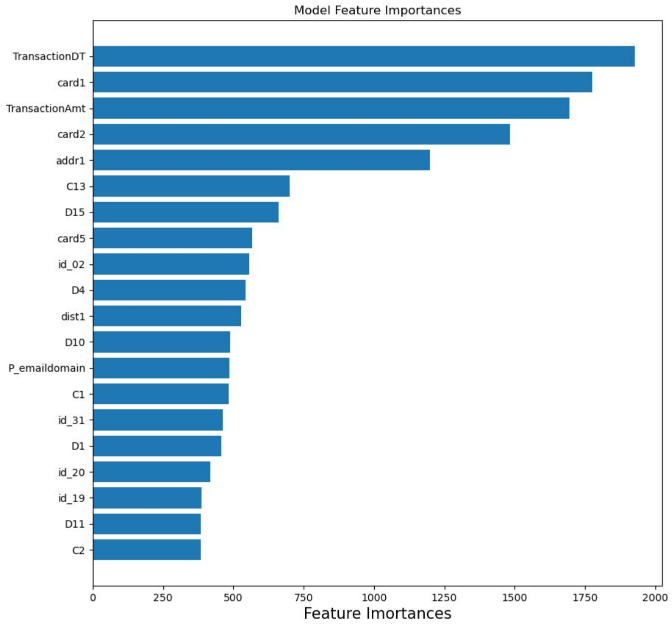
- 대체적으로 Accuracy 와 AUC 모두 LGBM이 제일 성능이 좋음
  - LGBM > XGBoost > DNN
- Vxxx 컬럼만 PCA 95%, 99%한 데이터와 안한 데이터의 성능 차이가 크게 없음
  - XGBoost는 PCA를 적용 안한 데이터가 0.5% 정도 낮음
  - LGBM은 모델 성능 차이가 거의 없음
- 최고 점수
  - Accuracy : PCA 적용 안한 데이터와 LGBM 모델 => **98.13%**
  - AUC : PCA 적용 안한 데이터와 LGBM 모델 => **94.38%**

06. 최종 결과

Model : LGBM
Data : without PCA

Accuracy : 98.0865%
AUC : 94.1472%






```
cv result :
    mean_test_AUC                                                                                                                    params
0        0.919487  {'subsample': 0.9, 'num_leaves': 50, 'n_estimators': 800, 'min_child_samples': 20, 'max_depth ': 15, 'learning_rate': 0.01, 'eval_metric': 'auc', 'colsample_bytree ': 0.7}
1        0.930177  {'subsample': 0.8, 'num_leaves': 50, 'n_estimators': 1000, 'min_child_samples': 20, 'max_depth ': 15, 'learning_rate': 0.05, 'eval_metric': 'auc', 'colsample_bytree ': 0.5}
2        0.930229  {'subsample': 0.8, 'num_leaves': 50, 'n_estimators': 800, 'min_child_samples': 60, 'max_depth ': 20, 'learning_rate': 0.05, 'eval_metric': 'auc', 'colsample_bytree ': 1}
3        0.928528  {'subsample': 0.9, 'num_leaves': 50, 'n_estimators': 500, 'min_child_samples': 40, 'max_depth ': 15, 'learning_rate': 0.05, 'eval_metric': 'auc', 'colsample_bytree ': 1}
4        0.875468  {'subsample': 0.9, 'num_leaves': 50, 'n_estimators': 1000, 'min_child_samples': 20, 'max_depth ': 6, 'learning_rate': 0.001, 'eval_metric': 'auc', 'colsample_bytree ': 0.5}
5        0.929292  {'subsample': 0.9, 'num_leaves': 50, 'n_estimators': 500, 'min_child_samples': 20, 'max_depth ': 6, 'learning_rate': 0.05, 'eval_metric': 'auc', 'colsample_bytree ': 0.5}
6        0.930396  {'subsample': 0.8, 'num_leaves': 50, 'n_estimators': 800, 'min_child_samples': 40, 'max_depth ': 6, 'learning_rate': 0.05, 'eval_metric': 'auc', 'colsample_bytree ': 1}
7        0.863392  {'subsample': 0.9, 'num_leaves': 50, 'n_estimators': 500, 'min_child_samples': 20, 'max_depth ': 20, 'learning_rate': 0.001, 'eval_metric': 'auc', 'colsample_bytree ': 0.5}
8        0.863392  {'subsample': 0.9, 'num_leaves': 50, 'n_estimators': 500, 'min_child_samples': 20, 'max_depth ': 15, 'learning_rate': 0.001, 'eval_metric': 'auc', 'colsample_bytree ': 1}
9        0.862854  {'subsample': 0.9, 'num_leaves': 50, 'n_estimators': 500, 'min_child_samples': 40, 'max_depth ': 10, 'learning_rate': 0.001, 'eval_metric': 'auc', 'colsample_bytree ': 0.7}
```

```
최적 하이퍼 파라미터 :  {'subsample': 0.8, 'num_leaves': 50, 'n_estimators': 800, 'min_child_samples': 40, 'max_depth ': 6, 'learning_rate': 0.05, 'eval_metric': 'auc', 'colsample_bytree ': 1}
최고 AUC : 0.9304
score :  0.9808647550237074
acc :  0.9808647550237074
AUC : 94.1472%
roc_auc : 0.9414722376579316
```

06. 개선사항

- 다양한 딥러닝 알고리즘 사용 (Conv1D)
  - PCA 적용한 데이터들도 딥러닝 적용해보기
- 머신러닝 (분류) 선형모델 사용 (SVM, Logistic Regression)
- 모델 앙상블을 통한 모델 성능 개선
  - 트리모델 + 선형모델
  - 트리모델 + DNN (함수형 모델)
- 전처리 과정에서 Vxxx 컬럼말고도 다른 컬럼도 PCA 진행해보기
- AutoML?

06. 프로젝트 일정

	11/26	11/27	11/28	11/29	11/30
데이터 전처리					
XGBoost					
LGBM					
DNN					
AutoKeras					
PT 준비					

06. 파일

- https://github.com/ym0179/bit_seoul/tree/master/individual%20project
  - 데이터 전처리 파일 : fraud_detect_data.py
  - Xgboost 학습 파일 : fraud_detect_xgb.py
  - LGBM 학습 파일 : fraud_detect_lgbm.py
  - DNN 학습 파일 : fraud_detect_dnn.py
  - AutoKeras 학습 파일 : fraud_detect_autokeras.py

07. 추가_캐글

```
#데이터 로드
x_train = np.load('/kaggle/input/ieee-submission/x_train.npy',allow_pickle=True)
y_train = np.load('/kaggle/input/ieee-submission/y_train.npy',allow_pickle=True)
test = np.load('/kaggle/input/ieee-submission/x_test.npy',allow_pickle=True)

#모델링
model = LGBMClassifier(
    subsample = 0.9,
    num_leaves= 50,
    n_estimators= 1000,
    min_child_samples=20,
    max_depth = 6,
    learning_rate= 0.05,
    eval_metric= 'auc',
    colsample_bytree = 0.7,
    tree_method='gpu_hist',
    predictor='gpu_predictor',
    n_jobs=-1,
    random_state=SEED
)

#학습
model.fit(x_train,y_train, verbose=1)

#예측
result = model.predict_proba(test)[:,1]

#제출
sub = pd.read_csv("/kaggle/input/ieee-fraud-detection/sample_submission.csv")
sub["isFraud"] = result
sub.to_csv("sub.csv",index=0)
```

07. 추가_캐글

Overview Data Notebooks Discussion Leaderboard Rules Team My Submissions **Late Submission**

Your most recent submission

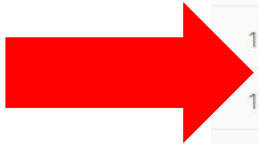
Name	Submitted	Wait time	Execution time	Score
sub (3).csv	a few seconds ago	0 seconds	3 seconds	0.929943

Complete

Jump to your position on the leaderboard

07. 추가_캐글

이쯤?



104	▲143	shishanuvic		0.930064	102	1y
105	▲72	jessky		0.929971	179	1y
106	▲160	yokotani		0.929963	163	1y
107	▲25	sasakama		0.929957	74	1y
108	▲127	Keith_SZE		0.929934	111	1y
109	▲5	Aham Fraudasmi		0.929867	273	1y
110	▼6	Evdilos_Ikaria		0.929839	245	1y
111	▲15	funkyboy	</> Find unique clients	0.929812	39	1y
112	▼20	bobocbf		0.929786	26	1y

108 / 6351 = 상위 1.7%...

더 올릴 수 있겠죠?


Q&A