Complex Audits and Frauds Analysis

This repository provides tools and functions to process data related to complex audits and fraud detection. It includes utilities to merge various datasets and a training pipeline to build and evaluate machine learning models for fraud detection and audit analysis.

Models

In the `models` directory we store the trained models that can be used for the fraud and audit prediction task. Both for the audit and fraud tasks, we select XGBoost as the best model. All the models are trained at most with the 50 most important features.

For example, when predicting `FRAUD_OR_CORRECT` (i.e. whether a TIN is either fraudolent or has submitted a major correction), the best model is the following: - `models/FRAUD_OR_CORRECT/training_years_[2022]/data_all/lnr_xgb.pkl`

A fully interpretable model can be found by loading the InterpretableAI Optimal Tree. The tree can be visualized either by openining `lnr_oct.html` or by running the following code:

```python
from interpretableai import iai
from feature_map import mapping_short

# Load Optimal Tree
lnr_oct = iai.read_json('models/FRAUD_OR_CORRECT/training_years_[2022]/data_all/lnr_oct.json')

# Get Features used by the model
features_used = lnr_oct.get_features_used()

# Select only the features that are actually used by the model
new_feature_map = {key: val for (key, val) in mapping_short.items() if key in features_used}

# Visualize in Browser with mapped features
iai.TreePlot(lnr_oct, feature_renames=new_feature_map).show_in_browser()
```

Feature Mapping

We mapped all the features available for the complex audit models. The mapping is accessible in `feature_map.py` file. - **mapping_long**: for each column has a lengthy explanation of its content - **mapping_short**: for each column has a short explanation of its content, used for visualization

---

Targets

We support the following targets for training and evaluation: - `AUDIT_ANY`: True if a company will receive a complex audit - `FRAUD_ANY`: True if a company will be fraudolent during a complex audit - `CORRECT_PROFIT`: True if a company will submit a significant change in its Shahutahark tax form - `FRAUD_OR_CORRECT`: True if a company will be fraudolent during a complex audit or submit a significant change in its Shahutahark tax form

Pipeline Overview

The pipeline is meant to be run in the following order: 1. `merge_data.py`: Merges and processes various datasets for audit and fraud analysis. 2. `training_pipeline.py`: Contains the pipeline for training machine learning models. 3. `evaluation_pipeline.py`: Contains the pipeline for evaluating machine learning models.

How the Configuration Works

The `config.py` file provides centralized configurations for the training and evaluation pipelines. Both `training_pipeline.py` and `evaluation_pipeline.py` load their respective configurations from `config.py` to streamline parameters and ensure consistency.

`training_pipeline.py`

This script constructs machine learning models to classify audits and detect potential fraud based on the merged data from `merge_data.py`.

Inputs

1. **Merged Data**: The output from `merge_data.py` with cleaned and processed features for training.
2. **config.py**: `training_config` is the configuration specifying the dataset path, target variable, training years, and the directory to save the trained models.

EXAMPLE CONFIG (`training_config` FROM `config.py`):

```
training_config = {
    'df_path': '/path/to/dataset.csv',
    'target': 'FRAUD_OR_CORRECT',
    'training_years': [2022],
    'save_path': '/path/to/save/models',
    'interpretable': True,
}
```

Outputs

- **Model Object**: The trained model object is saved using `pickle` for use in the evaluation step.

`evaluation_pipeline.py`

This script evaluates the performance of trained machine learning models on unseen data, generates interpretability insights, and produces evaluation metrics. It outputs ranked predictions, identifying the companies (TINs) that should be prioritized for audits based on the likelihood of fraud or anomalies.

Inputs

1. **Trained Models**: Models generated and saved by `training_pipeline.py` are loaded for evaluation.

2. **Merged Data**: The processed dataset, typically output from `merge_data.py`, filtered for the evaluation years.
3. **config.py**: `evaluation_config` is the configuration that defines the evaluation dataset, model path, and evaluation years.

EXAMPLE CONFIG (`evaluation_config` FROM `config.py`):

```
evaluation_config = {
    'df_path': '/path/to/dataset.csv',
    'inference': False,
    'models_path': '/path/to/models',
    'df_path': '/path/to/data.csv',
    'target': 'FRAUD_OR_CORRECT',
    'evaluate_years': [2023],
    'training_years': [2022],
    'figures_path': '/path/to/figures',
    'ranking': 'predicted_change_class',
    'compute_tin_level': True,
    'interpretable': True,
    'best_model': 'xgb',
}
```

Outputs

1. **Always Generated Outputs**:
2. **Ranked Predictions**: Probabilities or rankings for each instance in the evaluation dataset based on the trained model's predictions. Users can specify the ranking criteria in the `evaluation_config`. Supported options include:
   - `fraud_proba`: Sort by the probability of fraud (default for fraud models).
   - `predicted_change`: Sort by the predicted magnitude of corrections (useful for corrective tasks).
   - `predicted_change_class`: Sort by the predicted magnitude of corrections using a classification model (Low, Medium, High), empirically shown to be more effective.
3. **dict_tin_tax_year**: A dictionary mapping TINs to their corresponding tax years, with the following keys:
   - `predicted_prob`: The predicted probability of the event of interest.
   - `decile`: The decile in which the prediction falls.
   - `ranking`: The rank of the instance in the dataset based on the prediction.
   - `sample_shap_values`: SHAP values for the specific sample, explaining the model's prediction.
   - `expected_value`: The expected value of the model output.
   - `sample`: The actual data sample used for evaluation.
   - `sample_tree_explanation`: The TIN level explanation using InterpretableAI's Decision Tree.
4. **full_features**: The full set of features used during evaluation, including preprocessed and derived variables, for further analysis and debugging.
5. **When `inference` is `False`**:
6. **Evaluation Metrics**: Metrics such as accuracy, AUC, precision-recall, and confusion matrices are computed and saved.
7. **Interpretability Insights**: SHAP plots and other visualizations are generated to explain model predictions.

8. **Evaluation Summaries**: Reports summarizing model performance, including comparisons across multiple models, are generated for reporting and analysis.

The `inference` flag in the configuration dictates whether full evaluation metrics and insights are produced (`False`) or only ranked predictions are generated (`True`).

`example.ipynb`

This Jupyter notebook provides a working example of how to tweak the most important functions in this repository, covering: 1. Load large files from Google Drive 2. Data preparation and merging using `merge_data.py` (Only if data in 1. is not needed anymore) 3. Model training with `training_pipeline.py` 4. Model evaluation with `evaluation_pipeline.py` 5. Visualization and interpretation of model results, including global and TIN-level explanation of the models.

This notebook is a good starting point for understanding how to use the code in this repository for audit and fraud analysis.