Anomaly Detection using Neural Networks

Dean Langsam
What We Do

● Mission
  ○ Offer convenient and flexible access to working capital for small and medium sized businesses

● Products:
  ○ Revolving line of credit
  ○ Invoice Factoring (Receivables backed financing)
Data Science in BlueVine

- Bridge the gap between FAST and RISKY in approving loans
- Build credit models
- Analyze fraud and fraudulent behaviour
- Analyze financial strength of clients
- Incorporating 3rd party data
- Data pipelines for hundreds of fundamental variables
### Data Flow

<table>
<thead>
<tr>
<th>user_id</th>
<th>balance</th>
<th>income</th>
<th>feature_4</th>
<th>feature_5</th>
<th>feature_6</th>
<th>feature_7</th>
<th>feature_8</th>
<th>feature_9</th>
<th>feature_10</th>
<th>...</th>
<th>feature_491</th>
<th>feature_492</th>
<th>feature_493</th>
<th>feature_494</th>
<th>feature_495</th>
<th>feature_496</th>
<th>feature_497</th>
<th>feature_498</th>
<th>feature_499</th>
<th>internal_model_score</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1200.0</td>
<td>14000.0</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>medium</td>
</tr>
<tr>
<td>20</td>
<td>-1000.0</td>
<td>99000.0</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>medium</td>
</tr>
<tr>
<td>30</td>
<td>2000.0</td>
<td>25000.0</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>medium</td>
</tr>
<tr>
<td>40</td>
<td>10000.0</td>
<td>27000.0</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>medium</td>
</tr>
<tr>
<td>50</td>
<td>5000.0</td>
<td>13000.0</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>medium</td>
</tr>
<tr>
<td>60</td>
<td>7000.0</td>
<td>16000.0</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>medium</td>
</tr>
<tr>
<td>70</td>
<td>2000.0</td>
<td>30000.0</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>good</td>
</tr>
<tr>
<td>80</td>
<td>3000.0</td>
<td>85000.0</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>bad</td>
</tr>
</tbody>
</table>

8 rows x 499 columns
Data Flow

- Several primary data sources
- Dozens of models
- Hundreds of variables
- Thousands of total "things" to monitor
Anomaly Detection

Why we need it?

- Many data sources
- Big Team: Each model depends on results of other models
- Model scores are mission critical for the company
- Anomalies are proxies for corrupted data
In this Talk

- Anomaly detection project
  - My choices
  - Build it end to end
- My first neural network
- Useful and modern Pandas
- Python is a friend, not a foe.
Initialization

- **Explore**: A small subset of variables
- **Understanding**: monitor counts data
- **Realization**:
  - Complex data pipeline
  - Several different underlying tasks
  - Independent tasks
- **Solution**: “Microservice” architecture

Same variable counts data on different intervals
Pandas and datetimes

- Penny drop moments (choir moments)
  
  ```python
  x = (1, 2, 3, 4, 5, 6)
  ```

- Why not
  
  ```python
df2 = (df
      .groupby('col')
      .count())
  ```

- Index with date (df.loc[])
- df.resample()
- df.assign (col = lambda...)

```python
rs = (df
      .set_index('date_')
      .dropna()
      .loc['2017-1':]
      .resample('d')
      .apply('count')
      .rename(columns={'var_id': 'counts'})
      .loc[::1, ['counts']]
      .assign(weekend=lambda df: df.index.weekday >= 5)
      .assign(diff_=lambda df: df.counts.diff(1)))
```
Pandas date index

- `.dt accessor`
  - `df.col.dt.day`
  - `.floor('1h')` and `.ceil('3d')`
- `df.col.rolling(window).mean()`
- `df.col.rolling(window).std()`
- `df.col.expanding...`
- `df.col.ewm...`
Packages

- Luminol by LinkedIn
- Donut
  - Uses Tensorflow
- Skyline by Esty
  - No longer maintained
Update

Fit (Training)

Predict

Alert
Update

- Acts as a “sensor”
- Aggregates raw data into fixed-time, fixed-scope observations
- Reads production tables on fixed time-intervals and writes aggregated summaries
Update

- Some variables write more frequently than others
- Some variables show different patterns than others
df = (df
   .pipe(count_db_writes, freq='1h')
   .pipe(pad_db_counts, min_date=min_date, max_date=max_date, ids=ids)
   .pipe(expand_db_counts)
)

def pad_vector_counts (df):
    ret = df.copy()
    date_index = pd.date_range(start=min_date, end=max_date,
                                freq='H', tz='UTC', name='date_')
    ...  
    ret = ret.reindex(index=date_index.union(ret.index), columns=var_ids)
    return ret
Fit (Training and Retraining)

- Given aggregated data, creates many time series
- Processes time-series into sliding windows
- Train/test split
- Data cleaning pipeline
Update

**Predict** (Training and Retraining)

- Make u-turns: Use asserts
- One-stop-shop for cleaning
  - .pipe(clean)

```python
def retrain_model(data):
    assert complete_counts_history(data), 'meaningful message {}'\n        .format('')

    # ...
    train_start, train_end, test_start, test_end = get_train_test_period()
    train_data = create_sliding_windows(data,
        start=train_start, end=train_end, training=True)
    test_data = create_sliding_windows(data,
        start=test_start, end=test_end, training=False)
    scaler = MinMaxScaler().fit(train_data[['y']])
    X_train, y_train = process_input(train_data, scaler)
    X_test, y_test = process_input(test_data, scaler)
    # ...
    return model
```

**Alert**
**Update**

**Predict**

**Fit (Training and Retraining)**

- Learns a fully-connected Neural network for each variable on counts data
- Optimizes MAE
- A good trade off between practical and efficient

---

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>reshape_1 (Reshape)</td>
<td>(None, 14)</td>
<td>0</td>
</tr>
<tr>
<td>dense_1 (Dense)</td>
<td>(None, 32)</td>
<td>480</td>
</tr>
<tr>
<td>dropout_1 (Dropout)</td>
<td>(None, 32)</td>
<td>0</td>
</tr>
<tr>
<td>dense_2 (Dense)</td>
<td>(None, 16)</td>
<td>528</td>
</tr>
<tr>
<td>dropout_2 (Dropout)</td>
<td>(None, 16)</td>
<td>0</td>
</tr>
<tr>
<td>dense_3 (Dense)</td>
<td>(None, 1)</td>
<td>17</td>
</tr>
<tr>
<td>activation_1 (Activation)</td>
<td>(None, 1)</td>
<td>0</td>
</tr>
<tr>
<td>reshape_2 (Reshape)</td>
<td>(None, 1, 1)</td>
<td>0</td>
</tr>
</tbody>
</table>

Total params: 1,025
Trainable params: 1,025
Non-trainable params: 0
Update

Fit (Training and Retraining)

- Create model inside a function
- Use functools.partial to iterate over options

```python
def make_dense_model(X_train, y_train, nb=16, loss='mse', dropout=0.2, optimizer='adam'):
    in_shp = X_train.shape
    out_shp = y_train.shape[1]
    model = Sequential()
    model.add(Reshape((-1,), input_shape=(in_shp[1], in_shp[2])))
    model.add(Dense(2 * nb, activation='tanh'))
    model.add(Dropout(dropout))
    model.add(Dense(nb, activation='tanh'))
    model.add(Dropout(dropout))
    model.add(Dense(out_shp))
    model.add(Activation('relu'))
    model.add(Reshape((out_shp, 1)))
    model.compile(optimizer=optimizer, loss=loss)
    return model
```

```python
from functools import partial

make_model = partial(make_dense_model, nb_nodes=16, loss='mae')

###
models = [
    partial(make_dense_model, nb=16, loss='mae'),
    partial(make_lstm_model, nb=16, loss='mse'),
    ...
    partial(make_lstm_model, nb=4, loss='mse')
]
###

def make_and_fit(X, y, make_model):
    model = make_model(X, y)
    # Do more things
    fit_model(model, X, y)
    return model
```
Update

**Fit (Training and Retraining)**

- Uses test errors to compare model to previous models and chooses the best one
- Looks at errors of individual predictions
Update

**Fit (Training and Retraining)**

- On the test period, creates a histogram of prediction error per observation
- Fits the histogram to a non-central T distribution
- Finds thresholds that constitute anomalies

**Progression of Errors**

![Progression of Errors Chart]
Update

Fit (Training and Retraining)

- Distribution parameters and thresholds are saved to metadata table
- Models are saved to S3
- Model info is saved to metadata table

Distribution of Errors
Predict

- Counts data is written
- Predicted data is compared against real data
- Error is compared against thresholds
- Error and anomaly info are saved to database
Predict

- Counts data is written
- Predicted data is compared against real data
- Error is compared against thresholds
- Error and anomaly info are saved to database
Update

Fit (Training)

Predict

- Luckily we have the pipeline
- All model parameters exist
  - But not as python objects
- Make sure you can recreate everything from metadata
  - even the charts
- **Exception**: The model itself
- Make sure you can recreate the model itself from metadata
- `getattr()`

```python
ger = get_distribution_from_dict(db)
dist_params = get_dist_params_from_db(db)
dist = getattr(scipy.stats, db['dist_name'])
return dist(dist_params)
```

```python
ci = MinMaxScaler().fit([[db['scaler_min']], [db['scaler_max']]])
observation = create_sliding_windows(data, **db)
logger.info('Sliding windows created')
X, y = process_input(observation, scaler)
model = load_keras_model(**db)
```
Alert

- Take all anomalies from the database
- Create chart for each anomaly
- The chart captures all relevant information for current anomaly
  - Time series of actual vs. predicted
  - Time series of error progression
  - Histogram of errors with underlying distribution
Alert

- Support several types of anomalies
- Feedback mechanism will allow supervised learning in the future
Conclusions

- You don’t need to have a PHD in machine vision to deploy neural networks
- Pure Python has a lot more to offer than you use (But not dataframes)
- Diving into Pandas is a lot faster than inventing Pandas
Thank you

DeanLangsam

Dean_La

DeanLa

DeanLa.com