Consiglio Nazionale delle Ricerche

Polimi - Financial Engineering AY 2023/2024

Lab: Electricity Price and Load Forecasting

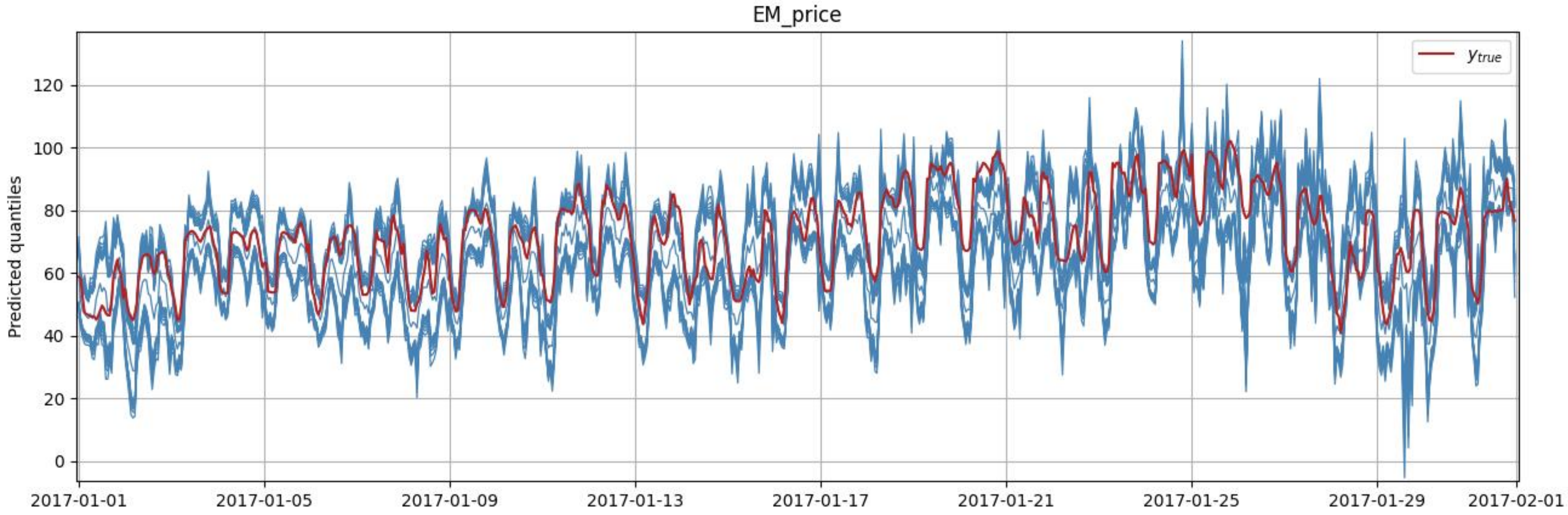
STIIMA

Alessandro Brusaferri

Summary

Goal:

- Learn how to develop probabilistic time series forecasting systems
- Applications to day-ahead electricity price/load forecasting
- Adopting SOTA developments and best practices in the literature (see prelab)
- Focusing ARX-l1(LEAR) and DNN (FFN) models in multi-step settings



Summary

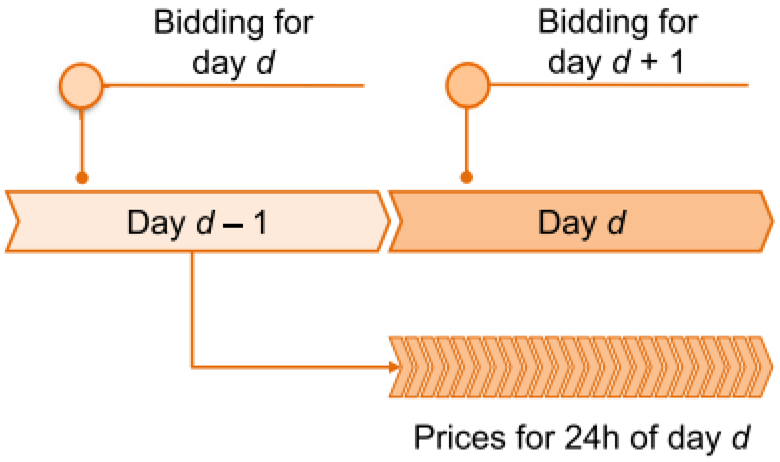
Today:

- Intro to main concepts: recalibration, moving window, batching, etc.
- Overview of the overall codebase (json settings, data processing, etc)
- Step-by-step implementation of the ARX-l1 model in TF and DNN intro

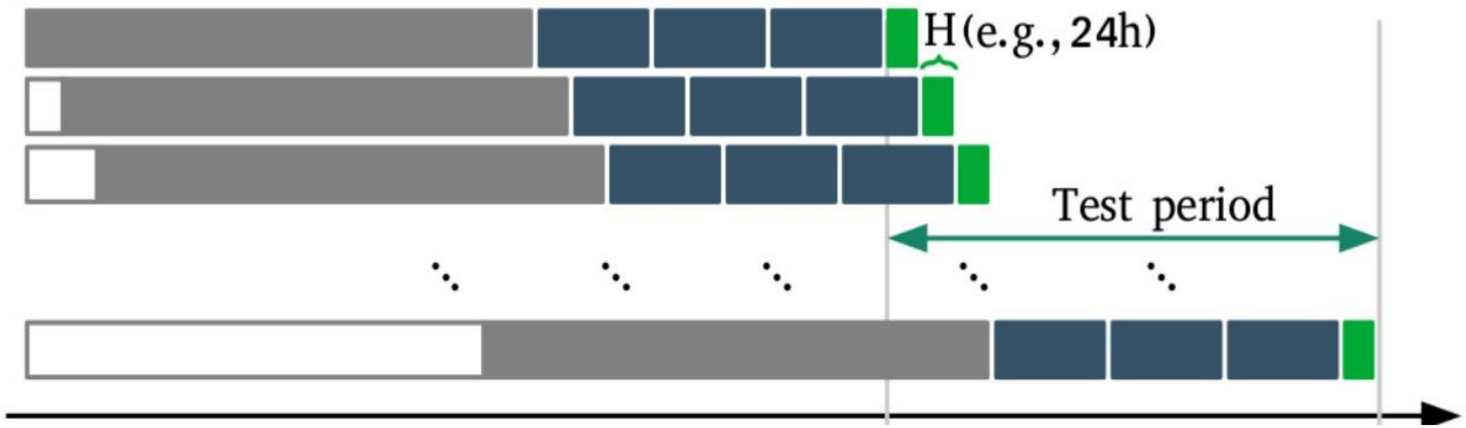
Next (subject to updates):

- L2: hyperparam tuning via optuna
- L3: probabilistic layers and ensembling
- L4: recent conformal inference developments

From time series to supervised learning



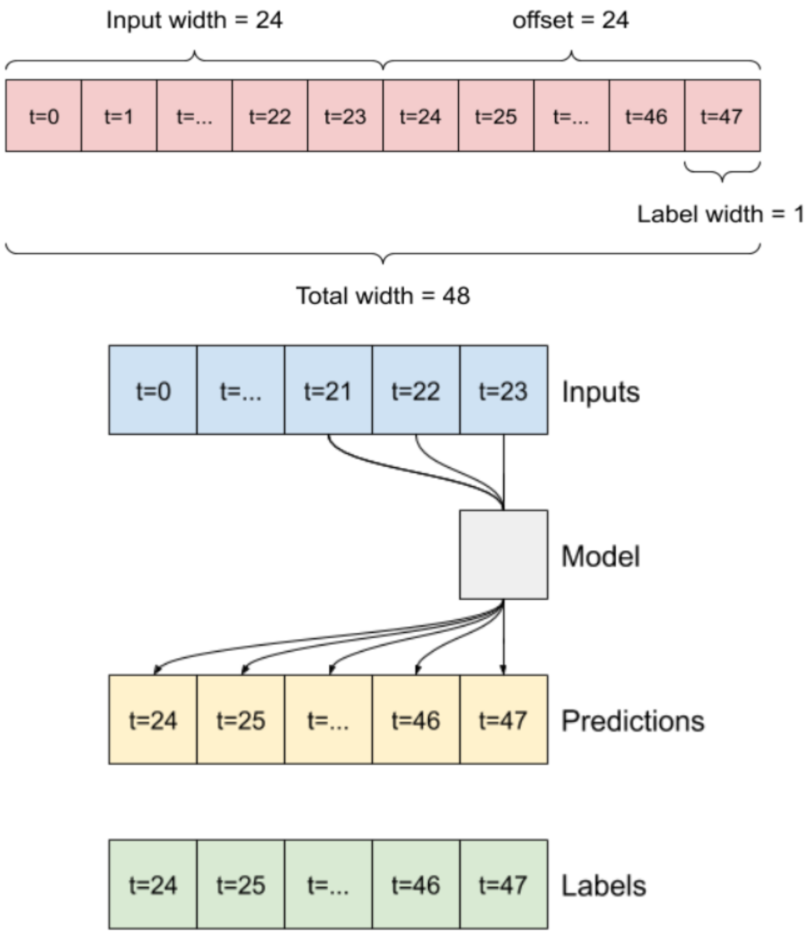
- **Target**: day-ahead hourly prices prediction
- **Input set**: past price values, load forecasts, generation forecasts, etc.
- **Recalibration**: incremental (e.g., daily) model update by including recent information

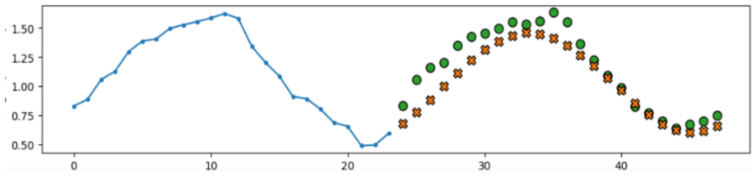


Lago et al: Forecasting day-ahead electricity prices: A review of state-of-the-art algorithms, best practices and an open-access benchmark

From time series to supervised learning

- **Moving window** to extract batches from TS
- Input **width** and prediction **offset** (multistep)

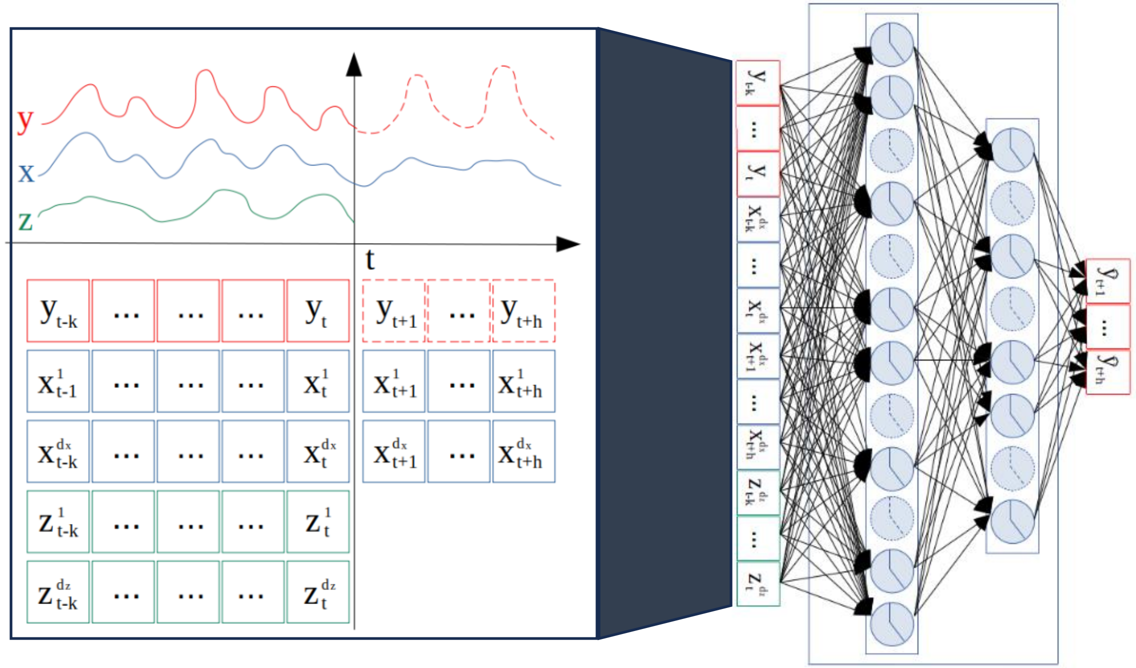


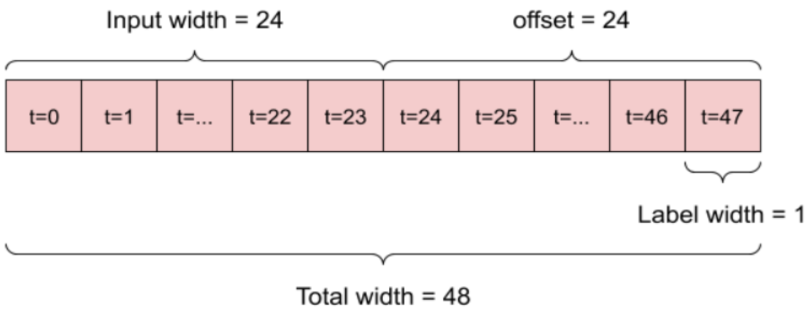


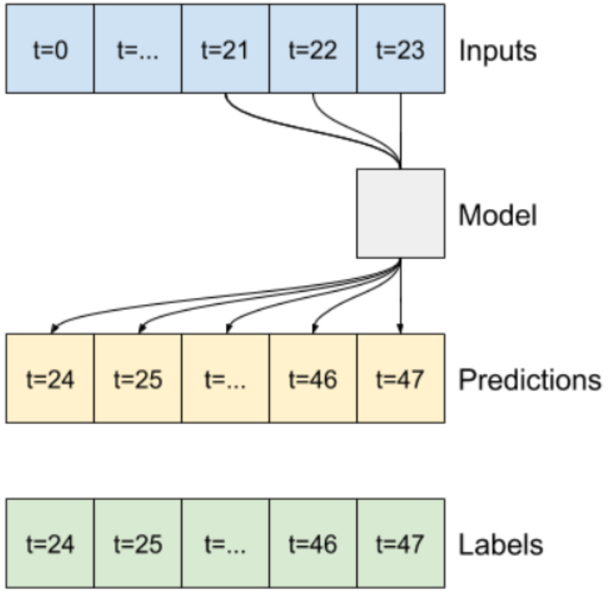
https://www.tensorflow.org/tutorials/structured_data/time_series

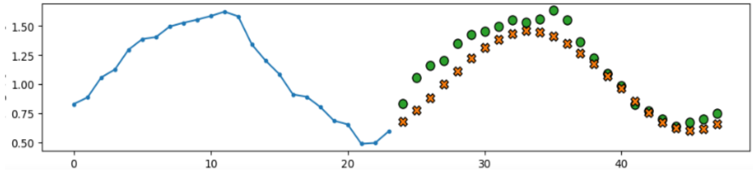
From time series to supervised learning

- **Moving window** to extract batches from TS
- Input **width** and prediction **offset** (multistep)
- **Past** vs **future** conditioning sets





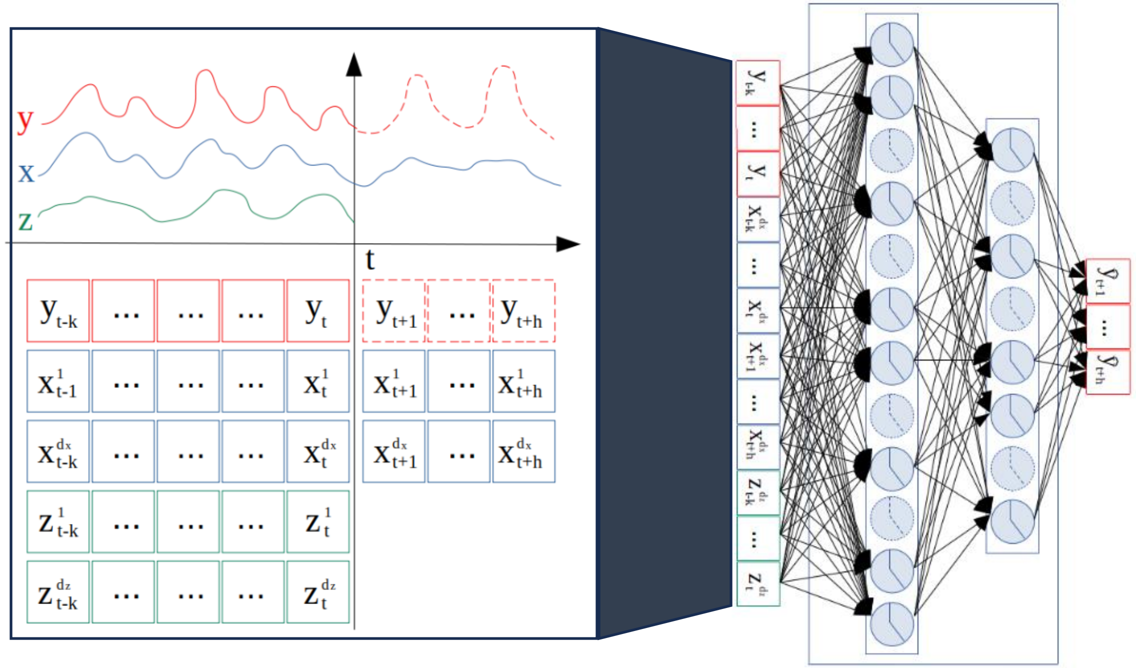




https://www.tensorflow.org/tutorials/structured_data/time_series

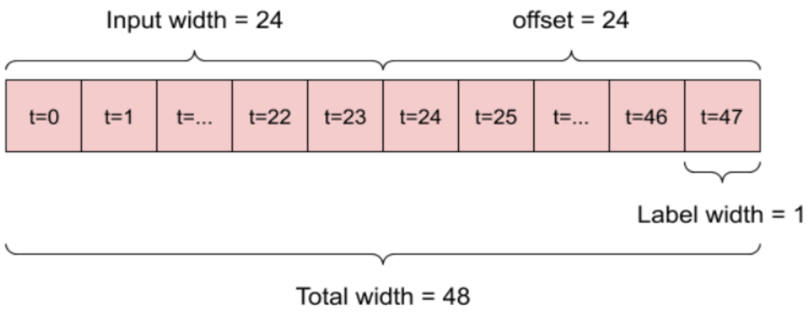
From time series to supervised learning

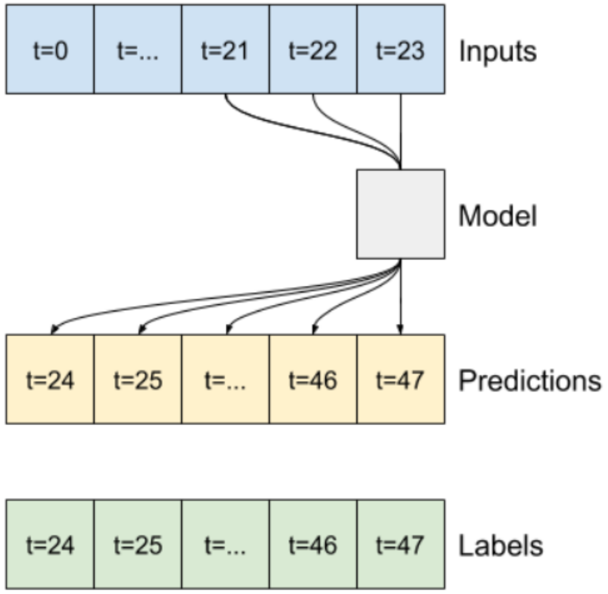
- **Moving window** to extract batches from TS
- Input **width** and prediction **offset** (multistep)
- **Past** vs **future** conditioning sets

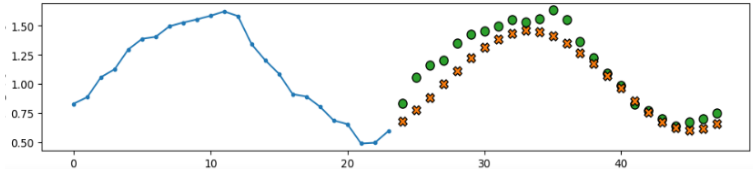


- Calendar features, e.g., weekday

$$c_t = [\sin(2\pi d_t/6)\,,\quad \cos(2\pi d_t/6)] \qquad d_t \in [0, ..., 6]$$

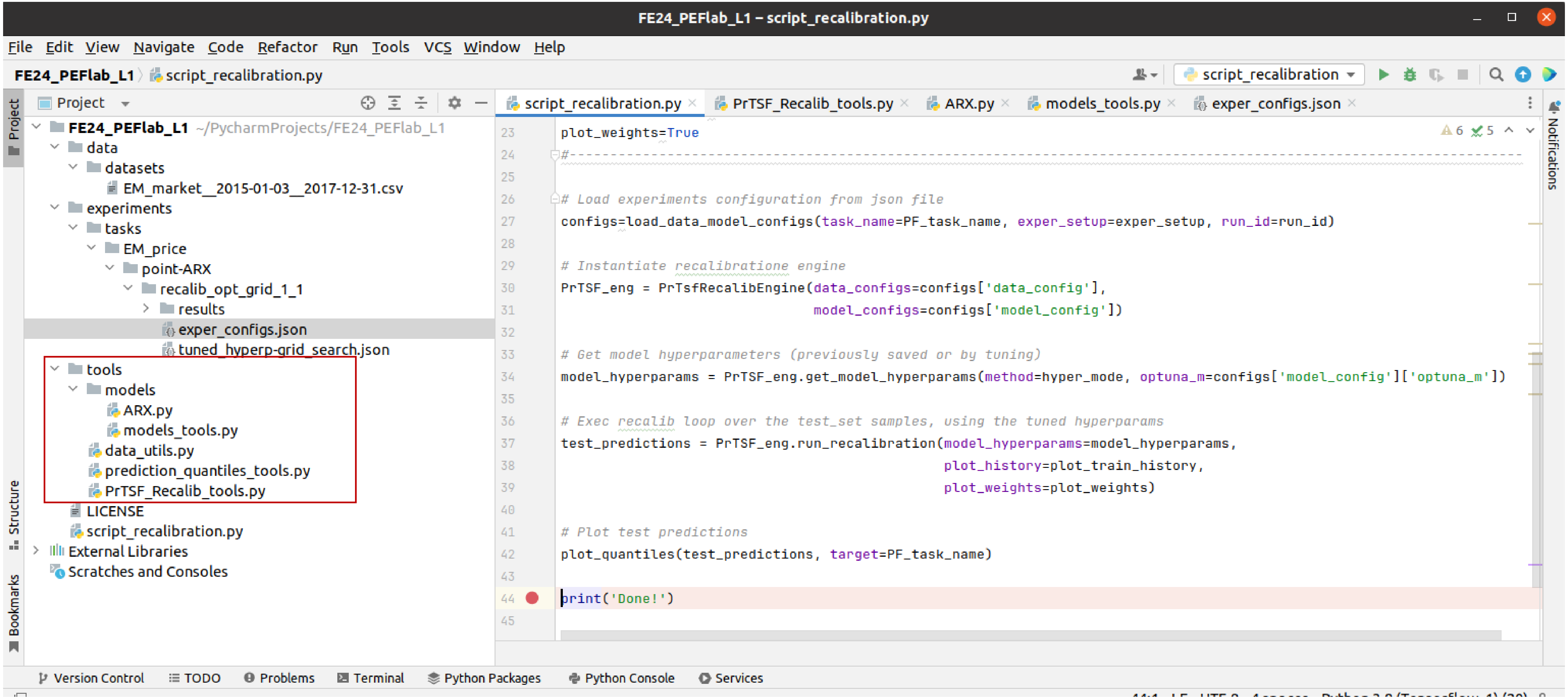






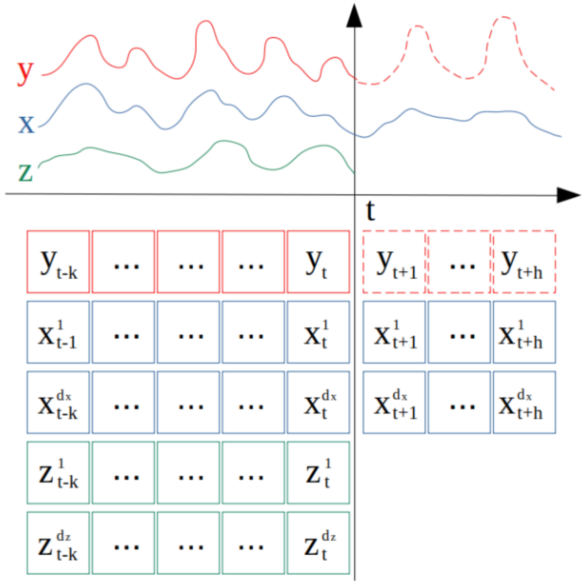
https://www.tensorflow.org/tutorials/structured_data/time_series

Codebase overview (goto PyCharm prj)



Simple ARX-l1 model in Tensorflow

```python
class ARXRegressor:
    def __init__(self, settings, loss):
        self.settings = settings

    @staticmethod
    def build_model_input_from_series(x, col_names: List, pred_horiz: int):
        # get index of target and past features
        past_col_idxs = [index for (index, item) in enumerate(col_names)
                         if features_keys['target'] in item or features_keys['past'] in item]

        # get index of const features
        const_col_idxs = [index for (index, item) in enumerate(col_names)
                          if features_keys['const'] in item]

        # get index of futu features
        futu_col_idxs = [index for (index, item) in enumerate(col_names)
                         if features_keys['futu'] in item]

        # build conditioning variables for past features
        past_feat = [x[:, :-pred_horiz, feat_idx] for feat_idx in past_col_idxs]
        # build conditioning variables for futu features
        futu_feat = [x[:, -pred_horiz:, feat_idx] for feat_idx in futu_col_idxs]
        # build conditioning variables for cal features
        c_feat = [x[:, -pred_horiz:-pred_horiz + 1, feat_idx] for feat_idx in const_col_idxs]

        # return flattened input
        return np.concatenate(past_feat + futu_feat + c_feat, axis=1)
```



Simple ARX-l1 model in Tensorflow

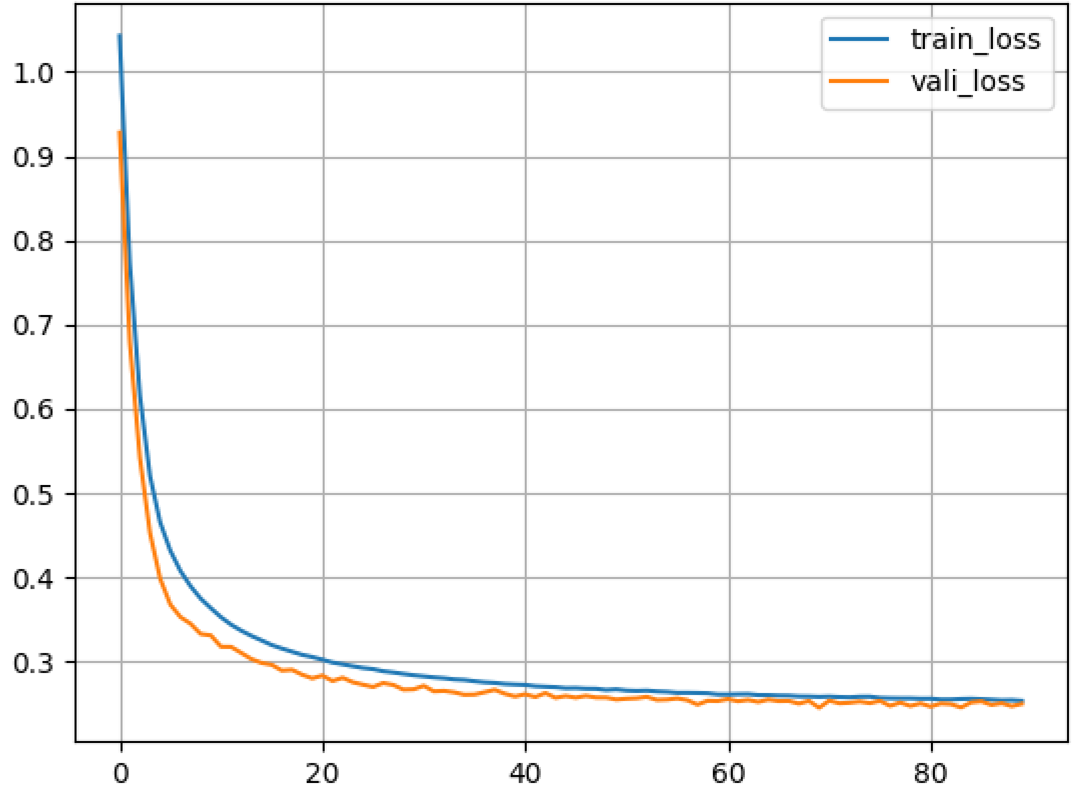
STIIMA

Keras Functional API

```
    self.__build_model__(loss)

def __build_model__(self, loss):
    x_in = tf.keras.layers.Input(shape=(self.settings['input_size']))
    logit = tf.keras.layers.Dense(self.settings['pred_horiz'],
                                  activation='linear',
                                  kernel_regularizer=tf.keras.regularizers.l1(self.settings['l1'])
                                  )(x_in)
    output = tf.reshape(logit, (-1, self.settings['pred_horiz'], 1))

    # Create model
    self.model= tf.keras.Model(inputs=[x_in], outputs=[output])
    # Compile the model
    self.model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=self.settings['lr']),
                       loss=loss
                       )
```

Simple ARX-l1 model in Tensorflow

```
def fit(self, train_x, train_y, val_x, val_y, verbose=0, pruning_call=None):
    # Convert the data into the input format using the internal converter
    train_x = self.build_model_input_from_series(x=train_x,
                                                 col_names=self.settings['x_columns_names'],
                                                 pred_horiz=self.settings['pred_horiz'])
    val_x = self.build_model_input_from_series(x=val_x,
                                               col_names=self.settings['x_columns_names'],
                                               pred_horiz=self.settings['pred_horiz'])

    es = tf.keras.callbacks.EarlyStopping(monitor="val_loss",
                                          patience=self.settings['patience'],
                                          restore_best_weights=False)

    # Create folder to temporally store checkpoints
    checkpoint_path = os.path.join(os.getcwd(), 'tmp_checkpoints', 'cp.ckpt')
    checkpoint_dir = os.path.dirname(checkpoint_path)
    if not os.path.exists(checkpoint_dir):
        os.makedirs(checkpoint_dir)

    cp = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                                            monitor="val_loss", mode="min",
                                            save_best_only=True,
                                            save_weights_only=True, verbose=0)

    if pruning_call==None:
        callbacks = [es, cp]
    else:
        callbacks = [es, cp, pruning_call]

    history = self.model.fit(train_x,
                             train_y,
                             validation_data=(val_x, val_y),
                             epochs=self.settings['max_epochs'],
                             batch_size=self.settings['batch_size'],
                             callbacks=callbacks,
                             verbose=verbose)

    # Load best weights: do not use restore_best_weights from early stop since works only in case it stops training
    self.model.load_weights(checkpoint_path)
    # delete temporary folder
    shutil.rmtree(checkpoint_dir)
    return history
```



2 usages (1 dynamic)

```
def predict(self, x):
    x = self.build_model_input_from_series(x=x,
                                           col_names=self.settings['x_columns_names'],
                                           pred_horiz=self.settings['pred_horiz'])
    return self.model(x)
```

2 usages (1 dynamic)

```
def evaluate(self, x, y):
    x = self.build_model_input_from_series(x=x,
                                           col_names=self.settings['x_columns_names'],
                                           pred_horiz=self.settings['pred_horiz'])
    return self.model.evaluate(x=x, y=y)
```

Run recalibration experiments

```
{
    "data_config": {
        "dataset_name": "EM_market__2015-01-03__2017-12-31.csv",
        "idx_start_train": {
            "y": 2015,
            "m": 1,
            "d": 3
        },
        "idx_start_oos_preds": {
            "y": 2017,
            "m": 1,
            "d": 1
        },
        "idx_end_oos_preds": {
            "y": 2017,
            "m": 1,
            "d": 3
        },
        "keep_past_train_samples": false,
        "steps_lag_win": 2,
        "pred_horiz": 24,
        "preprocess": "StandardScaler",
        "shuffle_mode": "none",
        "num_vali_samples": 100
    },
    "model_config": {
        "PF_method": "point",
        "model_class": "ARX",
        "optuna_m": "grid_search",
        "target_alpha": [
        ],
        "max_epochs": 800,
        "batch_size": 64,
        "patience": 20,
        "num_ense": 1
    }
}
```

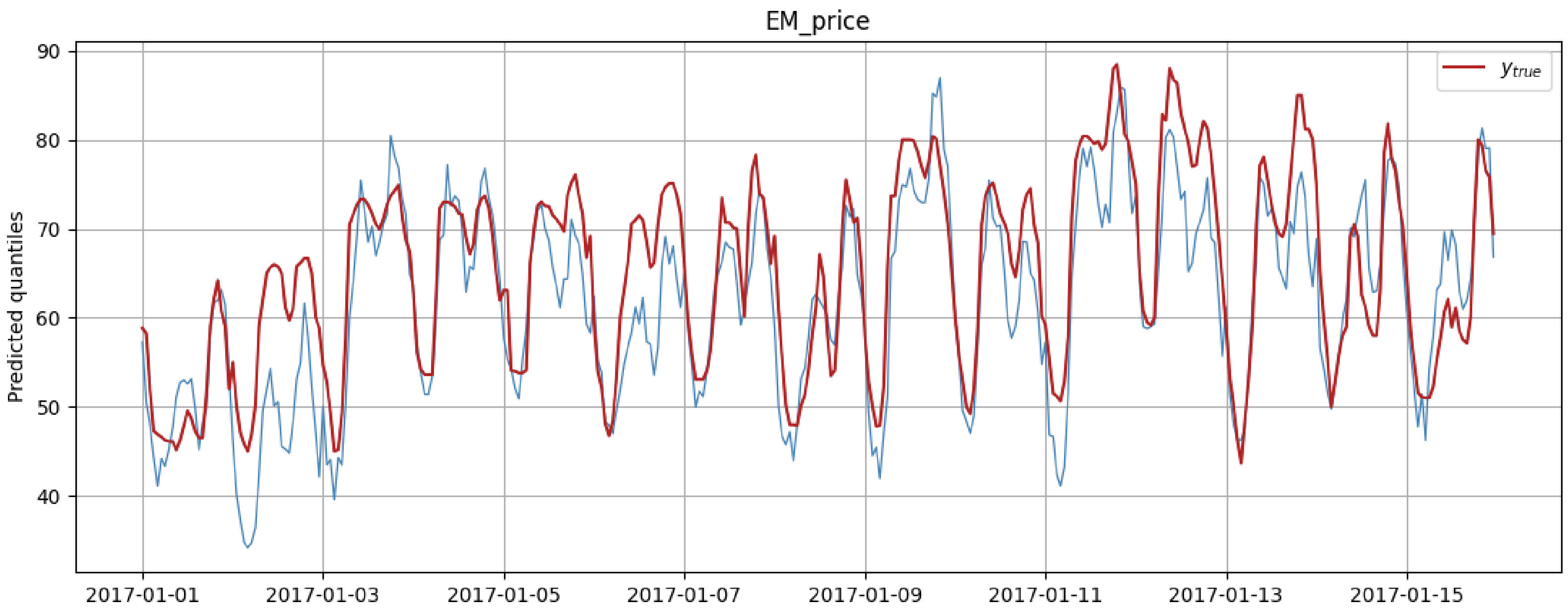
Set experiment configs in json

Play with the ARX-l1 hyperparams:

- L1 weight
- Learning rate

```
{"l1": 1e-03, "lr": 0.001}
```

Observe the impact on the results



Assignment PEF1

```
logit = tf.keras.layers.Dense(self.settings['pred_horiz'],
                              activation='linear',
                              kernel_regularizer=tf.keras.regularizers.l1(self.settings['l1'])
                              )(x_in)
```

- Experiment ARX-l1 using different l1 values (e.g., 1e-1, 1e-3, 1e-5)
- Plot and analyze the weights (e.g.: get_weights())
- Investigate Ridge/Elastic Net (l1 vs l2 vs l1l2)
- Provide a brief report of the performed experiments and observations

Facultative:

- Implement evaluation metrices (hourly and average): RMSE, MAE, sMAPE
- Compare the impact on the metrics of the hyperparameters setting

Consiglio Nazionale delle Ricerche

Thanks

Alessandro Brusaferri