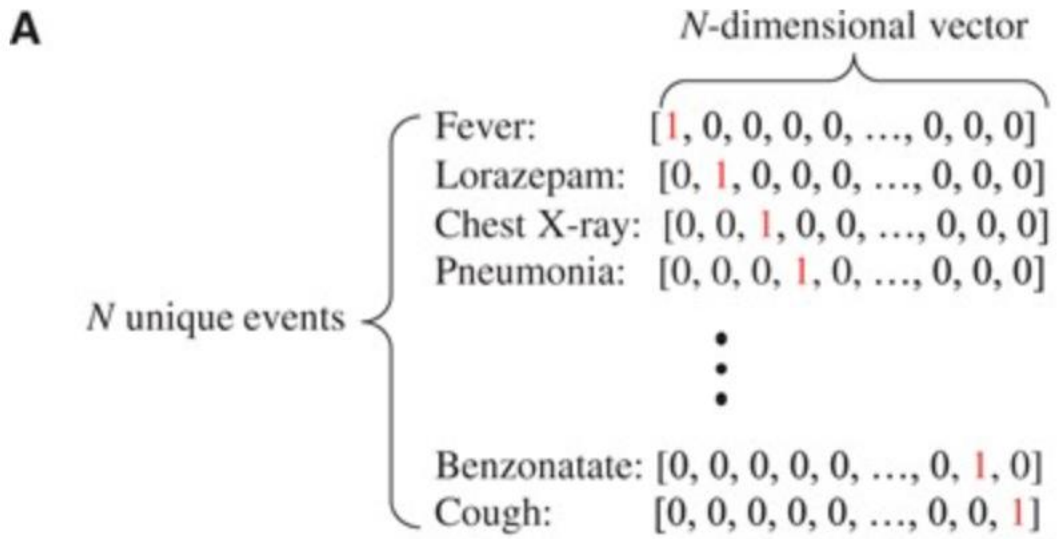
Week9 tutorial

Review: Prediction with temporal data - RNN

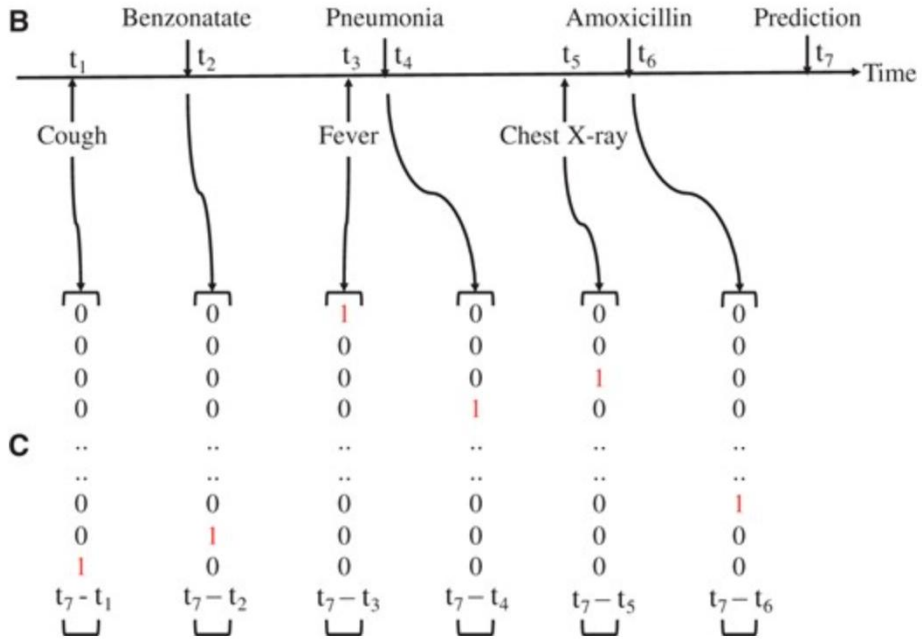
- Data preparation: convert medical events into one-hot code



A: an example of one-hot encoding for N unique medical events. 1 in one particular position means a particular event is occurred.

Review: Prediction with temporal data - RNN

- Data preparation: we can also add time information as the extra feature



B: indicates the time at which the event occurs, assuming we make the prediction at time t_7.
C: We appended the time duration features at the end of the one-hot vector.

RNN in Pytorch

- A **recurrent neural network** (**RNN**) is a class of neural network where connections between nodes form a directed or undirected graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior.

- It allows us to extra temporal information form the input.

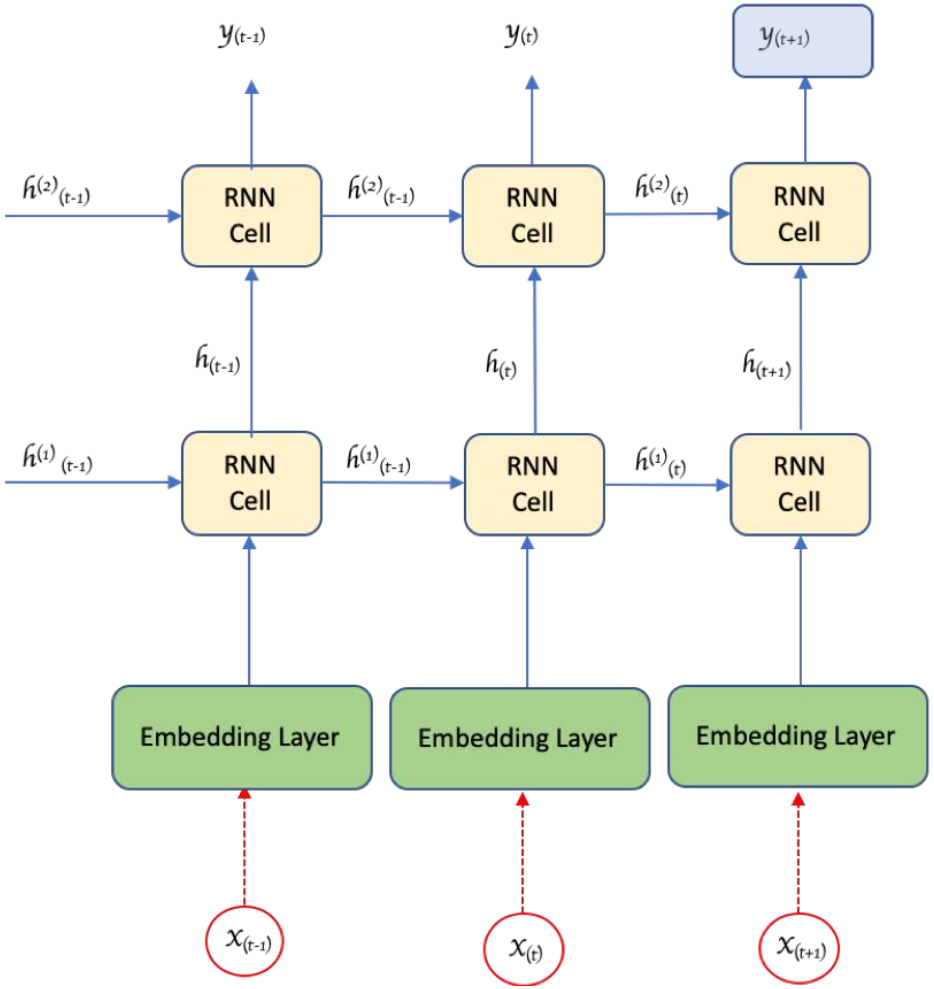
RNN in Pytorch

- Different with normal neural network such as CNN, which only consider the current input, RNN make decision based on the current input and the previous inputs.

- We achieve this by using the memory cell to store the information from the past.

$$h_t = \tanh(W_{ih}x_t + b_{ih} + W_{hh}h_{(t-1)} + b_{hh})$$

RNN in Pytorch



Target

- For each prediction, the a sequence of input will be passed to the memory cell h one by one, which store the information from the past.
- All information will combine together to get the final prediction
- h will be updated each time.

RNN in Pytorch

- Detail for the temporal data

As a temporal data, that means each sample is a sequence of data rather than one. For example, each patient has a sequence of EHR records. And we should have a sequence of output (expected results). It could be like: the probability of getting heart failure after each visiting, or the prediction of symptom in the next visiting.

RNN in Pytorch

- Detail for the temporal data

Assuming the length of a sequence is T, while all data have same length.

During the training progress, the input shape of the data should be like:

TxNxD,

Where the N is the number of data (e.g., the whole EHR of one patient),

D is the number of dimension of each timestamp (e.g., the record for each visiting).

RNN in Pytorch

- Detail for the temporal data

Assuming the length of a sequence is T, while all data have same length.

The ground truth data should have the shape:

TxNxd,

Where d is the number of dimension for the prediction in each timestamp.

We will discuss about the case when the sequence length is different for each patient later.

RNN in Pytorch

RNN

CLASS torch.nn.RNN(*args, **kwargs)

```
>>> rnn = nn.RNN(10, 20, 2)
>>> input = torch.randn(5, 3, 10)
>>> h0 = torch.randn(2, 3, 20)
>>> output, hn = rnn(input, h0)
```

RNN in Pytorch

- Parameters for RNN:

input_size – The number of expected features in the input *x*
which corresponding to the input dimension!

hidden_size – The number of features in the hidden state *h*

RNN in Pytorch

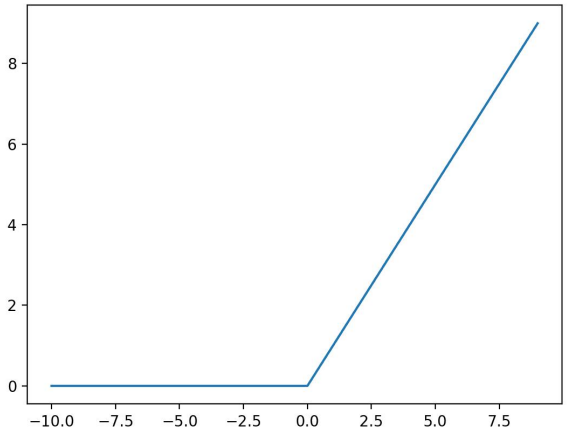
- Parameters for RNN:

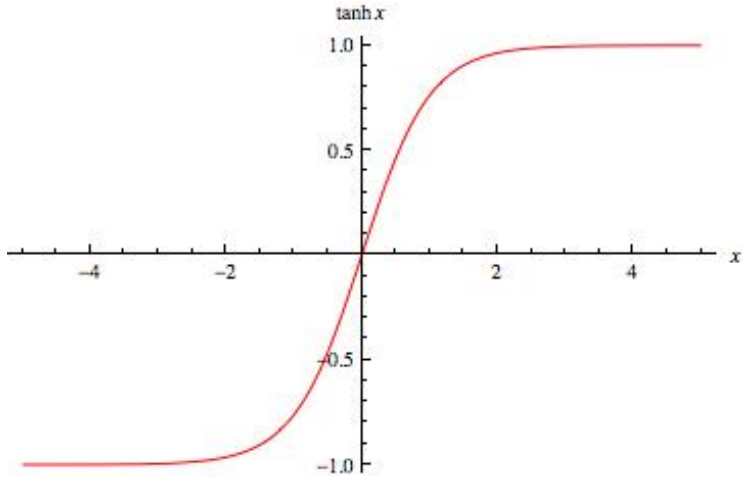
num_layers – Number of recurrent layers. E.g.,
setting num_layers=2 would mean stacking two RNNs together to form a *stacked RNN*, with the second RNN taking in outputs of the first RNN and computing the final results. Default: 1

RNN in Pytorch

- Parameters for RNN:

nonlinearity – The non-linearity to use. Can be either 'tanh' or 'relu'. Default: 'tanh'





$$h_t = \tanh(W_{ih}x_t + b_{ih} + W_{hh}h_{(t-1)} + b_{hh})$$

RNN in Pytorch

Parameters for RNN:

- **bias** – If False, then the layer does not use bias weights *b_ih* and *b_hh*. Default: True

Bias **allows you to shift the activation function by adding a constant (i.e. the given bias) to the input**. Bias in Neural Networks can be thought of as analogous to the role of a constant in a linear function, whereby the line is effectively transposed by the constant value.

RNN in Pytorch

Parameters for RNN:

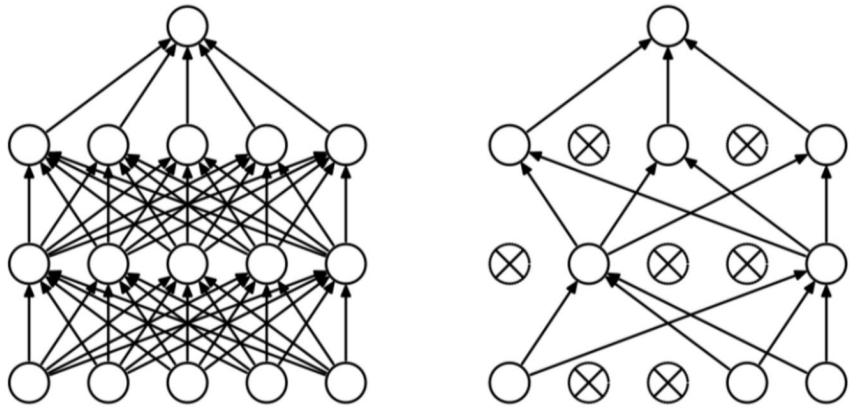
- **batch_first** – If True, then the input and output tensors are provided as *(batch, seq, feature)* instead of *(seq, batch, feature)*. Note that this does not apply to hidden or cell states. See the Inputs/Outputs sections below for details. Default: False

RNN in Pytorch

Parameters for RNN:

dropout – the dropout probability If non-zero, introduces a *Dropout* layer on the outputs of each RNN layer except the last layer, with dropout probability equal to dropout. Default: 0

For example, if dropout = 0.5, that means 50% neurons of the layer will be dropped. Appl ffect of overfitting.



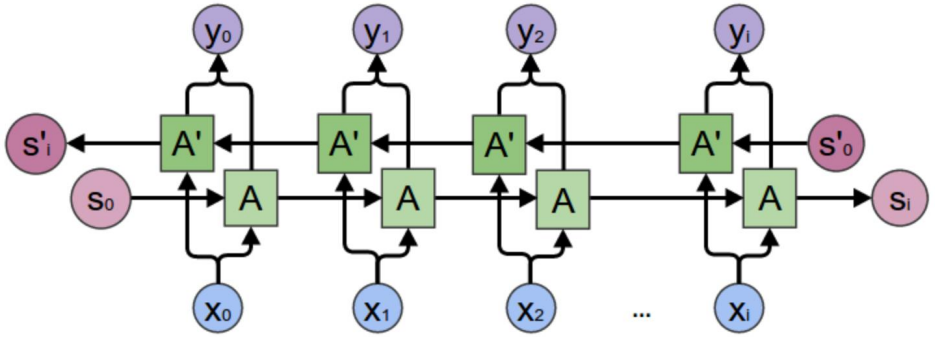
(a) Standard Neural Net (b) After applying dropout.

RNN in Pytorch

Parameters for RNN:

- **bidirectional** – If True, becomes a bidirectional RNN. Default: False

A BRNN is a combination of two RNNs - one RNN moves forward, beginning from the start of the data sequence, and the other, moves backward, beginning from the end of the data sequence.



Bidirectional RNN (Source: Colah)

RNN in Pytorch

Input for RNN model:

- Input: the input tensor. If batch_first=False: the shape of input should be (T,N,d), if batch_first=True, then the shape should be (N,T,d).

- h_0: the initial hidden state values. The shape should be like (D*num of layers, N, d). D=2 if bidirectional=True, D=1 if bidirectional=False. If we don’t provide the initial hidden state then h_0 will be zero values.

RNN in Pytorch

Output for RNN model:

- **output**: : the output tensor. If batch_first=False: the shape of input should be (T,N,D*d), if batch_first=True, then the shape should be (N,T,D*d).

Consider the input sequence has T stage: 1, 2, 3, …, T-1, T

The output are the prediction of next stage based on the previous stages: 2, 3, 4, …, T, T+1.

RNN in Pytorch

Output for RNN model:

- h_n: the final hidden state.

RNN in Pytorch

Example:

```
>>> rnn = nn.RNN(10, 20, 2)
>>> input = torch.randn(5, 3, 10)
>>> h0 = torch.randn(2, 3, 20)
>>> output, hn = rnn(input, h0)
```

RNN in Pytorch

- More about the input:

You might notice that the input sizes (the sequence length and dimension) are fixed. But the EHR record and visiting times could be different for each patient. How can we solve this problem?

RNN in Pytorch

- More about the input:

1. the EHR record

Since the HER will record all the symptom, the type and number of symptom should be different for each visiting and patient.

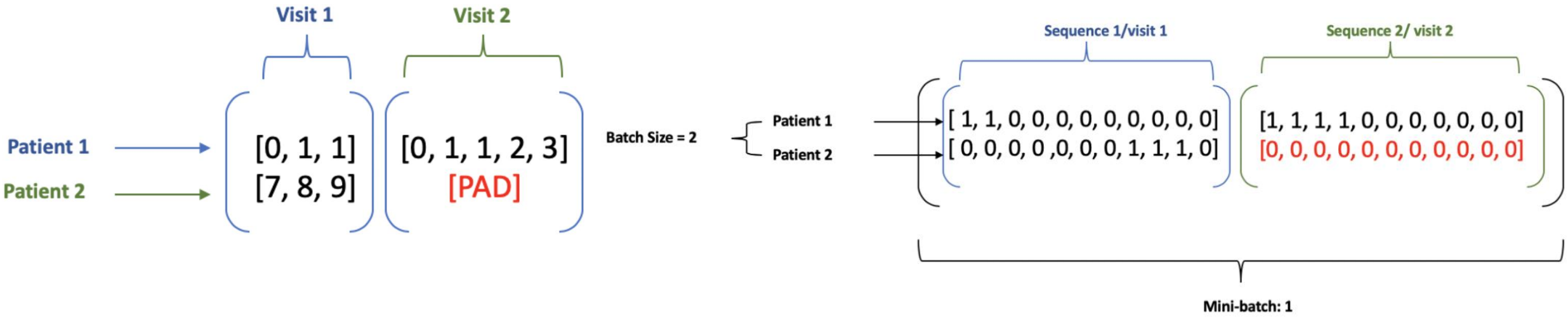
We can binarize each record to keep the same size

RNN in Pytorch

- More about the input:

1. the EHR record

We can binarize each record to keep the same size



RNN in Pytorch

- More about the input:

1. the EHR record

We can binarize each record to keep the same size

```
sklearn.preprocessing.MultiLabelBinarizer(*, classes=None, sparse_output=False)
```

Classes: *array-like of shape (n_classes,), default=None*Indicates an ordering for the class labels. All entries should be unique (cannot contain duplicate classes).

RNN in Pytorch

- More about the input:

1. the EHR record

We can binarize each record to keep the same size

```
>>> from sklearn.preprocessing import MultiLabelBinarizer
>>> mlb = MultiLabelBinarizer()
>>> mlb.fit_transform([(1, 2), (3,)])
array([[1, 1, 0],
       [0, 0, 1]])
>>> mlb.classes_
array([1, 2, 3])
```

RNN in Pytorch

- More about the input:

1. the EHR record

We can binarize each record to keep the same size

If you want the fixed number of class:

```
shapee = np.arange(1,numClass+1)
mlb = MultiLabelBinarizer(classes=shapee)
```

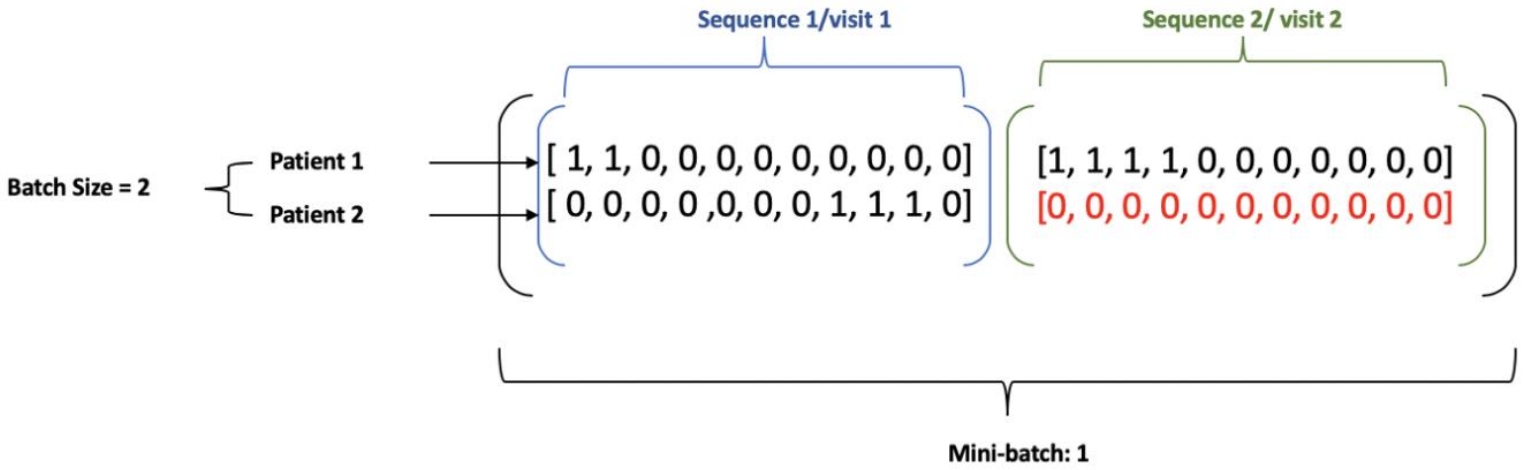
RNN in Pytorch

- More about the input:

The sequence length:

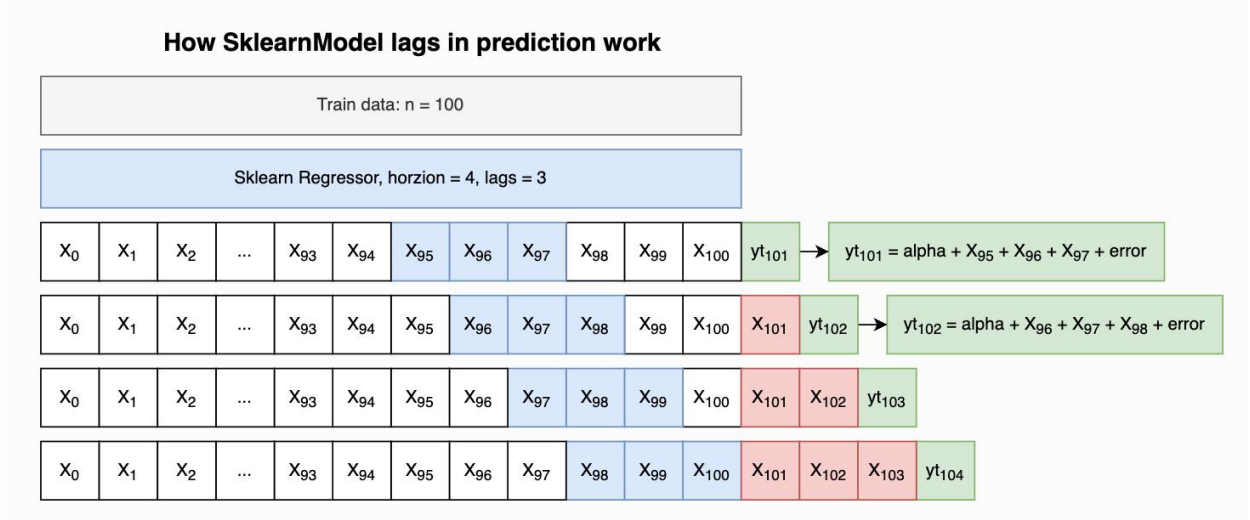
Idea:

We can apply padding to the shorter sample with 0s.



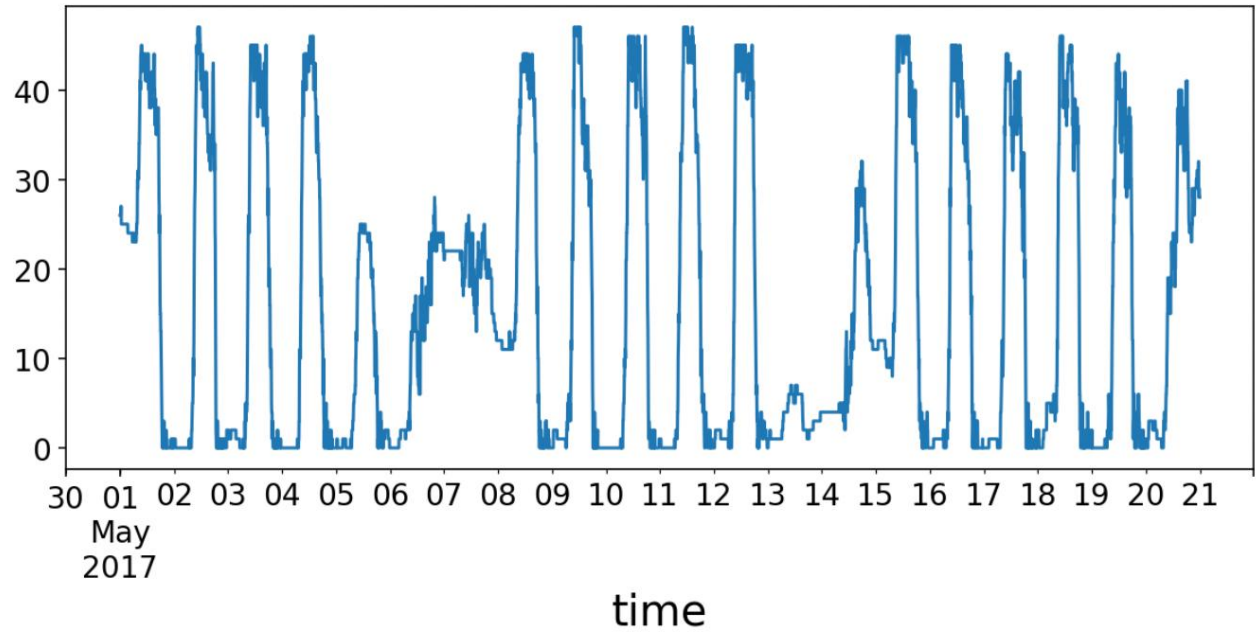
Autoregressive model in Sklearn

- Autoregressive is a traditional model for time-series predictions.
- The autoregressive model specifies that the output variable depends linearly on its own previous values and on a stochastic term (an imperfectly predictable term).



Autoregressive model in Sklearn

- Assuming the temporal data is a sequence of values changed in different stage.



Autoregressive model in Sklearn

- A regression model is required for autoregressive prediction.

- There are multiple regression are available in Sklearn, such as linear regression and random forest.

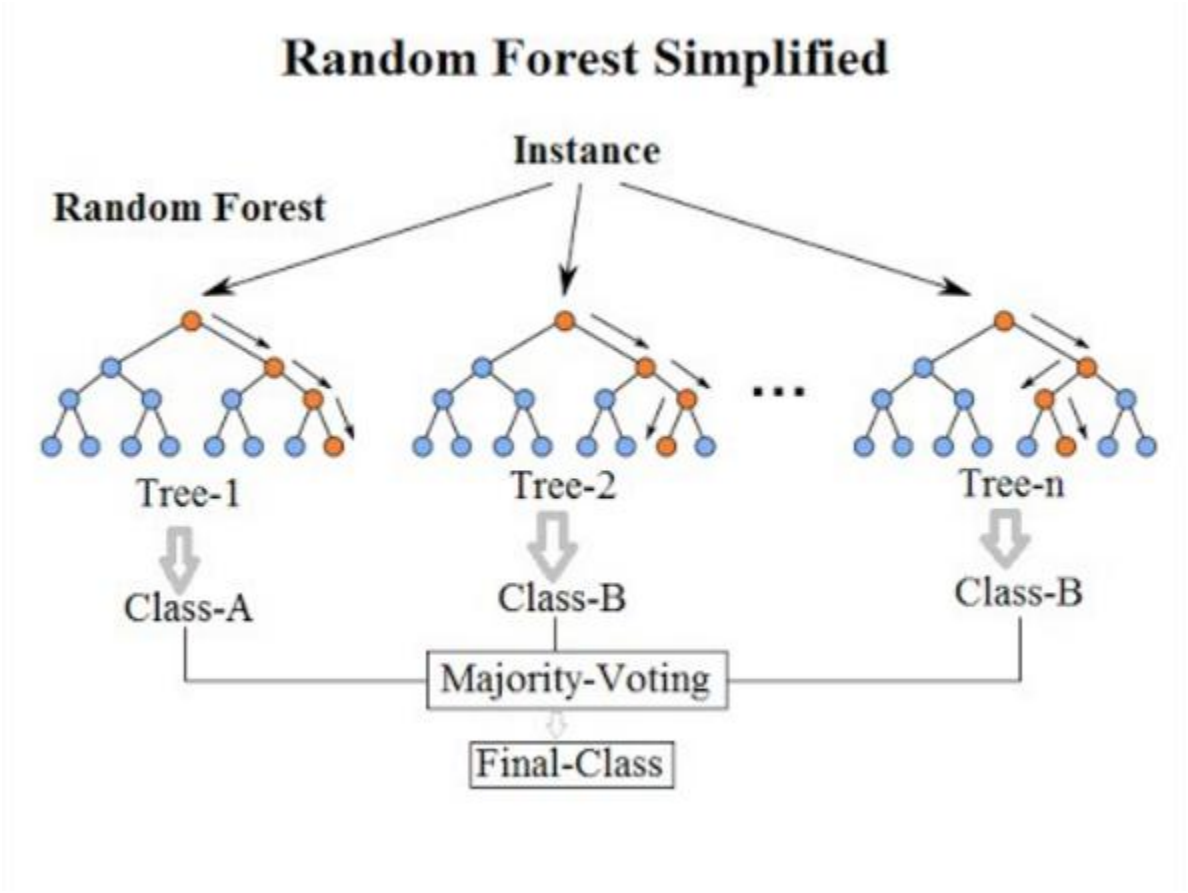
Autoregressive model in Sklearn

- Random forest: training multiple small predictor (called weak estimator) with randomly selected subset of the whole dataset, and we make the final prediction based on the result of all estimator.

- Using multiple predictor could provide more perspective and extract more information from the dataset, such as it might even better than a single huge prediction model.

- The final prediction could be made by majority voting: we pick the values that most of the estimator predicted.

Autoregressive model in Sklearn



Autoregressive model in Sklearn

sklearn.ensemble.RandomForestRegressor ¶

class `sklearn.ensemble.`**`RandomForestRegressor`**(*n_estimators=100, *, criterion='squared_error', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=1.0, max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, ccp_alpha=0.0, max_samples=None*) [source]

Autoregressive model in Sklearn

- **n_estimators** *int, default=100*

  The number of trees in the forest.

- Criterion: {“squared_error”, “absolute_error”, “poisson”}, default=”squared_error”

  The function to measure how good about the current estimator, and whether we need to further split the tree. Poisson is a method to measure the probability of the event occur in a specific time.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

$$|\boldsymbol{v}_A - \boldsymbol{v}_E|$$

Autoregressive model in Sklearn

- **max_depth** *int, default=None*

  The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

- **min_samples_split** *int, default=2*

  The minimum number of samples required to split an internal node

- **Bootstrap** *bool, default=True*

  Whether bootstrap samples are used when building trees. If False, the whole dataset is used to build each tree.

  In statistic, bootstrap means random sampling with replacement. E.g., random sample k samples within the dataset of N size, where k << N.

Autoregressive model in Sklearn

- **random_state** int**,** *default=None:*

  Only work when bootstrap=True, setting the random seed to random sampling.

- **n_jobs** *int, default=None*The number of jobs to run in parallel.

Autoregressive model in Sklearn

- Function for random forest:

  fit(X, y): Build a forest of trees from the training set (X, y)

  X: input data, shape like (num of samples, num of dimension).

  y: ground truth data, shape like (num of samples, 1)

Autoregressive model in Sklearn

- We use Hcrystal Ball package with Sklearn to achieve autoregressive.

- It is a wrapper of Sklearn regression model for time-series prediction tasks.

Autoregressive model in Sklearn

get_sklearn_wrapper

hcrystalball.wrappers.get_sklearn_wrapper(*model_cls*, ***model_params*) [source]

Autoregressive model in Sklearn

- **model_cls** (*class of sklearn compatible regressor*) – i.e. LinearRegressor, GradientBoostingRegressor

- **model_params** – model_cls specific parameters (e.g. max_depth), just like the params we introduced in randomforest! And/Or SklearnWrapper specific parameters (clip_predictions_lower)

  clip_predictions_lower int, default = None. If the value is given the minimum value of prediction result will be set as the given value.

Autoregressive model in Sklearn

```
from hcrystalball.wrappers._sklearn import _get_sklearn_wrapper
from sklearn.ensemble import RandomForestRegressor
est = get_sklearn_wrapper(RandomForestRegressor, max_depth=6, clip_predictions_lower=0.)
```

Pred = est.fit(X[:-10], y[:-10]) .predict(X[-10:])