Défis en Intelligence Artificielle

Défi 3 : L'IA pour l'analyse et la prévision de séries temporelles

Souhaib BEN TAIEB

UMONS
Université de Mons

Faculté
des Sciences

December 12, 2019

DataCamp courses/assignments

- Supervised Learning with scikit-learn
- Machine Learning for Time Series Data in Python
- Introduction to Deep Learning with Keras
- **Winning a Kaggle Competition in Python**
- (Optional) Hyperparameter Tuning in Python
- (Optional) Machine Learning with Tree-Based Models in Python

More details at Github link

- Python For Data Science Cheat Sheet
  - Pandas
  - Scikit-Learn
  - Keras
  - More at `https://www.datacamp.com/community/data-science-cheatsheets`

From time series forecasting to regression

- Forecasting strategies
  - Recursive
  - Direct
  - Multi-output
  - ...
- Regression methods
  - Linear regression methods: least-squares, ridge, lasso, etc
  - Machine learning methods: tree-based methods, kernel methods, nearest-neighbors, neural networks, etc

Recursive strategy

$$y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9, y_{10} \rightarrow ?, ?, ?$$

X			**y**
y_{t-2}	y_{t-1}	y_t	y_{t+1}
y_1	y_2	y_3	y_4
y_2	y_3	y_4	y_5
y_3	y_4	y_5	y_6
y_4	y_5	y_6	y_7
y_5	y_6	y_7	y_8
y_6	y_7	y_8	y_9
y_7	y_8	y_9	y_{10}
y_8	y_9	y_{10}	$?$

$$y_8, y_9, y_{10} \rightarrow \hat{y}_{11} \qquad y_9, y_{10}, \hat{y}_{11} \rightarrow \hat{y}_{12} \qquad y_{10}, \hat{y}_{11}, \hat{y}_{12} \rightarrow \hat{y}_{13}$$

Direct strategy

$$y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9, y_{10} \rightarrow ?, ?, ?$$

X			y
y_{t-2}	y_{t-1}	y_t	y_{t+1}
y_1	y_2	y_3	y_4
y_2	y_3	y_4	y_5
y_3	y_4	y_5	y_6
y_4	y_5	y_6	y_7
y_5	y_6	y_7	y_8
y_6	y_7	y_8	y_9
y_7	y_8	y_9	y_{10}
y_8	y_9	y_{10}	?

X			y
y_{t-2}	y_{t-1}	y_t	y_{t+2}
y_1	y_2	y_3	y_5
y_2	y_3	y_4	y_6
y_3	y_4	y_5	y_7
y_4	y_5	y_6	y_8
y_5	y_6	y_7	y_9
y_6	y_7	y_8	y_{10}
y_8	y_9	y_{10}	?

X			y
y_{t-2}	y_{t-1}	y_t	y_{t+3}
y_1	y_2	y_3	y_6
y_2	y_3	y_4	y_7
y_3	y_4	y_5	y_8
y_4	y_5	y_6	y_9
y_5	y_6	y_7	y_{10}
y_8	y_9	y_{10}	?

$$y_8, y_9, y_{10} \rightarrow \hat{y}_{11} \qquad y_8, y_9, y_{10} \rightarrow \hat{y}_{12} \qquad y_8, y_9, y_{10} \rightarrow \hat{y}_{13}$$

Multi-output strategy

$$y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9, y_{10} \rightarrow ?, ?, ?$$

X			**y**		
y_{t-2}	y_{t-1}	y_t	y_{t+1}	y_{t+2}	y_{t+3}
y_1	y_2	y_3	y_4	y_5	y_6
y_2	y_3	y_4	y_5	y_6	y_7
y_3	y_4	y_5	y_6	y_7	y_8
y_4	y_5	y_6	y_7	y_8	y_9
y_5	y_6	y_7	y_8	y_9	y_{10}
y_8	y_9	y_{10}	?		

$$y_8, y_9, y_{10} \rightarrow \hat{y}_{11}, \hat{y}_{12}, \hat{y}_{13}$$

$\rightarrow$ The multi-output strategy requires a model that can deal with multiple outputs, e.g. neural networks.

Cross-validation for time series forecasting

$$y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9, y_{10}$$

- You generally should **not use datapoints in the future to predict data in the past**. Use training data from the past to predict the future.
  - $y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9, y_{10} \rightarrow$ ☹
- You should **not shuffle** your data when making predictions with time series. Otherwise, you destroy the correlation structure in the data
  - $\underbrace{y_3, y_2, y_3, y_7, y_4, y_6}_{\text{Training}}, \underbrace{y_1, y_{10}, y_2, y_5}_{\text{Validation}} \rightarrow$ ☹
- The validation (forecast) error is computed by averaging over the validation set
  - $\underbrace{y_1, y_2, y_3, y_4, y_5, y_6}_{\text{Training}}, \underbrace{y_7, y_8, y_9, y_{10}}_{\text{Validation}} \rightarrow$ ☺
- **One-step** vs **multi-step** ahead forecasting
  - Traditional cross-validation can be used with **auto-regressive** data for one-step ahead forecasting

Cross-validation for time series forecasting

$$\underbrace{y_1, y_2, y_3, y_4, y_5, y_6}_{\text{Training}}, \underbrace{y_7, y_8, y_9, y_{10}}_{\text{Validation}}$$

$\mathbf{X}$			$\mathbf{y}$
y_{t-2}	y_{t-1}	y_t	y_{t+1}
y_1	y_2	y_3	y_4
y_2	y_3	y_4	y_5
y_3	y_4	y_5	y_6
y_4	y_5	y_6	y_7
y_5	y_6	y_7	y_8
y_6	y_7	y_8	y_9
y_7	y_8	y_9	y_{10}

$\rightarrow$ Simple training/validation split, also called the validation set approach (without shuffling)

Cross-validation for time series forecasting

$$\underbrace{y_1, y_2, y_3, y_4, y_5, y_6}_{\text{Training}}, \underbrace{y_7, y_8, y_9, y_{10}}_{\text{Validation}}$$

X			y
y_{t-2}	y_{t-1}	y_t	y_{t+1}
y_1	y_2	y_3	y_4
y_2	y_3	y_4	y_5
y_3	y_4	y_5	y_6
y_4	y_5	y_6	y_7

X			y
y_{t-2}	y_{t-1}	y_t	y_{t+1}
y_1	y_2	y_3	y_4
y_2	y_3	y_4	y_5
y_3	y_4	y_5	y_6
y_4	y_5	y_6	y_7
y_5	y_6	y_7	y_8

X			y
y_{t-2}	y_{t-1}	y_t	y_{t+1}
y_1	y_2	y_3	y_4
y_2	y_3	y_4	y_5
y_3	y_4	y_5	y_6
y_4	y_5	y_6	y_7
y_5	y_6	y_7	y_8
y_6	y_7	y_8	y_9

X			y
y_{t-2}	y_{t-1}	y_t	y_{t+1}
y_1	y_2	y_3	y_4
y_2	y_3	y_4	y_5
y_3	y_4	y_5	y_6
y_4	y_5	y_6	y_7
y_5	y_6	y_7	y_8
y_6	y_7	y_8	y_9
y_7	y_8	y_9	y_{10}

$\rightarrow$ This procedure is sometimes known as "evaluation on a rolling forecasting origin" because the "origin" at which the forecast is based rolls forward in time.

Cross-validation for time series forecasting

$$\underbrace{y_1, y_2, y_3, y_4, y_5, y_6}_{\text{Training}}, \underbrace{y_7, y_8, y_9, y_{10}}_{\text{Validation}}$$

X			y
y_{t-2}	y_{t-1}	y_t	y_{t+1}
y_1	y_2	y_3	y_4
y_2	y_3	y_4	y_5
y_3	y_4	y_5	y_6
y_4	y_5	y_6	y_7

X			y
y_{t-2}	y_{t-1}	y_t	y_{t+1}
y_2	y_3	y_4	y_5
y_3	y_4	y_5	y_6
y_4	y_5	y_6	y_7
y_5	y_6	y_7	y_8

X			y
y_{t-2}	y_{t-1}	y_t	y_{t+1}
y_3	y_4	y_5	y_6
y_4	y_5	y_6	y_7
y_5	y_6	y_7	y_8
y_6	y_7	y_8	y_9

X			y
y_{t-2}	y_{t-1}	y_t	y_{t+1}
y_4	y_5	y_6	y_7
y_5	y_6	y_7	y_8
y_6	y_7	y_8	y_9
y_7	y_8	y_9	y_{10}

$\rightarrow$ The training data has always the same number of observations

Cross-validation and hyperparameters optimization

- Sklearn for Machine Learning models
  - See `sklearn.model_selection`.
  - See `sklearn.model_selection.TimeSeriesSplit`.
- Keras for Deep Learning
  - See `keras.preprocessing.sequence.TimeseriesGenerator`.
- Keras model to Sklearn estimator
  - See `keras.wrappers.scikit_learn.KerasRegressor`.
  - DataCamp: Introduction to Deep Learning with Keras - Improving Your Model Performance

Cross-validating timeseries data

MACHINE LEARNING FOR TIME SERIES DATA IN PYTHON

Chris Holdgraf
Fellow, Berkeley Institute for Data Science

DataCamp

Cross validation with scikit-learn

```
# Iterating over the "split" method yields train/test indices
for tr, tt in cv.split(X, y):
    model.fit(X[tr], y[tr])
    model.score(X[tt], y[tt])
```

Cross validation types: KFold

- `KFold` cross-validation splits your data into multiple "folds" of equal size
- It is one of the most common cross-validation routines

```
from sklearn.model_selection import KFold
cv = KFold(n_splits=5)
for tr, tt in cv.split(X, y):
    ...
```

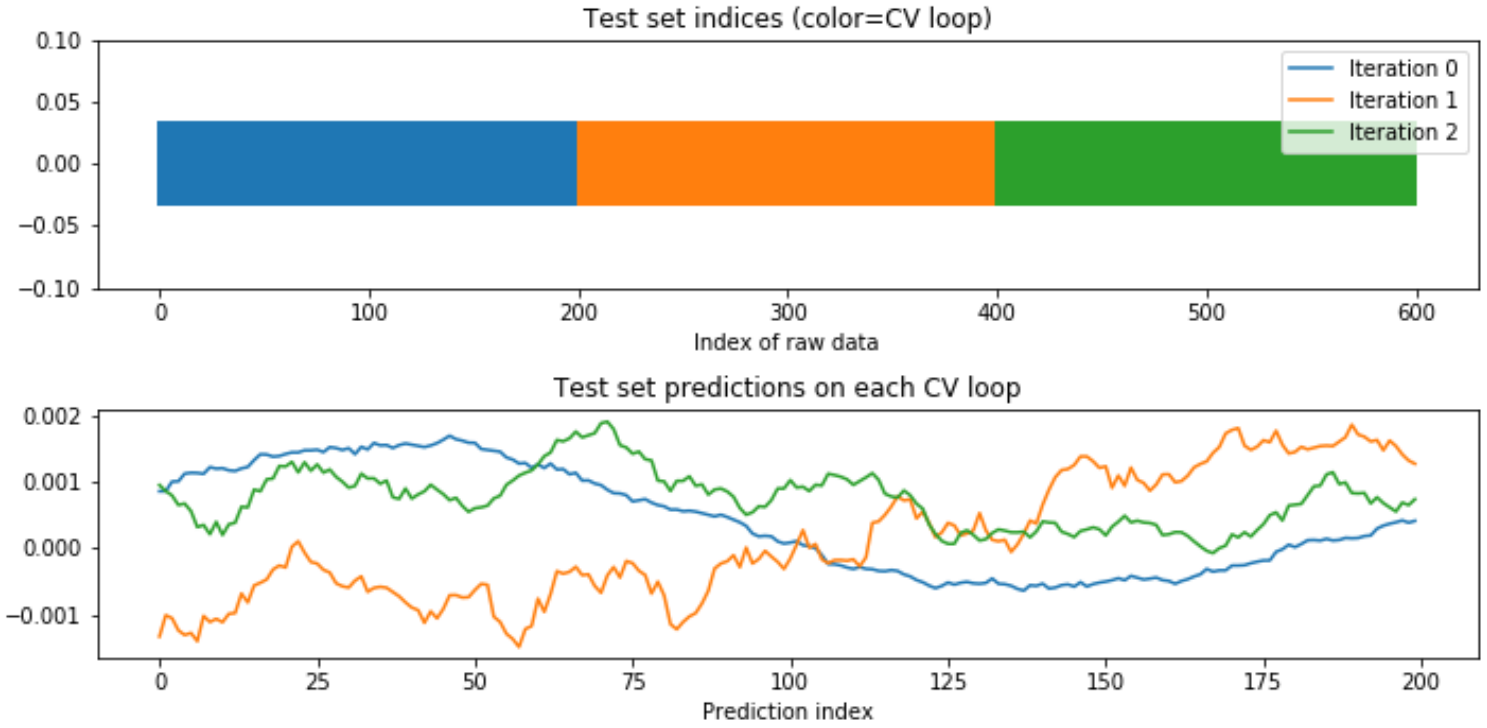
Visualizing model predictions

```
fig, axs = plt.subplots(2, 1)

# Plot the indices chosen for validation on each loop
axs[0].scatter(tt, [0] * len(tt), marker='_', s=2, lw=40)
axs[0].set(ylim=[-.1, .1], title='Test set indices (color=CV loop)',
           xlabel='Index of raw data')

# Plot the model predictions on each iteration
axs[1].plot(model.predict(X[tt]))
axs[1].set(title='Test set predictions on each CV loop',
           xlabel='Prediction index')
```

Visualizing KFold CV behavior



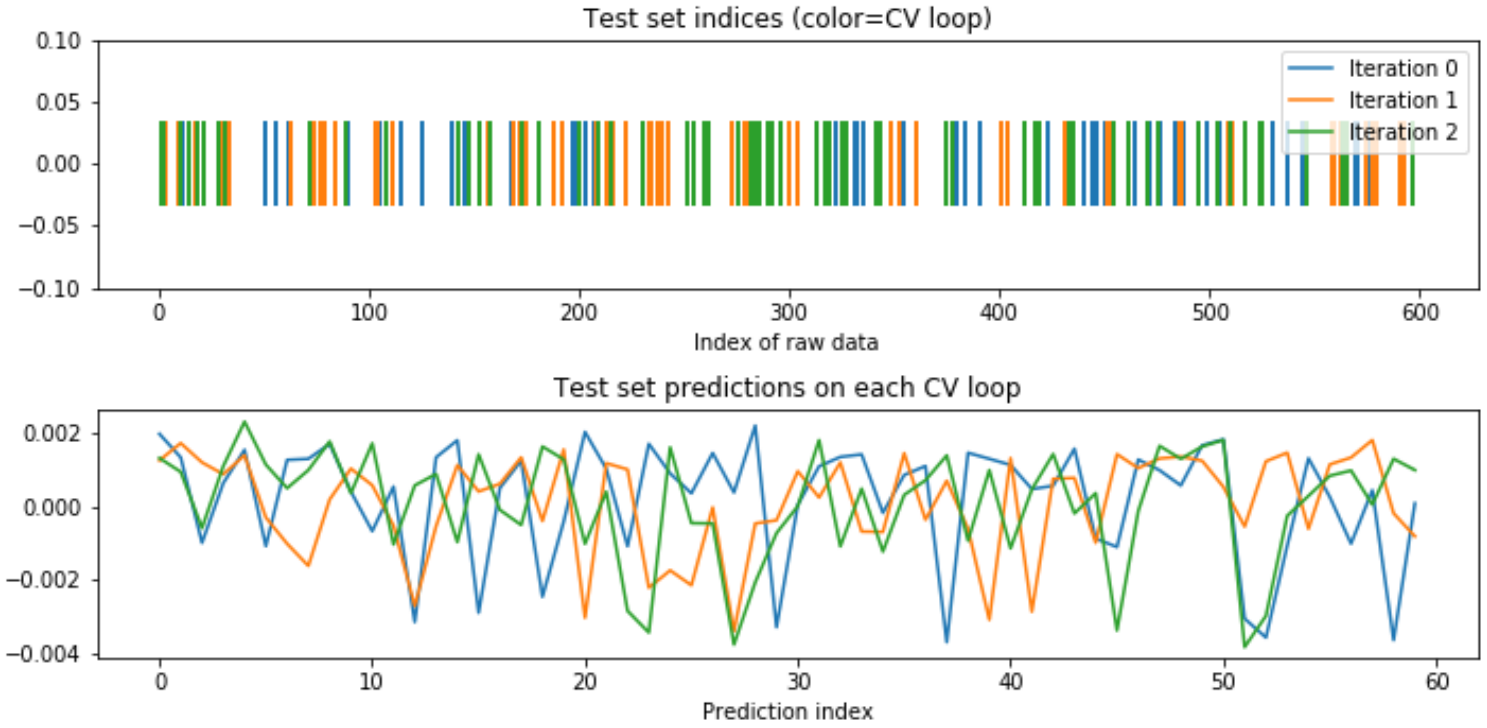
A note on shuffling your data

- Many CV iterators let you shuffle data as a part of the cross-validation process.
- This only works if the data is i.i.d., which timeseries usually is **not**.
- You should *not* shuffle your data when making predictions with timeseries.

```
from sklearn.model_selection import ShuffleSplit

cv = ShuffleSplit(n_splits=3)
for tr, tt in cv.split(X, y):
    ...
```

Visualizing shuffled CV behavior



Using the time series CV iterator

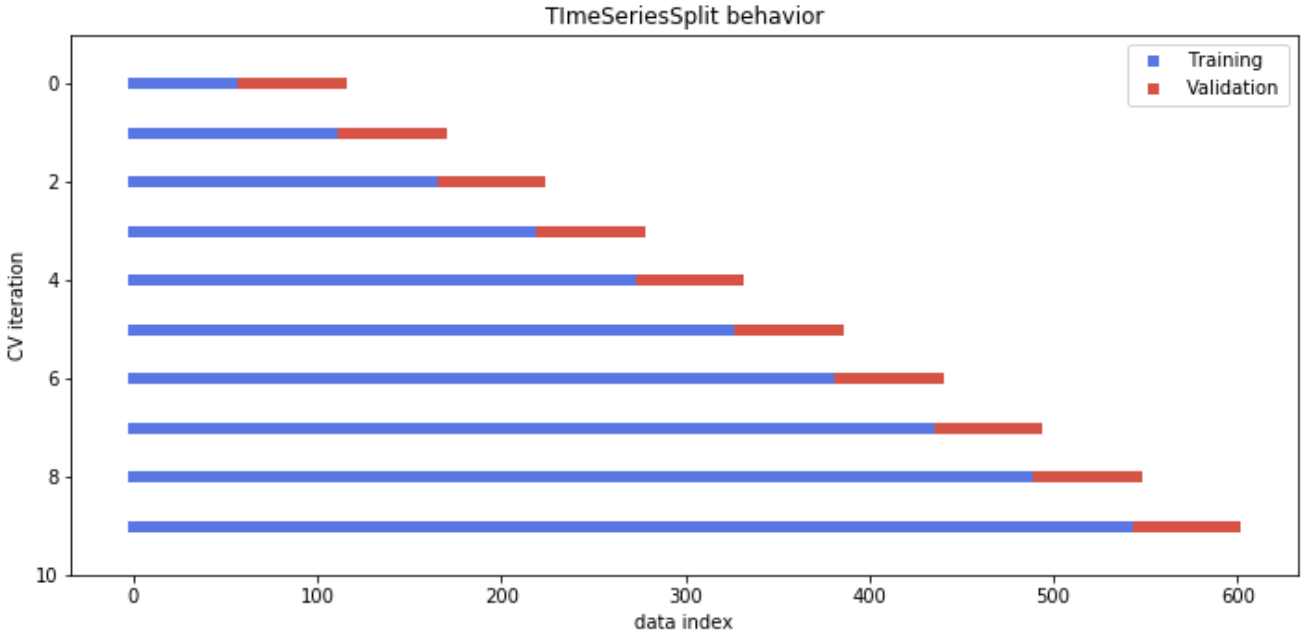
- Thus far, we've broken the linear passage of time in the cross validation
- However, you generally **should not** use datapoints in the future to predict data in the past
- One approach: Always use training data from the **past** to predict the **future**

Visualizing time series cross validation iterators

```
# Import and initialize the cross-validation iterator
from sklearn.model_selection import TimeSeriesSplit
cv = TimeSeriesSplit(n_splits=10)

fig, ax = plt.subplots(figsize=(10, 5))
for ii, (tr, tt) in enumerate(cv.split(X, y)):
    # Plot training and test indices
    l1 = ax.scatter(tr, [ii] * len(tr), c=[plt.cm.coolwarm(.1)],
                    marker='_', lw=6)
    l2 = ax.scatter(tt, [ii] * len(tt), c=[plt.cm.coolwarm(.9)],
                    marker='_', lw=6)
    ax.set(ylim=[10, -1], title='TimeSeriesSplit behavior',
           xlabel='data index', ylabel='CV iteration')
    ax.legend([l1, l2], ['Training', 'Validation'])
```

Visualizing the TimeSeriesSplit cross validation iterator



Custom scoring functions in scikit-learn

```
def myfunction(estimator, X, y):
    y_pred = estimator.predict(X)
    my_custom_score = my_custom_function(y_pred, y)
    return my_custom_score
```

Hyperparameter tuning

INTRODUCTION TO DEEP LEARNING WITH KERAS

Miguel Esteban
Data Scientist & Founder

Neural network hyperparameters

- Number of layers
- Number of neurons per layer
- Layer order
- Layer activations
- Batch sizes
- Learning rates
- Optimizers
- ...

Sklearn recap

```
# Import RandomizedSearchCV
from sklearn.model_selection import RandomizedSearchCV
# Instantiate your classifier
tree = DecisionTreeClassifier()
# Define a series of parameters to look over
params = {'max_depth':[3,None], "max_features":range(1,4), 'min_samples_leaf': range(1,4)}
# Perform random search with cross validation
tree_cv = RandomizedSearchCV(tree, params, cv=5)
tree_cv.fit(X,y)

# Print the best parameters
print(tree_cv.best_params_)
```

```
{'min_samples_leaf': 1, 'max_features': 3, 'max_depth': 3}
```

Turn a Keras model into a Sklearn estimator

```
# Function that creates our Keras model
def create_model(optimizer='adam', activation='relu'):
    model = Sequential()
    model.add(Dense(16, input_shape=(2,), activation=activation))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer=optimizer, loss='binary_crossentropy')
    return model

# Import sklearn wrapper from keras
from keras.wrappers.scikit_learn import KerasClassifier

# Create a model as a sklearn estimator
model = KerasClassifier(build_fn=create_model, epochs=6, batch_size=16)
```

Cross-validation

```
# Import cross_val_score
from sklearn.model_selection import cross_val_score

# Check how your keras model performs with 5 fold crossvalidation
kfold = cross_val_score(model, X, y, cv=5)
# Print the mean accuracy per fold
kfold.mean()
```

```
0.913333
```

```
# Print the standard deviation per fold
kfold.std()
```

```
0.110754
```

Tips for neural networks hyperparameter tuning

- Random search is preferred over grid search
- Don't use many epochs
- Use a smaller sample of your dataset
- Play with batch sizes, activations, optimizers and learning rates

Random search on Keras models

```
# Define a series of parameters
params = dict(optimizer=['sgd', 'adam'], epochs=3,
              batch_size=[5, 10, 20], activation=['relu','tanh'])

# Create a random search cv object and fit it to the data
random_search = RandomizedSearchCV(model, params_dist=params, cv=3)
random_search_results = random_search.fit(X, y)
# Print results
print("Best: %f using %s".format(random_search_results.best_score_,
random_search_results.best_params_))
```

```
Best: 0.94 using {'optimizer': 'adam', 'epochs': 3, 'batch_size': 10, 'activation': 'rel
```

Tuning other hyperparameters

```
def create_model(nl=1,nn=256):
    model = Sequential()
    model.add(Dense(16, input_shape=(2,), activation='relu'))
    # Add as many hidden layers as specified in nl
    for i in range(nl):
        # Layers have nn neurons
        model.add(Dense(nn, activation='relu'))
    # End defining and compiling your model...
```

Tuning other hyperparameters

```
# Define parameters, named just like in create_model()
params = dict(nl=[1, 2, 9], nn=[128,256,1000])

# Repeat the random search...

# Print results...
```

```
Best: 0.87 using {'nl': 2,'nn': 128}
```