Two Sigma Financial News Competition Official Getting Started Kernel

Introduction

In this competition you will predict how stocks will change based on the market state and news articles. You will loop through a long series of trading days; for each day, you'll receive an updated state of the market, and a series of news articles which were published since the last trading day, along with impacted stocks and sentiment analysis. You'll use this information to predict whether each stock will have increased or decreased ten trading days into the future. Once you make these predictions, you can move on to the next trading day.

This competition is different from most Kaggle Competitions in that:

- You can only submit from Kaggle Kernels, and you may not use other data sources, GPU, or internet access.
- This is a **two-stage competition**. In Stage One you can edit your Kernels and improve your model, where Public Leaderboard scores are based on their predictions relative to past market data. At the beginning of Stage Two, your Kernels are locked, and we will re-run your Kernels over the next six months, scoring them based on their predictions relative to live data as those six months unfold.
- You must use our custom `kaggle.competitions.twosigmanews` Python module. The purpose of this module is to control the flow of information to ensure that you are not using future data to make predictions for the current trading day.

In this Starter Kernel, we'll show how to use the `twosigmanews` module to get the training data, get test features and make predictions, and write the submission file.

TL;DR: End-to-End Usage Example

```
from kaggle.competitions import twosigmanews
env = twosigmanews.make_env()

(market_train_df, news_train_df) = env.get_training_data()
train_my_model(market_train_df, news_train_df)

for (market_obs_df, news_obs_df, predictions_template_df) in env.get_prediction_days():
  predictions_df = make_my_predictions(market_obs_df, news_obs_df, predictions_template_df)
  env.predict(predictions_df)

env.write_submission_file()
```

Note that `train_my_model` and `make_my_predictions` are functions you need to write for the above example to work.

In-depth Introduction

First let's import the module and create an environment.

In [ ]:
```
from kaggle.competitions import twosigmanews
# You can only call make_env() once, so don't lose it!
env = twosigmanews.make_env()
```

`get_training_data` function

Returns the training data DataFrames as a tuple of:

- `market_train_df` : DataFrame with market training data
- `news_train_df` : DataFrame with news training data

These DataFrames contain all market and news data from February 2007 to December 2016. See the competition's Data tab for more information on what columns are included in each DataFrame.

In [ ]:
```
(market_train_df, news_train_df) = env.get_training_data()
```

In [ ]:
```
market_train_df.head()
```

In [ ]:
```
market_train_df.tail()
```

In [ ]:
```
news_train_df.head()
```

In [ ]:
```
news_train_df.tail()
```

`get_prediction_days` function

Generator which loops through each "prediction day" (trading day) and provides all market and news observations which occurred since the last data you've received. Once you call **`predict`** to make your future predictions, you can continue on to the next prediction day.

Yields:

- While there are more prediction day(s) and `predict` was called successfully since the last yield, yields a tuple of:
  - `market_observations_df` : DataFrame with market observations for the next prediction day.
  - `news_observations_df` : DataFrame with news observations for the next prediction day.
  - `predictions_template_df` : DataFrame with `assetCode` and `confidenceValue` columns, prefilled with `confidenceValue = 0` , to be filled in and passed back to the `predict` function.
- If `predict` has not been called since the last yield, yields `None` .

In [ ]:
```
# You can only iterate through a result from `get_prediction_days()` once
# so be careful not to lose it once you start iterating.
days = env.get_prediction_days()
```

In [ ]:
```
(market_obs_df, news_obs_df, predictions_template_df) = next(days)
```

In [ ]:
```
market_obs_df.head()
```

In [ ]:
```
news_obs_df.head()
```

In [ ]:
```
predictions_template_df.head()
```

Note that we'll get an error if we try to continue on to the next prediction day without making our predictions for the current day.

In [ ]:
```
next(days)
```

`predict` function

Stores your predictions for the current prediction day. Expects the same format as you saw in `predictions_template_df` returned from `get_prediction_days` .

Args:

- `predictions_df` : DataFrame which must have the following columns:
  - `assetCode` : The market asset.
  - `confidenceValue` : Your confidence whether the asset will increase or decrease in 10 trading days. All values must be in the range `[-1.0, 1.0]` .

The `predictions_df` you send **must** contain the exact set of rows which were given to you in the `predictions_template_df` returned from `get_prediction_days` . The `predict` function does not validate this, but if you are missing any `assetCode` s or add any extraneous `assetCode` s, then your submission will fail.

Let's make random predictions for the first day:

In [ ]:
```
import numpy as np
def make_random_predictions(predictions_df):
    predictions_df.confidenceValue = 2.0 * np.random.rand(len(predictions_df)) - 1.0
```

In [ ]:
```
make_random_predictions(predictions_template_df)
env.predict(predictions_template_df)
```

Now we can continue on to the next prediction day and make another round of random predictions for it:

In [ ]:
```
(market_obs_df, news_obs_df, predictions_template_df) = next(days)
```

In [ ]:
```
market_obs_df.head()
```

In [ ]:
```
news_obs_df.head()
```

In [ ]:
```
predictions_template_df.head()
```

In [ ]:
```
make_random_predictions(predictions_template_df)
env.predict(predictions_template_df)
```

Main Loop

Let's loop through all the days and make our random predictions. The `days` generator (returned from `get_prediction_days`) will simply stop returning values once you've reached the end.

In [ ]:
```
for (market_obs_df, news_obs_df, predictions_template_df) in days:
    make_random_predictions(predictions_template_df)
    env.predict(predictions_template_df)
print('Done!')
```

`write_submission_file` function

Writes your predictions to a CSV file (`submission.csv`) in the current working directory.

In [ ]:
```
env.write_submission_file()
```

In [ ]:
```
# We've got a submission file!
import os
print([filename for filename in os.listdir('.') if '.csv' in filename])
```

As indicated by the helper message, calling `write_submission_file` on its own does **not** make a submission to the competition. It merely tells the module to write the `submission.csv` file as part of the Kernel's output. To make a submission to the competition, you'll have to **Commit** your Kernel and find the generated `submission.csv` file in that Kernel Version's Output tab (note this is *outside* of the Kernel Editor), then click "Submit to Competition". When we re-run your Kernel during Stage Two, we will run the Kernel Version (generated when you hit "Commit") linked to your chosen Submission.

Restart the Kernel to run your code again

In order to combat cheating, you are only allowed to call `make_env` or iterate through `get_prediction_days` once per Kernel run. However, while you're iterating on your model it's reasonable to try something out, change the model a bit, and try it again. Unfortunately, if you try to simply re-run the code, or even refresh the browser page, you'll still be running on the same Kernel execution session you had been running before, and the `twosigmanews` module will still throw errors. To get around this, you need to explicitly restart your Kernel execution session, which you can do by pressing the Restart button in the Kernel Editor's bottom Console tab: