



H₂O.ai

H2O Wave Training

H2O.ai

Contents

- Getting Started
- Developing a Batch Scoring App
 - Components
 - Batch Scoring App
- おまけ

Wave Document: <https://wave.h2o.ai/>

Getting Started

Waveの実行に必要な3つの要素

- Content Server running **Wave SDK**
 - The waved server is a ~10MB static binary executable that runs anywhere
 - Stores site content, transmits content changes to browsers, transmits browser events to apps

Download: <https://github.com/h2oai/wave/releases>

- Language Driver
 - The **h2o_wave python package** used by wave scripts and interactive apps
 - Allows developer to manage content on waved, completely in python

Download(pipインストール): <https://pypi.org/project/h2o-wave/>

- Browser Based Client
 - The user interface and components
 - Renders content to users and transmits events back to waved

環境構築

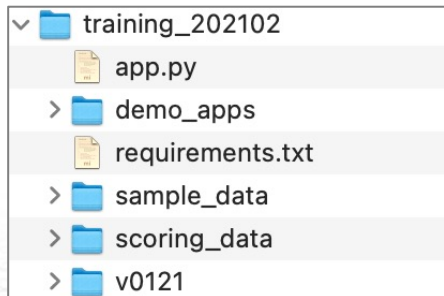
1. Wave SDKのダウンロードとフォルダの解凍

- <https://github.com/h2oai/wave/releases>

2. Python環境の構築

1. トレーニング用フォルダを作成し、Python3仮想環境を作成 `$ python3 -m venv your_env`
2. 作成した仮想環境にログイン `$ source your_env/bin/activate`
3. “requirements.txt”記載のPythonパッケージをインストール `(your_env) $ pip install -r requirements.txt`

トレーニング用フォルダ(training_202102)を作成
Python仮想環境(v0121)を作成



requirements.txt

```
h2o_wave==0.12.1
driverlessai==1.9.1
pandas
numpy
```

注意

- Wave SDKとh2o_waveのバージョンを揃える
- Driverless AIのバージョンにdriverlessaiのバージョンを揃える

Running App, 'Hello World'

1. Wave Server(Web Server)の起動

1. ダウンロードし解凍したWave SDKフォルダのwavedファイルを実行 (your_env) \$./waved

2. Appの実行

1. トレーニング用フォルダのdemo_appsフォルダ内のdemo_hello_app.pyを実行 (your_env) \$ wave run demo_hello_app.py

demo_hello_app.py

```
from h2o_wave import Q, main, app, ui

@app('/hello_app')
async def serve(q: Q):

    q.page['card_hello'] = ui.markdown_card(
        box = '1 1 2 3',
        title = 'Hello World!',
        content = '初めてのH2O Wave (wave app)',
    )

    await q.page.save()
```

ブラウザからアクセス

← → ↻ ⓘ localhost:10101/hello_app

Hello World!

初めてのH2O Wave (wave app)

Wave App or Wave Script?

インタラクティブ(ブラウザへのアクション)なアプリケーションの開発の場合はWave Appを用いる

demo_hello_app.py

```
from h2o_wave import Q, main, app, ui

@app('/hello_app')
async def serve(q: Q):

    q.page['card_hello'] = ui.markdown_card(
        box = '1 1 2 3',
        title = 'Hello World!',
        content = '初めてのH2O Wave (wave app)',
    )

    await q.page.save()
```

Wave Appの実行:

```
(your_env) $ wave run demo_hello_app.py
```

インタラクティブ無し(ブラウザからのアクションを反映させる必要がない)のダッシュボード開発の場合はWave Scriptを用いる

demo_hello_script.py

```
from h2o_wave import site, ui

page = site['/hello_script']

page['card_hello'] = ui.markdown_card(
    box = '1 1 2 3',
    title = 'Hello World!',
    content = '初めてのH2O Wave (wave script)',
)

page.save()
```

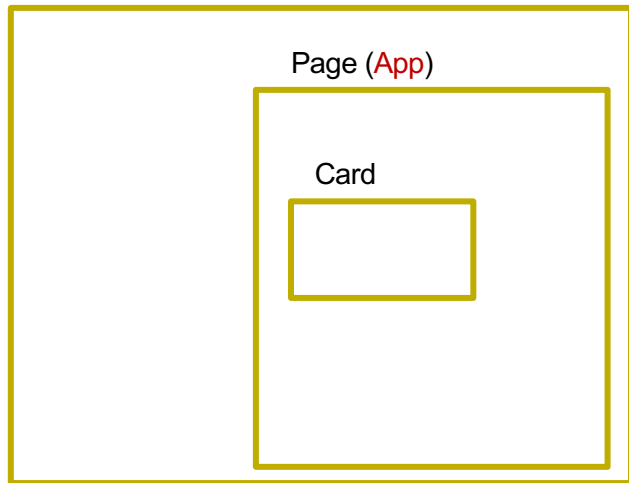
Wave Scriptの実行:

```
(your_env) $ python demo_hello_script.py
```

- ✓ 上記コードは同じ結果が得られる
- ✓ インタラクティブ無しのアプリケーションでも、Wave Appで開発可

Site/Pages/Cards

Site (Wave Server)



- Siteは複数のPageを持つことができる
- Pageはそれぞれディレクトリを持つ
- Cardがダッシュボード等のコンテンツとなる
- Pageの中に複数のCardを記述できる

appデコレータ(@app)

- Page(App)のディレクトリ('/hello_app')
- ブラウザからのアクションがあればappデコレータ下の関数(serve)が毎回実行される

```
from h2o_wave import Q, main, app, ui

@app('/hello_app')
async def serve(q: Q):

    q.page['card_hello'] = ui.markdown_card(
        box = '1 1 2 3',
        title = 'Hello World!',
        content = '初めてのH2O Wave (wave app)',
    )

    await q.page.save()
```

Card

- 複数配置できる

“Site”

AKA the **Wave** server

How it works

- The Wave server stores and manages content
 - Content is stored in a page cache, called a site
- The Wave server runs at a specific address – this is also the site
 - localhost:10101
 - yourpublicaddress:10101
- Multiple apps can be running at a single site, each of these is a page

How to use it

- Reference to browsers interacting with the server is handled with Q
- Q is the query object that comes from the browser and lets the developer know how the browser wants to interact with the content on the site

“Pages”

Wave Serve上の独立したアプリケーション

How it works

- Developers add content to specific pages
 - Static content
 - Content that updates in real time
 - Content that is an interactive application
- Use a specific address to reach each page, or **app**
 - localhost:10101/
 - localhost:10101/test
localhost:10101/myapp

How you use it

- Use the `@app('/page')` function to tell your code which page to update:
 - Indicates that the next function is where browser requests will be directed to
 - Use to route browsers to a specific Wave page
- Reference a page, make changes, and save
 - No explicit creation step is needed

[Learn More!](#)

“Cards”

アプリケーション上のコンテンツ

How it works

- Choose the type of card you want
 - 100s of cards to choose from for your exact needs
- Reference the page you want to add the card to, the name of the card, and the type of the card
- Make any content changes to what should be in the card
- Use `page['card_name']` to create and reference a new card

Examples

- Forms
 - This is the most used card in Wave apps
 - Used to show textboxes, dropdowns, and so on and get input from a user
- Content
 - Markdown, Markup (HTML), Image
- Control
 - Header, Navigation Panel, Tabs
- Graphics
 - Plots, graphs

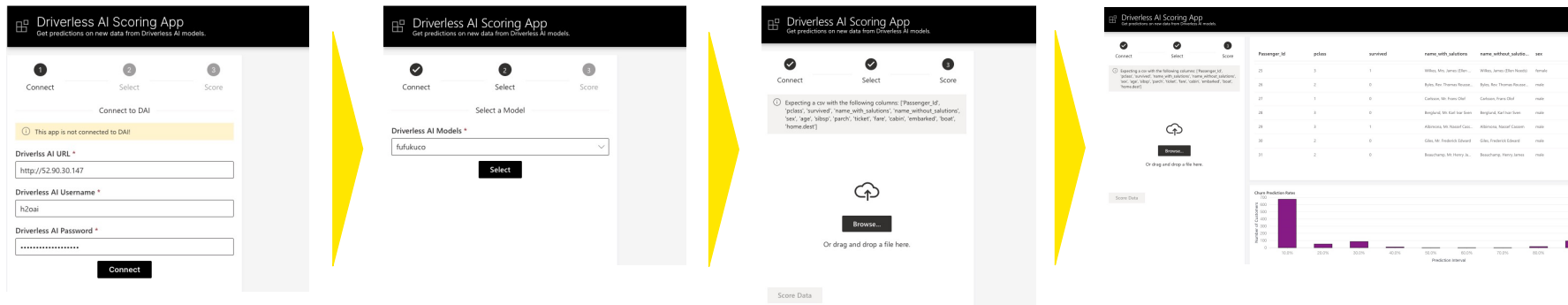
[Learn More!](#)

Developing a Batch Scoring App

App We Will Build

H2O.ai

開発する「Batch Scoring App」に関して



① Driverless AIへの接続

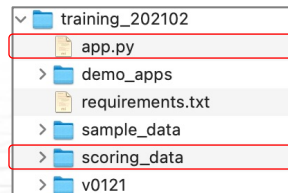
② モデル(学習済み)の選択

③ スコアリング用データのアップロード

④ スコアリング結果の確認、ダウンロード

必要機能

- データのアップロードとダウンロード
- Driverless AIへの接続とスコアリングの実施
- データやスコアリング結果の表示



アプリのプログラム

アップロード/スコアリング結果データのApp上保存先

Components

必要機能を個別に学習

レイアウトとアクション

コード: demo_layout.py

学習内容

- PageにCardsを配置し、カード内に色々なアイテムを配置する
- テキストボックスやボタンなどの動き(ブラウザからのアクション)を理解する

レイアウトとアクションの学習

これはデモAppです。

こんにちは。(text_xl)

こんにちは。(text_l)

こんにちは。(text_m)

こんにちは。(text_s)

こんにちは。(text_xs)

Text Box (必須) です。 *

なんか入力

Text Boxです。

なんか入力

Spin Boxです。

0

Button1です。

Button2です。

(ボタンを押すと入力情報などがWaveサーバに送られます。)

ここに分析結果とかいろいろ表示すると良いと思います。

以下、アクション情報

q.args.textbox1には「ABC」が入っています。

q.args.textbox2には「あいうえお」が入っています。

q.args.spinbox1には「0.5」が入っています。

q.args.button1には「True」が入っています。(Trueだとこのボタンが押されたと言うこと)

q.args.button2には「False」が入っています。(Trueだとこのボタンが押されたと言うこと)

“Query Arguments”

アクション結果（テキストの入力、ボタンのクリック）を保持

- Information based on how a user interacts with a page
 - Typing text
 - Clicking buttons
 - Making choices
- The information is available to the app
 - Did the user fill out this textbox and if so what did they write
- Query Arguments are stored in `q.args`
- The serve function often holds a series of if statements checking if various user components have been interacted with
 - If this button was clicked do this
 - Else if this other button was clicked do this
 - If this menu bar was selected do this

[Learn More!](#)

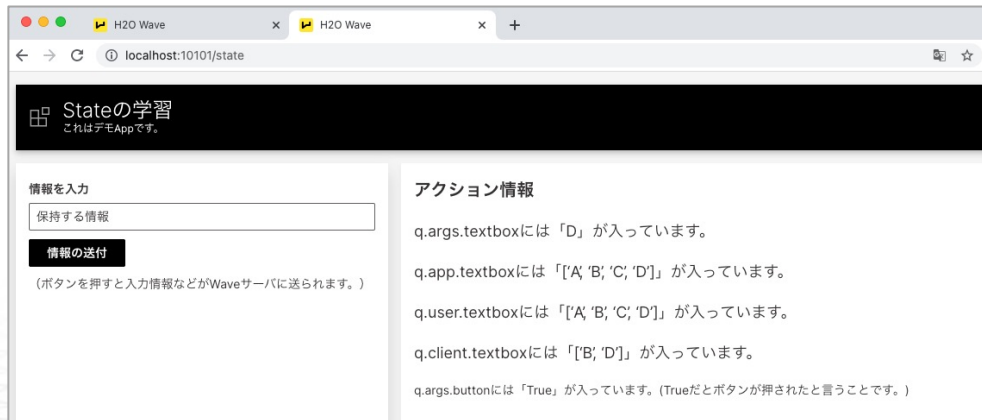
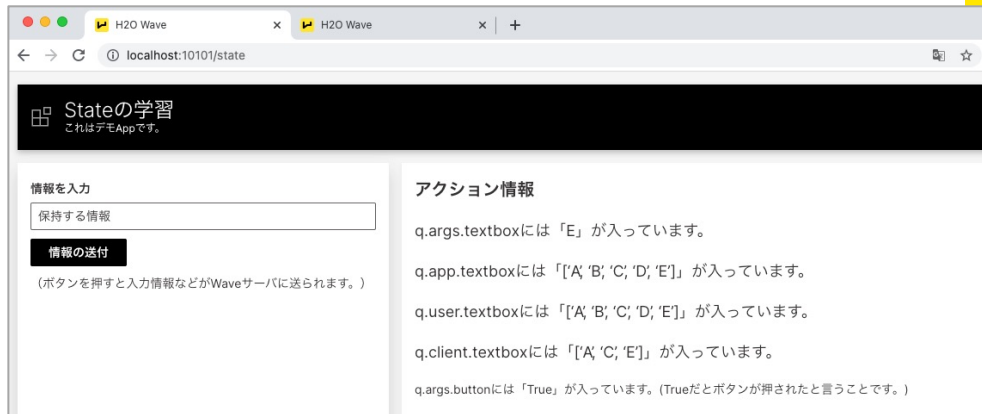
情報の保持

コード: demo_state.py

学習内容

- App実行中における、情報を保持する方法の理解

- 同一Appに別タブからアクセス
- q.argsはアクション毎に情報がリセットされる



State

ランタイムにおける情報のメモリへの保持

- **App-level:** 全てのユーザーへの情報共有
 - **User-level:** ユーザー単位の情報共有
 - **Client-level:** ブラウザ内タブ単位の情報共有
- App-level: `q.app`
 - Shared models
 - Item inventory
 - Stock images
 - User-level: `q.user`
 - Items in a shopping cart
 - Log in credentials
 - Client-level: `q.client`
 - Current displayed content

Routing

コード: demo_routing1.py

学習内容

- ルーティング(ブラウザからのアクションにしたがって表示を切り替える)の理解

```
async def serve(q: Q):  
  
    if q.args.button_A: # button_Aが押された場合実行。(q.args.button_A==True)  
        print('-- button_Aが押された --')  
        await do_button_A(q)  
  
    elif q.args.button_B: # button_Bが押された場合実行。(q.args.button_B==True)  
        print('-- button_Bが押された --')  
        await do_button_B(q)  
  
    else: # 初期画面。button_Aもbutton_Bも押されていない。  
        print('-- 初期画面です --')  
        await initial_display(q)  
  
    await q.page.save()
```



Routing 2

コード: demo_routing2.py

学習内容

- on, handle_onを用いたルーティング
- State(q.client)の理解

```
from h2o_wave import Q, main, app, ui, on, handle_on

@on('#heads')
async def on_heads(q: Q):
    q.page['sides'].items = [ui.message_bar(text='Heads!')]

@on('#tails')
async def on_tails(q: Q):
    q.page['sides'].items = [ui.message_bar(text='Tails!')]

async def setup_page(q: Q):
    q.page['sides'] = ui.form_card(
        box='1 1 4 4',
        items=[
            ui.button(name='#heads', label='Heads'),
            ui.button(name='#tails', label='Tails'),
        ],
    )

@app('/toss')
async def serve(q: Q):
    if not await handle_on(q):
        await setup_page(q)

    await q.page.save()
```

Document (Routing): <https://h2oai.github.io/wave/docs/routing/>

ファイルのアップロード/ダウンロード

コード: demo_fileup.py

学習内容

- ファイルのアップロードとダウンロード
- AppとWave Server間のデータのやりとり
- App内でのデータ(pandas.DataFrame)の表示
- App内でのデータ加工

ファイルをアップしましょう。



Or drag and drop files here.

Upload!

ファイルがサーバにアップされました。

アップされたWaveサーバ上のファイル: L3f647861c2-81af-437d-8533-2f156e595cc4/iris.csv

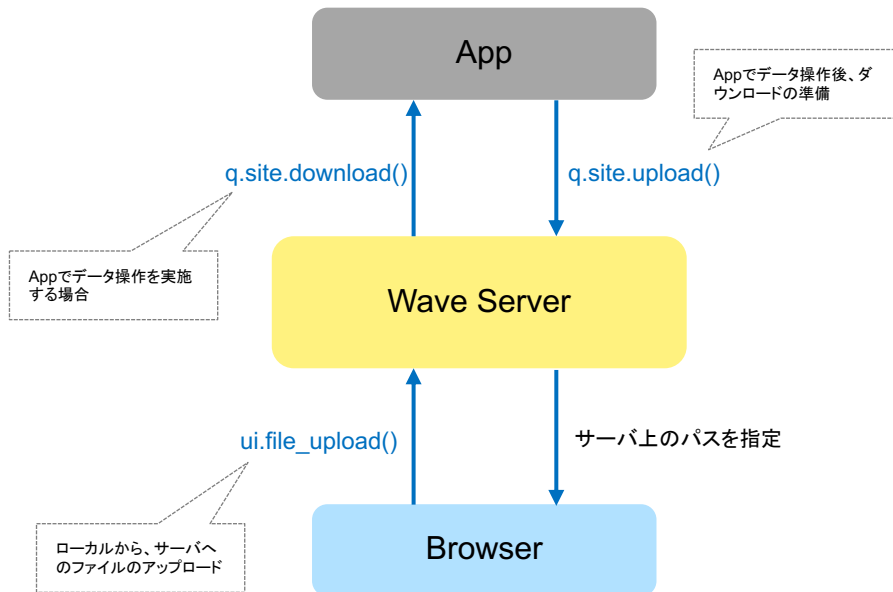
App実行ローカル上にDLされたファイル: /Users/YShimada-MBP16/Documents/Product/Wave/training_202102/demo_apps/iris.csv

サイズ(bytes): 3716

データシェープ: 150行、5列

(先頭10行の表示)

Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa



Driverless AI Client

H₂O.ai

Github: https://github.com/yukismd/Driverless_AI_ClientAccess

学習内容

- クライアント環境からDriverless AIへの接続
- スコアリングの実施

スコアリング実施例: https://github.com/yukismd/Driverless_AI_ClientAccess/blob/main/IID_Table/handling_experiment.ipynb

Python Clientからのスコアリングの実施

- Experimentの操作（実行済みExperimentの確認）
- 予測の実施（スコアリングデータのアップロードと、結果のダウンロード）

```
In [120]: import os
import driverlessai
import pandas
```

```
In [2]: # Driverless AIのuser nameとpasswordの読み込み
import json
with open('idpass.json') as f:
    idpass = json.load(f)
```

```
In [4]: # Driverless AIサーバーへの接続
dai = driverlessai.Client(address='http://54.157.227.21:12345', username=idpass['id'], password=idpass['pass'])
dai
```

```
Out[4]: <class 'driverlessai._core.Client'> http://54.157.227.21:12345
```

```
In [10]: # 接続先Driverless AIのDatasets
dai.datasets.list()
```

```
Out[10]: [<class 'Dataset'> 07c28a00-77f1-11eb-ae7b-0242ac110002 sample_unequal_interval_scoring1.csv,
<class 'Dataset'> fd62eeb2-77ee-11eb-ae7b-0242ac110002 sample_unequal_interval.csv,
<class 'Dataset'> 6e0cd35e-7677-11eb-b908-0242ac110002 dataset_temp.csv,
<class 'Dataset'> 6f800ef8-7664-11eb-b908-0242ac110002 walmart_scoring.csv,
<class 'Dataset'> 5eb74434-73d4-11eb-ab9b-0242ac110002 walmart_ts_6_fcst_grp_train.csv,
<class 'Dataset'> 5eb6cfc2-73d4-11eb-ab9b-0242ac110002 walmart_ts_6_fcst_grp_test.csv,
<class 'Dataset'> 0325cee0-678d-11eb-930d-0242ac110002 Covid_Chest.zip,
```

Batch Scoring App

① Driverless AIへの接続

The screenshot shows the 'Driverless AI Scoring App' interface. At the top, it says 'Get predictions on new data from Driverless AI models.' Below this is a progress bar with three steps: 1. Connect (active), 2. Select, and 3. Score. Under the 'Connect' step, there is a 'Connect to DAI' button. Below the button is a yellow warning box with an information icon and the text 'This app is not connected to DAI!'. Underneath the warning box are three input fields: 'Driverless AI URL *' with the value 'http://52.90.30.147', 'Driverless AI Username *' with the value 'h2oai', and 'Driverless AI Password *' with masked characters. At the bottom right of the form is a black 'Connect' button.

実行される機能(関数)

- serve
- initialize_app_for_new_client
- render_sidebar_content
- get_dai_configure_items

ボタンクリック後に返されるアクション情報(q.args)

- dai_url:'http://52.90.30.147'
- dai_username:'h2oai'
- dai_password:'i-0f12db80917c39394'
- dai_connect_button:True

② モデル(学習済み)の選択

The screenshot shows the 'Driverless AI Scoring App' interface. At the top, it says 'Get predictions on new data from Driverless AI models.' Below this, there are three steps: 'Connect' (marked with a checkmark), 'Select' (marked with a '2'), and 'Score' (marked with a '3'). Under the 'Select' step, there is a 'Select a Model' label and a dropdown menu labeled 'Driverless AI Models *'. The dropdown menu currently shows 'fufukuco'. Below the dropdown menu is a black button labeled 'Select'.

実行される機能(関数)

- serve
- handle_dai_connection
- create_dai_connection
- render_sidebar_content
- get_model_selection_items
- create_dai_connection

ボタンクリック後に返されるアクション情報(q.args)

- experiment_dropdown:'0559f2ec-678b-11eb-930d-0242ac110002'
- select_model_button:True

③ スコアリング用データのアップロード

The screenshot shows the 'Driverless AI Scoring App' interface. At the top, it says 'Get predictions on new data from Driverless AI models.' Below this, there are three steps: 'Connect' (checked), 'Select' (checked), and 'Score' (active, indicated by a '3' in a circle). A message box states: 'Expecting a csv with the following columns: ['Passenger_Id', 'pclass', 'survived', 'name_with_salutations', 'name_without_salutations', 'sex', 'age', 'sibsp', 'parch', 'ticket', 'fare', 'cabin', 'embarked', 'boat', 'home.dest']'. Below this, there is a cloud upload icon and a 'Browse...' button. At the bottom, it says 'Or drag and drop a file here.' and 'Score Data' is visible in the bottom left corner.

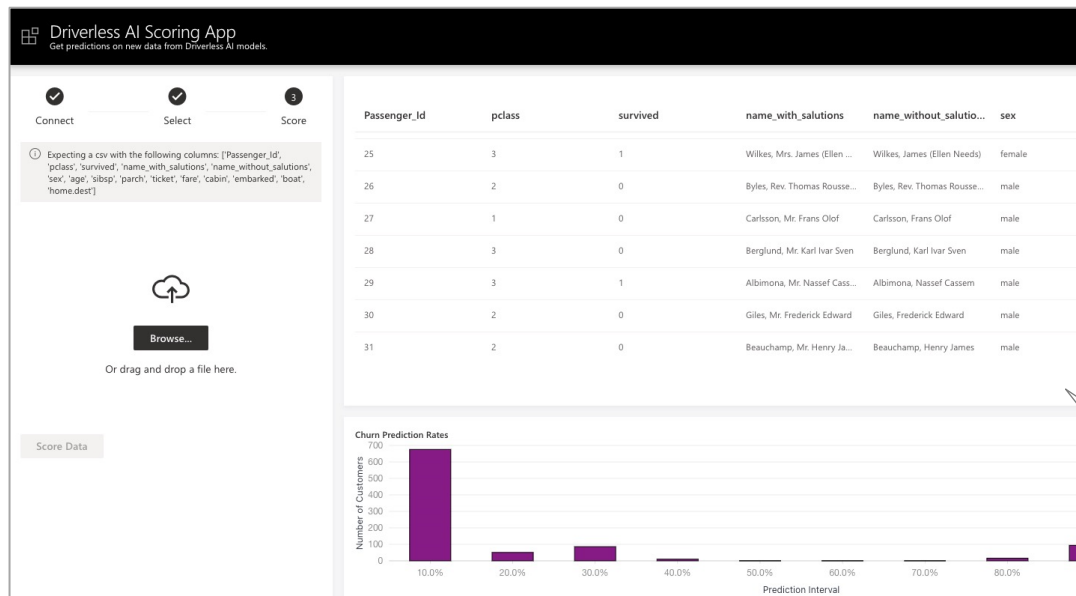
実行される機能(関数)

- serve
- handle_model_selection
- render_sidebar_content
- get_batch_score_items
- create_dai_connection

ボタンクリック後に返されるアクション情報(q.args)

- file_upload:['/_f/441bac5c-d925-4948-bfa0-35e51a23e334/TitanicData.csv']

④ スコアリング結果の確認、ダウンロード



実行される機能(関数)

- `serve`
- `handle_batch_scoring`
- `render_center_content`
- `get_dai_new_predictions`
- `create_dai_connection`

The screenshot shows a table with three columns: 'binary', 'ternary', and 'score'. The data rows are:

binary	ternary	score
ertrud Emilia	female	1.0
pl. Archibald IV	male	53.0

Below the table is a 'Download data' button with a download icon.

関数 – serve

WHAT HAPPENS WHEN USERS INTERACT WITH OUR APP

- Waiting for users at the root URL
 - localhost:10101/
 - myurl:10101/
- This function runs each time any user interacts with your app
 - Setup the browser tab if this is a new user
 - Handle any user events like button clicks or file uploads
 - Save new content to the server

```
@app('/')
async def serve(q: Q):

    if not q.client.initialized:
        await initialize_app_for_new_client(q)
    else:
        await handle_on(q)
    await q.page.save()
```

関数 – initialize_app_for_new_client

INFORMATION NEEDED THE FIRST TIME A CLIENT COMES TO THE APP

- Record that this tab has not completed any of the configuration steps
 - Each browser tab visiting your app can see different content!
 - Use `q.client.*` to create variables for saving information about this tab you want to reference throughout the app
- While developing, we can hard-code our DAI credentials for an easier experience
- Call two functions which render UI elements to the tab
- Record that this browser tab is setup

```
async def initialize_app_for_new_client(q: Q):
```

```
    # Save which activities each tab has successfully accomplished
```

```
    q.client.dai_connection = False
```

```
    q.client.model_selected = False
```

```
    q.client.data_uploaded = False
```

```
    # TODO: delete
```

```
    q.client.dai_url = 'http://54.84.100.125:12345'
```

```
    q.client.dai_username = 'h2oai'
```

```
    q.client.dai_password = 'i-0d7fabc9ac7b049ab'
```

```
    render_main(q)
```

```
    render_sidebar(q)
```

```
    q.client.initialized = True
```

関数 – render_main

UI ELEMENTS WE WANT USERS TO ALWAYS SEE

- Create a meta card
 - Information used in the app that is not an explicit card
 - Browser tab title
 - Color theme
 - Google Analytics
- Create a header card
 - Located at the top of the app, for the whole width of the screen

```
def render_main(q: Q):  
    q.page['meta'] = ui.meta_card(  
        box="",  
        title='DAI Scoring App',  
        theme='light'  
    )  
  
    q.page['header'] = ui.header_card(  
        box='1 1 11 1',  
        title='Driverless AI Scoring App',  
        subtitle='Get predictions on new data from Driverless AI  
models.'  
    )
```

関数 – render_sidebar

UI ELEMENT WE WANT USERS TO ALWAYS SEE, WITH DIFFERENT CONTENT

- Create a list of UI elements to show based on which steps this user has completed so far
 - Do they have a valid connection to Driverless AI?
 - Have they selected a model to batch score on?
 - Have they uploaded a dataset?
- Show a 1-2-3 Stepper with their progress in the app and the appropriate elements

```
def render_sidebar(q: Q):  
  
    if not q.client.dai_connection:  
        sidebar_items = ...  
    elif not q.client.model_selected:  
        sidebar_items = ...  
    elif not q.client.data_uploaded:  
        sidebar_items = ...  
    else:  
        sidebar_items = ...  
  
    q.page['sidebar'] = ui.form_card(  
        box='1 2 3 8',  
        items=[  
            ui.stepper(name='scoring_config_stepper', items=[  
                ui.step(label='Connect', done=q.client.dai_connection),  
                ui.step(label='Select', done=q.client.model_selected),  
                ui.step(label='Score', done=q.client.data_uploaded),  
                ui.step(label='Complete', done=False)  
            ])  
        ] + sidebar_items  
    )
```

関数 – handle_dai_credentials

WHAT HAPPENS WHEN A USER CLICKS THE CONNECT BUTTON

- If a button called dai_connect_button is clicked, this function is called
- Save the credentials to be used later in the app
- Call a function which attempts to connect to DAI and returns an error message if it fails
- Save information to use elsewhere
 - Connection error message
 - If connection is successful
 - List of available experiments
- Update the sidebar UI

```
@on('dai_connect_button')
async def handle_dai_credentials(q: Q):

    q.client.dai_url = q.args.dai_url
    q.client.dai_username = q.args.dai_username
    q.client.dai_password = q.args.dai_password

    dai, q.client.error = create_dai_connection(q)

    if q.client.error is None:
        q.client.dai_connection = True
        q.client.experiment_list = dai.experiments.list()

    render_sidebar(q)
```


関数 – handle_model_selection

WHAT HAPPENS WHEN A USER CLICKS THE CONNECT BUTTON

- If a button called `select_model_button` is clicked, this function is called
- Save these experiment id that was selected
 - Save that this tab has selected a model
- Call a function which attempts to connect to DAI and returns an error message if it fails
 - Get a list of expected columns so the user knows what their dataset should look like
- Update the sidebar UI

```
@on('select_model_button')
async def handle_model_selection(q: Q):

    q.client.experiment_key = q.args.experiment_dropdown
    q.client.model_selected = True

    dai, error = create_dai_connection(q)

    q.client.expected_columns =
        dai.experiments.get(q.client.experiment_key)
        .datasets['train_dataset'].columns

    render_sidebar(q)
```

関数 – handle_batch_scoring

WHAT HAPPENS WHEN A USER CLICKS THE CONNECT BUTTON

- If a button called `file_upload` is clicked, this function is called
- Use the `q.site.download` function to download the user's dataset
 - Save that this tab has uploaded data
- Call a function which gets predictions using the DAI python client
 - For binary classification problems, aggregate the data to show predictions
- Update the sidebar and center content UI

```
@on('file_upload')
async def handle_batch_scoring(q: Q):

    q.client.batch_data_path = await
q.site.download(url=q.args.file_upload[0], path='./data')
q.client.data_uploaded = True

q.client.scored_df = get_dai_new_predictions(q)

grouped_predictions = ...
q.client.distribution_data = data(
    fields=grouped_predictions.columns.tolist(),
    rows=grouped_predictions.values.tolist(),
    pack=True,
)

render_sidebar(q)
render_center(q)
```

関数 – render_center

THE CENTER TABLE CONTENT OF DATA AND PREDICTIONS

- The Wave Table widget allows us to create a table object which users can download the data
 - We take the pandas dataframe and create wave table columns and rows

```
def render_center(q: Q):
    df = q.client.df

    table = ui.table(
        name='my_table',
        columns=[ui.table_column(name=str(x),
                                   label=str(x)) for x in df.columns.values],
        rows=[ui.table_row(name=str(i),
                             cells=[str(df[col].values[i])
                                     for col in df.columns.values])
              for i in range(df.shape[0])],
        downloadable=True,
        height='400px'
    )

    q.page['predictions'] = ui.form_card(
        box='4 2 8 5',
        items=[table]
    )
```

おまけ

おまけ1

Rest APIによるスコアリングのサンプル

コード: demo_scoring.py

スコアリングApp

Driverless AIサーバ上にデPLOYしたモデルをスコアリング

API Point: <http://54.172.0.191:9090/model/score>

X1 [-10.0,10.0]

X2 [-10.0,10.0]

X3 [-10.0,10.0]

X4 [-10.0,10.0]

Scoring



Input: X1=0.4, X2=7, X3=-1.8, X4=0

Scoring Result: 3.996020476023356

Back

学習用データ: sample_data/sample_simple.csv

おまけ2

Rest APIによるスコアリングのサンプル

コード: demo_scoring.py

バッチスコアリングApp

Driverless AIサーバ上にデプロイしたモデルをスコアリング

API Point: <http://54.172.0.191:9090/model/score>

Chosen File



scoring_data.csv

Upload file

スコアリング用データ

サイズ(bytes) : 96

データシェーブ : 5行、4列

x1	x2	x3	x4
2.6	-3.21	0.62	1.36
0.0	10.0	0.3	1.0
-1.11	-1.11	2.22	2.22
0.0	0.0	0.0	0.0
-9.7	-0.4	6.98	2.1

Scoring this data

Back to file upload

スコアリング結果

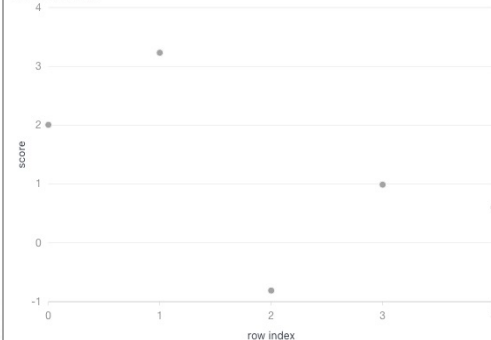
Scoring Result:

```
[["2.006951332092285"], ["3.232259194056193"],  
["-0.8104969635605812"], ["0.9894463221232096"], ["0.6018870572249094"]]
```

結果のダウンロード

Back to file upload

スコアリング結果



学習用データ: sample_data/sample_simple.csv

スコアリング用データ: sample_data/scoring_data.csv

Thank You