Darwin™ API User Manual

A SparkCognition™ Education Document

v 1.0 02.02.2018

Contents

This document contains the following sections:

About this guide

This manual describes the Darwin™ API and its use in automated model building. It is intended for data scientists, software engineers and analysts who want to use the Darwin API to interact with Darwin to create and train models, monitor jobs and perform analysis.

Darwin overview

Darwin is a SparkCognition® tool that automates model building processes to solve specific problems. This tool enhances data scientist potential because it automates various tasks that are often manually performed. These tasks include data cleaning, latent relationship extraction, and optimal model determination. Darwin promotes rapid and accurate feature generation through both automated windowing and risk generation. Darwin quickly creates highly-accurate, dynamic models using both supervised and unsupervised learning methods.
For additional information on Darwin, contact your local SparkCognition partner for access to see the white paper titled: *Darwin - A Neurogenesis Platform*.

Accessing the API

The Darwin API can normally be accessed through one of three methods:

- the Darwin Python SDK (preferred, recommended)
- through the `https://darwin-api.sparkcognition.com/v1` end point
- optionally, through user created `curl` commands

For additional information on the Darwin SDK, see the *Sparkcognition Darwin Python SDK Guide*.

Expectation

This document assumes experience of the data scientist or software engineer that is commensurate with data science techniques and associated programming tasks.

Technical routes

The Darwin API includes the following api operations:

- `analyze` - analyze a model or dataset
- `auth` - register and authenticate
- `download` - download or delete a generated artifact
- `lookup` - get model or dataset metadata
- `risk` - create risk information for a dataset
- `status` - return status on jobs
- `run` - run a model on a dataset
- `train` - train a model
- `upload` - upload or delete a dataset
  .

analyze

Request Type: POST

URI: /v1/analyze/model/{model_name}

Headers:

- Authorization: Bearer token

Form Data:

- *job_name*: The name of the job
- *artifact_name*: The name of the artifact

Description: Analyze a model.
Note: This API is capable of returning the structure of the model in the form of a png file.

Response Codes: 201, 400, 401, 403, 422

Successful Response:

```
{
  "job_name": "string",
  "artifact_name": "string"
}
```

Request Type: POST

URI: /v1/analyze/data/{dataset_name}

Headers:

- Authorization: Bearer token

Description: Analyze a dataset and return statistics/metadata concerning designated data.

Parameter Descriptions:

- *target*: String denoting target prediction column in input data.
- *job_name*: The job name
- *artifact_name*: The artifact name.
- *impute*: String alias that indicates how to fill in missing values in input data. Descriptions in following table.

ALIAS	DESCRIPTION	COMPLEXITY
'genetic'	Genetic Fill: Automatically determines the most appropriate fast imputation method using evolutionary methods.	Linear Fast
'ffill'	(**Default**) Forward Fill: Propagate values forward from one example into the missing cell of the next example. Can be useful for timeseries data, but also applicable for both numerical and categorical data.	Linear Fast
'bfill'	Backward Fill: Propagate values backward from one example into the missing cell of the previous example. Can be useful for timeseries data, but also applicable for both numerical and categorical data.	Linear Fast
'mean'	Mean Fill: Computes the mean value of all non-missing examples in a column to fill in missing examples. The result might not not be interpretable in terms of the input space for categorical variables.	Linear Fast
'median'	Median Fill: Computes the median value of all non-missing examples in a column to fill in missing examples. **Note**: Although the result is interpretable in terms of the input space for categorical variables, the approach might not not be appropriate for non-ordinal data.	Linear Fast
'mode'	Mode Fill: Uses the most common value on a column-by-column basis to fill in missing examples. The result is interpretable for both numerical and categorical variables.	Linear Fast
'spline'	Spline Fill: Interpolation using a spline (piecewise function). Can be useful for timeseries or sequential data.	Linear Fast
'linear'	Linear Interpolation Fill: Interpolation using a linear function. Can be useful for timeseries or sequential data.	Linear Fast
'knn'	K-Nearest Neighbors Fill: Fills in missing values by averaging the cell values of the k nearest neighbors in the reduced feature space defined by all non-missing columns.	Polynomial Slow
'rmf'	Robust Matrix Factorization Fill: Computes low-rank matrices L (observations x rank), R (features x rank), and E where X is input data, and $X = LR^T + E$.	Polynomial Slow
'mice'	Multiple Imputation by Chained Equations: First imputes missing values using Forward Fill. Then, column-by-column, missing values are reintroduced and regressed upon using the other (non-missing) columns. Continues iteratively.	Polynomial, Iterative Very Slow

- *drop*: Enables automatic pruning of input columns based on different criteria such as amount of missing data, number of unique values, and standard deviation.
  Note: This automatically drops identifier columns (unique value for each sample) and columns that do not contain sufficient data to aid prediction.

- *max_int_uniques*: Threshold for automatic encoding of categorical variables. If a column contains at least *max_int_uniques* unique values, it is treated as categorical and one hot encoded during preprocessing.

- *max_unique_values*: Threshold for automatic pruning of categorical columns prior to one hot encoding based on the number of unique values.
  Note: If a categorical column contains at least *max_unique_values*, it is dropped during preprocessing prior to one hot encoding.

- *feature_eng*: Enables automatic feature generation. Identifies an appropriate time window and augments input with new features derived in the frequency and time domains.
  Notes:

  - Can be applied only to timeseries data.
  - String aliases specify methods for window computation.

Alias	**Description**
None	No feature generation is applied.
‘mi’	Uses mutual information to estimate the window length.
‘auc’	*(Default)* Uses autocorrelation to estimate the window length.
‘user’	User specified window length: see *window_len*.

- *window_len*: User specified window length for feature generation.
  Note: This parameter is used only if *user* is provided for the *feature_eng* parameter.

- *feature_select*: A number in [0,1] that specifies the percentage of numerical features to maintain based on their dependency to the target. Ranks all features using mutual information and drops (1 - feature_select)% of the lowest-ranking features. Default is **1** (keep all features).

- *outlier*: A string alias that indicates that outlier detection be applied during preprocessing.
  Note: Outliers are removed and later filled using imputation.

Alias	**Description**
None	(Default) No outlier detection is applied.
‘mad’	Uses Median Absolute Deviation (mad) to detect outliers.
‘perc’	Uses Percentile-based outlier detection.
‘isol’	Uses Isolation Forest to detect outliers.

  Note: Outliers are removed and later filled using imputation.

Payload:

```
{
  "target": "string",
  "job_name": "string",
  "artifact_name": "string",
  "impute": "mean",
```

```
  "drop": true,
  "max_int_uniques": 15,
  "max_unique_values": 50,
  "feature_eng": "auc",
  “window_len”: 10
  "feature_select": 1,
  "outlier": "mad"
}
```

Response Codes: 201, 400, 401, 403, 422

Successful Response:

```
{
  "job_name": "string",
  "artifact_name": "string"
}
```

---

auth

Request Type: POST

URI: /v1/auth/login

Headers:

- Authorization: Bearer token

Description: Login as a service.

Form Data:

- *api_key*: The api key of the service
- *pass1*: The service level password

Response Codes: 201, 400, 401

Successful Response:

```
{
    'access_token': 'some_string'
}
```

Request Type: POST

URI: /v1/auth/login/user

- Authorization: Bearer token

Description: Login as a user.

Form Data:

- *username*: The end user name
- *pass1*: The end user level password

Response Codes: 201, 400, 401, 422

Successful Response:

```
{
    'access_token': 'some_string'
}
```

Request Type: POST

URI: /v1/auth/register

Headers:

- Authorization: Bearer token

Description: Register as a service.

Form Data:

- *api_key*: The api key of the service
- *pass1*: The service level password
- *pass2*: The service level password confirmation

Response Codes: 201, 400, 401

Successful Response:

```
{
    'access_token': 'some_string'
}
```

Request Type: POST

URI: /v1/auth/register/user

Headers:

- Authorization: Bearer token

Description: Register a user for your service.

Form Data:

- *username*: The end user name
- *pass1*: The end user level password
- *pass2*: The end user level password confirmation

Response Codes: 201, 400, 401, 422

Successful Response:

```
{
    'access_token': 'some_string'
}
```

---

download

Request Type: DELETE

URI: /v1/download/{artifact_name}

Headers:

- Authorization: Bearer token

Description: Delete an artifact.

Response Codes: 204, 401, 404, 422

Successful Response: None

---

lookup

Request Type: GET

URI: /v1/lookup/artifact

Headers:

- Authorization: Bearer token

Query Parameters:

- type: filter on the type of artifact (for example. Model, Dataset, Test, Run, or Risk)

Description: Get artifact metadata

Response Codes: 200, 401, 422

Successful Response:


```
[
  {
    "id": "string",
    "name": "string",
    "type": "string",
    "created_at": "2018-01-22T19:00:39.863Z",
    "mbytes": 0
  }
]
```

Request Type: GET

URI: /v1/lookup/artifact/{artifact_name}

Headers:

- Authorization: Bearer token

Description: Get artifact metadata for a single artifact

Response Codes: 200, 401, 422

Successful Response:


```
{
  "name": "string",
  "type": "string",
  "created_at": "2018-01-22T19:00:39.869Z",
  "mbytes": 0
}
```

Request Type: GET

URI: /v1/lookup/model

Headers:

- Authorization: Bearer token

Description: Get the model metadata for a user. This is useful if a user has forgotten certain model names.

Response Codes: 200, 401, 422

Successful Response:

```
[
    {
        "name": "model1_name",
        "updated_at": "2017-02-03T073000",
        "trained_on": ["dataset1_id", "dataset2_id"],
        "generations": 100,
        "loss": 0.8,
        "parameters": {},
    },
    {
        "name": "model2_name",
        "updated_at": "2017-08-22T175022",
        "trained_on": ["dataset3_id"],
        "loss": 0.82,
        "generations": 80,
        "parameters": {
            "target": "target1"
        },
    }
]
```

Request Type: GET

URI: /v1/lookup/client

Headers:

- Authorization: Bearer token

Description: Get a client's metadata.

Response Codes: 200, 401, 422

Successful Response:

```
{
                "username": "string",
                "tier": 0,
                "model_limit": 0,
                "job_limit": 0,
                "upload_limit": 0,
                "user_limit": 0
                }
```

Request Type: GET

URI: /v1/lookup/model/{model_name}

Headers:

- Authorization: Bearer token

Description: Get all of the model metadata for a particular model.

Response Codes: 200, 401, 404, 422

Successful Response:

```
{
    "updated_at": "2017-02-03T073000",
    "trained_on": ["dataset1_id", "dataset2_id"],
    "generations": 100,
    "loss": 0.8,
    "parameters": {},
}
```

Request Type: GET

URI: /v1/lookup/dataset

Headers:

- Authorization: Bearer token

Description: Get the dataset metadata for a user. This is useful if a user has forgotten certain dataset names.

Response Codes: 200, 401, 422

Successful Response:

```
[
    {
        "name": "dataset1_name",
        "mbytes": 0.2,
        "updated_at": "20170924T000000",
        "categorical": False,
        "sequential": True,
        "imbalanced": True,
    },
    {
        "name": "dataset2_name",
        "mbytes": 3.5,
        "updated_at": "20170902T010101",
        "categorical": True,
        "sequential": False,
        "imbalanced": False,
    }
]
```

Request Type: GET

URI: /v1/lookup/dataset/{dataset_name}

Headers:

- Authorization: Bearer token

Description: Get all of the dataset metadata for a particular dataset.

Response Codes: 200, 401, 404, 422

Successful Response:

```
{
    'mbytes': 0.2,
    'updated_at': '20170924T000000',
    'categorical': False,
    'sequential': True,
    'imbalanced': True,
}
```

Request Type: GET

URI: /v1/lookup/tier

Headers:

- Authorization: Bearer token

Description: Get all of the tier metadata.

Response Codes: 200, 401, 422

Successful Response:

```
[
  {
    "tier": 0,
    "model_limit": 0,
    "job_limit": 0,
    "upload_limit": 0,
    "user_limit": 0
  }
]
```

Request Type: GET

URI: /v1/lookup/tier/{tier_num}

Headers:

- Authorization: Bearer token

Description: Get the metadata for a particular tier.

Response Codes: 200, 401, 404, 422

Successful Response:

```
{
  "tier": 0,
  "model_limit": 0,
  "job_limit": 0,
  "upload_limit": 0,
  "user_limit": 0
}
```


---

risk

Notes concerning risk -
risk is a value used in calculating future events. A risk is calculated using algorithms based on sliding time frames and associated historical data that projects forward in time to predict the likelihood of the event. The outcome of the calculations is that the likelihood of an event occurring within a particular time frame becomes available for use. Note that risk values are dependent on the quality and extent of the historical data as well as the scope of the timeframe used for evaluation.

Request Type: POST

URI: /risk/{failure_data}/{timeseries_data}

Headers:

- Authorization: Bearer token

Description: Create risk information for a dataset.

Parameter Descriptions:

- *job_name*: The job name.
- *artifact_name*: The artifact name.
- *risk_columns*: A list of column names in the index.

- *shutdown_column*: Name of the column in the risk data that denotes the beginning of the predicted event of interest.

- *return_column*: Name of the column in the risk data that denotes the end of the predicted event *and* when all data can again be considered “normal”.

- *asset_column*: Name of the asset column in the risk data. This parameter is used when the datasets consist of multiple different assets.

- *lead_time*: Lead time in seconds. This value is half width of the risk function - this means. the risk index is 0 prior to *2* lead_time* and increases to 1 at a failure time.
  Note: The *lead_time*value must be greater than zero.

- *Functional_form*: Shape of a risk function, includes:

  - `step`: Step function
  - `linear`: Linear function
  - `sigmoid`: Sigmoid function

- `exponential`: Exponential function

Payload:

```
{
  "target": "string",
  "job_name": "string",
  "risk_columns": [
    "risk"
  ],
  "shutdown_column": "Shutdown Time",
  "return_column": "Return Time",
  "asset_column": "Asset",
  "lead_time": 1,
  "functional_form": "linear"
}
```

Response Code: 201, 400, 401, 403, 404, 422

Successful Response:

```
{
    "job_name": "job1_name",
    "artifact_name": "some string",
}
```

---

run

Request Type: POST

URI: /v1/run/model/{model_name}/{dataset_name}

Headers:

- Authorization: Bearer token

Form Data:

- *job_name*: The name of the job.
- *artifact_name*: The name of the artifact.
- *supervised*: A boolean (true/false) indicating whether the model is supervised or not, for example *unsupervised*.

Description: Run a model on a dataset and return the predictions/classifications/clusters found by the model.

Response Codes: 201, 400, 401, 403, 404, 408, 422

Successful Response:

```
{
    "job_name": "job1_name",
    "artifact_name": "artifact1_name"
}
```

---

status

Request Type: GET

URI: /v1/job/status

Headers:

- Authorization: Bearer token

Query Parameters:

- Age: List jobs that are less than X units old (for example, 3 weeks, 2 days)
- Status: List job of a particular status, for example *Running*

Description: Get the status for all jobs.

Response Codes: 200, 400, 401, 422

Successful Response:

```
[
    {
        "job_name": "job1_name",
        "status": "Requested",
                        "starttime": "2018-01-30T13:27:46.449865",
       "endtime": "2018-01-30T13:28:46.449865",
        "percent_complete": 0,
       "job_type": "TrainModel",
        "loss": 0,
        "generations": 0,
        "dataset_names": [
            "phone_data"
        ],
        "artifact_names": [
            "art1"
        ]
        "model_name": null,
    },
    {
        "job_name": "job2_name",
        "status": "Running",
        "starttime": "2018-01-30T13:27:46.449865",
        "endtime": "2018-01-30T13:28:46.449865",
        "percent_complete": 23,
        "job_type": "UpdateModel",
        "loss": 0.92,
        "generations": 50,
        "dataset_names": [
```

```
            "language_data"
        ],
        "artifact_names": null,
        "model_name": "test_model",
    }
]
```

Request Type: GET

URI: /v1/job/status/{job_name}

Headers:

- Authorization: Bearer token

Query Parameters:

- Age: List jobs that are less than X units old (for example, 3 weeks, 2 days)

Description: Get the status for a particular job.

Response Codes: 200, 400, 401, 404, 422

Successful Response:

```
{
    "status": "Requested",
        "starttime": "2018-01-30T13:27:46.449865",
    "endtime": "2018-01-30T13:28:46.449865",
    "percent_complete": 0,
    "job_type": "TrainModel",
    "loss": 0,
    "generations": 0,
    "dataset_names": [
        "language_data"
    ],
    "artifact_names": null,
    "model_name": None,
}
```

---

train

Request Type: POST

URI: /v1/train/model

Headers:

- Authorization: Bearer token

Description: Create a model trained on the dataset identified by dataset_name.

Parameter descriptions:

- *target*: String denoting target prediction column in input data.
- *dataset_names*: A list of dataset names to use for training.
  Note: only 1 is currently supported.
- *job_name*: The job name.
- *model_name*: The string identifier of the model to be trained.
- *max_train_time*: Sets the training time for the model in 'HH:MM' format.
  Note: This overrides any values set for *max_generation*.
- *max_generation*: Expected input/type: *numeric*. Sets the training time for the model in generations. If *max_train_time* is set, this parameter is ignored.
- *recurrent*: Expected input/type: *true/false*. Enables recurrent connections to be evolved in the model. This option can be useful for timeseries or sequential data.
  Note: This option is automatically enabled if a *datetime* column is detected in the input data. This can result in slower model evolution.
- *impute*: String alias that indicates how to fill in missing values in input data.

ALIAS	DESCRIPTION	COMPLEXITY
'genetic'	Genetic Fill: Automatically determines the most appropriate fast imputation method using evolutionary methods.	Linear Fast
'ffill'	**(Default)** Forward Fill: Propagate values forward from one example into the missing cell of the next example. Might be useful for timeseries data, but also applicable for both numerical and categorical data.	Linear Fast
'bfill'	Backward Fill: Propagate values backward from one example into the missing cell of the previous example. Might be useful for timeseries data, but also applicable for both numerical and categorical data.	Linear Fast
'mean'	Mean Fill: Computes the mean value of all non-missing examples in a column to fill in missing examples. The result may or might not be interpretable in terms of the input space for categorical variables.	Linear Fast
'median'	Median Fill: Computes the median value of all non-missing examples in a column to fill in missing examples. While the result is interpretable in terms of the input space for categorical variables, the approach might not be appropriate for non-ordinal data.	Linear Fast
'mode'	Mode Fill: Uses the most common value on a column-by-column basis to fill in missing examples. The result is interpretable for both numerical and categorical variables.	Linear Fast
'spline'	Spline Fill: Interpolation using a spline (piecewise function). Might be useful for timeseries or sequential data.	Linear Fast
'linear'	Linear Interpolation Fill: Interpolation using a Linear function. Might be useful for timeseries or sequential data.	Linear Fast
'knn'	K-Nearest Neighbors Fill: Fills in missing values by averaging the cell values of the k nearest neighbors in the reduced feature space defined by all non-missing columns.	Polynomial Slow

ALIAS	DESCRIPTION	COMPLEXITY
'rmf'	Robust Matrix Factorization Fill: Computes low-rank matrices L (observations x rank), R (features x rank), and E where X is input data, and $X = LR^T + E$.	Polynomial Slow
'mice'	Multiple Imputation by Chained Equations: First imputes missing values using *Forward Fill*. Then, column-by-column, missing values are reintroduced and regressed upon using the other (non-missing) columns. Continues iteratively.	Polynomial, Iterative Very Slow

- *drop*: Expected input/type: *true/false*. Enables automatic pruning of input columns based on different criteria such as amount of missing data, number of unique values, and standard deviation.
  Note:This automatically drops identifier columns (unique value for each sample) and columns that do not contain sufficient data to aid prediction.

- *max_int_uniques*: Expected input/type: *integer*. Threshold for automatic encoding of categorical variables. If a column contains at least *max_int_uniques* unique values, it is treated as categorical and one hot encoded during preprocessing.

- *max_unique_values*: Expected input/type: *integer*. Threshold for automatic pruning of categorical columns prior to one hot encoding based on the number of unique values.
  Note: If a categorical column contains at least *max_unique_values*, it is dropped during preprocessing prior to one hot encoding.

- *feature_eng*: Enables automatic feature generation. Identifies an appropriate time window and augments input with new features derived in the frequency and time domains.
  Note: Can only be applied to timeseries data. String aliases specify methods for window computation.

ALIAS	DESCRIPTION
None	No feature generation will be applied.
'mi'	Uses mutual information to estimate the window length.
'auc'	(**Default**) Uses autocorrelation to estimate the window length.
'user'	User specified window length: see* window_len*.

- *window_len*: Expected input/type: *integer*. User specified window length for feature generation.
  Note: This parameter is used only in the case that *user* is provided for the *feature_eng* parameter.

- *feature_select*: A number in [0,1] specifying the percentage of numerical features to maintain based on their dependency to the target. Ranks all features using mutual information and drops (1 - feature_select)% of the lowest-ranking features. **Default is 1** (keep all features).

- *outlier*: A string alias that indicates the outlier detection to apply during preprocessing.
  Note: Outliers are removed and later filled using imputation.

ALIAS	DESCRIPTION
None	(**Default**) No outlier detection will be applied.
'mad'	Uses Median Absolute Deviation to detect outliers.
'perc'	Uses Percentile-based outlier detection.
'isol'	Uses an Isolation Forest to detect outliers.

- *auto_save_per* (*supervised* only): Expected input/type: *integer*. Sets the checkpoint frequency. The model creation progress is recorded after every *auto_save_per* generations.

Note: If the model is retrained, the model begins from the last recorded checkpoint. The model is automatically saved at the end of evolution.

- *imbalance* (*supervised* only): Expected input/type: *true/false*. Enables automatic imbalance correction that selectively applies *random oversampling*, *random undersampling*, *synthetic minority oversampling* (SMOTE), or *adaptive synthetic sampling* (ADASYN) to the input data depending on problem characteristics.

- *n_clusters*(*unsupervised* only): Specifies the number of clusters to be used if *clustering* is enabled.
  Note: If this value is not provided, the number of clusters will be heuristically determined.

- *anomaly_prior* (*supervised* only): Expected input/type: *between [0,1]. *Significance level at which a point is defined as anomalous.
  Note: This parameter is used only for unsupervised problems if *clustering* is disabled.

Payload:

```
{
  "target": "string",
  "dataset_names": ["dataset_name1"],
  "job_name": "my_job",
  "model_name": "string",
  "max_train_time": "00:01",
  "max_generation": 100,
  "recurrent": true,
  "impute": "mean",
  "drop": true,
  "max_int_uniques": 15,
  "max_unique_values": 50,
  "feature_eng": "mi",
  "feature_select": 1,
  "outlier": "mad",
  "auto_save_per": 10,
  "imbalance": true,
  "n_clusters": 5,
  "anomaly_prior": 0.01
}
```

Response Codes: 201, 401, 403, 404, 408, 422

Successful Response:

```
{
    "job_name": "job_name1",
    "model_name": "model1_name",
}
```

Request Type: PATCH

URI: /v1/train/model/{model_name}

Headers:

- Authorization: Bearer token

Description: Resume training for a model on the dataset identified by *dataset_name*.

Parameter Descriptions:

- target: String denoting target prediction column in input data.
- dataset_names: A list of dataset names to use for training.
  Note: only 1 is currently supported.
- job_name: The job name
- max_train_time: Sets the training time for the model in 'HH:MM' format.
  Note: This overrides any values set for *max_generation*

Payload:

```
{
  "target": "string",
  "dataset_names": ["dataset_name1"],
  "job_name": "my_job",
  "max_train_time": "00:01"
}
```

Response Codes: 201, 401, 403, 404, 408, 422

Successful Response:

```
{
    "job_name": "job_name1",
    "model_name": "model1_name",
}
```

Request Type: DELETE

URI: /v1/train/model/{model_name}

Headers:

- Authorization: Bearer token

Description: Delete a model, possibly to prevent AI from becoming super intelligent and destroying the human race.

Response Codes: 204, 401, 404, 422

Successful Response: None

Upload

Request Type: POST

URI: /v1/upload

Headers:

- Authorization: Bearer token

Description: Upload a dataset, model, or a figure.

Form Data:

- *dataset*: a dataset file
- *dataset_name*: the name for the dataset

Response Codes: 201, 401, 403, 408, 413, 422

Successful Response:

```
{
    "dataset_name": "dataset1_name"
}
```

Request Type: DELETE

URI: /v1/upload/{dataset_name}

Headers:

- Authorization: Bearer token

Description: Delete a dataset, model, or a figure.

Response Codes: 204, 401, 404, 422

Successful Response: None

Request Type: GET

URI: /v1/download/{artifact_name}

Headers:

- Authorization: Bearer token

Description: Download an artifact.

Response Codes: 200, 401, 404, 422

Successful Response:

```
{
    'artifact': 'some string'
```

```
}
```

Examples

The following sections provide examples of how to use the Darwin API.

Login

1. Login using the `/v1/auth` routes. It is possible to login as either a *service* or a *user*.

Login	Request
Login as service:	**Request Type: POST** **URI**: /v1/auth/login **Form Data**: *api_key*: The api key of the service *pass1*: The service level password
Login as an end user:	**Request Type: POST** **URI**: /v1/auth/login/user **Form Data**: *api_key*: The api key of the service *username*: The end user name *pass1*: The end user level password

2. Receive token. If login is successful, a response arrives with an access token. This token is used in the *authorization header* for other requests:

```
{
    'access_token': 'some_string'
}
```

Note: The token (in this example “some string”) must be prepended with the string `Bearer`.
For example the token becomes:
`Bearer MyNewTokenString` - (that is: **Bearer(space)MyNewTokenString**).

Train a model - supervised

1. Upload a dataset using the following:

   Request Type: POST

   URI: /v1/upload

   Headers:

   - Authorization: Bearer token

Form Data:

- *dataset*: a dataset file
- *dataset_name*

Notes:

- Assign a name to the dataset. If no name is assigned, a random string is assigned in its place. It is necessary to keep track of lookup current datasets with the `/v1/lookup/dataset` route.
- Ensure not to exceed upload limits. If the limits are exceeded, a *403 forbidden error* is generated. To fix the exceeded limit, delete a dataset that is older or no longer required.

2. Set target parameter.
Use the uploaded dataset to train a model. Specify the dataset name in the URI of the train route. Note that training a supervised model requires the *target* parameter is set:

Request Type: POST

URI: /v1/train/model/{dataset_name}

Headers:

- Authorization: Bearer token

Payload:

```
{
  "target": "string",
  "dataset_names": ["dataset_name1"],
  "job_name": "job_name1",
  "model_name": "string",
  "max_train_time": "00:01",
  "max_generation": 100,
  "recurrent": true,
  "impute": "mean",
  "drop": true,
  "max_int_uniques": 15,
  "max_unique_values": 50,
  "feature_eng": "mi",
  "feature_select": 1,
  "outlier": "mad",
  "auto_save_per": 10,
  "imbalance": true,
  "clustering": true,
  "n_clusters": 5,
  "anomaly_prior": 0.01
}
```

Note: Because many of the payload parameters are optional, depending on your use case, it is possible to use a simple payload, for example:

```
{
  "target": "string",
  "dataset_names": ["dataset_name1"]
```

```
}
```

3. Consult return.
   In response to the payload, a job name and model name are returned. Note that if the dataset was not named, a random string is returned as its name. For example:

```
{
    "job_name": "job_name1",
    "model_name": "model1_name",
}
```

4. Check job status.
   When the job name is returned, use the `/v1/job/status` or `/v1/job/status/{job_name}` route together with the job name to check the job status. For example:

   Request Type: GET

   URI: /v1/job/status/{job_name}

   Headers:

   - Authorization: Bearer token

   Query Parameters:

   - Age: List jobs that are less than *X* units old, for example, *3 weeks*, *2 days*.
   - Status: List job of a particular status, for example *Running*.

   The query return contains information about how far the job has progressed, as a percent_complete and status, the number of generations run (so far), and the model loss. For example:

```
{
        "status": "Requested",
        "percent_complete": 0,
        "loss": 0,
        "generations": 0,
        "dataset_names": [
            "phone_data"
        ]
        "model_name": "model1_name",
    }
```

   When the job completes, the percent_complete shows `100` and status is set to `Complete`. The generated model can be used for additional tasks, described below.

Train a model - unsupervised

The process to train an unsupervised model follows the same procedure as the supervised model training procedure. The difference is the target parameter in the `/v1/train/model/{dataset_name}` route is left unspecified. Depending on the use case, it is possible to simplify the payload to a set of empty braces:

```
{}
```

Create risk data

The following example shows how to generate risk data.

1. Upload a failure dataset and time-series dataset through the /v1/upload route and follow the instructions above to create risk data. For example:

   Request Type: POST

   URI: /v1/risk/{failure_data}/{timeseries_data}

   Headers:

   - Authorization: Bearer token

   Payload:

   ```
   {
     "risk_columns": [
       "risk"
     ],
     "shutdown_column": "Shutdown Time",
     "return_column": "Return Time",
     "asset_column": "Asset",
     "lead_time": 1,
     "functional_form": "linear"
   }
   ```

2. Consult the return.
   The post action returns job and artifact names. The job name enables monitoring the job status.

3. Download the risk data.
   When the job completes, download the risk data via the `/v1/download/artifacts/{artifact_id}` route.

Analyze a dataset

1. If necessary, upload a dataset using the instructions and the `/v1/upload` route.
2. Analyze the dataset through the `/v1/analyze/data/{dataset_name}` route:<

   Request Type: POST

   URI: /v1/analyze/data/{dataset_name}

   Headers:

   - Authorization: Bearer token

   Payload:

```
{
  "target": "string",
  "impute": "mean",
  "drop": true,
  "max_int_uniques": 15,
  "max_unique_values": 50,
  "feature_eng": "mi",
  "feature_select": 1,
  "outlier": "mad"
}
```

3. Consult the return.
   The post action returns job and artifact names. The job name enables monitoring the job status.

4. Download the analysis.
   When the job completes, download the data analysis with the `/v1/download/artifacts/{artifact_id}` route.

Analyze a model

Analyze the trained model.

1. Use the the `/v1/analyze/model/{model_name}` route:

   Request Type: POST

   URI: v1/analyze/model/{model_name}

   Headers:

   - Authorization: Bearer token

   Form Data:

   - *job_name*: The name of the job
   - *artifact_name*: The name of the artifact

2. Consult the return.
   The post action returns job and artifact names. The job name enables monitoring the job status.
3. Download the analysis.
   When the job completes, download the model analysis with the `/v1/download/artifacts/{artifact_name}` route.
   Note: The model analysis takes the form of a decoded PNG. You must encode this yourself ("latin-1" encoding) or you can use the provided Python SDK to retrieve the results automatically.

Model uses

A trained model and dataset can be used for prediction, classification, or for detecting data clusters and / or anomalies.
After a model is trained, additional datasets can be uploaded and the model applied against those additional datasets. To perform these tasks:

1. Use the `/v1/run/model/{model_name}/{dataset_name}` route:

   Request Type: POST

   URI: /v1/run/model/{model_name}/{dataset_name}

   Headers:

   - Authorization: Bearer token

2. Consult the return.
   The post action returns job and artifact names. The job name enables monitoring the job status.
3. Download the results.
   When the job completes, download the results with the `/v1/download/artifacts/{artifact_id}` route.

Reference

Revision Table

Version	Author	Date	Notes
v 1.0	SparkCognition	02.02.2018	First Release