SMQTK-IQR

Release 0.15.1

Kitware, Inc.

Feb 23, 2022

CONTENTS

GitHub

This package provides the tools and web interface for using SMQTK's Interactive Query Refinement (IQR) platform.

CHAPTER

ONE

INSTALLATION

Please reference the SMQTK-Core installation documentation as such documentation for this package is nearly identical. Of course, replace uses of *smqtk-core* with *smqtk-iqr*.

CHAPTER

TWO

IQR CLASSES

Here we list and briefly describe the high level algorithm interfaces which SMQTK-IQR provides. Some implementations will require additional dependencies that cannot be packaged with SMQTK-IQR.

2.1 IqrController

class smqtk_iqr.iqr.iqr_controller.**IqrController**(*expire_enabled: bool = False*, *expire_check: float = 30*, *expire_callback: Optional[Callable] = None*)

Main controlling object for one or more IQR Sessions.

In order to interface with a web server, methods defined here are thread-safe.

This class may be used with the `with` statement. This will enable the instance's primary lock, preventing any other action from being performed on the instance while inside the with statement. The lock is reentrant, so nested with-statements will not dead-lock.

_handle_session_expiration() → None

Run on a separate thread periodic checks and removals of sessions that have expired.

add_session(*iqr_session:* smqtk_iqr.iqr.iqr_session.IqrSession, *timeout: float = 0*) → collections.abc.Hashable

Initialize a new IQR Session, returning the uuid of that session

This controller indexes the given session by its UUID.

Parameters

- **iqr_session** – The IqrSession instance to add
- **timeout** – The optional timeout, in seconds.

Returns UUID of new IQR Session

get_session(*session_uuid: collections.abc.Hashable*) → *smqtk_iqr.iqr.iqr_session.IqrSession*

Return the session instance for the given UUID

Raises **KeyError** – The given UUID doesn't exist in this controller.

Parameters **session_uuid** – UUID if the session to get

Returns IqrSession instance for the given UUID

has_session_uuid(*session_uuid: collections.abc.Hashable*) → bool

Check if this controller contains a session referenced by the given ID.

Performance using this function is faster compared to getting all UUIDs and performing a linear search (because hash tables).

This does NOT update session last access in regards to session expiration.

Parameters `session_uuid` – Possible UUID of a session

Returns True of the given UUID references a session in this controller and false if not.

`remove_session`(*session_uuid: collections.abc.Hashable*) → None
Remove an IQR Session by session UUID.

Raises `KeyError` – The given UUID doesn't exist in this controller.

Parameters `session_uuid` – Session UUID

`session_uuids`() → Tuple
Return a tuple of all currently registered IqrSessions.

This does NOT update session last access in regards to session expiration.

Returns a tuple of all currently registered IqrSessions.

`start_expiration_monitor`() → None
Initialize unique thread to check for session expiration if the feature is enabled. This does nothing if the feature is not enabled.

We stop the previous thread if one was started.

`stop_expiration_monitor`() → None
Stop the session expiration monitoring thread if one has been started. Otherwise this method does nothing.

2.2 IqrSession

class smqtk_iqr.iqr.iqr_session.**IqrSession**(*rank_relevancy_with_feedback: smqtk_relevancy.interfaces.rank_relevancy.RankRelevancyWithFeedback, pos_seed_neighbors: int = 500, session_uid: typing.Optional[typing.Union[str, uuid.UUID]] = None, distance_metric: typing.Callable[[numpy.ndarray, numpy.ndarray], numpy.ndarray] = <function euclidean_distance>, autoneg_select_ratio: int = 1*)
Encapsulation of IQR Session related data structures with a centralized lock for multi-thread access.

This object is compatible with the python with-statement, so when elements are to be used or modified, it should be within a with-block so race conditions do not occur across threads/sub-processes.

`_ordered_pos: Optional[Sequence[Tuple[smqtk_descriptors.interfaces.descriptor_element.DescriptorElement, float]]]`
Positively adjudicated descriptors in order of relevancy score.

`adjudicate`(*new_positives: Iterable[smqtk_descriptors.interfaces.descriptor_element.DescriptorElement] = (), new_negatives: Iterable[smqtk_descriptors.interfaces.descriptor_element.DescriptorElement] = (), un_positives: Iterable[smqtk_descriptors.interfaces.descriptor_element.DescriptorElement] = (), un_negatives: Iterable[smqtk_descriptors.interfaces.descriptor_element.DescriptorElement] = ()*) → None
Update current state of working set positive and negative adjudications based on descriptor UUIDs.

If the same descriptor element is listed in both new positives and negatives, they cancel each other out, causing that descriptor to not be included in the adjudication.

The given iterables must be re-traversable. Otherwise the given descriptors will not be properly registered.

Parameters

- **new_positives** – Descriptors of elements in our working set to now be considered to be positively relevant. collections.abc.Iterable[smqtk_descriptors.DescriptorElement]
- **new_negatives** – Descriptors of elements in our working set to now be considered to be negatively relevant. collections.abc.Iterable[smqtk_descriptors.DescriptorElement]
- **un_positives** – Descriptors of elements in our working set to now be considered not positive any more. collections.abc.Iterable[smqtk_descriptors.DescriptorElement]
- **un_negatives** – Descriptors of elements in our working set to now be considered not negative any more. collections.abc.Iterable[smqtk_descriptors.DescriptorElement]

external_descriptors(*positive: Iterable[smqtk_descriptors.interfaces.descriptor_element.DescriptorElement] = ()*, *negative: Iterable[smqtk_descriptors.interfaces.descriptor_element.DescriptorElement] = ()*) → None

Add positive/negative descriptors from external data.

These descriptors may not be a part of our working set.

TODO: Add ability to "remove" positive/negative external descriptors. See `adjudicate` method "**un_**…" parameters.

Parameters

- **positive** – Iterable of descriptors from external sources to consider positive examples.
- **negative** – Iterable of descriptors from external sources to consider negative examples. collections.abc.Iterable[smqtk_descriptors.DescriptorElement]

feedback_results() → List[smqtk_descriptors.interfaces.descriptor_element.DescriptorElement]

Return a list of all working-set descriptor elements that would benefit from further refinement. The list is in order of most to least useful.

If refinement has not yet occurred since session creation or the last reset, an empty tuple is returned.

Raises **RuntimeError** – If the end of the function is reached this means the feedback results have gotten into an invalid state.

get_negative_adjudication_relevancy() → List[Tuple[smqtk_descriptors.interfaces.descriptor_element.DescriptorElement, float]]

Return a list of the negatively adjudicated descriptors as tuples of `(element, score)` in order of descending relevancy score.

This does *not* include external negative adjudications, only negatively adjudicated descriptors in the working set.

If refinement has not yet occurred since session creation or the last reset, an empty list is returned.

Cache is invalidated when: - A refinement occurs. - Negative adjudications change.

get_positive_adjudication_relevancy() → List[Tuple[smqtk_descriptors.interfaces.descriptor_element.DescriptorElement, float]]

Return a list of the positively adjudicated descriptors as tuples of `(element, score)` in order of descending relevancy score.

This does *not* include external positive adjudications, only positively adjudicated descriptors in the working set.

If refinement has not yet occurred since session creation or the last reset, an empty list is returned.

Cache is invalidated when: - A refinement occurs. - Positive adjudications change.

`get_state_bytes`() → bytes
Get a byte representation of the current descriptor and adjudication state of this session.

This does not encode current results or the relevancy index's state, but these can be reproduced with this state.

Returns State representation bytes

`get_unadjudicated_relevancy`() → List[Tuple[smqtk_descriptors.interfaces.descriptor_element.DescriptorElement, float]]
Return a list of the non-adjudicated descriptor elements as tuples of `(element, score)` in order of descending relevancy score.

If refinement has not yet occurred since session creation or the last reset, an empty list is returned.

`ordered_results`() → List[Tuple[smqtk_descriptors.interfaces.descriptor_element.DescriptorElement, float]]
Return a tuple of all working-set descriptor elements as tuples of `(element, score)` in order of descending relevancy score.

If refinement has not yet occurred since session creation or the last reset, an empty tuple is returned.

`refine`() → None
Refine current model results based on current adjudication state

Raises

- **RuntimeError** – No working set has been initialized. *update_working_set()* should have been called after adjudicating some positive examples.
- **RuntimeError** – There are no adjudications to run on. We must have at least one positive adjudication.

`reset`() → None
Reset the IQR Search state

No positive adjudications, reload original feature data

`set_state_bytes`(*b: bytes*, *descriptor_factory: smqtk_descriptors.descriptor_element_factory.DescriptorElementFactory*) → None
Set this session's state to the given byte representation, resetting this session in the process.

Bytes given must have been retrieved via a previous call to `get_state_bytes` otherwise this method will fail.

Since this state may be completely different from the current state, this session is reset before applying the new state. Thus, any current ranking results are thrown away.

Parameters

- **b** – Bytes to set this session's state to.
- **descriptor_factory** – Descriptor element factory to use when generating descriptor elements from extracted data.

Raises **ValueError** – The input bytes could not be loaded due to incompatibility.

update_working_set(*nn_index: smqtk_indexing.interfaces.nearest_neighbor_index.NearestNeighborsIndex*) → None

Initialize or update our current working set using the given `NearestNeighborsIndex` instance given our current positively labeled descriptor elements.

We only query from the index for new positive elements since the last update or reset.

Parameters **nn_index** – `NearestNeighborsIndex` to query from.

Raises **RuntimeError** – There are no positive example descriptors in this session to use as a basis for querying.

CHAPTER

THREE

WEB SERVICE AND DEMONSTRATION APPLICATIONS

Included in this package are a few web-based service and demonstration applications, providing a view into the functionality provided by the included algorithms and utilities.

3.1 runApplication

This script can be used to run any conforming (derived from *SmqtkWebApp*) SMQTK web based application. Web services should be runnable via the `bin/runApplication.py` script.

Runs conforming SMQTK Web Applications.

```
usage: runApplication [-h] [-v] [-c PATH] [-g PATH] [-l] [-a APPLICATION] [-r]
                      [-t] [--host HOST] [--port PORT] [--use-basic-auth]
                      [--use-simple-cors] [--debug-server] [--debug-smqtk]
                      [--debug-app] [--debug-ns DEBUG_NS]
```

3.1.1 Named Arguments

-v, --verbose Output additional debug logging.

Default: False

3.1.2 Configuration

-c, --config Path to the JSON configuration file.

-g, --generate-config Optionally generate a default configuration file at the specified path. If a configuration file was provided, we update the default configuration with the contents of the given configuration.

3.1.3 Application Selection

-l, --list List currently available applications for running. More description is included if SMQTK verbosity is increased (-v | –debug-smqtk)

Default: False

-a, --application Label of the web application to run.

3.1.4 Server options

-r, --reload Turn on server reloading.

Default: False

-t, --threaded Turn on server multi-threading.

Default: False

--host Run host address specification override. This will override all other configuration method specifications.

--port Run port specification override. This will override all other configuration method specifications.

--use-basic-auth Use global basic authentication as configured.

Default: False

--use-simple-cors Allow CORS for all domains on all routes. This follows the "Simple Usage" of flask-cors: https://flask-cors.readthedocs.io/en/latest/#simple-usage

Default: False

3.1.5 Other options

--debug-server Turn on server debugging messages ONLY. This is implied when -v|–verbose is enabled.

Default: False

--debug-smqtk Turn on SMQTK debugging messages ONLY. This is implied when -v|–verbose is enabled.

Default: False

--debug-app Turn on flask app logger namespace debugging messages ONLY. This is effectively enabled if the flask app is provided with SMQTK and "–debug-smqtk" is passed. This is also implied if -v|–verbose is enabled.

Default: False

--debug-ns Specify additional python module namespaces to enable debug logging for.

Default: []

3.2 Sample Web Applications

3.2.1 IQR Demo Application

Interactive Query Refinement or "IQR" is a process whereby a user provides one or more exemplar images and the system attempts to locate additional images from within an archive that a similar to the exemplar(s). The user then adjudicates the results by identifying those results that match their search and those results that do not. The system then uses those adjudications to attempt to provide better, more closely matching results refined by the user's input.

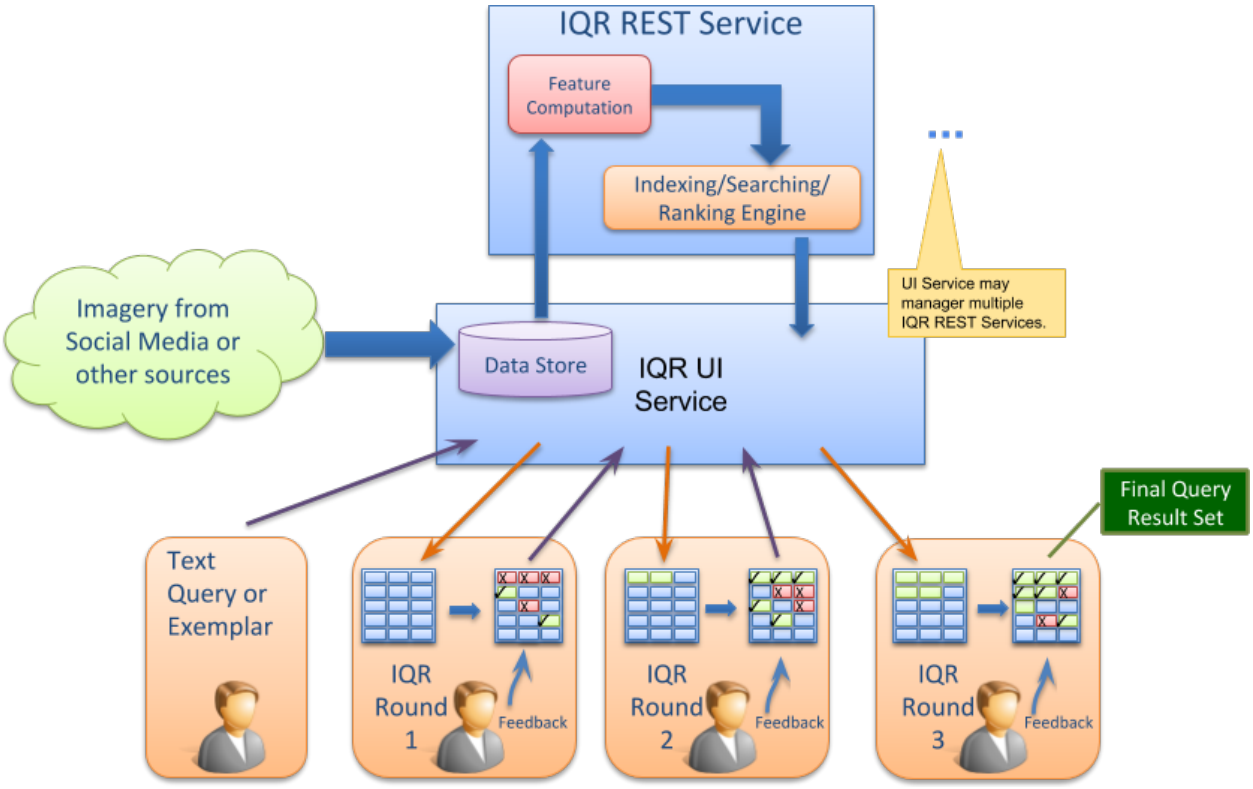

Fig. 1: **SMQTK IQR Workflow**
Overall workflow of an SMQTK based Interactive Query Refinement application.

The IQR application is an excellent example application for SMQTK as it makes use of a broad spectrum of SMQTK's capabilities. In order to characterize each image in the archive so that it can be indexed, the `DescriptorGenerator` algorithm is used. A `NearestNeighborsIndex` algorithm is used to understand the relationship between the images in the archive and a `RelevancyIndex` algorithm is used to rank results based on the user's positive and negative adjudications.

SMQTK comes with a pair of web-based application that implements an IQR system using SMQTK's services as shown in the *SMQTK IQR Workflow* figure.

Running the IQR Application

The SMQTK IQR demonstration application consists of two web services: one for hosting the models and processing for an archive, and a second for providing a user-interface to one or more archives.

In order to run the IQR demonstration application, we will need an archive of imagery. SMQTK has facilities for creating indexes that support 10's or even 100's or 1000's of images. For demonstration purposes, we'll use a modest archive of images. The Leeds Butterfly Dataset will serve quite nicely. Download and unzip the archive (which contains over 800 images of different species of butterflies).

SMQTK comes with a script, `iqr_app_model_generation`, that computes the descriptors on all of the images in your archive and builds up the models needed by the `NearestNeighborsIndex` and `RelevancyIndex` algorithms.

```
usage: iqr_app_model_generation [-h] [-v] -c PATH PATH -t TAB GLOB [GLOB ...]
```

Positional Arguments

GLOB Shell glob to files to add to the configured data set.

Named Arguments

-v, --verbose Output additional debug logging.

Default: False

-c, --config Path to the JSON configuration files. The first file provided should be the configuration file for the `IqrSearchDispatcher` web-application and the second should be the configuration file for the `IqrService` web-application.

-t, --tab The configuration "tab" of the `IqrSearchDispatcher` configuration to use. This informs what dataset to add the input data files to.

The `-c/--config` option should be given the 2 paths to the configuration files for the `IqrSearchDispatcher` and `IqrService` web services respectively. These provide the configuration blocks for each of the SMQTK algorithms (`DescriptorGenerator`, `NearestNeighborIndex`, etc.) required to generate the models and indices that will be required by the application. For convenience, the same configuration files will be provided to the web applications when they are run later.

The SMQTK source repository contains sample configuration files for both the `IqrSearchDispatcher` and `IqrService` services. They can be found at `smqtk_iqr/web/search_app/sample_configs/runApp.IqrSearchDispatcher.json` and `smqtk_iqr/web/search_app/sample_configs/runApp.IqrService.json` respectively. The `iqr_app_model_generation` script is designed to run from an empty directory and will create the sub-directories specified in the above configurations requires when run.

Since these configuration files drive both the generation of the models and the web applications themselves, a closer examination is in order.

Present in both configuration files are the `flask_app` and `server` sections which control Flask web server application parameters. The `runApp.IqrSearchDispatcher.json` contains the additional section `mongo` that configures the MongoDB server the UI service uses for storing user session information.

```
{
    "flask_app": {
        "BASIC_AUTH_PASSWORD": "demo",
        "BASIC_AUTH_USERNAME": "demo",
```

(continues on next page)

(continued from previous page)

```
        "SECRET_KEY": "MySuperUltraSecret"
    },
    "iqr_tabs": {
        "LEEDS Butterflies": {
            "data_set": {
                "smqtk_dataprovider.impls.data_set.memory.DataMemorySet": {
                    "cache_element": {
                        "smqtk_dataprovider.impls.data_element.file.DataFileElement": {
                            "explicit_mimetype": null,
                            "filepath": "models/image_elements.dms_cache",
                            "readonly": true
                        },
                        "type": "smqtk_dataprovider.impls.data_element.file.
↪DataFileElement"
                    },
                    "pickle_protocol": -1
                },
                "type": "smqtk_dataprovider.impls.data_set.memory.DataMemorySet"
            },
            "iqr_service_url": "localhost:5001",
            "working_directory": "data/iqr_app_work"
        }
    },
    "mongo": {
        "database": "smqtk",
        "server": "127.0.0.1:27017"
    },
    "server": {
        "host": "0.0.0.0",
        "port": 5000
    }
}
```

The `runApp.IqrSearchDispatcher.json` configuration has an additional block "iqr_tabs" (line 7). This defines the different archives, and matching IQR REST service describing that archive, the UI is to provide an interface for. In our case there will be only one entry, "LEEDS Butterflies" (line 8), representing the archive that we are currently building. This section describes the data-set container that contains the archive imagery to show in the UI (line 10) as well as the URL to the RESTful service providing the IQR functions for the archive (line 23).

In the `runApp.IqrService.json` configuration file (shown below) we see the specification of the algorithm and representation plugins the RESTful IQR service app will use under `iqr_service -> plugins`. Each of these of these blocks is passed to the SMQTK plugin system to create the appropriate instances of the algorithm or data representation in question. The blocks located at lines 41, 96, and 174 configure the three main algorithms used by the application: the descriptor generator, the nearest neighbors index, and the relevancy index. For example the `nn_index` block that starts at line 97 specifies two different implementations: `FlannNearestNeighborsIndex`, which uses the Flann library.

(jump past configuration display)

```
{
    "flask_app": {
        "BASIC_AUTH_PASSWORD": "demo",
        "BASIC_AUTH_USERNAME": "demo",
        "SECRET_KEY": "MySuperUltraSecret"
```

(continues on next page)

(continued from previous page)

```
    },
    "iqr_service": {
        "plugin_notes": {
            "classification_factory": "Selection of the backend in which classifications␣
↪are stored. The in-memory version is recommended because normal caching mechanisms␣
↪will not account for the variety of classifiers that can potentially be created via␣
↪this utility.",
            "classifier_config": "The configuration to use for training and using␣
↪classifiers for the /classifier endpoint. When configuring a classifier for use, don't␣
↪fill out model persistence values as many classifiers may be created and thrown away␣
↪during this service's operation.",
            "descriptor_factory": "What descriptor element factory to use when asked to␣
↪compute a descriptor on data.",
            "descriptor_generator": "Descriptor generation algorithm to use when␣
↪requested to describe data.",
            "descriptor_set": "This is the index from which given positive and negative␣
↪example descriptors are retrieved from. Not used for nearest neighbor querying. This␣
↪index must contain all descriptors that could possibly be used as positive/negative␣
↪examples and updated accordingly.",
            "neighbor_index": "This is the neighbor index to pull initial near-positive␣
↪descriptors from.",
            "relevancy_index_config": "The relevancy index config provided should not␣
↪have persistent storage configured as it will be used in such a way that instances are␣
↪created, built and destroyed often."
        },
        "plugins": {
            "classification_factory": {
                "smqtk_classifier.impls.classification_element.memory.
↪MemoryClassificationElement": {},
                "type": "smqtk_classifier.impls.classification_element.memory.
↪MemoryClassificationElement"
            },
            "classifier_config": {
                "smqtk_classifier.impls.classify_descriptor_supervised.sklearn_logistic_
↪regression.SkLearnLogisticRegression": {
                },
                "type": "smqtk_classifier.impls.classify_descriptor_supervised.sklearn_
↪logistic_regression.SkLearnLogisticRegression"
            },
            "descriptor_factory": {
                "smqtk_descriptors.impls.descriptor_element.postgres.
↪PostgresDescriptorElement": {
                    "binary_col": "vector",
                    "create_table": false,
                    "db_host": "/dev/shm",
                    "db_name": "postgres",
                    "db_pass": null,
                    "db_port": 5432,
                    "db_user": "smqtk",
                    "table_name": "descriptors_resnet50_pool5",
                    "uuid_col": "uid"
                },
```

(continues on next page)

(continued from previous page)

```
                "type": "smqtk_descriptors.impls.descriptor_element.postgres.
↪PostgresDescriptorElement"
            },
            "descriptor_generator": {
                "smqtk_descriptors.impls.descriptor_generator.caffe1.
↪CaffeDescriptorGenerator": {
                    "batch_size": 10,
                    "data_layer": "data",
                    "gpu_device_id": 0,
                    "image_mean": {
                        "smqtk_dataprovider.impls.data_element.file.DataFileElement" : {
                            "explicit_mimetype": null,
                            "filepath": "/home/smqtk/caffe/msra_resnet/ResNet_mean.
↪binaryproto",
                            "readonly": true
                        },
                        "type": "smqtk_dataprovider.impls.data_element.file.
↪DataFileElement"
                    },
                    "input_scale": null,
                    "load_truncated_images": true,
                    "network_is_bgr": true,
                    "network_model": {
                        "smqtk_dataprovider.impls.data_element.file.DataFileElement" : {
                            "explicit_mimetype": null,
                            "filepath": "/home/smqtk/caffe/msra_resnet/ResNet-50-model.
↪caffemodel",
                            "readonly": true
                        },
                        "type": "smqtk_dataprovider.impls.data_element.file.
↪DataFileElement"
                    },
                    "network_prototxt": {
                        "smqtk_dataprovider.impls.data_element.file.DataFileElement" : {
                            "explicit_mimetype": null,
                            "filepath": "/home/smqtk/caffe/msra_resnet/ResNet-50-deploy.
↪prototxt",
                            "readonly": true
                        },
                        "type": "smqtk_dataprovider.impls.data_element.file.
↪DataFileElement"
                    },
                    "pixel_rescale": null,
                    "return_layer": "pool5",
                    "use_gpu": false
                },
                "type": "smqtk_descriptors.impls.descriptor_generator.caffe1.
↪CaffeDescriptorGenerator"
            },
            "descriptor_set": {
                "smqtk_descriptors.impls.descriptor_set.postgres.PostgresDescriptorSet":
↪{
```

(continues on next page)

(continued from previous page)

```
                    "create_table": false,
                    "db_host": "/dev/shm",
                    "db_name": "postgres",
                    "db_pass": null,
                    "db_port": 5432,
                    "db_user": "smqtk",
                    "element_col": "element",
                    "multiquery_batch_size": 1000,
                    "pickle_protocol": -1,
                    "read_only": false,
                    "table_name": "descriptor_set_resnet50_pool5",
                    "uuid_col": "uid"
                },
                "type": "smqtk_descriptors.impls.descriptor_set.postgres.
↪PostgresDescriptorSet"
            },
            "neighbor_index": {
                "smqtk_indexing.impls.nn_index.faiss.FaissNearestNeighborsIndex": {
                    "descriptor_set": {
                        "__note__": "Using real descriptor index this time",
                        "smqtk_descriptors.impls.descriptor_set.postgres.
↪PostgresDescriptorSet": {
                            "create_table": false,
                            "db_host": "/dev/shm",
                            "db_name": "postgres",
                            "db_pass": null,
                            "db_port": 5432,
                            "db_user": "smqtk",
                            "element_col": "element",
                            "multiquery_batch_size": 1000,
                            "pickle_protocol": -1,
                            "read_only": false,
                            "table_name": "descriptor_set_resnet50_pool5",
                            "uuid_col": "uid"
                        },
                        "type": "smqtk_descriptors.impls.descriptor_set.postgres.
↪PostgresDescriptorSet"
                    },
                    "factory_string": "IDMap,Flat",
                    "gpu_id": 0,
                    "idx2uid_kvs": {
                        "smqtk_dataprovider.impls.key_value_store.postgres.
↪PostgresKeyValueStore": {
                            "batch_size": 1000,
                            "create_table": true,
                            "db_host": "/dev/shm",
                            "db_name": "postgres",
                            "db_pass": null,
                            "db_port": 5432,
                            "db_user": "smqtk",
                            "key_col": "key",
                            "pickle_protocol": -1,
```

(continues on next page)

(continued from previous page)

```
                            "read_only": false,
                            "table_name": "faiss_idx2uid_kvs",
                            "value_col": "value"
                        },
                        "type": "smqtk_dataprovider.impls.key_value_store.postgres.
↪PostgresKeyValueStore"
                    },
                    "uid2idx_kvs": {
                        "smqtk_dataprovider.impls.key_value_store.postgres.
↪PostgresKeyValueStore": {
                            "batch_size": 1000,
                            "create_table": true,
                            "db_host": "/dev/shm",
                            "db_name": "postgres",
                            "db_pass": null,
                            "db_port": 5432,
                            "db_user": "smqtk",
                            "key_col": "key",
                            "pickle_protocol": -1,
                            "read_only": false,
                            "table_name": "faiss_uid2idx_kvs",
                            "value_col": "value"
                        },
                        "type": "smqtk_dataprovider.impls.key_value_store.postgres.
↪PostgresKeyValueStore"
                    },
                    "index_element": {
                        "smqtk_dataprovider.impls.data_element.file.DataFileElement" : {
                            "filepath": "models/faiss_index",
                            "readonly": false
                        },
                        "type": "smqtk_dataprovider.impls.data_element.file.
↪DataFileElement"
                    },
                    "index_param_element": {
                        "smqtk_dataprovider.impls.data_element.file.DataFileElement" : {
                            "filepath": "models/faiss_index_params.json",
                            "readonly": false
                        },
                        "type": "smqtk_dataprovider.impls.data_element.file.
↪DataFileElement"
                    },
                    "ivf_nprobe": 64,
                    "metric_type": "l2",
                    "random_seed": 0,
                    "read_only": false,
                    "use_gpu": false
                },
                "type": "smqtk_indexing.impls.nn_index.faiss.FaissNearestNeighborsIndex"
            },
            "rank_relevancy_with_feedback": {
                "smqtk_relevancy.impls.rank_relevancy.margin_sampling.
↪RankRelevancyWithMarginSampledFeedback": {
```

(continues on next page)

(continued from previous page)

```
176                     "rank_relevancy": {
177                         "smqtk_relevancy.impls.rank_relevancy.wrap_classifier.
    ↪RankRelevancyWithSupervisedClassifier": {
178                             "classifier_inst": {
179                                 "smqtk_classifier.impls.classify_descriptor_supervised.
    ↪sklearn_logistic_regression.SkLearnLogisticRegression": {
180                                 },
181                                 "type": "smqtk_classifier.impls.classify_descriptor_
    ↪supervised.sklearn_logistic_regression.SkLearnLogisticRegression"
182                             }
183                         },
184                         "type": "smqtk_relevancy.impls.rank_relevancy.wrap_classifier.
    ↪RankRelevancyWithSupervisedClassifier"
185                     },
186                     "n": 10,
187                     "center": 0.5
188                 },
189                 "type": "smqtk_relevancy.impls.rank_relevancy.margin_sampling.
    ↪RankRelevancyWithMarginSampledFeedback"
190             }
191         },
192         "session_control": {
193             "positive_seed_neighbors": 500,
194             "session_expiration": {
195                 "check_interval_seconds": 30,
196                 "enabled": true,
197                 "session_timeout": 86400
198             }
199         }
200     },
201     "server": {
202         "host": "0.0.0.0",
203         "port": 5001
204     }
205 }
```

Once you have the configuration file set up the way that you like it, you can generate all of the models and indexes required by the application by running the following command:

```
iqr_app_model_generation \
    -c runApp.IqrSearchDispatcher.json runApp.IqrService.json \
    -t "LEEDS Butterflies" /path/to/butterfly/images/*.jpg
```

This will generate descriptors for all of the images in the data set and use them to compute the models and indices we configured, outputting to the files under the `workdir` directory in your current directory.

Once it completes, you can run the `IqrSearchApp` and `IqrService` web-apps. You'll need an instance of MongoDB running on the port and host address specified by the `mongo` element on line 27 in your `runApp.IqrSearchDispatcher.json` configuration file. You can start a Mongo instance (presuming you have it installed) with:

```
mongod --dbpath /path/to/mongo/data/dir
```

Once Mongo has been started you can start the `IqrSearchApp` and `IqrService` services with the following commands

in separate terminals:

```
# Terminal 1
runApplication -a IqrService -c runApp.IqrService.json

# Terminal 2
runApplication -a IqrSearchDispatcher -c runApp.IqrSearchDispatcher.json
```

After the services have been started, open a web browser and navigate to `http://localhost:5000`. Click lick on the `login` button in the upper-right and then enter the credentials specified in the default login settings file `source/python/smqtk/web/search_app/modules/login/users.json`.

Fig. 2: *Click on the login element to enter your credentials*

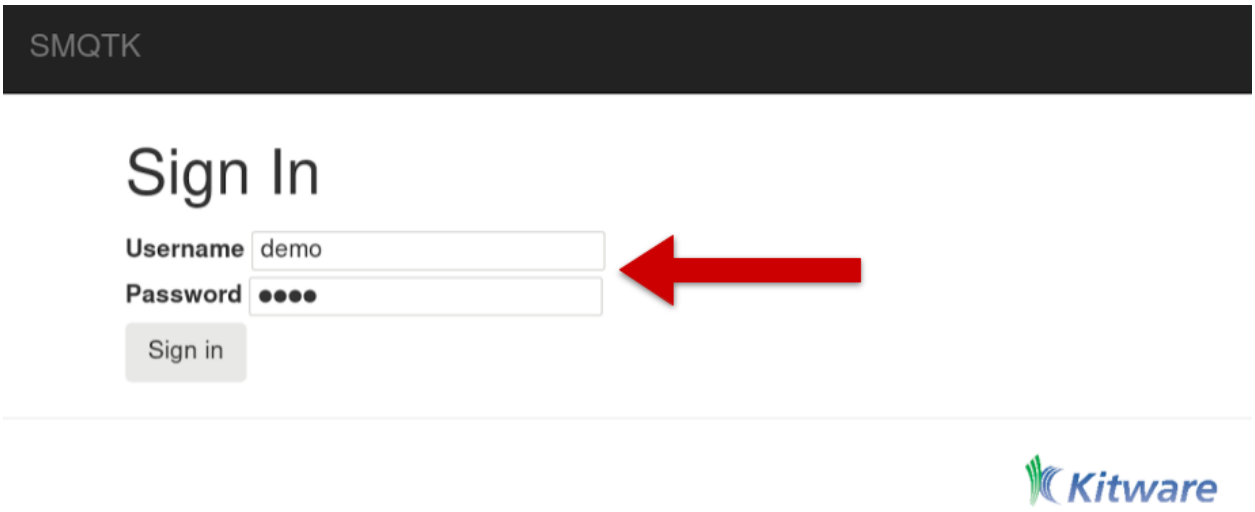

Fig. 3: *Enter demo credentials*

Once you've logged in you will be able to select the `LEEDS Butterfly` link. This link was named by line 8 in the `runApp.IqrSearchDispatcher.json` configuration file. The `iqr_tabs` mapping allows you to configure interfacing with different IQR REST services providing different combinations of the required algorithms – useful for example, if you want to compare the performance of different descriptors or nearest-neighbor index algorithms.

To begin the IQR process drag an exemplar image to the grey load area (marked 1 in the next figure). In this case we've uploaded a picture of a Monarch butterfly (2). Once uploaded, click the `Initialize Index` button (3) and the system will return a set of images that it believes are similar to the exemplar image based on the descriptor computed.

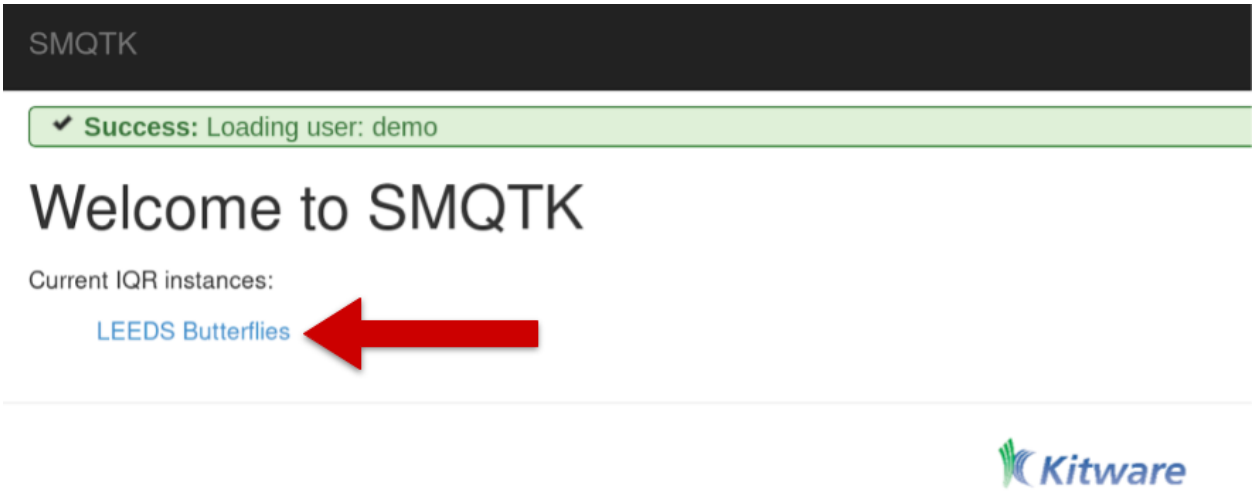

Fig. 4: *Select the "LEEDS Butterflies" link to begin working with the application*

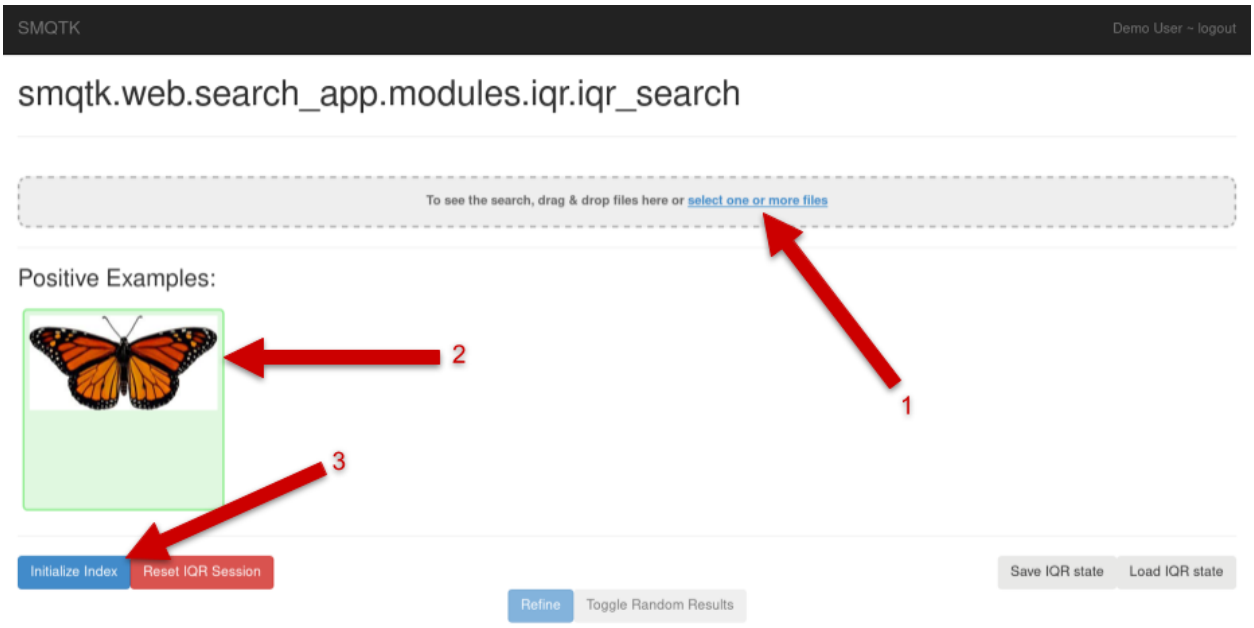

Fig. 5: *IQR Initilization*

The next figure shows the set of images returned by the system (on the left) and a random selection of images from the archive (by clicking the `Toggle Random Results` element). As you can see, even with just one exemplar the system is beginning to learn to return Monarch butterflies (or butterflies that look like Monarchs)

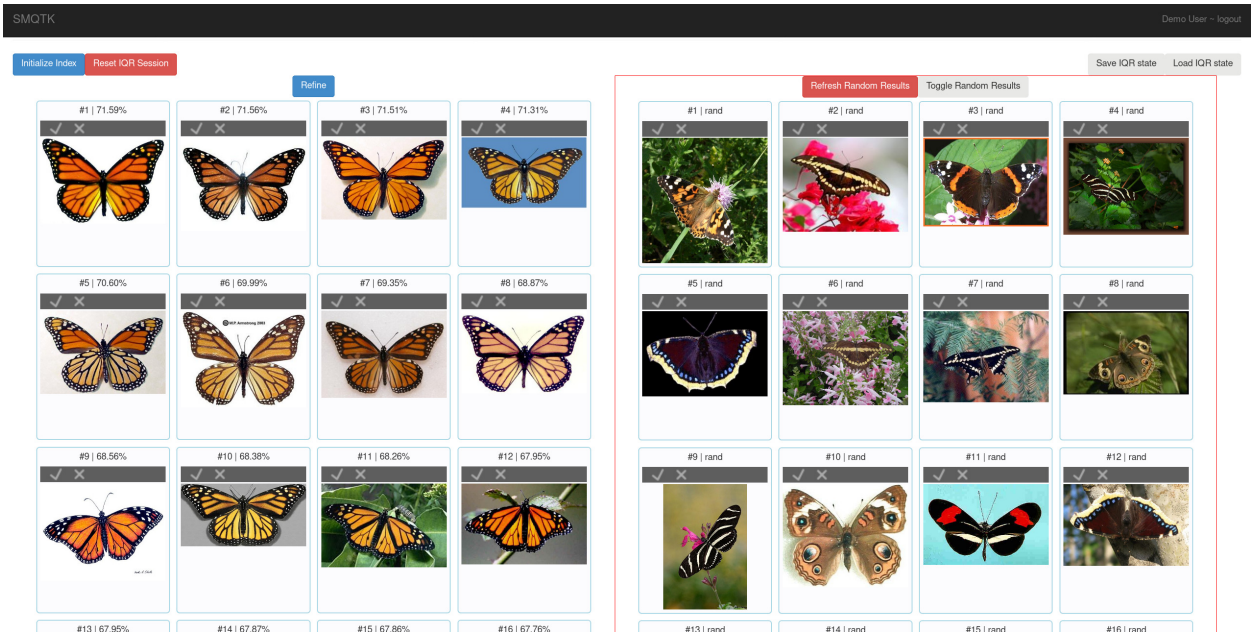

Fig. 6: *Initial Query Results and Random Results*

At this point you can begin to refine the query. You do this by marking correct returns at their checkbox and incorrect returns at the "X". Once you've marked a number of returns, you can select the "Refine" element which will use your adjudications to retrain and rerank the results with the goal that you will increasingly see correct results in your result set.

You can continue this process for as long as you like until you are satisfied with the results that the query is returning. Once you are happy with the results, you can select the `Save IQR State` button. This will save a file that contains all of the information requires to use the results of the IQR query as an image classifier. The process for doing this is described in the next session.

Using an IQR Trained Classifier

Before you can use your IQR session as a classifier, you must first train the classifier model from the IQR session state. You can do this with the `iqrTrainClassifier` tool:

```
usage: iqrTrainClassifier [-h] [-v] [-c PATH] [-g PATH] [-i IQR_STATE]
```

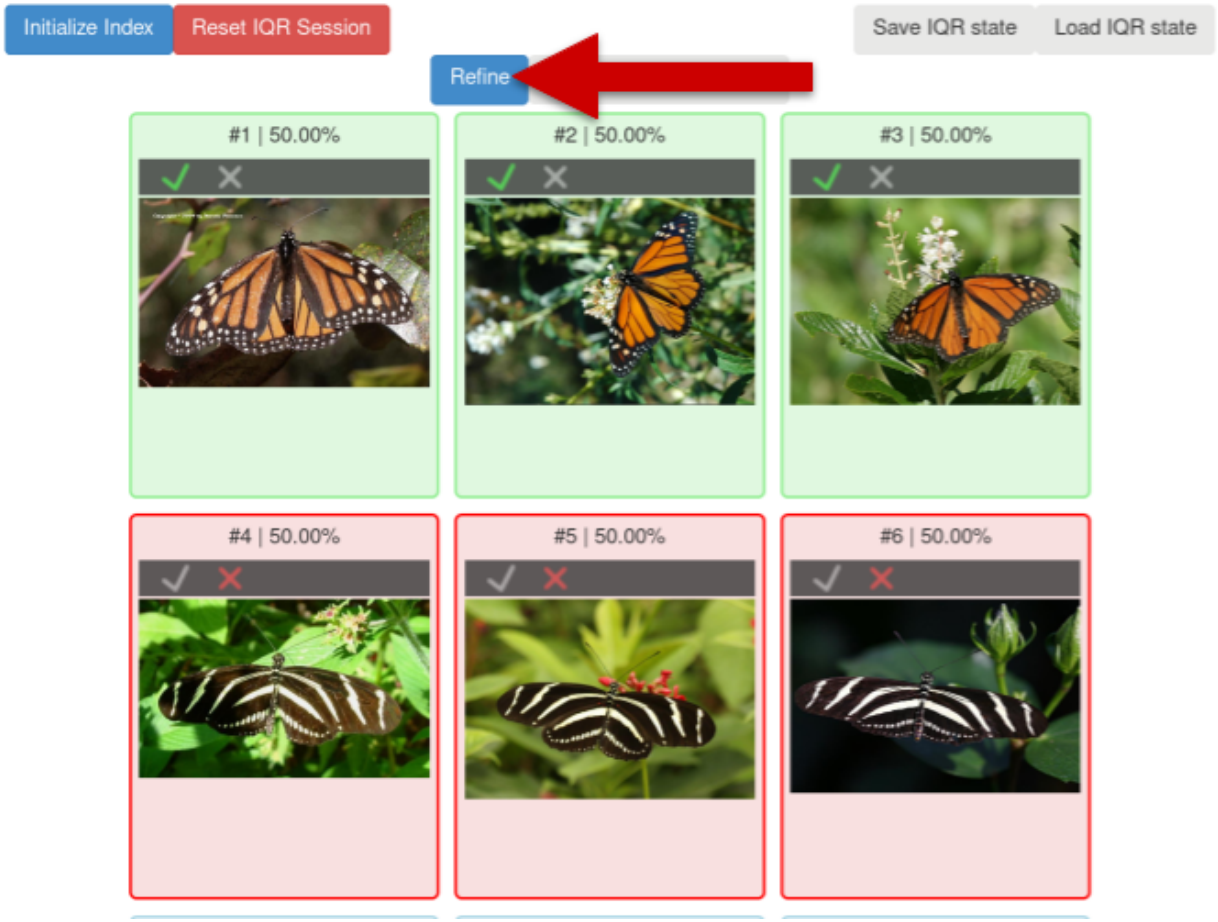

Fig. 7: *Query Refinement*

Named Arguments

-v, --verbose Output additional debug logging.

Default: False

-i, --iqr-state Path to the ZIP file saved from an IQR session.

Configuration

-c, --config Path to the JSON configuration file.

-g, --generate-config Optionally generate a default configuration file at the specified path. If a configuration file was provided, we update the default configuration with the contents of the given configuration.

As with other tools from SMQTK the configuration file is a JSON file. An default configuration file may be generated by calling `iqrTrainClassifier -g example.json`, but pre-configured example file can be found `here` and is shown below:

```
{
    "classifier": {
        "smqtk_classifier.impls.classify_descriptor_supervised.sklearn_logistic_
↪regression.SkLearnLogisticRegression": {
        },
        "type": "smqtk_classifier.impls.classify_descriptor_supervised.sklearn_logistic_
↪regression.SkLearnLogisticRegression"
    }
}
```

The above configuration specifies the classifier that will be used, in this case the `LibSvmClassifier`. Let us assume the IQR session state was downloaded as `monarch.IqrState`. The following command will train a classifier leveraging the descriptors labeled by the IQR session that was saved:

```
iqrTrainClassifier.py -c config.iqrTrainClassifier.json -i monarch.IqrState
```

CHAPTER

FOUR

RELEASE PROCESS AND NOTES

4.1 Steps of the SMQTK Release Process

Three types of releases are expected to occur: - major - minor - patch

See the `CONTRIBUTING.md` file for information on how to contribute features and patches.

The following process should apply when any release that changes the version number occurs.

4.1.1 Create and merge version update branch

Patch Release

A patch release should only contain fixes for bugs or issues with an existing release. No new features or functionality should be introduced in a patch release. As such, patch releases should only ever be based on an existing release point.

1. Create a new branch off of the `release` branch named something like `release-patch-{NEW_VERSION}`.
   - Increment patch value in `python/smqtk/__init__.py` file's `__version__` attribute.
   - Rename the `docs/release_notes/pending_patch.rst` file to `docs/release_notes/v{VERSION}.rst`, matching the value in the `__version__` attribute. Add a descriptive paragraph under the title section summarizing this release.
   - Add new release notes RST file reference to `docs/release_notes.rst`.
2. Tag branch (see *Tag new version* below).
3. Merge version bump branch into `release` and `master` branches.

Major and Minor Releases

Major and minor releases may add one or more trivial or non-trivial features and functionalities.

1. Create a new branch off of the `master` or `release` named something like `release-[major, minor]-{NEW_VERSION}`.

a) Increment patch value in `VERSION` file.

b) Rename the `docs/release_notes/pending_release.rst` file to `docs/release_notes/v{VERSION}.rst`, matching the value in the `VERSION` file. Add a descriptive paragraph under the title section summarizing this release.

c) Add new release notes RST file reference to `docs/release_notes.rst`.

2. Create a pull/merge request for this branch with master as the merge target. This is to ensure that everything passes CI testing before making the release. If there is an issue then branches should be made and merged into this branch until the issue is resolved.
3. Tag branch (see *Tag new version* below) after resolving issues and before merging into `master`.
4. Reset the release branch (–hard) to point to the new branch/tag.
5. Merge version bump branch into the `master` branch.

4.1.2 Tag new version

Release branches should be tagged in order to record where in the git tree a particular release refers to. The branch off of `master` or `release` is usually the target of such tags.

Currently the `From GitHub` method is preferred as it creates a "verified" release.

From GitHub

Navigate to the releases page on GitHub and click the `Draft a new release` button in upper right.

Fill in the new version in the `Tag version` text box (e.g. `v#.#.#`) and use the same string in the `Release title` text box. The "@" target should be the release branch created above.

Copy and past this version's release notes into the `Describe this release` text box.

Remember to check the `This is a pre-release` check-box if appropriate.

Click the `Public release` button at the bottom of the page when complete.

From Git on the Command Line

Create a new git tag using the new version number (format: `v<MAJOR.<MINOR>.<PATCH>`) on the merge commit for the version update branch merger:

```
$ git tag -a -m "[Major|Minor|Patch] release v#.#.#"
```

Push this new tag to GitHub (assuming origin remote points to SMQTK on GitHub:

```
$ git push origin v#.#.#
```

To add the release notes to GitHub, navigate to the tags page on GitHub and click on the "Add release notes" link for the new release tag. Copy and paste this version's release notes into the description field and the version number should be used as the release title.

4.1.3 Create new version release to PYPI

Make sure the source is checked out on the newest version tag, the repo is clean (no uncommited files/edits), and the `build` and `dist` directories are removed:

```
$ git checkout <VERSION_TAG>
$ rm -r dist python/smqtk.egg-info
```

Create the `build` and `dist` files for the current version with the following command(s) from the source tree root directory:

```
$ python setup.py sdist
```

Make sure your `$HOME/.pypirc` file is up-to-date and includes the following section with your username/password:

```
[pypi]
username = <username>
password = <password>
```

Make sure the `twine` python package is installed and is up-to-date and then upload dist packages created with:

```
$ twine upload dist/*
```

4.2 Release Notes

4.2.1 v0.15.0

Updates / New Features

CI

- Add Github actions workflow for CI.
- Setup Read the Docs project

Misc.

- Now standardize to using Poetry for environment/build/publish management.
  - Collapsed pytest configuration into the `pyproject.toml` file.

Fixes

- Update CI configurations to use Poetry.
- Fixed MyPy and flake8 issues
- Fixed pytest issues

4.2.2 v0.15.1

Updates / New Features

Build

- Updated `smqtk-*` dependency version requirements to be up-to-date with the latest versions.

Fixes

Tests

- Fixed unit tests that were broken due to API changed in the updated `smqtk-*` packages.
- Added more unit tests for IQRSession

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

Symbols

A

E

F

G

H

I

O

R

S

U