Documentation

Mavroudo

March 25, 2020

1 Model

Describing all the classes that are represented in the model package

- **AugmentedDetail:** Represents an detail that is augmented by the event name to which it corresponds. Constructor (eventName,key,value). Compare, Equals, toString, hashCode.
- **Completion:** Represents a Completion returned through the quick_stats endpoint. Constructor (step,completions,averageDuration,lastCompletedAt). *The step index of the completed funnel (e.g. in a funnel A-¿B-¿C a completion with step index 1 corresponds to a completion of the A-¿B sub-funnel).* Getters,Setter,toString.
- **Detail:** A Step Detail as provided by the JSON input. Is a Builder has 3 fields: key, operator, value. toString returns a json file.
- **DetailedCompletion:** Represents a single Detailed Completion returned through the export_completions endpoint. Constructor (step, completed_at, duration, session_id, app_id, device_id, user_id). The duration of the funnel, completed_at = the timestamp of the last event in the completion.
- **Event** Represents a funnel Event. 3 different constructors. Fields: *name* (Funnel event name (prefixed by AppID and LogtypeID, i.e. appID_logtypeID_eventName)) and *details* (TreeSet< *AugmentedDetail* >). Compare, matches, toString, equals, hashcode, getPrefixedAppID.
- **EventPair:** 2 events as string_name. Equals, hashcode, equals, toString.
- **EventPairFrequency:** Represents an EventPairFrequency returned by probing the "count" index tables. Constructor (EventPair, sum_duration(sum of all durations of this event pair), comp_count(sum of all completions of this event pair)). Compare (which is not consistent with equals).
- **Funnel:** A Funnel body as provided by the JSON input (i.e. not wrapped by a funnel tag. Contains a Builder. fields: *steps List* < *Step* > and *maxDuration*. A toString that returns a json of this object.

- **FunnelWrapper:** A Funnel Wrapper as provided by the JSON input (i.e. the funnel tag that wraps a funnel body). Contains Builder. has only one field: Funnel.
- **LifeTime:** Represents the lifetime of a DetailedCompletion. Constructor (start_date, end_date, duration, appID).
- **Name:** A Step Name as provided by the JSON input. Builder, fields: *logName*, *applicationID*, *logType*.
- **Proposition:** Represents a Proposition corresponding to a continuation provided by the explore/* endpoint. 3 Constructors. Fields: *applicationID*, *logType*, *logName*, *completions* , *averageDuration*, *lastCompleted*. toString, compareTo.
- **QueryPair:** A pair of events (along with their details) corresponding to a query pair (e.g. a 3-sized funnel has 3 query pairs). Fields: *first_ev*, *second_ev*. hashCode, equals, toString.
- **QuickStatsResponse:** Represents the output of an quick_stats endpoint query. Fields: *completions List < Completion >*. equals, toString, hashCode.
- **Sequence:** Events are stored in an ArrayList in the order they appear in the funnel. Fields: *seq List < Event >*. Methods:
  - *appendToSequence(Event)*
  - *prependToSequence(Event)*:add it to the first position
  - *removeLastEvent*
  - *insert(event,position)*
  - *getList*
  - *getFirstEvent*
  - *getLastEvent*
  - *getQueryTuples()*: return a *List < QueryPair >* (a list of all query pairs for the funnel. For example, a funnel A-B-C has the following query pairs: (A,B), (B,C), (A,C)),
  - *getEvent(position)*
  - *getSize()*
  - *fitsTimestamps(List < TimestampedEvent >, maxDuration)*, Check if this sequence exists in a (superset) sequence of events. Furthermore, the sequence must exist within a time limit. Return an array of two values [timestamp of the first event occurrence, timestamp of the last event occurrence] in the list of timestamped events. If this sequence is not contained in the greater list, [-1,-1] is returned
  - *getUniqueEvents*. Return a Set¡Event¿

- *getSecondLastEventDelimited(String delim).* Get the second to last event of the sequence concatenated with a delimiter (Currently not used).

- **Status:** Represents the output of an @code status endpoint query. Fields: $List < String > user$, $List < String > device$

- **Step:** A Step as provided by the JSON input. Fields: *names* $List < Name >$, *details* $List < Detail >$, which are matched from the json query. Created with Builder.

- **TimestampedEvent:** An event annotated with a ink java.util.Date timestamp. Fields: *timestamp*, *event*. compare, hashCode, equals, toSting.

2 Query

All the classes that are contained in the query package.

- **SequenceQueryHandler:** Class designed to handle the low-level needs of a query evaluation. Each time a query fetches data from the index (not count) tables the incoming rows are handled by this class first. Each row is handled by extracting info about the entity (device or user) ID that achieved the completion and then handling the completion itself in one of two different ways (i.e. as an event pair or as a detail triplet).
  Fields:

  - protected Cluster
  - protected Session
  - protected KeyspaceMetadata
  - protected final string Delimiter
  - $ConcurrentHashMap < String, List < TimestampedEvent >$ allEventsPerSession.

  Methods:

  - public **stripAppIdAndLogtype**
  - public **getYearMonths**. return all months between 2 provided dates + periods per/ 10 days.
  - protected **handleSessionRow**(Sequence, first, second, startDate, endDate,$List < String >$ candSessioons, putInMap). Running through the information per Session and handle each row differently, depending on the data that contains.

- private **handleEventPairs** (dateFormat, sessionID, string[] times, first, second, startDate, endDate). times contains all the different times for this pair of events in this session id. If session id is contained in the allEventsPerSession, then TimestampedEvents will be added, if the timestamps are between limits.
- private **handleDetailPairs**, same as before only now contains a event_key_value.

Classes:

- **CandSessionsCallback** implements FutureCallback< $ResultSet$ >. oveerides onSuccess and onFailure based on if the query in cassandra was successfull or not.

- **SequenceQueryEvaluator** extends **SequenceQueryHandler:** The class responsible for accurately evaluating a query. Each query is evaluated on a per month basis and then all monthly results are aggregated in a single candidate list. All the initial candidates are then tested and any false positives are discarded. **Fields**: No new field, same as the superclass. **Methods**:

  - public Set< $String$ > **evaluateQuery**(startDate, endDate, Sequence, Map< $Integer, List < AugmentedDetail >>$, isUsersQuery). One table per month in Cassandra (dvc_idx_year_month for not userQuery else usr instead of dvc). If table exists then we take the candidates from this month by using **evaluateQueryOnMonth**.
  - private Set< $String$ > **evaluateQueryOnMonth**(tableName, $List <$ $QueryPair >$, Sequence, startDate, endDate, Map< $Integer, List <$ $AugmentedDetail >>$, isUsersQuery). queryPairs was calculated based on the sequence in the previous. Query to Cassandra the first query pair, in the given table, the 2 fields and the third is null. Then continues for the other pairs in a multi threading, suing the **CandSessionsCallBack** from the superclass. Closes all the threads and returns the month candidates.
  - public **findTruePositivesAndLifetime(Sequence, candidates, maxDuration)**. For each candidate we check for two requirements: 1) The events of that candidate occur in the correct order, 2) They occur within the timeframe of the "maxDuration" variable. For every candidate, gets the candidates from the allEventsPerSession. Then using **fitsTimestamps** from Sequence, to find if a pair is contained in a superset. Returns a Map of $< String, LifeTime >$, where string is the candidate pair and lifetime is representation of a DetatiledCompletion.
  - getDeviceIds(startDate, endDate, userIDS), querying Cassandra for every month, every userId and returns the device id.

- getUserIds(startDate, endDate, deviceIDS), querying Cassandra for every month, every deviceID and returns the user id.

- **SequenceQueryExplorer** extends **SequenceQueryEvaluator: Fields**: No new field, same as the superclass.
  Methods:

  - public **exploreQueryAccurateStep**(startDate, endDate, Sequence, Map< $Integer, List < AugmentedDetail >>$, position, maxDuration, isUsersQuery). Returns a Map< $String, Map < String, Long >>$, which is a set of true positives (along with their completion duration) for each possible continuation. Position defines the index on the query sequence where continuations are to be explored.
    * $0 < position < query.size$: Intermediate explore, calls the **getAllPossibleIntermediateEvents** method
    * $position = query.size$: Forward explore, calls **getEventFrequencyCounts** method.
    * $position = 0$: Prefix explore, calls **getAllPossiblePrecedingEvents** method.

    All these methods return a $List < EventPairFrequency >$. For every EventPairFrequency we perform the followings:
    After dismissing all the app ids that we don't care, creates a temporal sequence from the given one and:
    * $0 < position < query.size$: insert the second event in the given position
    * $position = query.size$: add event at the end of the sequence
    * $position = 0$: add event at the beginning of the sequence

    Then this query is passed to the **evaluateQuery** method, which belongs to the superclass. finds true positives with **findTruePositives** method from the superclass and append $second\_event, truePositives >$ in the final results.
  - public **exploreQueryFastStep**(startDate, endDatem Sequence, Map< $Integer, List < AugmentedDetail >>$, position, targetAppID, lastCompletions, isUsersQuery). Performs a fast evaluation of each possible continuation. Used in the /export/fast and /export/hybrid endpoints. Starts by creating a $List < Proposition >$ which will be the final results. Depending on the position performs the follows:

    * $0 < position < query.size$: Intermediate explore. lastEvent is the event in position (position-1). calls method **getEventFrequencyCounts** passing as event this lastEvent. Then for every EventPairFrequency in the returned values, dismisses the events that has not the given targetID. Creates a Proposition object and adds it the final results if it is interesting. The rest procedure is the same **comment:** might need to change that

* *position* >= *query.size*: Forward explore. lastEvent is the last event of the Sequence.
* *the rest*: Prefix explore. Calls the **getReverseEventFrequencyCounts** method passing as event the first event of the Sequence. The rest procedure is the same **comment:** might need to change that

Below are the private methods that return $List < EventPairFrequency >$

- private **getEventFrequencyCounts**(year_months, queryEvent, isUsersQuery). Creates 2 hashMaps to hold allDurationSums and allCompSums. For every month perform a query in *Cassandra* in the user_count_+year_month for users or dvc in case it is not a usersQuery. This *table* contains $< event\_name, sum\_duration, comp\_count >$ data. Then all the data has been collected, create EventPairFrequency by add as eventPair (queryEvent, eventFound), allDurationSums (Corresponding to this event), allCompCounts corresponding to this event. Sorts reverse and return the list of EventPairFrequency.
- private **getDetailFrequencyCounts**(year_months, queryEvent, queryKey, isUsersQuery). We will not discuss this further cause it is not used in our Example.
- private **getAllPossiblePrecedingEvents**(year_month, secondEvent, isUsersQuery). All eventPairFrequency will have as first event "n/a". as second the a possible precedingEvents and the 2 counters will be set to 0.0. They query *Cassandra* usr_idx_+year_month or device, by the second field and return all the first events from this query.
- private **getAllPossibleInternediateEvents**(year_months, eventFirst, isUsersQuery). It is the opposite of the method above. Query *Cassandra* in the index table, the first field and return all the second events. Again the EventPairFrequency will have "n/a" as first event, eventFound as second and 0.0 in both counters.
- private **getReverseEventFrequencyCounts**(year_months, queryEvent, isUsersQuery). Same as **getEventFrequencyCounts**, but it is just querying a different table in *Cassandra*, the reverse_usr_count_+year_month. **And 2 public functions**
- public **getCountForQueryPair**(startDate, endDate, QueryPair, isUsersQuery). Calls **getEventFrequencyCounts** for the first event and then counts only the events that have as second event the second event of the pair. Return the counter.
- public **getEventForEventKeyValueTriplet**(startDate, endDatem event, key, value, isUsersQuery). Uses **getDetailFrequencyCounts** that will be not discussed in this example.

- **ResponseBuilder**Class responsible for generating a response to any POST query made through the application. More specifically, the class returns

the following objects depending on the POST query:

- com.fa.funnel.rest.model.QuickStatsResponse for the 'quick_stats' endpoint
- com.fa.funnel.rest.model.ExploreResponse for the 'explore/mode' endpoint
- java.lang.String for the 'export_completions' endpoint

Fields: cluster, session, KeyspaceMetadata, cassandra_keyspace_name, entityMapping, steps, startDate, endDate, maxDuration, fileHeader, IS_USERS_Query
Methods:

- A basic **Constructor**, based onthe fields above. Passing a **Funnel object**, which contains the necessary information.

  Below are the functions that build a Response to a query
- public **buildQuickStatsResponse()**. Returns a QuickStatsResponse, in which is passed the returned $List < Completion >$ from **getCompletions** method.
- public **buildExploreRespons**(strPosition, targetAppID). Builds a com.fa.funnel.rest.model.ExploreResponse object according to the provided funnel and start/end dates. Furthermore, the parameter strPosition corresponds to the search location inside the funnel. If it is equal to 0, we are searching for preceding events, if it is less than the funnel's step count we are searching for intermediate events and if it is equal to the funnel step count we are searching for following events. This evaluation corresponds to the 'accurate' explore method. First the **getFollowUps** method is called with the given position and then the result, which is a $List < Proposition >$ is passed in a ExploreResponse object, which finally returned.
- public **buildExploreResponseFast**(strPosition, targetAppID). Returns a com.fa.funnel.rest.model.ExploreResponse object according to the provided funnel and start/end dates. Furthermore, the parameter strPosition corresponds to the search location inside the funnel. If it is equal to 0, we are searching for preceding events, if it is less than the funnel's step count we are searching for intermediate events and if it is equal to the funnel step count we are searching for following events. This evaluation corresponds to the 'fast' explore method. First calculates the lastCompletions with **getCompletionCountOfFullFunnel** and then this int is passed in the **getFollowUpsFast** method. The returned $List < Proposition >$ is added to the ExploreResponse and returned.

- public **buildExploreResponseHybrid**(strPosition, targetAppID, strTopK). Returns a link com.fa.funnel.rest.model.ExploreResponse object according to the provided funnel and start/end dates. Furthermore, the parameter strPosition corresponds to the search location inside the funnel. If it is equal to 0, we are searching for preceding events, if it is less than the funnel's step count we are searching for intermediate events and if it is equal to the funnel step count we are searching for following events. This evaluation corresponds to the 'hybrid' explore method where the initial results are evaluated using the 'fast' approach and then strTopK results are evaluated using the 'accurate' approach. First calculates the lastCompletions with **getCompletionCountOfFullFunnel** method, the resulted int is passed in the **getFollowUpsHybrid** method and then the results is passed to a ExploreResponse object, which finally returned.
- public **buildCSVFile**(path). Builds a CSV file according to the JSON information specified for the 'export_completions' endpoint. Uses **getFullRetroactiveCompletions** to get a $List < DetailedCompletion >$, the file is saved in path (parameter) and return its name.

private methods:

- **copyOriginalStepList**($List < Step >$). A a function that deep copies a list of steps into another.
- **generateAllSubqueries**($List < List < Step >>$) Creates all the smaller subqueries for this sequence.
- **getCompletionCountOfFullFunnel**($List < Step >$, startDate, endDate, maxDuration). First simplyfies the sequence of steps, in a way that there are no OR operations, so creates a list of lists of events. Then creates a **SequenceQueryExplorer** object. Takes one, by one the steps and create a Sequence. Then get all the tuples from **getQueryTuples()** method of the sequence. The last one will return a $List < QueryPair >$ object. For every QueryPair, we call **getCountForQueryPair** method of SequenceQueryExplorer object, keep the smaller number and add it. At the end return the sum of the smaller numbers.
- **getCompletions**($List < Step >$, startDate, endDate, maxDuration). Simplyfies and finds all subqueries with **simplifySequences and generateAllSubsequeries** methods. For every subquery, creates a **SequenceQueryEvaluator**, from wich first calls **evaluateQuery** to find the candidates and then **findTruePositives** to find the true positives. Then add the true positives in a List and returns them.