Appendix

Fig. 6: Final tagging algorithm.

Input: Some Text t, NLTK Brown Corpus Trained Tagger, T
Output: A list of tags extracted from t as $extractedTags$

$g =$ A list of regular expressions for grouping part of speech tags;
$blacklist =$ A list of characters to remove;
$segmenters =$ A list of sentence breaking characters;
$tokenisedSet =$ Tokenise t by white space;
$textSegments$, $segmentBuffer$, $extractedTags = \emptyset$;

foreach *word* w *in* $tokenisedSet$ **do**
  foreach *character* c *in* w **do**
    if *If* $c \in blacklist$ **then**
      Remove c from w;
    end
    else if *If* $c \in segmenters$ **then**
      Remove c from w;
      if *If* $segmentBuffer$ *is not empty* **then**
        Append $segmentBuffer$ as an element to $textSegments$;
        $segmentBuffer = \emptyset$;
      end
    end
    else
      break loop;
    end
  end
  Repeat above for characters of w in reverse from right to left;
  Append w to $segmentBuffer$;
end
Append $segmentBuffer$ as an element to $textSegments$;
foreach *segment* s *in* $textSegments$ **do**
  Tag each member of s with a part of speech classification using T;
  $parsedTree =$ Parse each tagged member of s against grammar g to create a tree;
  traverse node n in $parsedTree$
  if *Levenshtein edit-distance of* n *against all current members of* $extractedTags > 2$ **then**
    Append n to $extractedTags$;
  end
end
Return $extractedTags$;

Fig. 7: Finalised external APIs for related semantic content.

API	**Media Type**	**Content Retrieved**	**Quota**
Flickr (Yahoo! 2013)	Images	Public photos from flickr.photos.search endpoint	3600 requests per hour
Youtube (Google, Inc 2013d)	Videos	Public videos from youtube/v3/search endpoint	5,000,000 units per day
Yahoo Pipes (Yantsiou 2013)	News	Aggregated News from: • Google Blog Search • Google News • MSN News • Yahoo News • Yahoo Search Through pipe 'News Aggregator'.	200 requests per 10 minutes
Freebase (Google, Inc 2013a)	Entity Descriptions	Public titles, alias', descriptions and notable types of entities (e.g. people, objects, events etc) through the Search Overview and Topic API endpoints.	100,000 requests per day

Fig. 8: JavaScript libraries used for the experiment applications

Library	**Use in the project**
jQuery	Used as the framework for the experiment environment and independent application logic for facilitating faster and more stable development over traditional JavaScript.
jQuery UI	Used to facilitate 'draggable' sorting for time lines in the time line application
Hammer.js	Used to enable touch-gesture support in experiment applications.
jQuery Masonry	Used in tile wall application for aesthetics in placing tiles in appripriate positions according to their dynamic height.
spin.js	Used to replace an image based loading indicator for AJAX requests for speed optimisation.
jQuery Nicescroll	Used to replace minor aesthetic changes in the individual scrollbars of time linesin the time line application.

Fig. 9: Python libraries used for the experiment API platform and tagging algorithm implementation

Library	**Use in the project**
Python Twitter Tools	Used as a lightweight wrapper to the Twitter API for authentication calls and content calls.
Django	Used as the model-view-controller framework for integrating Python with a HTTP web server.
TastyPie	Used as an additional framework on top of Django for enable the creation of a REST web service as well as traditional HTTP web pages.
Natural Language Toolkit	Used as a framework to implement various generic natural language processing methods that are used in parts of the tagging algorithm.

Fig. 10: API platform authentication endpoints.

Endpoint	**Input Parameters**	**Output**	**Purpose**
/auth/login	'via' - The service to retrieve authentication request for e.g. Twitter. 'format' - The output format to return the content in e.g. XML or JSON.	For 'via=twitter' the output will be a URL to Twitter's OAuth authenticate URL with a unique token.	To facilitate a one-click sign-in to an authenticated service e.g Twitter.
/auth/validate	'type' - A reference to the service the credentials are being registered for. Relevant authentication variables given a service. For twitter these are 'oauth_token' and 'oauth_verifier'	The API key the credentials were registered to and a boolean field indicating a success or failure to do so.	A callback function to register a services authentication credentials (including finishing of OAuth dance if applicable) for a user with that user's API platform key.
/auth/authorise	'type' - A reference to the service the credentials are being registered for. 'apikey' - A user specific API key to register authentication credentials for a service to.	The API key of the user and the URL to finish authentication of a service for.	An endpoint to register credentials additional services for a user which already has an API key. e.g. retrieved an API key from /auth/login/ but did not finish registering or wants to register a service different other than Twitter (beyond project scope).

Fig. 19: Details of the personalised tile wall experiment application.

Item	**Description**
Design	The main element of this application is to provide Twitter connection influenced content in the form of tile blocks that are positioned for tightest fit. The content of these tile blocks include. Tweets from the authenticated user's home time line which have at least 5% keyword precision against the keywords of the authenticated user's personal time line. Additionally, any URL links and images in the authenticated user's home time line and news stories found from each Tweet's keywords that have at least 10% keyword precision in the title and description.
Libraries Used	• jQuery • jQuery Masonry
API Platform Endpoints Used	• /experimental/extendedTweets/ • /toolkit/substringTree/ • /semantics/news/
Pre-requisities for Use	Twitter account and authenticated API platform for use with account.

Fig. 20: Experiment details to supplement details in report

Item	**Description**
Environment Equality Measures	To reduce the possibility for debatable bias, all experiments were performed on the same device, the reasoning being that the graphical user interface adapts for different screen resolutions (for a better mobile experience).
Task Equality Measures	Additionally, for the time line application, the user was prevented with the same time line initially, the home time line with keywords from their personal time line affecting the ranking. For both applications, the tiles were generated from the same number of home time line and personal time tweets.

Fig. 11: API platform keyword and entity extraction endpoints.

Endpoint	Input Parameters	Output	Purpose
/toolkit/keywordExtraction	'q' - A query either containing text to tag or formatted in the form of $service : endpoint : parameters$ to retrieve the text to be tagged from the endpoint of a source such as Twitter given a set of parameters. 'apikey' - A user specific API key to lookup Twitter credentials for. 'format' - The output format to return the content in e.g. XML or JSON.	A list of tags.	To facilitate calling of the entire tagging operation.
/toolkit/entityExtraction	'q' - A query either containing text to tag or formatted in the form of $service : endpoint : parameters$ to retrieve the text to be tagged from the endpoint of a source such as Twitter given a set fo parameters. 'apikey' - A user specific API key to lookup Twitter credentials for. 'format' - The output format to return the content in e.g. XML or JSON. 'output' - A Freebase API field to return entity information if available.	A list of keywords and closest matched Freebase entity details if available for that keyword.	To facilitate retrieving additional semantic details such as alias' and descriptions about extracted keywords. Could also be described as applicable as a /semantics/ endpoint.

Fig. 12: API platform keyword and entity extraction endpoints. (Continued)

Endpoint	Input Parameters	Output	Purpose
/toolkit/substringTree	'q' - A query either containing text to tag or formatted in the form of $service : endpoint : parameters$ to retrieve the text to be tagged from the endpoint of a source such as Twitter given a set of parameters. 'apikey' - A user specific API key to lookup Twitter credentials for. 'format' - The output format to return the content in e.g. XML or JSON.	A tree structure of keywords where the roots are entities and sub-branches are keywords which are sub-strings of its parent. e.g. {'Cardiff University Student'} {'Cardiff University', 'Student'} {'Student'}	To reduce the number of requests that would need to be called to cover all related semantics in the content; i.e. may not need to query 'Student' if content for 'Cardiff University Student' was found.
/toolkit/stringClean	'q' - A piece of text to clean blacklisted characters from. 'blacklist' A list of characters wishing to be removed from the text. (Optional). Default {\/[('"# 'segmenters' A list of characters which cause the piece of text to be split into pieces. (Optional). Default @, . :;])}!? 'format' - The output format to return the content in e.g. XML or JSON.	A list of segments with blacklisted characters removed.	A sub-operation of the tagging algorithm.

Fig. 13: API platform keyword and entity extraction endpoints. (Continued 2)

Endpoint	Input Parameters	Output	Purpose
/toolkit/posTag	'q' - A comma separated or JSON encoded list of strings. 'format' - The output format to return the content in e.g. XML or JSON.	A list of tags and their part of speech classifications	A sub-operation of the tagging algorithm
/toolkit/posChunk	'q' - A JSON encoded list of ordered keywords and their part of speech classifications. 'type' - If set to 'chunks' (not by default) it will return keywords which contain more than one word, if omitted will return all keywords including those unchunked. 'format' - The output format to return the content in e.g. XML or JSON.	A list of tags either only containing multiple words or all tags.	A sub-operation of the tagging algorithm

Fig. 14: API platform semantic content endpoints.

Endpoint	Input Parameters	Output	Purpose
/semantics/youtube	'q' - A query to retrieve YouTube videos (e.g. related to a set of comma separated tags). 'format' - The output format to return the content in e.g. XML or JSON.	A list of public YouTube video details including the title, thumbnail url, video url, description and publish timestamp of each.	To facilitate the ability to find relevant semantic video content to a given set of tags.
/semantics/news	'q' - A query to retrieve aggregated news articles (e.g. related to a set of comma separated tags). 'format' - The output format to return the content in e.g. XML or JSON.	A list of news article details including the title, source url, description and publish timestamp of each.	To facilitate the ability to find relevant semantic news content to a given set of tags.
/semantics/flickr	'q' - A query to retrieve Flickr images (e.g. related to a set of comma separated tags). 'format' - The output format to return the content in e.g. XML or JSON.	A list of public Flickr image details including the title, thumbnail url, image url, description, Flickr tags, author and publish timestamp of each.	To facilitate the ability to find relevant semantic image content to a given set of tags.

Fig. 15: API platform Twitter-specific endpoints.

Endpoint	Input Parameters	Output	Purpose
/extendedTweets	'source' - The Twitter endpoint to retrieve Tweets from. 'apikey' - A user specific API key to lookup Twitter credentials for. 'precisionSets' - A list of either sources to retrieve tags from and/or lists of tags. Either comma separated or JSON encoded. (Optional) 'rank' A boolean value stating whether to rank the source Tweets or not. (Optional) Default is true. 'format' - The output format to return the content in e.g. XML or JSON.	A list of tweets split between a Tweet's original Twitter content and extended values derived from it in accordance to the input parameters sent.	To facilitate the ability to calculate values for ranking Tweets.
/twittertools	'function' - The Twitter endpoint to call using the Twitter Tools Library. 'apikey' - A user specific API key to lookup Twitter credentials for. 'format' - The output format to return the content in e.g. XML or JSON.	The response from Twitter wrapped around the content structure of the API Platform.	To facilitate the ability to call Twitter endpoints that do not return Tweet or authentication content e.g. user details.

Fig. 16: Linking personal account of other services to the API platform.

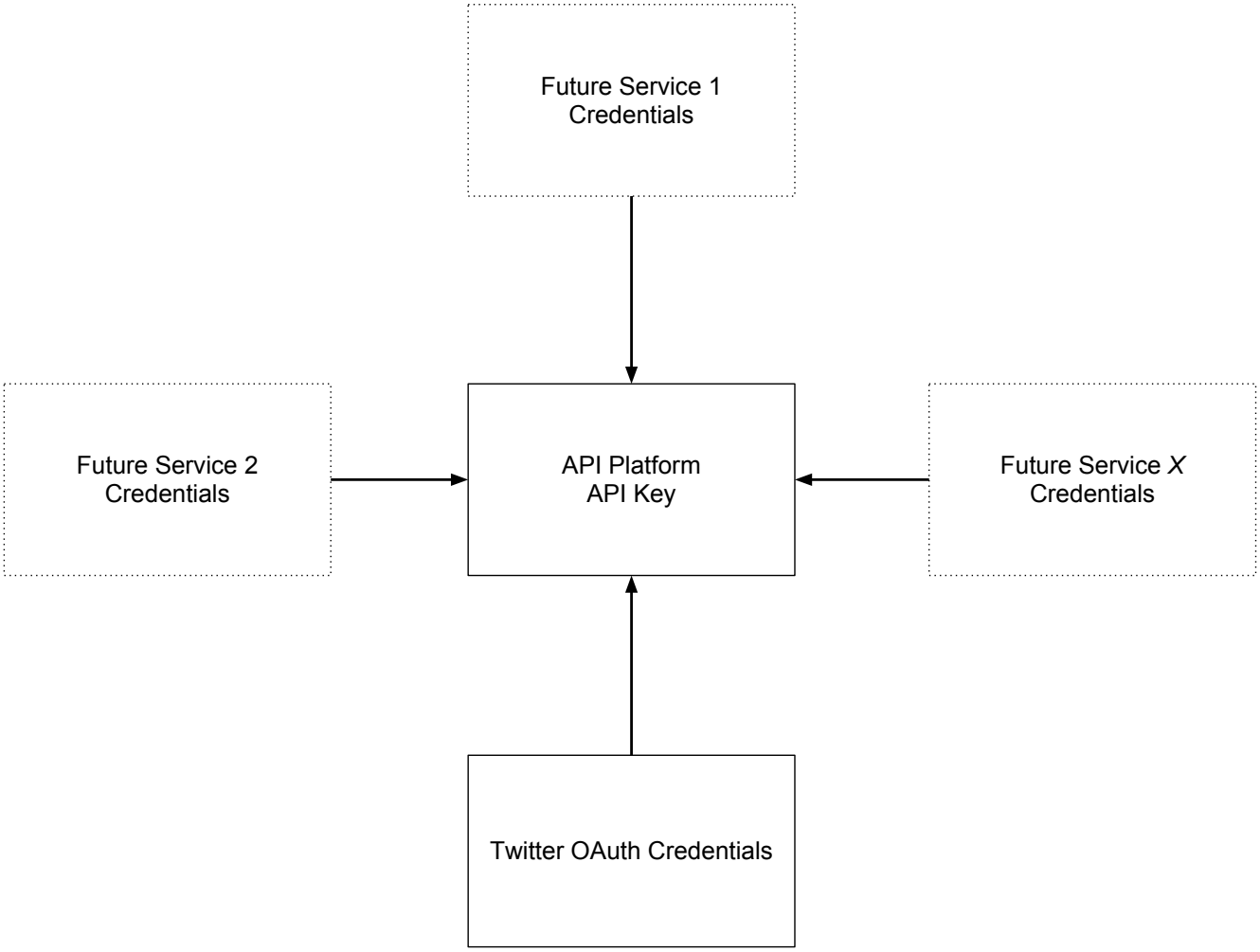


Fig. 17: Details of the application wrapper.

Item	**Description**
Purpose	The purpose of the application wrapper was to encompass both experiment applications together for ease of usability for the experiment participants.-
Concept	The application wrapper would have a 'landing page' showing details of the experiment environment and its applications. The second page would have a header for navigation back to the landing page, as well as a 'Sign in with Twitter' button for a single authentication process. This authentication would apply for both experiment applications so that the user did not have to authenticate for each. The body of the page would consist of a menu to the left hand side serving as navigation through the experiment applications and a window on the right which would hold the selected experiment application. Additional design features are the ability to toggle the experiment application window between full screen and the default window mode. The wrapper also has the capability to adapt its design appropriately based upon screen width, applicable from 'standard' 480px landscape smart phone through to tablets and netbooks through to 1920px desktop displays. Screenshots of the implementation can be found in the Appendix Figure 32, Weekly Supervisor Notes for Week 16/17.
Libraries Used	• jQuery • spin.js
API Platform Endpoints Used	• auth/validate/ • auth/login/
Pre-requisities for Use	None

Fig. 18: Details of the Twitter timeline based experiment application.

Item	**Description**
Design	Main element of the app being adding Twitter time lines as columns, based on a particular Twitter endpoint as the source of the Tweets. The content is then ranked by interest, then by integrity, then by time by default, although this is changeable on a per time line basis. The size of the times are the full height of the window they are in, with the time line being scrollable for hidden information. Tweet entities that are links are made into hyperlinks. Clicking on a tweet will expand the area below it and provide the user with a drop-down menu consisting of: Entities, Images, Videos and News. Clicking on a tweet with an expanded area will toggle the expanded area so that it can be shown or hidden freely. Selecting an option will load that particular content into the space below the drop-down menu in the expanded area. As multiple items of content for a particular medium is expected to be returned, the space is occupied by one of these items at a time. Performing a swipe gesture left-to-right or right-to-left on the area will then show next or previous items of content according to the gesture.
Libraries Used	• jQuery • spin.js • Hammer.js • jQuery Nicescroll
API Platform Endpoints Used	• /toolkit/substringTree • /extendedTweets/ • /semantics/youtube • /semantics/news • /semantics/flickr
Pre-requisities for Use	Twitter account and authenticated API platform for use with account.

Fig. 21: An example instance where multiple definitions of a word can result in incorrect context when the Freebase API performs generalisation to best fit a tag with an entry.

Fig. 22: Weekly Supervisor Notes for Week 12 - Page 1

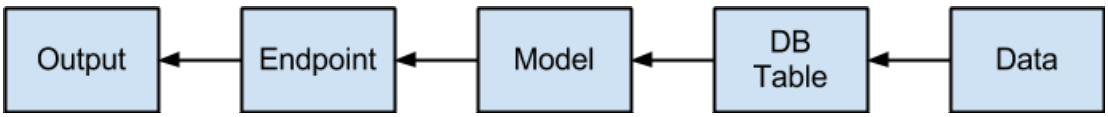
Project Meeting Notes - Week 12

Progress Summary:

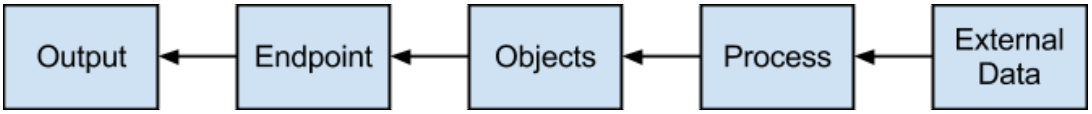
- Prototype implementation has been isolated to its own Git branch.

- Two new Git branches have been created for full implementation; one for unstable code builds and one for stable builds (as per the interim report).

- Creation of system infrastructure has begun with:
    - Local hosted PostGreSQL database created and synced into the locally hosted Django project.
    - A single Django 'App' for the api has been instantiated within the Django project.
    - A simple model used for initial testing with outputting instances of that model from the database with JSON.
    - (For full build notes see p3).
- The default setup for tastypie assumes the following:
    - Data is static and contained within object-relational mappings (ORM) between Django models and PostGreSQL tables.
    - The intention of the API is to output the data in a formatted paginated way.



- The project requirements are the following:
    - Data is not stored as instances of models in the database tables (with exception to user authentication details). I.e. The data used will be Non-ORM.
    - External Data does not equal desired output and is processed in real-time before output.



- Fortunately TastyPie allows for flexibility with non-ORM data sources and the above requirements should be implementable, although with a deeper and more conclusive understanding of the syntax and documentation than previously envisaged. Many automatic configurations such as fetching and filtering have to completed manually which increases flexibility but increases unstableness of code. A very basic mock up was implemented using Python dictionaries (which is what json is decoded to using most Twitter Python libraries). See build notes 0.0.5.

- The extra layer in the system model (i.e. the API) is likely to cause challenges regarding authentication with Twitter, particularly with the API only outputting formatted text (e.g. JSON). The internal processes should be able to be developed relatively independently until a solution is finalised.

- TastyPie by default allows output to be returned in JSON, XML and YAML, satisfying the desired feature stated in the interim report. Other outputs can be manually added if necessary, but this seems unnecessary given the project scope.

- Brief research into implementing backbone.js for a client beyond the interim report, predominantly being that the application could either be independent and therefore hosted remotely or could be tied into a Django App which would import backbone.js for internal functionality, hosted with the api.

Fig. 23: Weekly Supervisor Notes for Week 12 - Page 2

Focus Summary:

[Immediate Proposed Focus]

- On reflection of API tree structure defined in the interim report, the method produces redundant endpoints; where the same result could be achieved through different URIs. Changing the tree structure to be two clusters of endpoints would resolve this, as well as suit the TastyPie resource framework better and bring benefits such as increased simplicity and looser coupling. An example of this change would be:
    - Basic endpoint cluster - ../api
      /toolkit
      /stringfunctions
      /keyword_extraction
      /social
      /twitter
      /tweets
      /users
    - Complex endpoint cluster - ../api
      /discover
      /tweets
      /semantics

- Establishing a concrete, albeit barebone, API infrastructure by:
    - Establishing a basic template for custom TastyPie resources (API endpoints)
        - Override schema defaults regarding meta-data such as pagination, offsets etc.
        - Override schema defaults regarding all object meta-data such as individual URIs.
        - Set schema defaults such as response type unless overridden (JSON)
    - Create initial resources/endpoints from the template.

- Create Python classes for representing Tweets and Semantic Objects utilising Object-oriented modelling inheritance to ensure flexibility in additions/modifications.

[Further Focus]

- Draw up a plan for user authentication with initial focus on determining where authentication will be conducted either:
    - On the client with OAuth credentials finally passed to the client:
        - Increases flexibility and loose coupling.
        - Computational complexity affecting API server load is decreased.
        - However OAuth credentials would have be validated if 3rd party applications were made possible as the integrity of data would not be able to be determined.
    - On the server using Twitter's pin authentication:
        - Increased complexity between API and client for user authentication.
        - Potential hindrance on one-click login feature (interim report).
        - More complex to implement given implementation time.
        - Integrity of OAuth credentials would be ensured.
        - Allow for simpler authentication for clients that do not have easy to use libraries for Twitter OAuth (although there are few).
    - Both approaches after OAuth dance was completed would issue the client with a unique API key for that user (stored server-side) which would remain for whatever client the user was on, so whatever throttling measure was put in place would persist across all potential clients. API Key would be sent to distinguish user requests for each method call (either in header, by GET or by POST request).

Fig. 24: Weekly Supervisor Notes for Week 12 - Page 3

Build Notes: **(Most recent first)**

[Unstable Branch]

- - - - - -
Author: lt99399 <1029310@gmail.com>
Date: Mon Dec 24 14:28:46 2012 +0000

Build: 0.0.5
- Initial experimentation with a non-ORM API resource implemented, using 3 instances of a basic 2 attribute data structure.
- Assigning a value to the key 'b' in a HTTP GET request to the resource endpoint will form the value of the attribute 'name' of the 3rd instance object, if specified.

- - - - - -

Author: lt99399 <1029310@gmail.com>
Date: Thu Dec 20 17:44:02 2012 +0000

Build: 0.0.4
- Fixed an issue where the basic API model for initial implementation was not imported correctly in urls.py.
- Database connection parameters in settings.py updated following configuration changes for stable performance.
- Basic API access implemented with 2 instances of a basic model.

- - - - - -

Author: lt99399 <1029310@gmail.com>
Date: Wed Dec 19 16:32:29 2012 +0000

Build: 0.0.3
- Basic API model created for initial structure implementation.
- Unrequired files removed from project directory.
- Project url and view files cleaned from initial test implementations as further implementation will take place with files in the 'api' application.

- - - - - -

Author: lt99399 <1029310@gmail.com>
Date: Wed Dec 19 14:17:57 2012 +0000

Build: 0.0.2
- Django app 'api' created.

- - - - - -

Author: lt99399 <1029310@gmail.com>
Date: Mon Dec 17 14:57:38 2012 +0000

Build: 0.0.1
- PostGreSQL Database Created and Synced.

Fig. 25: Weekly Supervisor Notes for Week 13 - Page 1

Project Meeting Notes - Week 1 (13)

Progress Summary:

- Relatively slow due to exam period, but nothing outside of what was allocated in the timeplan.
- Created a ‘template’ API endpoint supporting read only operations through HTTP GET on entire collections of data only (lists).
- Response meta-data section of the section has been remodelled to remove references to redundant pagination.
- Created a ‘known issues’ template for early demonstration of the API using a dynamic non-ORM data source (comma-delimited CSV file) and using the template. Output below (manual indentations for readability):


```
{
	"content": [
		{
			"detail": "Pagination parameters are still active after removal from response meta
				element",
			"priority": 2
		},
		{
			"detail": "Responses formatted in XML have content elements labeled as object
				regardless of  contents",
			"priority": 2
		},
		{
			"detail": "Responses formatted in JSON are not indented",
			"priority": 1
		},
		{
			"detail": "Responses formatted in YAML are forces a download file always named
				'download'",
			"priority": 1
		}
	],
	"meta": {
		"http": 200,
		"results_count": 4,
		"timestamp": "Wed Jan 30 20:57:34 +0000 2013"
	}
}
```


Focus Summary:

[Review]

- Having taken time to reflect on the system implementation, I believe that as it stands it would be better to make the API endpoints more abstract and flexible. Therefore I propose that the client specific cluster of endpoints ‘discover’ be removed and the functionality merged with the more generic endpoints. With the right parameters the same result could be achieved.
- In regards to the client, the current designed nature uses the front-end client as merely a wrapper to display the JSON; merely anything more than just applying navigation and CSS. I propose that with the above point, that the demonstration nature of the client should be brought out more. This would be achieved by implementing prototypes of dissimilar individual applications; using different combinations of the APIs functionality to emphasise the APIs use as a platform.

- This may give the impression that this would increase the workload a great amount but all demo prototypes would use the same end-points; just with slightly different parameters to achieve potentially large differentiations between combinations. Some example ‘demos’ could include:

    - A simple traditional Twitter home timeline ranked by the NSGA-II prototype (both interest and integrity focused), rather than time exclusively.
    - A global tweet and relevant semantic mashup focusing on content with high calculated interest to the user. Could be displayed in a visual focused manner, similar to Flipboard’s tiles of varying sizes.
    - A Twitter feed taking aggregated user and home timeline keywords as a base for searching for most recent related news stories. The content of these news stories would then be used in a global tweet search, the results of which would form the stream; ranked by integrity.
    - A search function that would return a formatted wikipedia style semantically focused content, the content would only be shown if similar information was being tweeted globally.
    - An ‘endless’ global tweet stream ranked by NSGA-II. Firstly based on API provided values for keywords (based on timelines or global/local trends); interactions with the tweets would manipulate the keyword set in an attempt to provide a very basic mutation feature, purely client side. When new tweets are loaded, the mutated keyword set is sent and used for new global tweets, still ranked by NSGA-II. This would show that any keyword set can be used for endpoint flexibility. Clicking on a tweet could also ‘expand’ the area below to a certain length containing related semantic information.
    - A photo stream based on interesting images (tag based) that were similar to keywords extracted from a user's timeline and home timeline. Viewing a thumbnail to a full size image would also show brief semantic snippets of information related to the tags.

- The first stable implementation would aim to use my personal Twitter OAuth credentials so that API functionality is complete before, integration of individual user based access.

- Propose that each API response contain two blocks of elements, response meta-data (timestamp, http response code, amount of content data entries etc) and content data (e.g. tweets, semantic content links).

- In regards to the building and filtering of relevant semantic content, I propose that open linked data is utilised to minimise unrelated content being combined together (although it would not stop the issue of potentially incorrect context in the initial starting point selection). Potential services to use include Dbpedia, arguably one of the biggest sources of RDF content and links to other repositories; a REST API is available, but may require use of additional technologies such as SPARQL and heavy XML parsing. Another is the ‘Freebase’ API collection which in addition to RDF endpoints, also provides a percentage relevancy lookup to aid in getting the correct context and entity recognition and a lightweight API for getting a single short paragraph definition of an entity. 100,000 API calls allowed a day, the only potential negative is getting semantics for an entity requires knowing their unique id for an entity; however if the entity lookup is to be used then this information can be grabbed for the second call.

[Decisions]

- Should basic tweet interaction functionality be client side (like intended authorisation) or pass through the server through wrapper end-points?

- In regards the API endpoints, should a library be used to wrap endpoint urls around more user-friendly set of blackbox javascript functions for sending requests and parsing results into a fully compatible backbone.js dictionary format?

Fig. 27: Weekly Supervisor Notes for Week 13 - Page 3

Build Notes: **(Most recent first)**

[Unstable Branch]

- - - - - -

Author: lt99399 <1029310@gmail.com>
Date: Wed Jan 30 21:04:10 2013 +0000

Build: 0.0.7

- First demonstration prototype using the template developed, which will serve as the experimental endpoint for any future endpoint functionality upgrades, such as POST, PUT implementations. Involves:
  - Shows the known issues with the API in its current state, using two attributes one for detail of an issue and one for current priority in addressing.
  - Demonstrates use of API with a dynamic non-ORM data source, an external .csv file.
  - Contents of .csv file are processed after import in a Python class and directly passed through the API without any further hydration or dehydration TastyPie functions implemented.

- - - - - -

Author: lt99399 <1029310@gmail.com>
Date: Fri Jan 25 16:55:29 2013 +0000

Build: 0.0.6

- Template custom resource created for 'read' operations, incomplete but provides:
  - The 'objects' sub-element of 'response' has been renamed to 'content'.
  - Object sub-elements within 'content' no longer have the redundant 'resource_uri' sub-element.
  - Redundant sub-elements under the 'meta' element relating to pagination have been removed.
  - A timestamp sub-element has been added under the 'meta' element.
  - A placeholder sub-element for the HTTP response code has been added under the 'meta' element.
  - The sub-element of 'meta' denoting the number of objects within the 'content' element has been renamed to 'results_count'.

- - - - - -

Fig. 28: Weekly Supervisor Notes for Week 14 - Page 1

Project Meeting Notes - Week 2 (14)

Progress Summary:

- Implemented keyword extractor as a functioning API HTTP GET end-point. Minor code optimisation performed, however primarily regarding Python syntax; rather than runtime complexity in the algorithms.
- Processes have been split into several libraries and using object-oriented methodologies for modularity.
- Integration of a demo client using backbone.js will need further research, due to complexities in the non-ORM nature of the API response data.
- Felt that it would be better to have continual reading for integrating JS frameworks (such as backbone) and non-ORM TastyPie; and so continued on API endpoint functionality.
- Implemented further API endpoints surrounding keyword extraction, by allowing each notable action in the process being accessibly standalone; i.e. string cleaning, part-of-speech tagging, part-of-speech chunking etc. These endpoints are designed to be as flexible as possible to promote loose coupling, allowing for greater customisation in keyword extraction or in actions that make it. For example, character 'blacklists' can be overridden, segmentation can be overridden or removed; grammars can be overridden etc.
- It is noted that the practical use of each of these 'sub' endpoints varies greatly; however this is just to illustrate the 'platform' nature of the project.

Focus Summary:

[Immediate]

- Integrating prototype tweet ranking end-point.
- Experiment with freebase API for semantics.
- Perform brief testing on keyword extraction endpoints and push to first stable build (v0.1)

[On-Going]

- Continue to gain the knowledge for client implementation.
- Attempt to improve run-time complexities in libraries powering the currently implemented endpoints.

Fig. 29: Weekly Supervisor Notes for Week 14 - Page 2

Build Notes: **(Most recent first)**

[Unstable Branch]

- - - - - -

Author: lt99399 <1029310@gmail.com>
Date: Wed Feb 6 15:15:17 2013 +0000

Build: 0.0.9

- Implemented object-oriented approach to the keyword extraction API. Keywords are returned if request contains a HTTP GET variable named 'q' with a string for keyword extraction.
- Minor code optimisation implemented from prototype of keyword extraction.

- - - - - -

Author: lt99399 <1029310@gmail.com>
Date: Mon Feb 4 22:24:35 2013 +0000

Build: 0.0.8

- Prototype keyword extractor (using part-of-speech chunking) implemented as ready-only HTTP GET endpoint. Only response format parameter currently implemented.

- - - - - -

Fig. 30: Weekly Supervisor Notes for Week 15 - Page 1

Project Meeting Notes - Week 3 (15)

Progress Summary:

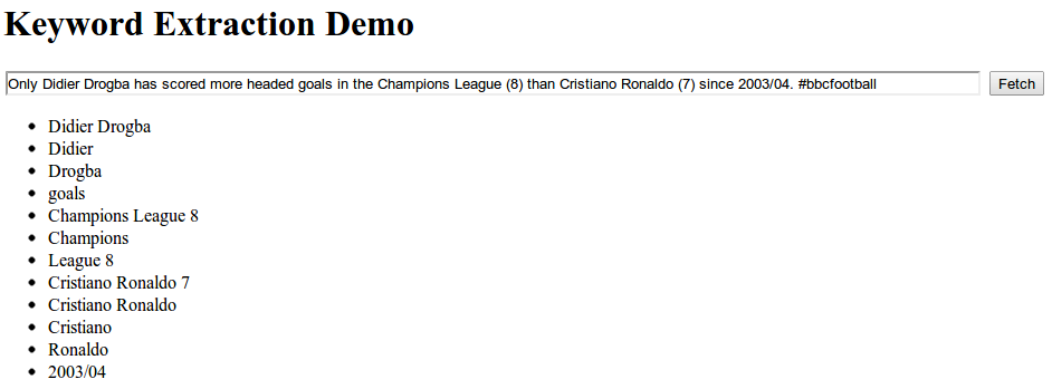
- Converted prototype libraries for extended tweet stats, interest/integrity scoring and NSGA-II ranking for similar modularity to the libraries for previous endpoint implementations. Some modules still contain prototype code for the time being, such as calculating string cosine similarity with numpy; but working as intended.

- The strict rules of non-ORM TastyPie data dehydration slowed progress towards implementing all endpoints and possible component endpoints for the extended tweet stats and ranking; due to poor documentation. However, issues regarding dynamic attributes and the datatypes of nested object attributes were resolved; a tidy up of the code will be necessary, potentially during stress testing of the endpoint in the near future.

- Investigated into individual user logins for Twitter specific endpoints and confident in the eventual ease of integration.

- Attempted implementation of a client for the keyword extraction endpoint using backbone.js. The non-ORM nature of data coupled with the redundancy of create, update and delete operations caused significant problems. Attempted to implement in a micro-MVC JavaScript framework instead, with the implementation being relatively successful, however with one particularly unavoidable drawbacks of being unable to work with nested JSON responses. Cannot be conclusive that it would not be possible as I do not have full competence in Backbone.js; but given time constraints and the push for novelty features, this was but on the back-burner for this week.

- Therefore, reflecting on the heavy nature of a JavaScript MVC framework, given the nature of its use being to display a collection of non-ORM models; this initial basic demo was implemented with an AJAX call using jQuery:



- The speed of the response after inputting a Tweet and clicking fetch was instantaneous. It would be very flexible to implement all clients in this manner, given that interaction with the server is only a GET request. However management of all clients in this manner (many of which will be more complex in parsing responses) while simple, could get messy.

Fig. 31: Weekly Supervisor Notes for Week 15 - Page 2

Focus Summary:

[Immediate]

- Create the barebones of the client, following agreed focus for JavaScript MVC implementation or not.
- Create libraries for gathering, processing and mashing up data for relevant semantics endpoints.
- Extend granularity of endpoints with all potential overridable parameters.

[Near Future]

- Tidy up endpoint resources that use libraries for extended tweet data.
- Optimise of various implemented libraries methods taken straight from the prototypes.

Fig. 32: Weekly Supervisor Notes for Week 16/17 - Page 1

Project Meeting Notes - Week 4,5 (16,17)

Progress Summary:

- Endpoints and libraries for YouTube videos, Flickr photos implemented.

- Endpoint for aggregated news implemented using a Yahoo 'Pipe' which searches through Google and Yahoo news among others to return unique results about a set of keywords (AND, OR seemingly available).

- Endpoint for semantics of an entity has been implemented a combination of endpoints part of the Freebase API. At the moment values returned are fields available for all results e.g name, alias', description, most recent image etc. The names of hierarchical branches the entity (if found) is classed under are extracted if they are not generalised terms for all entries. A sort of 'pre-extraction' keyword set. Currently, the API performs a search based on the input and the result with the highest relevancy score is chosen (this can be easily changed). API has the major limitation of only allowing 'AND' query parameters, but does perform some minor generalisation at nameand alias level. e.g. swim -> swimming, Anakin Skywalker -> Darth Vader.

- A new inheritance based model for API endpoints and data containers was created and all endpoints converted over too for better testing, maintainability, code redundancy minimisation and expandability of classes easier due to looser coupling.

- The extended tweet info/ranking API now accepts specially formatted string input that represents external APIs and their methods (and parameters) to call to retrieve information, in the form of:

    - [Prefix]:[Method]:[Parameters]
    - Where Prefix is the name of the service (e.g. Twitter), Method is the API endpoint of that service, and parameters are the parameters of that method to pass when called.

- These formatted strings are decoded, a OO 'factory' like class creates the relevant instance of a connection class (e.g. Twitter), the method is then checked against a list of 'shortcuts' (e.g. 'trends' instead of the actual path 'trends/place.json') and the request sent with the parameters. Relevant decoding is then performed depending on whether its getting tweets for the source or keyword extraction for the comparable precision sets.

- The Twitter API is now accessible through its own endpoint, by passing parameters in a similar manner to the above specially formatted strings. Any response is returned as an attribute of this APIs schema for conformity. This will allow for any read or write operation to be sent to twitter. (The process uses the TwitterTools library, the manner in which it works is a simple wrapper of creating relevant URLs, making it future proof to any changes; a similar method of doing this could be implemented for services without implemented APIs e.g. Flickr if time allows). Any parameters that are passed to the endpoint will also be passed with the twitter api call (this API specific parameters such as 'format' are removed) allowing for any optional Twitter parameter to be passed easily.

- Other endpoints such as the modular keyword extraction endpoints (just clean, just tag, just chunk etc) have been updated with additional optional parameters which override defaults such as: which characters to remove; which causes segmentations; a different regex grammar etc.

- A demonstration client has been implemented, with scalable/mutatable CSS based on screen pixel width, 480px and above supported (typical smartphone landscape or small tablet portrait). Consists of a

launch page and an apps list page based on the interim report designs. All graphical elements (e.g. backgrounds and icons) are done in pure CSS(3), no images or sprites exist, which removes full compatibility for older browsers but achieves much lower overhead in filesize. An app has been started which currently takes a term and aggregates semantic information and tweets into a dynamic single page (wikipedia inspired). Currently no personalisation exists, but could once semantic ranking has been finalised, could also implement localisation in wording if possible (e.g. pants, chips, football) if the user's country of residence could be determined.

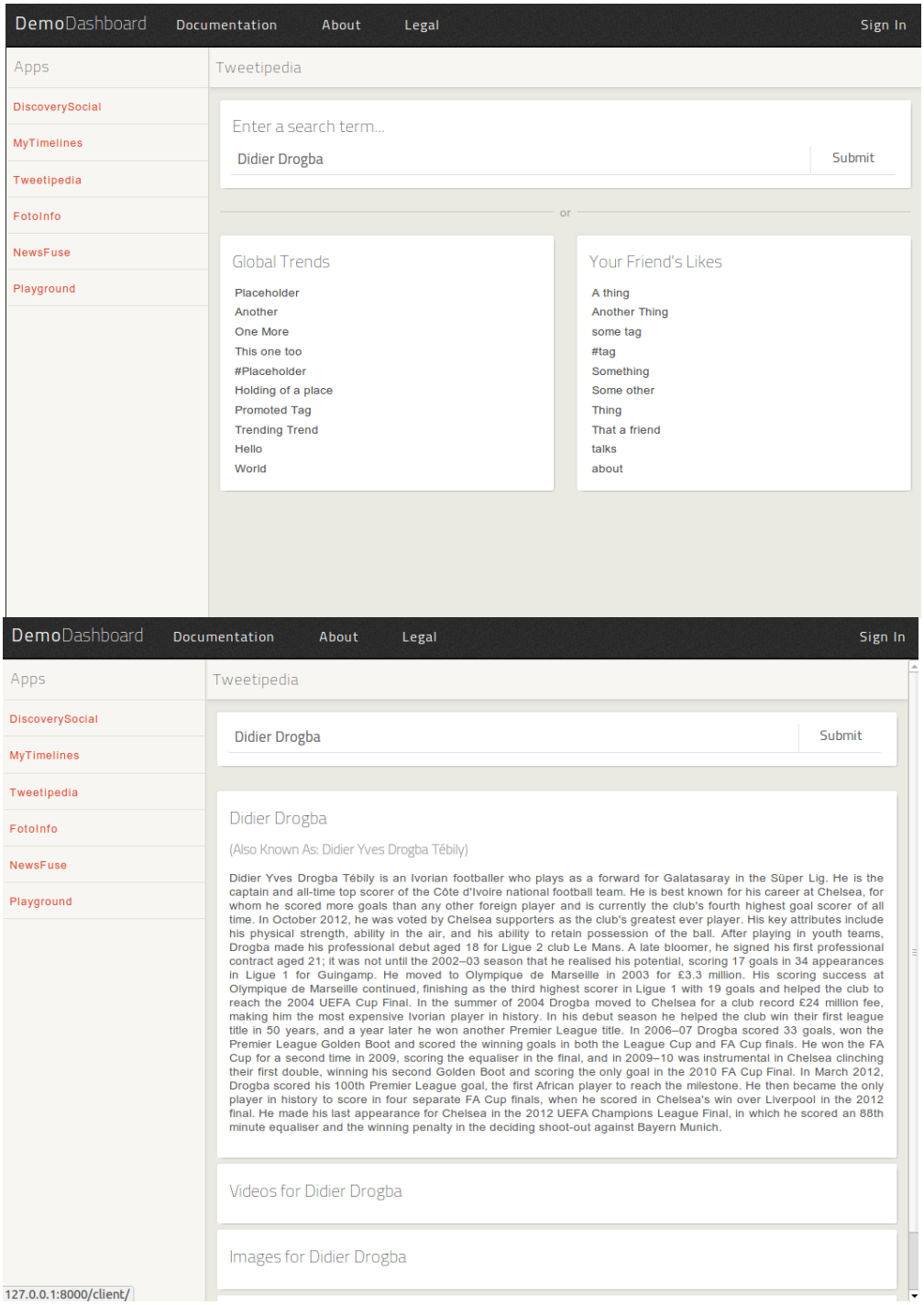
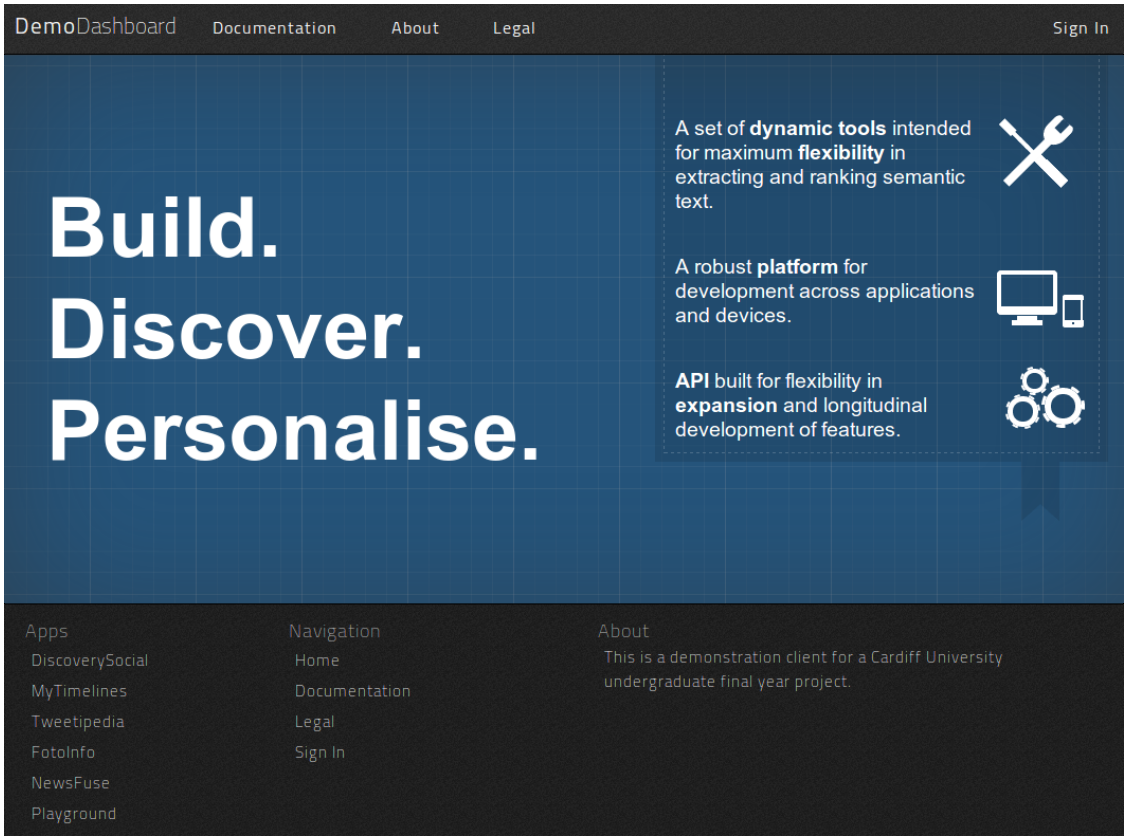


Fig. 34: Weekly Supervisor Notes for Week 16/17 - Page 3



Focus Summary:

[Immediate]

- Create solid focus for applications based upon those currently proposed.
    - ‘DiscoverySocial’ - An infographic single page application with interactive elements such as ‘Show me X’, showing statistics based upon the user’s Twitter.
    - ‘My Timelines’ - A traditional simple Twitter application with tweets ranked by NSGA-II rather than time, tweets could be ‘expandable’ to show brief semantic information based on the contents. An additional list of ‘current interests’ could be formed based on tweets the user interacts with the most, on refresh this is either added to the comparable precision sets or becomes parameters for a twitter public search to the ‘source’ tweets for the next list.
    - ‘Tweetipedia’ - An aggregation of tweets and semantic content for either a searchable topic or selectable current global trends /keywords from the user’s friends most recent posts.
    - ‘FotoInfo’ - An image wall created from keywords derived from user’s timeline, favorites and recent friend tweets. Clicking on the thumbnail would enlarge the image and provide brief semantic information relevant to the tags.
    - ‘News Fuse’ - A news wall ‘Flipboard’ style but aggregated for simplicity, based on keywords extracted from user timeline, ranked by relevance to friends recent posts and local trends. Related video, images and any links found in recent friends tweets which could be relevant would be shown on expansion of a news story.
    - ‘Playground’ - Similar to Google’s data visualisation playground for their graph/chart API, will contain a list of all accessible API endpoints in a pane, upon selecting one another pane will show all relevant and optional parameters to pass (like YouTube and Flickr endpoint experimentation) and a pane for the output.

- Need to decide on objectives for ranking different semantic data, the objectives should be generic but with potentially different calculations based upon the type of data e.g. videos, news etc.

[Near Future]

- The OO ‘factory’ class which creates instances of connections to external API services could be an appropriate way to handle authorisation. For example, if a user of the API was given a unique API key, such user could register oauth credentials of any number of external APIs including but not limited to Twitter. Upon detection of one of the specially formatted string as highlighted above in an endpoint, before creation of an instance, the API key could be looked up with the ‘prefix’ and if the user has registered their OAuth credentials then an instance could be created, otherwise not. This could allow for a move away from a solid Twitter backbone to any external API (if appropriate classes and methods were implemented), focuses on the notion of a platform. Other APIs would probably be out of scope, but the facilitation of this at the moment is not too far off.

Build Notes: (Most recent first)

[Unstable Branch]

- - - - - -

Author: lt99399 <1029310@gmail.com>
Date: Wed Feb 27 19:34:46 2013 +0000

Build: 0.1.7
- Integrated extended tweet information and tweet ranking endpoint with new inheritance structure to be more in line with the semantic and keyword endpoints.
- Tweet extended info/ranking endpoint now has the following features:
  - Some parameters now accept string input in the form of:
    - [prefix]:[method]:[parameters (optional)]
    - Where prefix is a string representation of an external service e.g. twitter, method is the api method string and parameters is the set of parameters for that api method.
    - The string is decoded by a 'factory' class creating an instance of a connection to the relevant API, the method call and parameters are then formatted and sent.
  - The required 'source' parameter now takes the string input above in addition to a json encoded document of tweets.
  - The optional 'comparableSets' parameter now accepts a json encoded string containing a single list of either/a mix of:
    - Dynamic string input (as above).
    - A comma delimited string
  - The endpoint now takes an optional parameter:
    - 'ranking', a boolean parameter which when set to true (by default) performs NSGA2 ranking on a currently pre-defined set of objectives.
- The Twitter connection library class now allows for any Twitter API endpoint to be accessed through an API endpoint, with the response formatted back into this API's schema.
- The demonstration client UI has been updated, including the introduction of a informative launch screen.
- The demonstration client will now dynamically scale/alter for screen pixel widths of 480px and higher.
- An app for the demonstration client has been started 'Tweetipedia', which takes input keyword(s) and returns aggregated semantic information, currently unranked.
- Formatting of keyword extraction library to TastyPie compatible objects now takes place in the endpoint, to allow for lower overhead in the parsing of comparable sets in the extended tweet information API.

- - - - - -
Author: lt99399 <1029310@gmail.com>
Date: Sat Feb 23 14:11:28 2013 +0000

Build: 0.1.6
- Implemented a library and endpoint for entity information using the Freebase API.

- - - - - -
Author: lt99399 <1029310@gmail.com>
Date: Fri Feb 22 13:48:15 2013 +0000

Build: 0.1.5
- Part-of-speech chunker library method now accepts an additional parameter:
  - 'grammar' - converts a list of regular expressions into a grammar for the nltk regular expression chunk parser.
- The demonstration client homepage now has updated CSS.

- - - - - -

Author: lt99399 <1029310@gmail.com>
Date: Thu Feb 21 14:06:13 2013 +0000

Build: 0.1.4
- Keyword extraction endpoint library now follows new standardised object structure.
- String cleaning, keyword part-of-speech tagging and part-of-speech chunking endpoints now follows new standardised object structure.
- String cleaning endpoint now accepts additional optional parameters:
  - 'blacklist' - a string containing a list of characters to remove at the prefix or suffix of each word in the string; must be specified to override default. Declaring but not giving a value will r
  - 'segmenters' - a string containing a list of characters to cause a segmentation break; must be specified to override default. Declaring but not giving a value will remove segmentation.
- Part-of-speech chunking endpoint now accepts an additional optional parameter:
  - 'type' - When declared and set to 'chunks' will return only strings containing multiple words, any other value will default to return all strings considered 'keywords' by the grammar.

- - - - - -

Author: lt99399 <1029310@gmail.com>
Date: Wed Feb 20 14:11:49 2013 +0000

Build: 0.1.3
- Bare-bones client implemented using Django template inheritance, smartphone friendly for CSS3 compatible browsers.
- New libraries and endpoints added for retrievable of related: YouTube, Flickr and aggregated news articles to a set of comma-delimited, or JSON encode list of, keywords.
- New python object structure for API implemented for better code management and standardisation across endpoint types, new endpoints currently using it only.

- - - - - -

Project Meeting Notes - Week 6,7 (18,19)

Progress Summary:

- Progress somewhat slow due to heavy coursework load, however the final pieces of the project have been implemented to some degree.

- API 'Playground' app implemented, full API endpoint list not available at the moment but framework is in place, only need to specify the text boxes to draw for each parameter.
    - Output currently only available in indented JSON, need to look into a lightweight XML indent JS library (or create one if time is available).

- Django models created for user account and extended models for service specific credentials (e.g. Twitter OAuth token and secret).

- API endpoints added for creating a user account (retrieving an API key), registering credentials from a service with that API Key and logging in via those credentials . The structure is so that you would be able to authenticate with any service once and then get access to all services you have registered credentials for.

Focus Summary:

[Immediate]

- Final semester week before break focused on tying up loose ends and beginning testing.

- Propose to change scope regarding client apps to include less, but more substantial and diverse apps, current proposal:
    - Traditional Twitter client with extended features such as ranking and semantic snippets on tweet 'expansion'.
    - API Playground, already implemented and would facilitate demos of any endpoints not covered by the other apps.
    - TweetCategoriser: Would aim to provide similar functionality as 'TweetDeck' i.e. tweets organised by category. However, would use keyword extraction and 'type-of' terms from entity extraction to group tweets autonomously. E.g. Tweets about 'Didier Drogba' and 'Cristiano Ronaldo' would be under a 'Football' section and a 'Sports' and an 'Athletes' section. Tweets could then be ranked within their own sections to integrity or interest or both.
    - GraphSearch: Ambitious final client and would require definite scope, but would essentially provide some similar functionality to Facebook graph search, but using this API as a platform. Would extend the already implemented content aggregator for a particular entity. Would involve simple queries that would encourage experimentations with. E.g. Focusing on all or a subset of: What, Why, Where and Who
        - *'Why* could *this* be trending?' -> News stories
        - *'What* is the Harlem Shake?' -> Content aggregator (e.g. text, images, videos, news).
        - *'What* are the current *trends amongst* my *friends*?' -> Terms most commonly talked about in home timeline.
        - *'What topics* are my *friends* talking about that *I would like*?' -> keywords and 'type of' classes for these keywords e.g. 'Drogba' -> Footballer therefore talking about Football or Sports.
        - *'What news* would interest me, *based upon* what my *friends* are talking about? - News stories based upon extracted keywords of home timeline.

Fig. 39: Weekly Supervisor Notes for Week 18/19 - Page 2

    - *'What* can me and *this person* talk about? -> 'Classic' Google Glasses example - List of topics containing both extracted keywords and 'type of' classes from those keywords, as well as news and videos relating to them.

[Near Future]

- Final report writing slated for the first two weeks of the Easter break, allowing the final weeks for further evaluation to bulk the report; any bug fixes and implementing more features to the client apps if time allows.

Build Notes: (Most recent first)

[Unstable Branch]

- - - - - -

Author: lt99399 <1029310@gmail.com>
Date: Thu Mar 14 20:04:48 2013 +0000

Build: Chassk 0.1.10

- Demonstration client now has a 'playground' app implemented , which provides a three pane interface for: choosing an endpoint; adding parameters to a query of that endpoint and getting the response when
- Django models have been added for user accounts, which follows a inheritance tree architecture for user credentials for external user services and subsequently for authorisation to endpoints using those
- An API endpoint has been added for registering as a user and recieving an API Key.
- An API endpoint has been added for signing in and authenticating to external services (currently only Twitter implemented). Returns a user-specific API-key to be sent as a parameter for access to other e

- - - - - -
Author: lt99399 <1029310@gmail.com>
Date: Fri Mar 1 15:49:56 2013 +0000

Build: 0.1.9

- Entities parsed from the Freebase API are now given the notable type 'Unknown' if not provided in the response.
- The most notable type attribute has been added to the entity_information API endpoint response (previously not added to output).
- The most notable type attribute has been added to the semantic section of the 'Tweetipedia' app response.
- Alias' of an entity now appear next to the default name on the semantic section of the 'Tweetipedia' app response.

- - - - - -
Author: lt99399 <1029310@gmail.com>
Date: Fri Mar 1 15:25:35 2013 +0000

Build: 0.1.8

- Updated Tweetipedia 'app' to include YouTube, News and Flickr endpoints, loading performed with asynchronously with AJAX.
- jQuery plugin for a CSS3 powered loading spinner implemented to show loading for AJAX calls.
- Flickr library updated to include a new attribute 'thumbnail' as well as the full size image 'url'.
- News Yahoo pipe library updated to also remove any HTML tags found in article titles.

For the final weeks, written meeting notes was replaced by either: emails following supervisor availability clashes or visual updates through demonstrations of the implementations.