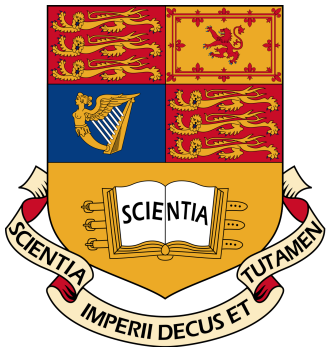


Goldman Sachs

C362 - Software Engineering Project

Aspect-Based Sentiment Analysis

Imperial College London
The Goldman Sachs Group, Inc.

Authors:
Harry Brown
Shashwat Dalal
Marcel Kenlay
Ilyas Saltykov
Lewis Treacy
Thomas Yung

Imperial College Supervisor:
Dr. Anandha Gopalan
Goldman Sachs Supervisor:
Michael Carneciu

Autumn 2018

Acknowledgments

We would like to thank our supervisor, Dr. Anandha Gopalan, for his invaluable guidance and support throughout the sprints. We would also like to thank GSAM's engineers, in particular Michael Carneciu, Adam Atkins and Abhischek Thottakara, for inviting us to see first-hand the function of engineering in an asset management business, and for making the project such an enjoyable experience.

Executive Summary

Goldman Sachs' Asset Management line of business, GSAM, is constantly looking for novel investment opportunities to help achieve high returns on their clients' investments. As such, internal analysts and external researchers alike produce vast sums of research about their investment ideas for a portfolio manager to later either exercise or dismiss.

GSAM believes it can leverage the recent advances in natural language processing to streamline the large quantity of research — which can exceed a dozen pitch-books an hour — that a portfolio manager is presented with. GSAM has been working on a search and analytics platform which catalogues and organises their knowledge repository. Portfolio managers can already look up research pertaining to a certain security, or research produced by a particular individual. GSAM would like to add aspect-based sentiment analysis, assigning a sentiment score for every entity mentioned in a document, as an additional feature to this research platform so that analysts and managers can look "for data to support or refute [their] investment thesis regarding an opportunity". This is particularly useful as "evaluating the sentiment trend from a set of reliable authors/publishers/sources for a particular company or industry provides an additional data point to strengthen [their] analysis".

Our product successfully implements an aspect-based sentiment analysis pipeline which takes a document and outputs a graph representing the sentiment of named entities in the document, as well the semantic relation between them. Our product builds upon existing NLP vendor products and performs sentiment aggregation based on contextual aspects of a named entity, as well as aggregation on semantic relationships between named entities. These additional aggregation techniques produce a more refined sentiment score for named entities.

The relationships between entities within a document, along with the sentiment directed at each, create a visual summary of the article. Also, by highlighting noteworthy articles that deviate from the 'trend' for a topic, it allows potential analysts to stay alerted to information they might otherwise miss. As such, our product acts as a tool to make the consumption of written media quicker and will hopefully give GSAM an edge in managing assets.

This core functionality that might one day be integrated into GSAM's analytics platform, and then also the ability to train our unsupervised aspect model, are both wrapped in a web app for demonstration purposes.

Contents

1 Introduction

1.1 Project Specification

Goldman Sachs' Asset Management division, GSAM, had initially provided us with a project specification they said should be used as a guideline to help steer the direction of the project. That being said, they were open to any approach we wanted to take, as they were interested to see what was possible to achieve as a proof of concept for the use of natural language processing in asset management. The main aims of the project are outlined below:

What you should build: The GSAM Engineering team wants to find and aggregate sentiment on entities mentioned within documents or aspects of them. Given an opinionated text about a/multiple target entities, identify all the opinion tuples with the following information:

1. Identification of entity/attribute pairs towards which an opinion is expressed in a given sentence.
2. Extraction of the linguistic expression used in the text to refer to the Entity—Attribute pair. (Sentence/Evidence that describes the sentiment of the recognized pair).
3. Given a document about a/multiple target entities the goal is to identify a set of entity—polarity tuples that summarize the opinions expressed in the review. These tuples can be weighted in relation to their importance of the sentiment expression.
4. Given a set of entity—polarity tuples in the document then establish the aggregate polarity of this entity across all documents in the dataset.
5. Propose a measure to objectively evaluate the performance of your model.

1.2 Proposed Solution

From the outlined specification, the project could either be focused on building and training a model that would associate sentiment to entity mentions, or instead be focused on sentiment aggregation, for gathering more refined sentiment scores once an initial sentiment is calculated for entities and aspects. Thus, to clarify the direction we took for our product, we decided it would first help to clarify what our product does *not* do.

Definition 1.1. *Entity* is a noun-phrase: "a phrase that has a noun (or indefinite pronoun) as its head or performs the same grammatical function as such a phrase" [1]

Our product does *not* tackle the problem of associating a sentiment score within a document for all mentions of entities. We believed re-engineering this was not an efficient use of our time as many vendors, such as *Google Cloud NLP*, *IBM Watson*, and *Amazon Comprehend*, provide accurate models as a service for this entity-sentiment pair association. [2] [3] [4]

What our product does do is aggregate sentiment across entities to produce refined sentiment scores for *named-entities*. The described problem of associating a sentiment-score

for all mentions of entities, is nonetheless still a prerequisite for our work on sentiment aggregation.

We break down sentiment aggregation on two levels:

1. Named Entity — Aspect Relationships (to be referred to as NE-A relationships)
2. Named Entity — Named Entity Relationships (to be referred to as NE-NE relationships)

When receiving the sentiment of entities from our vendor product of choice, we distinguish a named-entities from aspects based on the following definitions.

Definition 1.2. *Named-Entity* is an entity that has a *Wikipedia* reference.

Definition 1.3. *Aspect* is an entity that does not have a *Wikipedia* reference.

We found these simple definitions worked well as non-proper nouns (often non-capitalised nouns) such as battery and screen, usually yield ambiguous *Wikipedia* searches whilst proper nouns (usually capitalised nouns) such as *Apple Inc.*, usually yield unambiguous *Wikipedia* searches. [6]

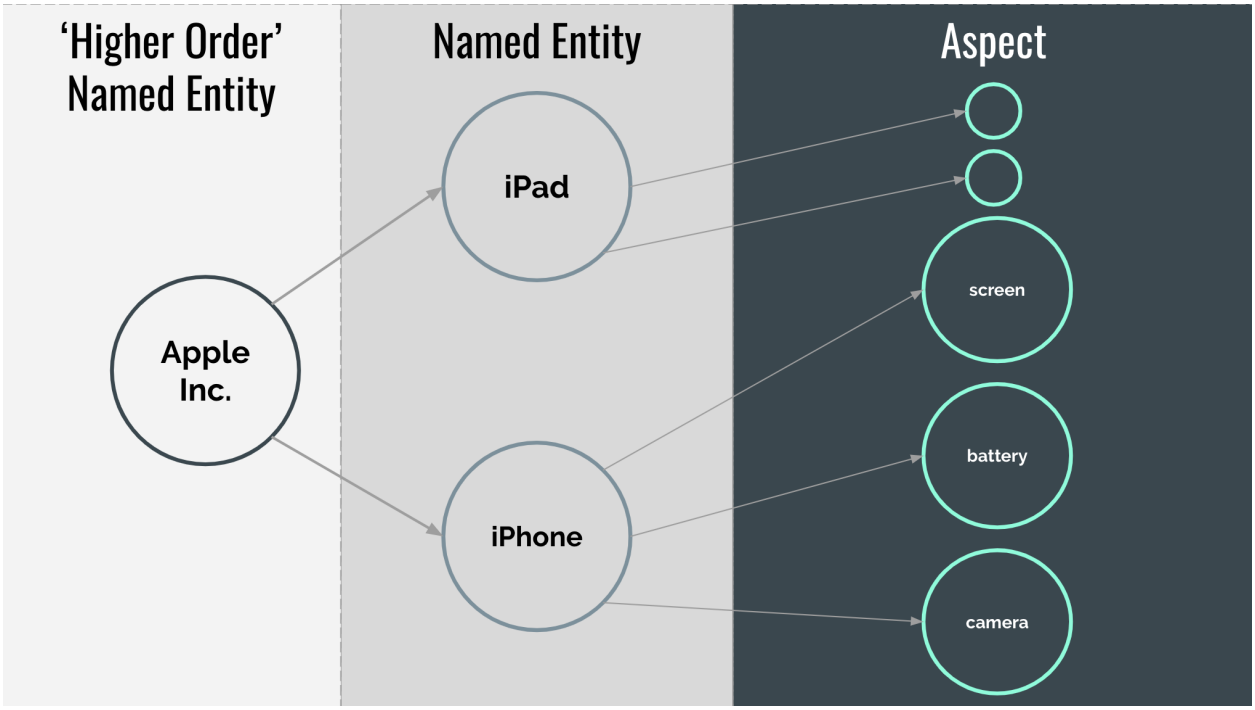


Figure 1: Definitions. Note *iPhone* and *iPad* are examples of non-capitalised proper nouns

Looking at figure 1, NE-A aggregation works by updating the sentiment of named entities based on associated aspects. For example in this case, aspects of the *iPhone* include screen, battery and camera.

NE-NE aggregation works by updating the sentiment of 'higher-order' named entities based on other named entities that influence it. In the figure above, *Apple Inc.* is the 'higher-order' named entity whose sentiment is influenced by both *iPad* and *iPhone*.

To highlight the importance of both these levels of aggregation it helps to think about how a human might associate sentiment for the following sentences.

- *"Apple has revealed the new iPhone XS. On the inside just about everything has changed, including an improved faster processor."*

  For this sentence, a naive aspect based sentiment analysis product would return a positive-sentiment for the entity of 'processor'. A human reader, however, would infer that the processor is belonging to the product *iPhone XS*, i.e. is an aspect of *iPhone XS*, and the positive sentiment should be associated with *iPhone XS*. This is the refinement the NE-A aggregation step aims to perform.

- *"The FAANG stocks fell into a bear market last week, each declining more than 20 percent from recent highs."*

  From just the knowledge within the text, it is difficult for any purely sentiment-analysis model to associate *Facebook*, *Apple Inc.*, *Amazon*, *Netflix*, and *Google* to *FAANG*.[5] Nonetheless, a human reader would associate a negative sentiment for the stocks that constitute *FAANG*. Hence, external knowledge of real-world relationships need to be used to establish this 'influenced by' relationship which in-turn is used to update the sentiment for these 'higher-order' named entities. This is what NE-NE aggregation aims to achieve.

1.3 Main Achievements

- Using a probabilistic machine learning model, attribute aspects to named entities and further propagate the sentiment of these aspects to the named entity using learned weights
- Build a persistent model of the world, considering named entities and their relationships to one another. Similarly, sentiment is propagated to other entities using the associated weight
- Track how our refined sentiment for an entity changes over time, using this information to flag articles which deviate from the previous sentiment of any contained entity
- Display our findings on a web-UI along with an interface to perform validation for the machine learning model

2 Project Management

2.1 Agile Methodology

Given the bi-weekly milestones and previous experiences, we felt it logical to employ the scrum development methodology with two-week sprints for this project. In order to make full use of this methodology, the group project leader, Harry Brown, was nominated as the scrum master, and Shashwat Dalal was assigned the role of product manager.

Harry's role was to organise internal meetings, such as stand-ups, to ensure that the backlog was up-to-date, and to ensure that work was distributed evenly between group members for each sprint. On the other hand, Shashwat Dalal kept the stakeholders updated, and made sure there was full transparency in the progress we were making, whilst ensuring we kept their requirements in mind.

2.2 Meetings

2.2.1 Group Meetings

One aspect of the scrum methodology we followed rigorously was the 'daily' stand-up which occurred three times a week. The brevity of the stand-up encouraged members to be concise yet precise about their progress and blockers. The use of *Azure Boards* (see 2.3.2) not only allowed us to easily structure our stand-ups, but also allowed us to efficiently document changes to the sprint plan.

Alongside the short stand-up meetings, we also held longer sprint meetings. This is discussed in depth in 2.3.2.

2.2.2 Client Meetings

Every two weeks, we would spend a couple of hours with GSAM's engineering team in their offices. This was a great way for us to understand the stakeholder's perspective in developing the product whilst providing us with an opportunity to receive feedback about what we had delivered after each sprint. Additionally, during our final visit to the offices we delivered a 30-minute presentation to the team about the work we had produced and our future plans.

2.2.3 Supervisor Meetings

As well as our visits to GSAM's offices, each sprint concluded with a meeting with our supervisor, Dr. Anadha Gopalan. Dr. Gopalan helped us reflect on the previous sprint and focus our efforts on features that would be most beneficial for the forthcoming sprint.

2.3 Development Cycle

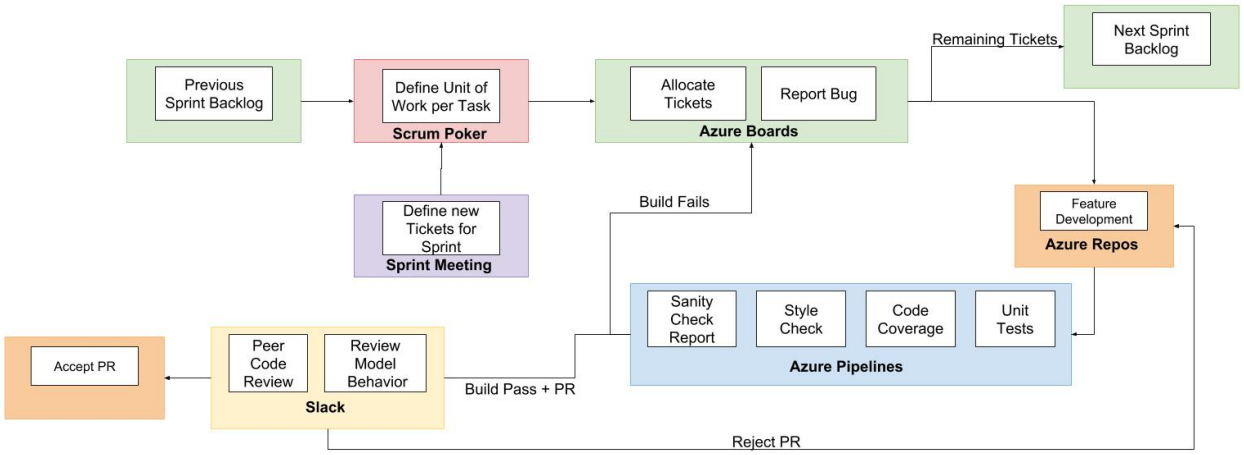
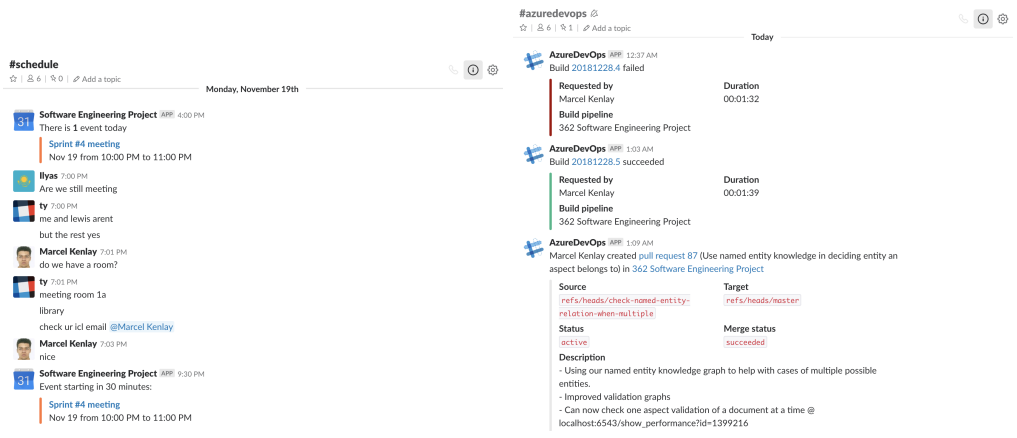


Figure 2: Development Cycle and Tools Used

Figure 2 encapsulates our process of development, and the following section aims to demonstrate the motivation for each stage of the process.

2.3.1 Communication Channels

Even though we tried to meet as regularly as possible, most of the development was done independently or in pairs. As such, communication and transparency in the group was highly important. We used *Slack* as our medium of communication. The myriad of app interactions, as show in figure 3, also allowed us to use various *Slack* channels for planning meetings and requesting peer-reviews for pull-requests. Additionally, we were able to invite some of GSAM's engineers onto a *Slack* channel which allowed us to quickly ask them any questions we might have had.



(a) Google Calender (b) Azure DevOps

Figure 3: Slack Integrations

2.3.2 Sprint Planning

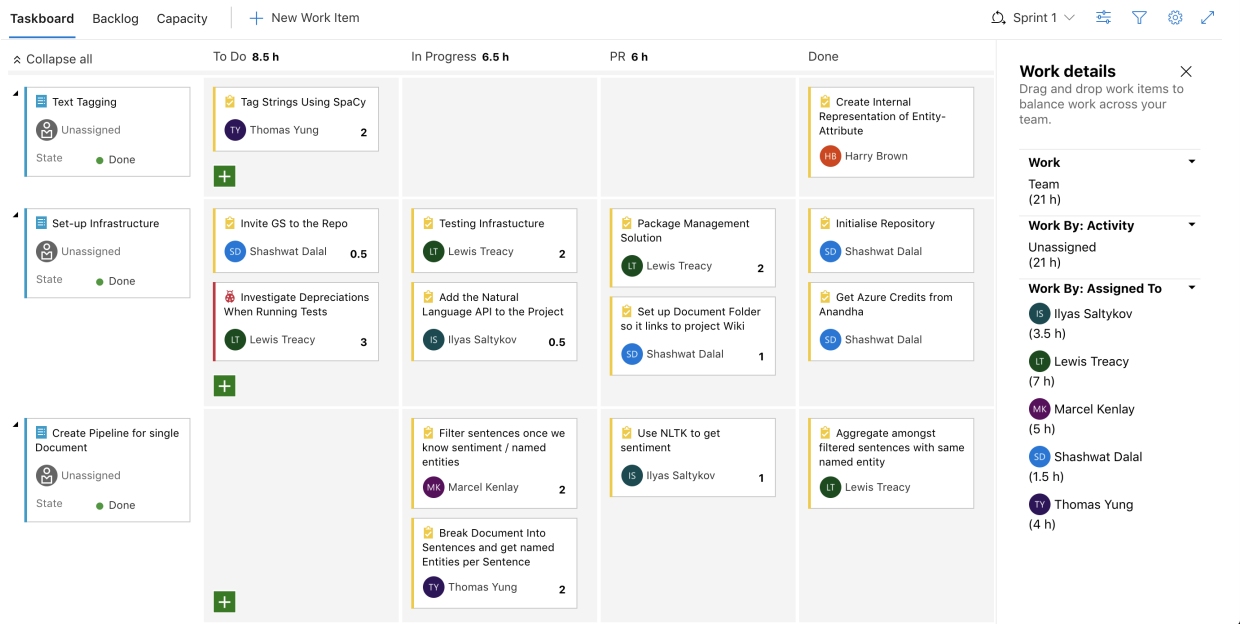


Figure 4: Snapshot of *Azure Boards* for Sprint 1

At the beginning of each sprint, we held a longer meeting to scope out the sprint. As shown in figure 2, we would create user stories and respective tasks based on the backlog of tickets that were not completed in the previous sprint, as well as new tickets based on the current sprint plan. After defining the user stories and tasks on *Azure Boards* (figure 4), we used 'scrum poker' as a tool to aid quantifying the amount of value we were going to produce for the coming two weeks. It helped us gauge how much work to assign to a sprint. Should the project have gone on for further sprints, this metric for work produced would allow us to calculate team-momentum — a comparative measure of our weekly output against projected output.

The discussions that were invited as a result of using 'scrum poker' also proved to be extremely useful as they often revealed a quicker way to solve a problem or a potential blocker that lay ahead. It also allowed members to show interest in certain pieces of work which made work allocation easier.

2.3.3 Continuous Integration

After our initial software engineering consultation workshop with Dr. Robert Chatley, we decided to get a naive pipeline working before iteratively improve the parts of our pipeline. As much of the product we were engineering was a model, we did not have the luxury of having a binary pass-fail test suite for all parts of the product. Instead, to ensure that each subsequent iteration of the project was indeed improving the model, our CI pipeline automatically generated a model-behaviour report. In addition to a peer code-review, the group discussed the generated report on *Slack* before accepting a pull request.

The report, an example attached in *Appendix A*, included the following sections:

1. Graph and Summary
2. Decisive Mentions in Text
3. Change of Sentiment after Aggregation
4. Accuracy and Loss Before and After Aggregation

In addition to model-behavior report we had automated tests, for parts of our product that could be unit-tested, that had to pass as a pre-condition for a pull-request. This allowed us to be confident that our master branch was always functionally correct.

2.4 Milestone Recap

2.4.1 Sprint 1

Aim: By the end of the first sprint, we wanted to have a basic algorithm that, for any input document, extracted any mentioned entities and their corresponding attributes. The attributes should have been assigned a polarity between +1 and -1. We knew a large portion of this sprint would be geared towards infrastructure and set-up and therefore kept this in mind during planning.

Outcome: We successfully met this goal, and after a meeting with Goldman Sachs we refocused our targets for the next sprints.

2.4.2 Sprint 2

Aim: The main goal of our second sprint was to begin storing the results of running our algorithm on a document in a database. Then, when running the algorithm subsequently, we should have been able to use the data we had stored to calculate prior aggregates to compare the input document to. We had also seen the value of being able to validate the performance of our algorithm, therefore we planned to develop some sort of UI to make validation easier for us. This UI would allow someone to mark areas of sentiment within a document, and then see how running our algorithm on the document compared to this. A final focus of the sprint was to create links between named entities we find appearing together in documents often.

Outcome: As planned, the results of sending a document through the pipeline were successfully stored in a database. We also had been able to make the components to recognise and assign sentiment for entities mentioned in provided documents, build relationships (with associated weights) between different named entities, and build relationships between named entities and their aspects (something originally planned for the third sprint). Along with this, we had a simple web UI which allowed us to easily evaluate the performance of our system.

2.4.3 Sprint 3

Aim: We aimed to integrate the new features into our existing pipeline, and to make iterative improvements to the current pipeline components to improve both their precision and performance. Key areas of focus were on the metrics for entity relevance, and caching data to avoid duplicated work. We intended to move to a persistent graph stored as a graph-database and, over time, build a model of the world of entities and their relations, and use this to aggregate sentiment for the parent entities (i.e the sentiment of *Apple* should take into account the sentiment for *iPhone*)

Outcome: We implemented all intended features of this sprint. We were then in a position to experiment with different mathematical functions used in the metrics for entity relevance and aspect relevance, along with separate weighting functions used in aggregating sentiment. We also had two persistent databases (one graph and one relational) which allowed us to build a model of the world with regards to named entities and their relationships (graph) and entities and their aspects (relational).

2.4.4 Sprint 4

We completed the evaluation of our product and delivered a presentation to the stakeholders to demonstrate the progress we have made in the previous two months. The UI took a slightly different direction in that it was used to aid model assessment alongside providing an interface for the client to use. We did not add the feature to display the sentences from which sentiment scores were derived however this is something we can feasibly do in the run up to the final presentation and demonstration. The fourth sprint also saw us make the switch from Google Cloud back to *spaCy* for entity tagging, since an external update to *Google Cloud*'s NLP API made its precision significantly worse in the technology domain.

3 Design and Implementation

3.1 System Design

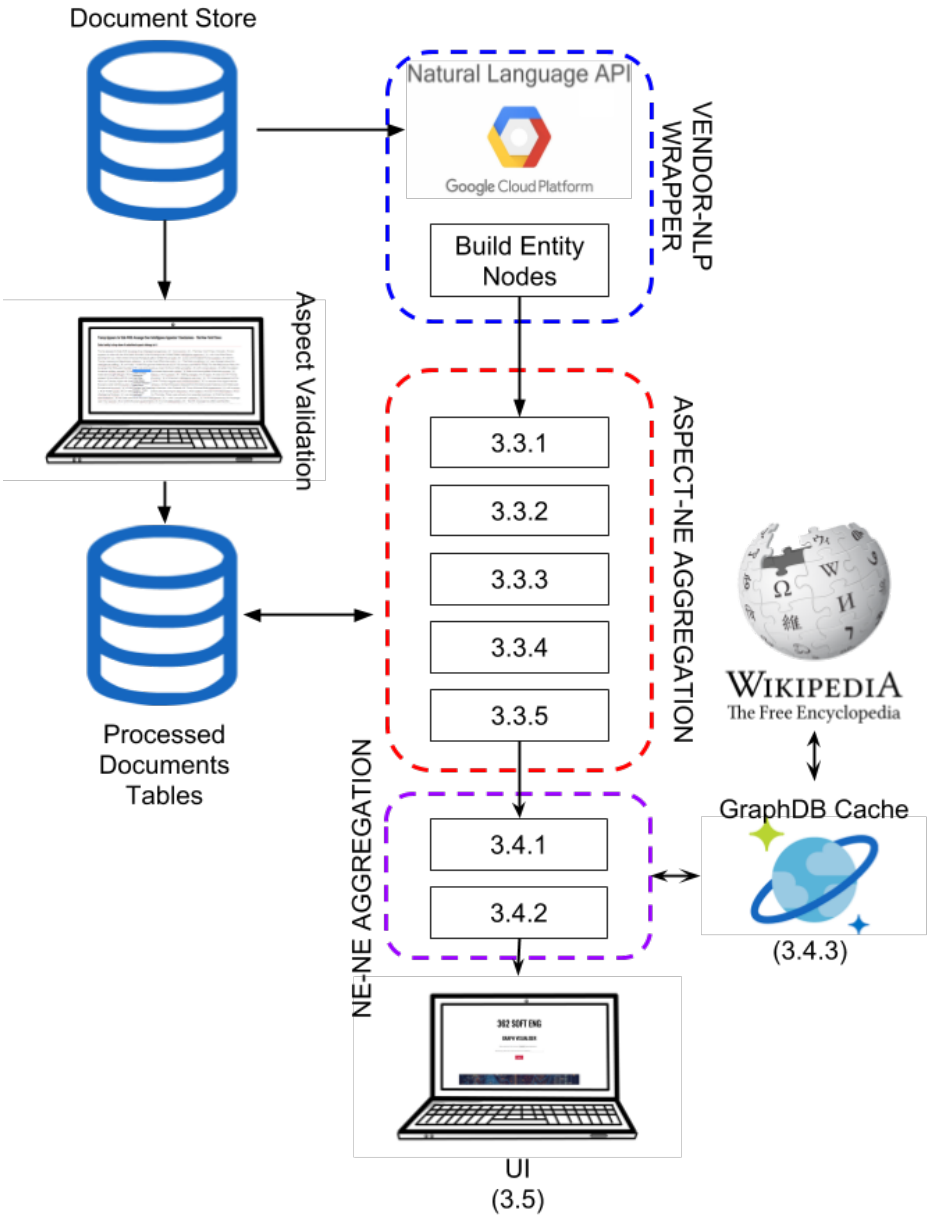


Figure 5: System Architecture

As discussed, our product takes a document as input and outputs a graph representing the sentiment of named-entities in the document, as well the semantic-relation between them. To achieve this, we had to develop a data-pipeline as shown in figure 5.

Breaking down the logical steps of the data-processing into respective logical modules had the following advantages:

- The architecture had high cohesion and low coupling. This in-turn allowed us to develop in parallel as pairs/individual only had to be concerned with the module they were working on.
- We could easily build a naive end-to-end solutions first and then iteratively improve modules that would improve the pipeline the most based on our metrics.
- The architecture could be transformed into a distributed service with each module acting as a micro-service and by having a messaging-queue coordinating the flow of the pipeline.

Additionally as shown in figure 5, we attempted to make our architecture as hexagonal as possible. We wrote wrappers and interfaces whenever we used external services and libraries. This reduction in coupling between our core pipeline and other services, allowed some team-members to focus on integration while others could simultaneously focus on developing our model.

3.2 Sentiment Analysis

In order to make it easier to use different sentiment analysis services and libraries, we wrapped each with an adapter which transformed the output of analysis into our own internal `EntityNode` class which is sent through the pipeline. This meant that in order to swap a new service into the pipeline all we had to do was define the adapter for it so that it coordinated with the rest of the system.

The two main contenders for the sentiment analysis service we would use were *NLTK* and *Google Cloud*, the latter of which we decided to keep in the long run. In both cases adapters were made to extract the information we desired from the respective services' output whilst discarding any excess data.

Following the decision to use *Google Cloud*, we were able to take advantage of the more advanced features of the service. In each analysis of a document we were returned more rich data that we could use to make more precise summaries about the input. In order to make full use of the data available, our pipeline slowly became more and more dependent on the extra (in terms of a comparison to *NLTK*) data made available to us and thus more tightly coupled to *Google Cloud*.

Clearly tighter coupling is an undesirable property of our code and thus we had to be sure that *Google Cloud* was the service we wished to use. However, with the modular design in mind, it is possible to set the 'missing' data from other services with appropriate null-representing values so that the pipeline can proceed to run as expected, should we change to another sentiment analysis service in the future.

3.3 Named Entity (NE) - Aspect Relationships

The following sections will cover how we implemented the detection of NE - Aspect relationships and the aggregation of sentiment from aspects. This was implemented by three

components in our system:

1. NE - Aspect relationship likelihood model (3.3.1)
2. NE - Aspect relationship identification (3.3.3)
3. NE - Aspect salience and sentiment aggregation (3.3.5)

3.3.1 NE - Aspect Relationship Likelihood Model

In order to recognise named entity - aspect (NE-A) relations, we needed to develop a model which would hold the likelihood of these relations. To perform this we chose to develop an unsupervised learning algorithm; this would extract information from all the documents which we encounter in order to recalculate the likelihoods.

For every document provided as input we performed probabilistic updates on the likelihoods using a technique similar to a Bayesian approach.[7] The update has the form:

$$P(R|\text{RID}) = \frac{\alpha(R)P(R) + \beta(D)P(\text{RID})}{\alpha(R) + \beta(R, D)}$$

In this formula:

- **R** represents a NE-A relation
- **P(R)** is the prior likelihood of the relation. The value of this is:
  - If this relation has been considered previously, the likelihood that we have stored in our database
  - If the NE has been encountered but the relation has not, the lowest likelihood we have in the database which contains the NE and any other aspect
- **P(RID)** represents the likelihood of the relation using information in the document, explained further in 3.3.2.
- $\boldsymbol{\alpha}(\mathbf{R})$ is the weighting for the prior likelihood in the update
  - This takes into consideration the quantity of mentions for the named entity within previously seen documents
- $\boldsymbol{\beta}(\mathbf{R}, \mathbf{D})$ is the weighting for the likelihood obtained from the document
  - The salience (importance) of both the NE and the aspect within the document are used here - with greater salience resulting in a higher weight

Once we have calculated $P(R|\text{RID})$ we then store this in our database to be used as the prior likelihood in the next update. An update is performed for all relations belonging to each named entity which occurs in the document.

3.3.2 NE-Aspect Relation Likelihood in Document Calculation

When calculating P(RID), we consider the relative locations of both the named entity and the aspect within the document. For every aspect mention in the document we calculate a score between 0 and 1 based upon the location of the closest mention of the named entity.

After some research, discussed in 4.2.3, we settled on a method which gave co-occurrences in which the NE appears prior to the aspect a higher value compared to when the aspect is before the NE.

Once we have a score between 0 and 1 for every aspect mention we calculate the mean of this. This mean value is P(RID) which we use in the update of P(R).

3.3.3 NE - Aspect Relationship Identification

What we needed to implement in this component was the ability to detect relations between named entities and their aspects within a document. Therefore this section was dependent on the likelihood values we that we calculate for NE - Aspect relations (as discussed in 3.3.1).

Within this component we had to decide upon what would make a relation likely. When we encountered the mention of an aspect we considered several factors:

- The named entities which have been mentioned recently enough to be a possibility; we considered these entities to be the active entities
- How much prior knowledge we need to have on the active entities to be confident in our likelihood value
- The threshold for the relation likelihood value for us to assume a relation
- What to do when there are multiple entities meeting the threshold

The flowchart in Figure 6 best represents the decision logic we implemented within this component. The flowchart shows how we handle each occurrence of an aspect when attempting to find the named entity which it belongs to.

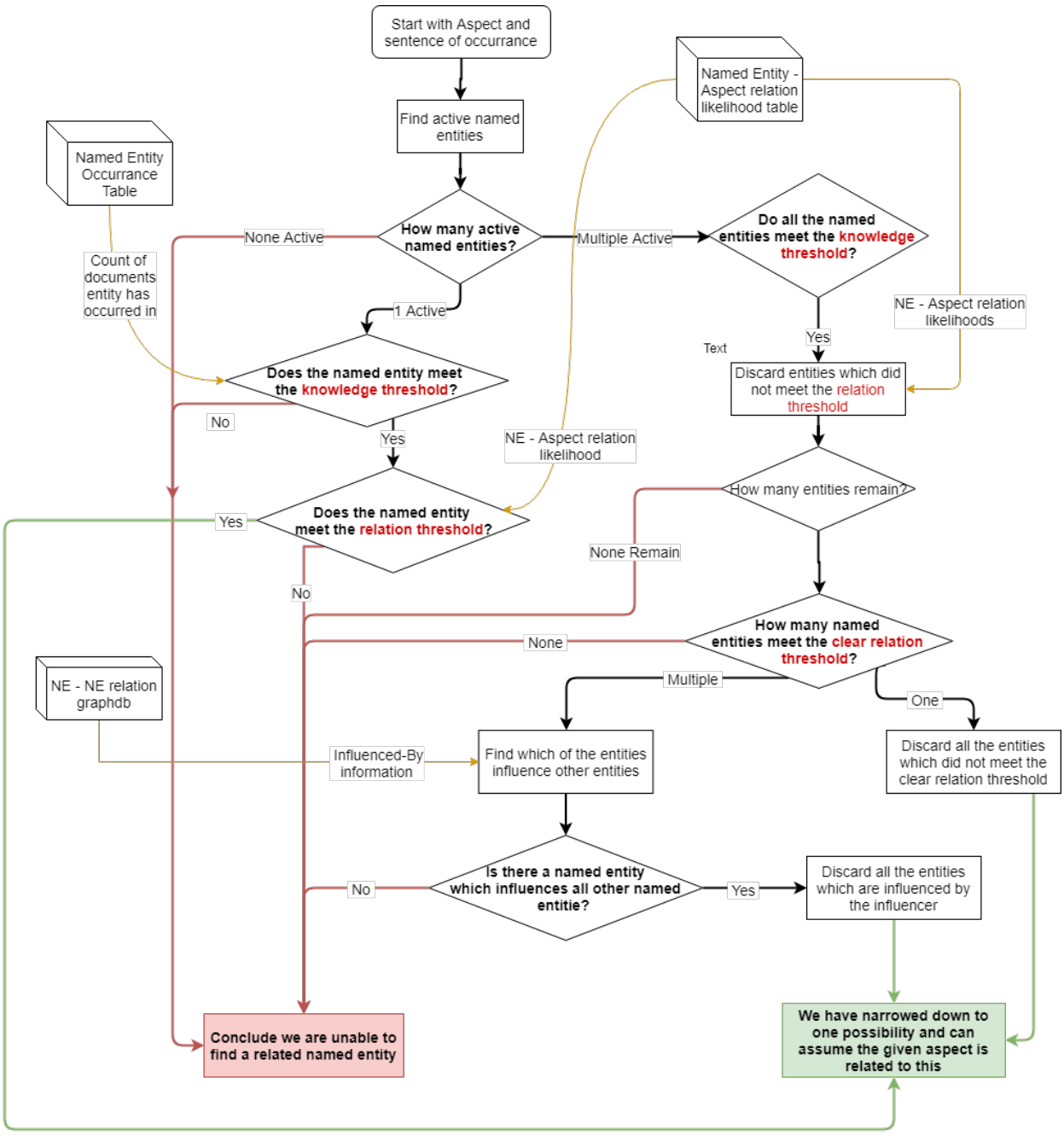


Figure 6: Flowchart showing how we attempt to identify the named entity that an aspect belongs to

All of the functionally for this decision logic is held within the NE - Aspect relationship identification component. However there are some notable differences from the flowchart within our implementation.

Firstly, whereas the flowchart shows behaviour on a single aspect mention, we do not process one mention at a time we instead process sentence by sentence. The reason for this design choice is based on how we calculate the active named entities; we select the active named entities from a sliding window as we pass through the document. The window size we eventually settled on was the current sentence and the previous two. How we determined this window size is discussed in 4.2.

This method of processing the document also benefits us in computation time due to the fact that it reduces the amount of times we check the named entity against against the knowledge threshold as we avoid repeating the check on every aspect mention.

Highlighted within the flowcharts are thresholds used:

- **Knowledge Threshold**: this threshold controls the number of documents we need to have previously seen all the named entities within to continue with finding the related named entity
- **Relation Threshold**: this threshold is a value between 0 and 1. Whether the relation likelihood for the NE and aspect is above or below this threshold states whether we believe a relation between the two is likely or unlikely respectively
- **Clear Relation Threshold**: this is a combination of two conditions that need to be satisfied and is used to decide whether or not a relation between the NE and aspect is highly likely as opposed to a weaker likelihood which has met the first relation threshold.
  This is used when multiple entities meet the first relation threshold. The first factor it must meet is a minimum likelihood value and the second factor means that it must be a scalar multiple greater than all the likelihood of all the other entities

The above thresholds are implemented using several program constants. The process of tweaking these constants to get optimal behaviour is discussed in 4.2.

3.3.4 Usage of NE-NE Relation Database in NE - Aspect Relation Finding

A notable part of the flowchart (Figure 6) is the usage of the NE-NE relation database. This stage typically helps us distinguish in cases where there is two name entities which have met the relation threshold.

Given that we have a directional graph of NE-NE relations (3.4) we can use this to know which direction sentiment will be propagated at the later NE-NE aggregation stage. The logic we implemented is to assign the aspect to the NE which is an influence of the other in this case given that it will eventually propagate the sentiment of the aspect to the other sentiment through the later stages (3.4.2).

We became confident that in most cases, if there is a link between these two named entities, it is very likely the child in the relationship is the reason for the high likelihood between the parent and the aspect. This was brought to our attention during validation, see 4.2.

3.3.5 NE - Aspect Sentiment and Salience Aggregation

Once we have detected the relations we need to aggregate both the sentiment and salience from the aspects to the entities which own them.

This component works by considering all of an entity's aspects in order to update both the sentiment and salience of the named entity, for each named entity.

Salience Aggregation

We were required to update the salience as, although it is not something we output to the user, we use it in the NE-NE sentiment aggregation stages. Therefore if we have found a lot of aspects for the NE we wish this increased knowledge to be accurately reflected in any NE-NE aggregation.

The update to salience is simply a summation of the salience values of all the aspects and adding this to the salience of the NE. We chose to do it this way given that the response we receive from Google has a normalised salience where all of the salience values sum to 1. Therefore this idea seems logical as it reflects the additional text within document we have found belonging to the NE.

$$\mathrm{NE}_{\mathrm{salience}} = \mathrm{NE}_{\mathrm{salience}} + \sum_{\alpha \in \mathrm{NE}_{\mathrm{aspects}}} \alpha_{\mathrm{salience}}$$

Sentiment Aggregation

To propagate sentiment, we begin by retrieving a list of sentiment scores for each mention of the NE and another list of sentiment scores for each of the aspect mentions belonging to the NE which we are processing.

This part of processing is held within the entity node class as opposed to the sentiment aggregation component. We chose to implement it in this way to follow the 'Law of Demeter' in order to minimise the fragility of this component, as it allows us to not be concerned with the how mentions are represented within the entity node class.

Once we have the two lists of sentiments we begin by filtering out all the sentiments of very low magnitude. When doing this we are assuming that these aspects had no sentiment assigned to them and are therefore not important at this stage.

$$\mathrm{NE}_{\mathrm{sentiments}} = \{M_{\mathrm{sentiment}} | M \in \mathrm{NE}_{\mathrm{mentions}} \wedge M_{\mathrm{sentiment}} > 0.05\}$$

$$\mathrm{A}_{\mathrm{sentiments}} = \{A_{\mathrm{sentiment}} | A \in \mathrm{NE}_{\mathrm{aspects}} \wedge A_{\mathrm{sentiment}} > 0.05\}$$

Once the two updated lists of sentiments have been calculated we can then use these lists to calculate our new sentiment value for the NE. This calculation is represented by the formula:

$$\mathrm{NE}_{\mathrm{sentiment}} = \frac{\sum \mathrm{NE}_{\mathrm{sentiments}} + \beta \sum \mathrm{A}_{\mathrm{sentiments}}}{\#\mathrm{NE}_{\mathrm{sentiments}} + \#\mathrm{A}_{\mathrm{sentiments}}}$$

$$\beta = \frac{\#\mathrm{A}_{\mathrm{sentiments}}}{\#\mathrm{A}_{\mathrm{sentiments}} + \#\mathrm{NE}_{\mathrm{sentiments}}}$$

- β is a factor we introduced to reduce the influence of the aspect sentiments based on the ratio of aspect mentions to total mentions
- We divide by the total number of sentiments to normalise the sums in the numerator thus ensuring the new sentiment is a value between 0 and 1

Once we have calculated this new sentiment, the sentiment and salience of the `EntityNode` we were processing is now updated to hold the new values.

3.4 Named Entity (NE) Relationships

This section covers the implementation of the last stage of analysis in the pipeline. Having processed the document, extracted the named entities with their aspects and aggregating the aspect sentiment into the respective NEs, we are left with a list of `EntityNode` objects. We now perform more abstract, higher level analysis by determining relationships between different NEs, for example: *Apple* and *iPhone 6*. This stage can be broken down into 2 main parts:

1. Forming NE-NE relationships based on real world semantics
2. Sentiment propagation based upon the newly formed relationships

By the end of this stage of the pipeline, having had a list of `EntityNode` objects, we construct a directed graph with weighted edges, where the nodes are `EntityNode`s representing NEs and the edges are NE-NE relationships.

3.4.1 NE-NE Relationships

We believed we could leverage *Wikipedia*'s knowledge of the real world by looking at the hyperlinks on *Wikipedia* pages. Hyperlinks provide a natural source of relationship between two *Wikipedia* pages, as well as some sense of hierarchy.

How do we determine the existence of a relationship between two NEs A and B given that both have corresponding *Wikipedia* pages? Assuming we have scraped both of the pages and collected all the hyperlinks, we know the number of occurrences of each hyperlink and the total count of all hyperlinks.
Let:

$$\text{Number of mentions } B \text{ on page } A \equiv \text{numOfMent(B, A)} \tag{1}$$

$$\text{Largest number of mentions for a hyperlink in } A \equiv \text{maxFreq(A)} \tag{2}$$

$$\text{Relationship exists between } A \text{ and } B \equiv \text{isRelation(A, B)} \tag{3}$$

$$A \text{ contains hyperlink to } B \equiv \text{isPresent(B, A)} \tag{4}$$

Then:

$$\begin{aligned} \text{isPresent(B, A)} \wedge \text{numOfMent(B, A)} > \max(1, \text{maxFreq(A)} * 0.05) \\ \iff \text{isRelation(A, B)} \end{aligned} \tag{5}$$

This can be interpreted into English as: "If B is mentioned on A's *Wikipedia* page **and** B is mentioned more times than a given threshold, then there exists a relationship **from** A to B". This relationship is directed in terms of the *Wikipedia* pages but not (yet) in terms of which entity is influenced by its counterpart.

We can also deduce that *isRelation* is non-symmetric, non-reflexive and non-transitive due to the nature of *Wikipedia*.

Extracting knowledge from *Wikipedia* is the first step in building NE-NE relationships. We collect extra information by gathering **all** NEs off A's *Wikipedia* page (that qualify the conditions necessary for *isRelation* to hold) and store them in the graph database for persistent caching, since the external API provided by *Wikipedia* is heavily throttled. Of course if the NEs being analyzed have been seen before, we will not perform the *Wikipedia* analysis logic as the results of it will be present in the graph database.

Having collected all the data from *Wikipedia* and storing it in the graph database, the algorithm for the second step is relatively straightforward:
For each NE A in document:

1. Get the corresponding sub-graph (all NEs B such that isRelation(A, B) holds) from *Wikipedia* or the graph database
2. If A and B are both present in the document, then add a directed edge to the final graph, such that the direction is from the the "larger" to the "smaller" NE

The notion of ordering seen in step 2 is implemented in the `EntityType` class that represents the different types of NEs. These types can be described by the set in Figure 7. These are derived from the external library *spaCy* that we use and we do admit that this introduces inevitable coupling. `EntityType` is an `Enum` class where the ordering is defined using the ordinal values seen in the set above. The ordering of the types resembles real world hierarchy based on their economic importance. For example, an *Organization* is of larger economic importance than *Consumer Good*. In fact, this ordering **defines the direction** of relationships within our final graph.

3.4.2 NE-NE Sentiment Propagation

Having established directed relationships between NEs in the form of a directed graph, it is natural to consider exploiting the graph structure in the context of the problem. In this case, we decided that, since edges represent real world relationships (e.g *iPhone 6* is a product of *Apple Inc.*), we can use them to incorporate real world semantics into our analysis and, potentially, correct bias and inaccuracies within the sentiment scores.

In a nutshell, what we do with sentiment propagation is: if a node has a lot of negatively scored neighbours, then that node becomes more negative and vice versa. A good example

Figure 7: `EntityType` enum

{*Organization = 10, Consumer Good = 7, Person = 6, Location = 4, Event = 3, Other = 2, Work of Art = 1, Unknown = 0*}

would be an article that talks positively about *Microsoft* in one paragraph but then talks very negatively about *XBOX* and *Microsoft Surface* in the following paragraphs. The naive sentiment analysis would say that Microsoft has a positive sentiment score, but we say that, because *Microsoft*'s products *XBOX* and *Microsoft Surface* are presented negatively, *Microsoft* should not be as positive as in the naive analysis.

Figure 8 is an example of sentiment propagation on a simple graph.

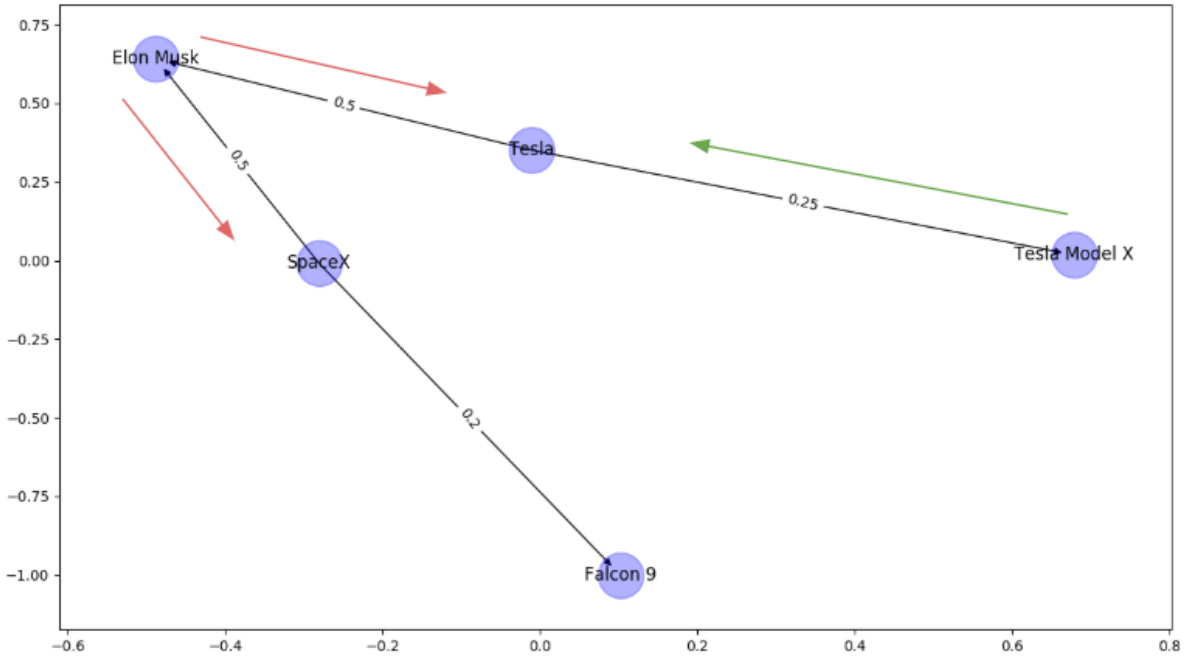


Figure 8: **Elon Musk**: `-0.500`, **Falcon 9**: `+0.600`,
Tesla Model X: `+0.000`,
SpaceX: `+0.300` → `+0.189`,
Tesla: `+0.300` → `+0.130`

It is easy to see that NE "Elon Musk" has a -0.5 negative sentiment and, through propagation, "Elon Musk" affects both "SpaceX" and "Tesla" by making their respective scores less positive. However, since "Falcon 9" has a positive +0.6 sentiment, the negative change on "SpaceX" is reduced.

Before we introduce the algorithm for performing propagation, we need to introduce the use of weight attributes for graph edges. As seen in figure 8, all edges have a weight attribute. Weight is calculated during the *Wikipedia* knowledge extraction and is therefore also cached in the graph database for each relationship. Weights represent the "strength" of the relationship based on the frequency of mentions of an entity on a page.

$$\text{weight}(\text{A}, \text{B}) = \frac{\text{numOfMent}(\text{B}, \text{A})}{\text{maxFreq}(\text{A})} \tag{6}$$

After building a directed graph with `EntityNodes` representing NEs and directed edges with weights representing relationships, performing propagation becomes very simple:
For every node A:

1. Visit all adjacent nodes B, where adjacent means **outgoing edge**
2. Having visited every B propagate sentiments into A by updating A's sentiment score using the following function

$$A_{score} = \alpha * A_{score} + (1 - \alpha) * \text{adjacentAvg(A)} \tag{7}$$

where:

$$\alpha = 1 - \frac{1}{(\text{neighbourCount(A)} + 1)^2} \tag{8}$$

$$\text{adjacentAvg(A)} = \frac{\text{adjacentSum(A)}}{\text{neighbourCount(A)}} \tag{9}$$

$$\text{adjacentSum(A)} = \sum_{B \in adjacent(A)} (B_{score} * B_{mag} * weight(A, B)) \tag{10}$$

3.4.3 Caching & Building a Persistent Model

In order to reduce the number of requests made to the *Wikipedia* API and improve the performance of the pipeline, we decided to build a persistent store of named entity relationships. To achieve this we used *Microsoft*'s multi-model database, *Cosmos DB*. [11] We chose *Cosmos DB* because it was easy to setup and manage with our existing *Azure* infrastructure and its close ties to *Azure* also means that it was easy for us to build in redundancy and would be easy for us to distribute globally.

Our API for this database was built using the graph traversal language, *Gremlin*. *Gremlin* allows us to use basic graph traversal and pattern matching to build up complex graph queries on our entity relationship cache.

In a *Cosmos* DB graph database we store nodes with an entity ID and knowledge as to whether they have previously had their *Wikipedia* pages scraped. These entity node are joined with edges indicating the influence relationship between them and a weighting to indicate the significance of the relationship. Having this information means that the next time the program comes across an entity, if it has had its Wikipedia page scraped for entities already, there is no need to do it again and the relationships can be read from the graph.

3.5 Graphical User Interface

3.5.1 Why design a GUI?

Although not part of the formal project requirements from *Goldman Sachs*, we felt that it would be particularly beneficial to implement some form of user interface, to allow the end user to have some form of visualisation of the data we are outputting and possibly then to allow them to uncover potential new links in the data that would not be immediately obvious just from the raw data.

Throughout the development cycle, we were using a basic method of outputting the links between entities visually with *Networkx* — a *Python* library for studying graphs and networks. [10] This was a fairly crude method but, as it was only for our own internal

validation, it worked perfectly well and provided us with all the information we needed. This eventually became the basis for the web app.

Initially, it was proposed that we could make a minor adaption to the code in order to simply produce the *Networkx* graph for any given input document, as we had been doing internally, and serve this as the UI. While certainly this would have worked, there were a few flaws with this which we had been happy to overlook for our testing purposes but which would not have stood up were this to be implemented as our final front end:

- Most importantly for us, the generation of the *Networkx* graph was something that was done at the end of running the pipeline through the command line interface. We wanted a more fully-fledged web app, which would enable the user to upload or input text graphically, rather than via commands, which we felt was less intuitive.
- The graph produced with *Networkx* is static — there is no way to interact (i.e. pan and zoom, hover over elements for more information, etc) with it. We wanted a more responsive UI than what *Networkx* would have offered.

3.5.2 Building a Web Application

With the above issues in mind, we decided to create a simple web interface with which the user would be able to interact with our product.

We opted for *Pyramid*, a lightweight *Python* web framework, to build our web-application. [8] Simply put, *Pyramid* works using a series of function decorators, allowing you to receive requests at a given URL, execute a function, and then return a page. From the index page, the user is able to upload a news article in XML format, which is then passed into a function as a `POST` request. The function receives the document and passes it through the pipeline, as one would do via the command line. The graph would then be generated and saved in the local directory for us to open.

In addition to setting up the routing on the web-app, we had to change the final step in the pipeline so that we were able to display this interactive graph. This is where the *Holoviews* library came in. [9] *Holoviews* allows us to embed the graph directly into an HTML page, meaning it can seamlessly integrate into a web app, exactly as desired. There are a number of useful customisation options available with *Holoviews* that we were also able to use to improve the accessibility of the data. For example, we are able to colour the arcs based on the weight, or the nodes according to category.

An interesting discussion point that was raised by *Goldman Sachs* during one of our initial meetings was how best to outwardly represent the weightings of the links between entities. Internally, each arc carries a value between 0 and 1, representing the strength of the connection. However, to an end user, what does this value actually represent? Is, for example, a value of 0.4 noticeably better than a 0.3 or a 0.35? Given that the values are represented as decimal numbers with a precision of tens of digits, is it not also almost certainly the case that the numbers are too precise to really be reasonable, especially given the subjective nature of entity relations? *Goldman* actually already use a similar confidence scale in one of their internal products, and they found it was better to display the values in three colour bands (e.g. with red representing any values from 0–0.333, yellow 0.333–0.666, ...) rather than showing the actual value. We liked this approach and decided to implement

it ourselves, and so weakly-linked entities are connected with red arcs, entities with some reasonable link have yellow arcs, and strongly-linked entities are connected with green arcs.

The completed web app (screen-shots follow in Appendix C) thus allows the user to upload a news article in XML format, have it pass through the pipeline, and then be taken to a page on which the entity-relation graph is displayed. A later decision was taken to merge the user validation tool (see section 3.6) into the web app.

3.6 Validation User Interface Implementation

We were aware that for the validation of Named Entity - Aspect relations we would need data to compare the output of the system against.

We decided that the best way to do this was by validating against what a user believed to be the relations in the document and therefore concluded that we would need the ability to generate this validation data ourselves and then compare against it.

We knew that for this method of validation to be effective we would need an interface with two key features:

1. We need to be able to easily tag NE-A relations within a document
2. We need to be able to easily see the results of the pipeline in comparison to tags defined by the user

3.6.1 Document Tagging UI

For the first requirement we decided the best way we could implement this given the time constraints was by giving drop- down menus next to each aspect identified in the document giving the ability to select which named entity. This can be seen in figure 16.

The technologies used in implementing this were again *Pyramid*, the *Python* web framework, which was used in the GUI for usage by users (3.5.2). Here the usage of *Pyramid* was to retrieve the relevant data from the database based on the document ID passed via a `POST` request we could submit from the validation home page (visible in figure 15).

Alternatively, for this page we made use of *Mako* templates to render the page for it's simplistic ways to implement dynamic web page generation. Using *Mako* made solving the main implementation task easy, this was to insert the drop down options in the correct positions within the document text.

3.6.2 Storage of User Defined NE - Aspect Relations

Once the user has completed tagging the document on the UI, they then submit their form which sends a `POST` request to the backend of the application. This is then processed, parsing the results of the form and extracting the required data.

3.6.3 Validation against the User Document Tags

Now that the tags are stored permanently in the database, the user can request for a document to be ran through our system and compared to existing user defined tags via the UI.

When running these comparisons between the two sets of tags (system and user), we group the results into 4 categories, these are:

1. `Matches` — When the user and system both selected the same named entity for an aspect

2. `Incorrect Entity` — When the user and system both selected a named entity for an aspect but they were not the same named entity

3. `Additional` — When the system selected a named entity for an aspect which the user had selected no named entity

4. `Missed` — When the user defined an NE - Aspect relation and the system did not assign this aspect to any named entity

3.6.4 NE - Aspect Validation Results UI

With our intended use of the validation UI, which will be explained in 4.2, it was important that we could see the results of validation in a clear and concise manner.

Our implementation for this is shown in the demonstration in Figure 17. There are three ways in which we displayed results of validation:

1. **Bar chart plot**: this allows us to see the change in performance of the system against the user defined tags over time. Each bar on the plot is colour coded based on the four categories defined in 3.6.3. This means we can make a change and then reload the page to re-run the system on all the tagged documents and immediately see the results of the change in comparison to before the change.

   Bokeh was used to produce the plot, generating the HTML from the data for the graph, which is inserted into the *Mako* template upon rendering the page

2. **Table of results**: this groups the results based upon the four categories and can be interpreted as an expansion of the final bar on the plot. The purpose of this was to allow us to quickly spot trends in the results and identify areas of weakness in the algorithm.

3. **Statistic summaries** (above the table): this allows us the quickest possible comparisons when evaluating simple performance targets of our system, for example a minimum % marking of correct relations

4 Validation and Evaluation

4.1 Selection of External Sentiment API

A key part of our pipeline is the tool we use for sentiment analysis. We tested both *NLTK Vader* sentiment analysis tool and *Google Cloud*'s Natural Language API and ended up settling for *Google Cloud*'s offering.

One major reason for this was that *Google Cloud* would perform both entity tagging and sentiment analysis with one API call, returning to us the entities that appear in a document and their sentiment. On the other hand, with *NLTK* we would have to first tag the document, locating the entities, and then take the sentiments of the sentences they appear in.

NLTK is limited by the fact that it gives the sentiment of a sentence whereas *Google Could* looks more specifically at the words relating to the entity and gives the sentiment of that entity. An example of this is the sentence *"I really like the Galaxy S9 but I didn't like the iPhone X"*. *Google Cloud* gave a sentiment for 'Galaxy S9' of 0.4 and for 'iPhone X' of -0.8, where as *NLTK* gave the sentence a sentiment of 0.0. We get less information by using *NLTK* and often the sentiment of a sentence is based upon one or more entities meaning it's not a particularly good representation of either entity.

We also regularly witnessed poor sentiment analysis by *NLTK*, one example was the sentence: *"The Justice Department has been criticized for years as being too easy on the banks that caused the financial meltdown and set off a major recession."*, a clearly negative sentence. *Google Cloud* gave 'Justice Department' a sentiment of -0.8 and *NLTK* gave the sentence a sentiment of 0.25.

4.2 Named Entity - Aspect Relations Validation

In performing validation of the Named Entity - Aspect relation detection part of the system, we made use of the Validation UI we developed, see 3.6.

He we will be referring back to the four categories that we used to categorise results: `Matches`, `Wrong Entity`, `Additional` and `Missed`. See 3.6.3 for description of the categories.

When we were performing our validation and tuning process we had two main goals in mind:

1. **Maximise** `Matches`: this was for obvious reasons as the desired behaviour would be to match user data perfectly

2. **Minimise** `Wrong Entity`: we identified `Wrong Entity` as the greatest concern in the incorrect cases as it means we would be likely to attribute sentiment to the wrong entities.

For `Additional`, if the user did not identify an NE for the aspect, then it is unlikely it was an important aspect and will therefore be unlikely to have sentiment attached to it.

Similarly, `Missed` cases were also not as much of a concern given that, in this case, there is no chance we have introduced incorrect information but instead we have missed information which we believe would not have as adverse an effect.

4.2.1 Usage in Threshold Altering

When altering the threshold values that were used in NE - Aspect Relations Identification component (3.3.3), we used the validation UI in order to confirm that the effect of alterations had the desired effect. In this section I am going to step through the results used to decide the final values for these thresholds.

Beginning with the knowledge threshold, when altering this value we believed that we had a trade-off between the quality and quantity of information we could extract. If we have a high knowledge threshold, then we will only assign aspects to an NE if the knowledge we have of all the active NEs is at this greater level, so we would be more confident in the correctness of results. However, we would also expect an issue arising in that if an NE is present that we have very little knowledge of, then it will not meet the threshold and it may be the case that if we disregarded the threshold then we would have collected correct information.

To test this theory we began altering the knowledge threshold to find a suitable level. Firstly, with an initially low threshold (which required us to have seen NEs in just one document previously), we typically got a lot of correct entities but the percentage of aspects marked for the wrong named entity was higher than we would be satisfied with. For the large thresholds we did see that the percentage of relations marked for the wrong entity had decreased but we also saw for many documents a dramatic decrease in the percentage of correct relations detected, as seen in Table 4.

After testing at the different values we decided the value which had the best result with regards to the trade-off was 2. This is the current value in our system however we would expect that we would be able to increase this threshold in the future as we develop a wider knowledge base.

The same trade-off exists for both the relation threshold and clear relation threshold. The different values we tested for the relation threshold can be seen in table 5. After analysing these results we settled on 0.1 for the relation threshold.

We then proceeded to perform the same process for the clear relation threshold, this time altering the two values which make up this threshold: the minimum value and the multiplicative factor that it must be greater than in comparison to all other relations. After seeing these results (Table 6) we decided to use 0.3 and 2 for these two values respectively.

We did not use 0.3 and 1.5 respectively due to our cautious approach to solving this problem. This case had a greater increase in mean percentage of cases when the model selected the wrong entity proportionally compared to the increase in correct cases. Therefore we saw 0.3 and 2 to be a better solution for our trade-off

We performed similar tests on sizes for the sliding window which we select the active named entities from, see Table 7. From these results we chose a sliding window consisting of 3 sentences, the current sentence and two prior.

4.2.2 Detection of New Algorithm Additions

Our initial algorithm differed from the flowchart that is present in 3.3.3, as originally we had no interaction with the NE-NE graph at that stage of the pipeline. However, the use of the validation UI brought our attention towards a poor performing document. The document was about the *Playstation Neo*, a name given initially to the *Playstation 4 Pro*. Therefore within this article there were mentions of *Sony*, *Playstation 4* and *Playstation Neo*. We noticed that our first run managed to correctly identify just 10% of relations (figure 9).

We then decided to look into the logs, from running our pipeline, and found that there were many cases in which both *Sony* and *Playstation 4* have a high relation likelihood for a given aspect. This was easy to justify as *Playstation 4* is a key product of *Sony* making it likely that the key aspects of a *Playstation 4* are also going to be key to *Sony*. Therefore upon seeing this we then decided that it would be useful for us to make use of NE - NE knowledge graph which we had built as we concluded in this case (and we assumed many more) it makes sense to assume relation is to the product (*Playstation 4*) which will then have it's sentiment aggregated to the organisation (*Sony*).

Once we had implemented this we then returned to our validation UI. We saw an improvement in the matches detected in document the issue was raised in, now correctly detecting 33% of NE - Aspect relations for this document (figure 10). The validation also allowed us to immediately see that it had not had a negative effect in any of the other validation documents and therefore we decided to keep this change to the algorithm permanently.

4.2.3 Relation in document likelihood validation

Using the documents we had tagged we then wrote a simple program to provide us with statistical analysis of Named Entity - Aspect relations. This program gives us figures on the character distances between aspects and the closest occurrence of the named entity which was tagged by the user. These results are in Table 8.

We used these results as information on how we should score the likelihood of an NE - Aspect relation at the document level. Due to the fact that the closest NE before was on

average closer than the closest NE after, we decided that Named Entity occurrences before the aspect should be scored higher than NE occurrences after.

The summaries calculated also helped in deciding upon how exactly we scaled the inverse distance part of scoring. With a score of 0 being approximately equivalent to the upper quartile values we observed.

4.2.4 Example of Learning

The validation UI also allowed us to see evidence of our algorithm learning for many of the documents. Typically it was the case that we would know very little about a document and then by finding other documents for the subjects of the document and training our model on this we could gradually see the performance of our algorithm improve. For example in figure 11, we see the improvement of the performance on a document, whilst all thresholds in the system were kept constant, after feeding the system other documents containing the NEs present in the document.

4.2.5 Evaluation of NE - Aspect Detection

Overall, keeping in mind that a majority of the content of components making up Named Entity - Aspects recognition being based upon data we have collected, we are confident that the final algorithm we have now is of a good standard and works in a logical manner. With the use of the validation UI we have been able to tweak values in the system to try to optimise performance across documents with a variety of topics which has increased our confidence in our product further.

In general we have taken a cautious approach to this problem which is enforced by the knowledge thresholds and clear relation thresholds. Across the validation documents we have, at a time of writing, an average rate of 9% of aspects assigned to the wrong entity, while 45% were assigned to the correct NE. These are values we are very happy with given that it's clearly a low error rate but also a high success rate. From the numbers we saw we believe we can produce a considerable amount of useful data and that the user does not have to be concerned of the validity of this data.

We are also happy to have observed several examples of the model improving after collecting more data on named entities. This gives us great confidence that over time the machine learning algorithm we have implemented would see results improve further.

4.3 Named Entity Relations Validation

In order to assess the performance of our algorithm for building relationships between named entities in the world, we had to define a metric which would allow us to draw meaningful comparisons between iterations of the algorithm.

Within this section of the pipeline there are two areas of interest: named entity extraction from the *Wikipedia* page and the criteria for building a relationship between two entities. The former of these issues is specific to an entity tagger library and thus something we did not feel worth assessing; this would only be important to us in picking a service to use as our entity tagger rather than any performance improvements we could directly make to our algorithm. On the other hand, adjusting our acceptance criteria for determining if entities are linked is an area we could have complete control over and therefore would be the main focus for this section of model validation.

Initial discussions introduced the idea of manually tagging *Wikipedia* pages with their related entities and comparing this to the output of our pipeline however it quickly became apparent that this was an infeasible approach. The most significant issue with this approach is that two individuals may have vastly conflicting ideas on what is deemed relevant when considering the relationships between entities.

With the aim of reducing this conflict during validation, we moved towards an accuracy scoring system whereby we look at the output of the algorithm and mark a returned related entity as correct or not. Since human validators consider a much smaller set of entities, should two people come across the same entities to validate the relationship between, it is significantly more likely that the two people will agree on an answer than if they had to list the entities linked to a given named entity.

As mentioned previously, the influence of the entity tagger on the output of the algorithm is beyond our control and thus erroneously tagged named entities that appeared in the result were omitted. Once this noise had been removed, we produced an accuracy scored based on the proportion of the returned entities which are relevant to the *Wikipedia* page from which they appeared.

Naturally as we tightened our acceptance criteria, the number of mentions required for a connection to be made, the output set cardinality decreased — a factor we had to take into consideration when deciding which criteria conditions worked best for our purpose.

After adjusting our parameters to what we considered most optimal we were able to finalise accuracy scores for different sectors of the stock market. We considered sectors in which companies usually have a lot of linked entities, such as products and subsidiaries, to give us richer data and then picked some significant players in each sector to find an average accuracy score. The sectors picked were technology (a sector the project focus was shifted towards), consumer staples and consumer discretionary. The full results of the *Wikipedia* scraping for relevant entities can be found in appendix B.3.

Company	Apple Inc.	Microsoft	Sony	Samsung
Correctly tagged	32	19	38	38
Accuracy (%)	87.5	84.21	84.21	63.16

Table 1: The technology sector results.

Within the technology market we looked at the results for *Apple*, *Microsoft*, *Sony*, and *Samsung*. Whilst the accuracy score came out at 79.77%, we noticed that a lot of the returned entities were dated products which may have been significant in the company's growth but are virtually extinct in the present day. Whilst this is not ideal, the appearance of these entities in the result do not negatively impact the main purpose of the product when aggregating sentiment since they are so unlikely to ever be mentioned, so it is not a problem that required much attention. It is worth noting that the strong score of this sector is helped by iterations of a company's products appearing, such as *IPhone 4S* and *IPhone XS* for *Apple*. It is a reasonable move to keep both when determining relationships but this characteristic of the technology sector leads to a stronger score than in sectors where this iterative product cycle is not evident.

Company	The Coca-Cola Co.	Procter & Gamble	Walmart	Unilever
Correctly tagged	2	16	38	25
Accuracy (%)	100	50	47.37	84

Table 2: The consumer staple sector results.

For the consumer staples section of the stock market we picked *The Coca-Cola Company*, *Proctor & Gamble*, *Walmart* and *Unilever*. This sector achieved an accuracy score of 70.34%. However, included in this figure is the result for *The Coca-Cola Company* which, when ran through the algorithm, returned so little data that it achieved a perfect accuracy score. Omitting this 'false positive' gives a more reliable average accuracy score of 60.46%. One problem evident form the results is that a company is often linked to competitors and their products. Given the purpose of the overall pipeline it is detrimental to the cause to link these as sentiment propagation between them is nonsensical. Other than this flaw, the algorithm performs reasonably well as there are clear-cut distinctions between companies and their products and there is less frequent collaboration between companies in this sector meaning overlap of *Wikipedia* pages is less likely.

Company	Disney	Ford	Amazon.com
Correctly tagged	3	1	21
Accuracy (%)	66.67	100	57.14

Table 3: The consumer discretionary sector results.

Lastly, we ran our algorithm on *Disney*, *Ford* and *Amazon.com* to assess our algorithm's performance in the consumer discretionary sector. On average there was a 74.60% accuracy score which can be deemed a false positive. As seen in the table there are very few entities returned by the algorithm, regardless of correctness, for two of the companies ran. This is because the acceptance criteria for keeping entities in the relevant entity set is calculated using the most mentioned entity on the page. For the three companies we ran through our algorithm, two returned a variant of their company name in the final set which would make the acceptance criteria too high for other entities to meet. This is just one example of the weakness that *Wikipedia* has as a data source, discussed below.

A common problem across all of these sectors is that the algorithm heavily relies on the quality and reliability of the *Wikipedia* articles we use. During research we were unable to find a better objective data source to use however this does not mean that *Wikipedia* is a good source, but instead it is a better source that its rivals. For example, the section on the *Sony* page on *Wikipedia* there is a brief short paragraph for '2016 Onwards' which omits most of the products released by the company in this time period meaning our model cannot build relationships well for that company. Whilst this may be a case specific problem, it highlights the fragility of the data source which has a close knock-on effect on the output of the algorithm. Potential solutions and improvements regarding this issue are outlined later in this report.

The performance of this stage in our product's pipeline appears to rely heavily on its data source and entity-tagging library. We believe that given a better data source and more refined entity tagger, our algorithm could work well assuming some parameters were tweaked. Whilst some of the accuracy scores may not groundbreaking upon first look, we recognise that there is a trade off associated with the number of entities linked to an input entity and the correctness of this result. A working compromise comes from allowing bad links to be made, with the hope that two incorrectly linked entities do not appear too frequently in the same input article of the product.

4.4 Ability To Solve the Problem

Originally we set out to create a product which could process a document to produce a summary of the sentiment directed towards the named entities within. In both machine learning models for micro-level and marco-level relationship building we found we had to proceed with extreme caution when considering how our respective input data was structured. Regardless of the level of relationship being built, it is unsafe to make any assumptions about the way in which the writer of the document uses language to express their point. Since the two relationship building problems differ in nature, we made an early decision to approach each problem separately, leading to completely different approaches.

We believe our product performs well in either case and our validation has provided us with reasonably strong metrics showing our success. Whilst there are some suggestions for future work in the next section of this report, these are simply ideas and, as has happened often in the natural language processing domain, may not necessarily help overcome the many problems that language processing presents.

The bulk of our work came in building relationships between entities such that we can meaningfully aggregate sentiment between them; something we believe we have achieved to a comfortable degree of accuracy.

5 Reflection and Future Work

5.1 Conclusions

Throughout this project we have learned a lot about sentiment analysis and natural language processing through our experience in writing code to deal with it.

One of the most clear and important lessons we can draw from this area is just how fragile natural language processing is. Not only is it extremely difficult to generalise about language use and abstract rules about the way people use language, but state-of-the-art systems and models for this purpose may only work well in certain cases.

At the very start of our fourth sprint, we believe the entity tagger, *Google Cloud*, we used in the named entity relationship building stage of our pipeline had undergone an update which significantly reduced its performance during our use of it, thus prompting a switch to *spaCy*. It is reasonably safe to assume the team at *Google* believed the latest iteration of their service was the best so far, hence the deployment, but that is not to say it would yield improvements in our use of it. We do not blame *Google* for any poor results but instead used these unfortunate circumstances to highlight just how inconsistent human communication is.

Following this, we expect the performance of our product to improve along with the state of natural language processing however we are aware that improvements are slow and research in this field is in its infancy.

5.2 Reflections

Given the nature of our product, there are few ethical issues to consider. As is the case for the current GSAM application that our product could be integrated into, the software at *Goldman Sachs* runs in-house and has no data stored externally, for security reasons. This drastically reduces the population of users we need to consider when we contemplate the uses and impacts of our product.

The project itself is currently open-source meaning there are many possibilities for people to use the product in ways that we have not currently considered and indeed it would be impossible to keep track of every use of the product. Ultimately, each user must provide their own credentials for using any APIs we have included in the product, so any uses of these APIs which break the terms and conditions will be the user's fault and not our own.

Our product is inherently experimental due to the unpredictable and complex field it resides in — natural language processing. With this comes an implied warning to the users; to use this tool as an aid for research but not as a decision making application. Should the work be merged into the current GSAM system, we would make sure users were not using our product as a source of truth. In a similar light, our product does not make any entity specific conclusions and works as generally as possible. We have kept it as objective as possible, so that the ethical concern of promoting particular companies is avoided.

5.3 Future Work

In such a complex field as natural language processing, there is a lot of opportunity to try new ideas and see what results they give. Under a time constraint we not able to explore all of our ideas however outlined below are some which we believe could have brought some more precision to the product.

Following the development of two machine learning algorithms for the two levels of relationship building, future work could entail a hybrid model handling both with different weightings to reflect significance of the two current models in each context.
When considering the relationships between named entities, articles and research documents cannot be trusted to be as uniform as the current data source, Wikipedia, however we believe a probabilistic approach to them could extract some value from them. Throughout the project we have had an abundance of articles to analyse and thus opens up the opportunity to use the probabilistic approach to building macro-level relationships. We can agree at a high level that companies and their related entities will appear frequently in the same articles and thus we could build a probabilistic model of the links between named entities in a similar way to is done for entities and aspects. As more and more articles are seen, the system builds a better view of which named entities are linked and can use this to contribute to the arc weight between them.
This idea was shelved as we believed using *Wikipedia* as a source of truth was a more reliable approach to the named-entity relationship problem.

Conversely, *Wikipedia* articles are likely to discuss aspects of entities and thus make it a decent source of data to pass into the aspect-entity relationship building algorithm. Due to the more objective nature of these articles, it could be possible to extend the algorithm such that this as a data source is given higher prominence. Future work would entail the weighting of data sources and the conversion of *Wikipedia* articles into forms the micro-level relationship algorithm accepts.

With regards to the GUI, there were a couple of limitations with *Holoviews* which prevented further customisations to the GUI that we had wanted to make. The sizes of the nodes unfortunately cannot be adjusted, but we felt that this would have been a nice way to visualise the importance (e.g. number of relations) of each individual node in the graph. We also wanted to display the output of the pipeline in the webpage while the inputted document is being processed, to show the user what exactly is going on behind the scenes. This ended up being very non-trivial to implement, so was left just to the terminal. Given more time, we would have liked to have found an effective way to get this to work, but there were unfortunately higher-priority issues to deal with throughout the project.

References

[1] Wikipedia contributors. "Noun phrase." *Wikipedia, The Free Encyclopedia.* Wikipedia, The Free Encyclopedia, 5 Jan. 2019. Web. 7 Jan. 2019.

[2] "Cloud Natural Language, Cloud Natural Language API, Google Cloud." Google, Google, cloud.google.com/natural-language/. Web. 7 Jan. 2019.

[3] "Amazon Comprehend - Natural Language Processing (NLP) and Machine Learning (ML)." Amazon, Amazon, aws.amazon.com/comprehend/. Web. 7 Jan. 2019.

[4] "Natural Language Understanding." The Analytics Maturity Model IBM Corporation, 28 Nov. 2016, www.ibm.com/watson/services/natural-language-understanding/. Web. 7 Jan. 2019..

[5] Pontiki, Maria, et al. "SemEval-2016 task 5: Aspect based sentiment analysis." Proceedings of the 10th international workshop on semantic evaluation (SemEval-2016). 2016.

[6] Wikipedia contributors. "Named-entity recognition." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 3 Jan. 2019. Web. 7 Jan. 2019.

[7] Wikipedia contributors. "Bayesian inference." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 2 Jan. 2019. Web. 7 Jan. 2019.

[8] "The Pyramid Web Framework." The Pyramid Web Framework - The Pyramid Web Framework v1.10.1, docs.pylonsproject.org/projects/pyramid/en/1.10-branch/.

[9] contributors, HoloViews. "HoloViews." HoloViews - HoloViews, holoviews.org/.

[10] "NetworkX." NetworkX - NetworkX, networkx.github.io/.

[11] SnehaGunda. "Introduction to Azure Cosmos DB." Microsoft Docs, docs.microsoft.com/en-us/azure/cosmos-db/introduction.

Appendices

A Model Behavior Report

Model Behavior Report

In [1]:
```
from src.pipeline import Pipeline
from matplotlib.pyplot import figure
import pandas as pd
import numpy as np
from src.util.util import graph_summary, sentiment_of_mentions
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
pd.set_option('display.max_colwidth', -1)
```

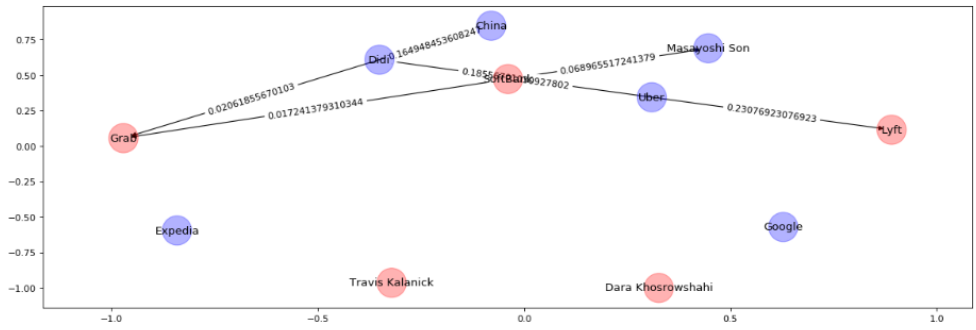
Uber and Softbank

In [3]:
```
%%time
%%capture
pipeline = Pipeline(1437148)
result_data_three, graph_three = pipeline.execute()
```

```
CPU times: user 40.3 s, sys: 1.99 s, total: 42.3 s
Wall time: 1min 27s
```

Graph and Summary

In [4]:
```
figure(num=None, figsize=(18, 6), dpi=80, facecolor='w', edgecolor='k')
graph_three.draw()
```



In [5]:
```
graph_summary(graph_three)[['Entity','Saliance','Sentiment Score','Sentiment Magnitude']]
```

Out[5]:

	Entity	Saliance	Sentiment Score	Sentiment Magnitude
0	China	0.001149	0.000000	0.0
1	Dara Khosrowshahi	0.021915	0.400000	-0.1
2	Didi	0.000765	0.000000	0.0
3	Expedia	0.000822	0.000000	0.0
4	Google	0.020729	0.200000	0.0
5	Grab	0.001149	0.000000	0.0
6	Lyft	0.001150	0.000000	0.0
7	Masayoshi Son	0.001643	0.700000	-0.7
8	SoftBank	0.018287	0.393739	0.0
9	Travis Kalanick	0.093862	1.100000	0.0
10	Uber	0.363802	2.500000	-0.1

Examples in Text

In [6]:
```
%%time
mentions_three = sentiment_of_mentions(1437148,result_data_three['text'])
```

```
CPU times: user 93.7 ms, sys: 3.16 ms, total: 96.9 ms
Wall time: 2.03 s
```

In [7]:
```
sorted_mentions = mentions_three.iloc[np.argsort(mentions_three['Sentiment Score'] * mentions_thre
e['Sentiment Magnitude'])]
```

Top Positive Mentions

In [8]:
```
sorted_mentions[-5:][['Entity','Sentence']]
```

Out[8]:

	Entity	Sentence
4	Uber	The SoftBank investment was key to reforming Uber's dysfunctional board and bringing a detente between warring factions: former Uber CEO Travis Kalanick and early Uber investor Benchmark.
1	Uber	It also paves the way for an overhaul for Uber after a turbulent year, complete with criminal probes, a lawsuit over trade secrets allegedly stolen from Google (GOOGL)'s self-driving car division and internal investigations into its toxic company culture.
18	Travis Kalanick	SoftBank first publicly expressed interest about investing in Uber over the summer, while the startup was still looking for Kalanick's replacement.
35	Didi	SoftBank has previously pumped billions into several ride-hailing companies abroad, including Didi in China, Grab in Southeast Asia and 99 in Brazil.
15	Travis Kalanick	Uber's board also adopted a new voting system that will further lessen Kalanick's influence.

Top Negative Mentions

In [9]:
```
sorted_mentions[:5][['Entity','Sentence']]
```

Out[9]:

	Entity	Sentence
2	Uber	The SoftBank investment was key to reforming Uber's dysfunctional board and bringing a detente between warring factions: former Uber CEO Travis Kalanick and early Uber investor Benchmark.
30	Masayoshi Son	But SoftBank CEO Masayoshi Son had repeatedly teased the possibility -- some might say threat -- of investing in Lyft instead, including as recently as last month.
19	Dara Khosrowshahi	Dara Khosrowshahi, the former CEO of Expedia (EXPE), took Kalanick's place in August after a lengthy search.
10	Travis Kalanick	The SoftBank investment was key to reforming Uber's dysfunctional board and bringing a detente between warring factions: former Uber CEO Travis Kalanick and early Uber investor Benchmark.
21	Dara Khosrowshahi	But SoftBank CEO Masayoshi Son had repeatedly teased the possibility -- some might say threat -- of investing in Lyft instead, including as recently as last month.

Sentiment Analysis Evaluation

In [10]:
```
grouped_sentiment = sorted_mentions.groupby('Entity').describe()[[('Sentiment Score','mean'),('Sentiment Magnitude','mean')]]
sorted_grouped_sentiment = grouped_sentiment.iloc[np.argsort(grouped_sentiment[('Sentiment Magnitude','mean')]
                                                             * grouped_sentiment['Sentiment Score','mean'])]
sorted_grouped_sentiment.round(5)
```

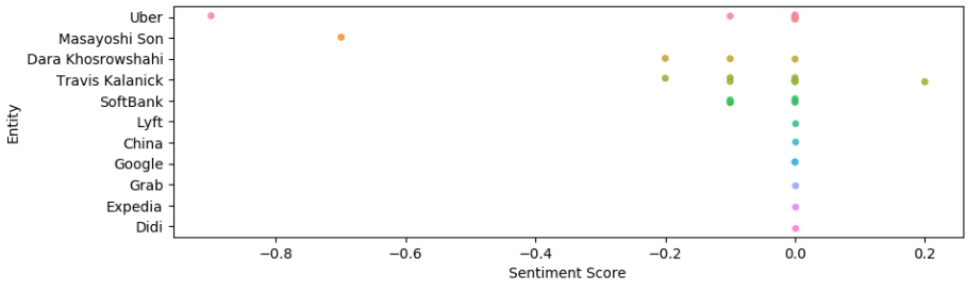
Out[10]:

	Sentiment Score	Sentiment Magnitude
	mean	mean
Entity		
Masayoshi Son	-0.70000	0.70000
Dara Khosrowshahi	-0.10000	0.10000
Uber	-0.10000	0.10000
SoftBank	-0.05000	0.05000
Travis Kalanick	-0.02222	0.06667
China	0.00000	0.00000
Didi	0.00000	0.00000
Expedia	0.00000	0.00000
Google	0.00000	0.00000
Grab	0.00000	0.00000
Lyft	0.00000	0.00000

Sentiment Score Distribution

In [11]:
```
figure(num=None, figsize=(10, 3), dpi=100, facecolor='w', edgecolor='k')
sns.stripplot(x="Sentiment Score", y="Entity",
              data=sorted_mentions, dodge=True, jitter=True,
              alpha=.9, zorder=1)
```

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x146d28ba8>



Change in Sentiment Score and Magnitude after Aggregation

In [12]:
```
joined_analysis = graph_summary(graph_three).set_index('Entity').join(sorted_grouped_sentiment)
change_in_sentiment = pd.DataFrame({
    'Change in Sentiment Score': joined_analysis[ 'Sentiment Score'] - joined_analysis[('Sentiment
 Score', 'mean')],
    'Change in Sentiment Magnitude':  joined_analysis[ 'Sentiment Magnitude'] - joined_analysis[(
'Sentiment Magnitude', 'mean')]}).round(5)
change_in_sentiment = change_in_sentiment.iloc[np.argsort(change_in_sentiment['Change in Sentiment
 Score']
                                                          * change_in_sentiment['Change in Sentiment
 Magnitude'])]
change_in_sentiment
```

Out[12]:

Entity	Change in Sentiment Score	Change in Sentiment Magnitude
Masayoshi Son	1.40000	-1.40000
Uber	2.60000	-0.20000
Dara Khosrowshahi	0.50000	-0.20000
Travis Kalanick	1.12222	-0.06667
SoftBank	0.44374	-0.05000
China	0.00000	0.00000
Didi	0.00000	0.00000
Expedia	0.00000	0.00000
Google	0.20000	0.00000
Grab	0.00000	0.00000
Lyft	0.00000	0.00000

Loss Evaluation

In [13]:
```
ground_truth = {'Masayoshi Son': (-1,1), 'Dara Khosrowshahi': (-1,1),'Uber': (0,1),'SoftBank':(0,1
),'Travis Kalanick':(0,1),
                'China': (0,0),'Didi': (0,0),'Expedia':(0,0), 'Google':(0,0), 'Grab':(0,0), 'Lyft'
:(0,0)  }
```

In [14]:
```
pre_agg_loss = 0
post_agg_loss = 0
for index, row in sorted_grouped_sentiment.iterrows():
    post_agg_loss = abs(row[('Sentiment Score','mean')] - ground_truth[index][0]) + abs(row[('Sent
iment Magnitude','mean')] - ground_truth[index][1]) + post_agg_loss
for index, row in graph_summary(graph_three).iterrows():
    pre_agg_loss = abs(row['Sentiment Score'] - ground_truth[row['Entity']][0]) + abs(row['Sentime
nt Magnitude'] - ground_truth[row['Entity']][1]) + pre_agg_loss
```

In [15]:
```
print('[Pre Aggregation Loss] : %.3f' % pre_agg_loss)
print('[Post Aggregation Loss]: %.3f' % post_agg_loss)
```

```
[Pre Aggregation Loss] : 13.194
[Post Aggregation Loss]: 5.356
```

B NE - Aspect Validation Results

B.1 Threshold Altering & Tagged Document Analysis

knowledge_threshold	1	2	3	4	5
Mean Matches	45.3%	39.9%	25.2%	16.4%	15.2%
Mean Wrong Entity	16.4%	8.0%	7.6%	5.3%	5.1%

Table 4: Performance for different knowledge thresholds — mean percentages calculated from 8 documents

relation_threshold	0.05	0.1	0.2	0.3	0.5
Mean Matches	37.2%	39.9%	32.2%	28.2%	20.9%
Mean Wrong Entity	9.2%	8.0%	13.0%	5.3%	1.6%

Table 5: Performance for different relation thresholds — mean percentages calculated from 8 documents

clear_relation_threshold	> 0.3, 1.5*	> 0.3, 2*	> 0.5, 1.5*	> 0.5, 2*
Mean Matches	44.1%	39.9%	31.5%	30.1%
Mean Wrong Entity	11.2%	8.0%	7.6%	7.3%

Table 6: Performance for different clear relation thresholds (> 0.3, 1.5* represents that to achieve, the likelihood for an NE must be greater than 0.3 and 1.5 times the likelihood of all other NEs) — mean percentages calculated from 8 documents

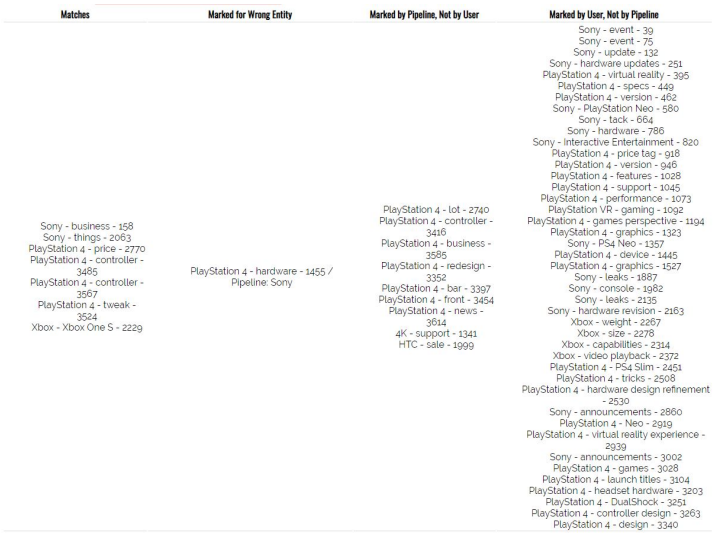
Sliding Window Size	1	2	3	4
Mean Matches	28%	37%	40.1%	32.1%
Mean Wrong Entity	10.0%	9.2%	9.9%	9.5%

Table 7: Performance for different sizes of sliding window, a sliding window contains current sentence and previous

	Lower Quartile	Median	Upper Quartile
Closest NE	25	65	113
Closest NE Before	34	96	190
Closest NE After	87	186	460

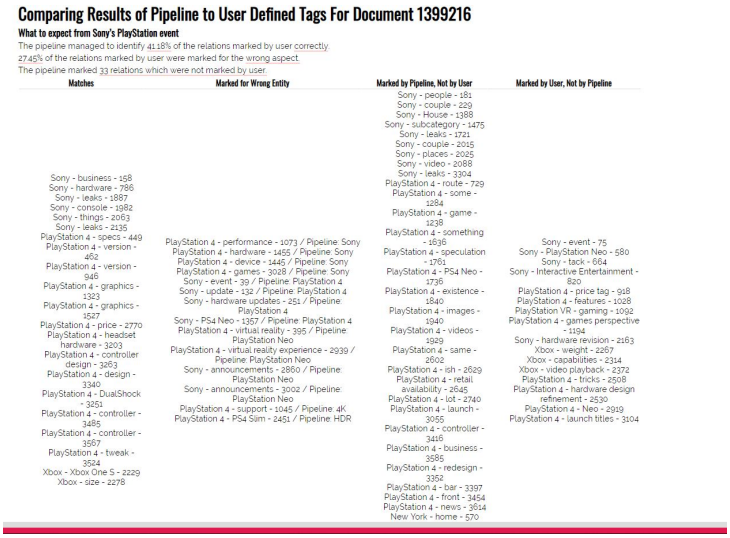
Table 8: Statistical summaries of distance between aspects and named entities — results of 8 documents tagged by human analysed to form results

B.2 Algorithm Modification



Matches	Marked for Wrong Entity	Marked by Pipeline, Not by User	Marked by User, Not by Pipeline
Sony - business - 158 Sony - things - 2063 PlayStation 4 - price - 2770 PlayStation 4 - controller - 3485 PlayStation 4 - controller - 3567 PlayStation 4 - tweak - 3524 Xbox - Xbox One S - 2229	PlayStation 4 - hardware - 1455 / Pipeline: Sony	PlayStation 4 - lot - 2740 PlayStation 4 - controller - 3416 PlayStation 4 - business - 3585 PlayStation 4 - redesign - 3352 PlayStation 4 - bar - 3397 PlayStation 4 - front - 3454 PlayStation 4 - news - 3614 4K - support - 1341 HTC - sale - 1999	Sony - event - 39 Sony - event - 75 Sony - update - 132 Sony - hardware updates - 251 PlayStation 4 - virtual reality - 395 PlayStation 4 - specs - 449 PlayStation 4 - version - 462 Sony - PlayStation Neo - 580 Sony - tack - 664 Sony - hardware - 786 Sony - Interactive Entertainment - 820 PlayStation 4 - price tag - 918 PlayStation 4 - version - 946 PlayStation 4 - features - 1028 PlayStation 4 - support - 1045 PlayStation 4 - performance - 1073 PlayStation VR - gaming - 1092 PlayStation 4 - games perspective - 1194 PlayStation 4 - graphics - 1323 Sony - PS4 Neo - 1357 PlayStation 4 - device - 1445 PlayStation 4 - graphics - 1527 Sony - leaks - 1887 Sony - console - 1982 Sony - leaks - 2135 Sony - hardware revision - 2163 Xbox - weight - 2267 Xbox - size - 2278 Xbox - capabilities - 2314 Xbox - video playback - 2372 PlayStation 4 - PS4 Slim - 2451 PlayStation 4 - tricks - 2508 PlayStation 4 - hardware design refinement - 2530 Sony - announcements - 2860 PlayStation 4 - Neo - 2919 PlayStation 4 - virtual reality experience - 2939 Sony - announcements - 3002 PlayStation 4 - games - 3028 PlayStation 4 - launch titles - 3104 PlayStation 4 - headset hardware - 3203 PlayStation 4 - DualShock - 3251 PlayStation 4 - controller design - 3263 PlayStation 4 - design - 3340

Figure 9: Performance of algorithm without contacting the NE-NE relation graph to distinguish in multiple entity cases



Comparing Results of Pipeline to User Defined Tags For Document 1399216

What to expect from Sony's PlayStation event

The pipeline managed to identify 41.18% of the relations marked by user correctly.
27.45% of the relations marked by user were marked for the wrong aspect.
The pipeline marked 33 relations which were not marked by user.

Matches	Marked for Wrong Entity	Marked by Pipeline, Not by User	Marked by User, Not by Pipeline
Sony - business - 158 Sony - hardware - 786 Sony - leaks - 1887 Sony - console - 1982 Sony - things - 2063 Sony - leaks - 2135 PlayStation 4 - specs - 449 PlayStation 4 - version - 462 PlayStation 4 - version - 946 PlayStation 4 - graphics - 1323 PlayStation 4 - graphics - 1527 PlayStation 4 - price - 2770 PlayStation 4 - headset hardware - 3203 PlayStation 4 - controller design - 3263 PlayStation 4 - design - 3340 PlayStation 4 - DualShock - 3251 PlayStation 4 - controller - 3485 PlayStation 4 - controller - 3567 PlayStation 4 - tweak - 3524 Xbox - Xbox One S - 2229 Xbox - size - 2278	PlayStation 4 - performance - 1073 / Pipeline: Sony PlayStation 4 - hardware - 1455 / Pipeline: Sony PlayStation 4 - device - 1445 / Pipeline: Sony PlayStation 4 - games - 3028 / Pipeline: Sony Sony - event - 39 / Pipeline: PlayStation 4 Sony - update - 132 / Pipeline: PlayStation 4 Sony - hardware updates - 251 / Pipeline: PlayStation 4 Sony - PS4 Neo - 1357 / Pipeline: PlayStation 4 PlayStation 4 - virtual reality - 395 / Pipeline: PlayStation Neo PlayStation 4 - virtual reality experience - 2939 / Pipeline: PlayStation Neo Sony - announcements - 2860 / Pipeline: PlayStation Neo Sony - announcements - 3002 / Pipeline: PlayStation Neo PlayStation 4 - support - 1045 / Pipeline: 4K PlayStation 4 - PS4 Slim - 2451 / Pipeline: HDR	Sony - people - 181 Sony - couple - 229 Sony - House - 1388 Sony - subcategory - 1475 Sony - leaks - 1721 Sony - couple - 2015 Sony - places - 2025 Sony - video - 2088 Sony - leaks - 3304 PlayStation 4 - route - 729 PlayStation 4 - some - 1284 PlayStation 4 - game - 1438 PlayStation 4 - something - 1636 PlayStation 4 - speculation - 1761 PlayStation 4 - PS4 Neo - 1736 PlayStation 4 - existence - 1840 PlayStation 4 - images - 1940 PlayStation 4 - videos - 1929 PlayStation 4 - same - 2602 PlayStation 4 - ish - 2629 PlayStation 4 - retail availability - 2645 PlayStation 4 - lot - 2740 PlayStation 4 - launch - 3055 PlayStation 4 - controller - 3416 PlayStation 4 - business - 3585 PlayStation 4 - redesign - 3352 PlayStation 4 - bar - 3397 PlayStation 4 - front - 3454 PlayStation 4 - news - 3614 New York - home - 570	Sony - event - 75 Sony - PlayStation Neo - 580 Sony - tack - 664 Sony - Interactive Entertainment - 820 PlayStation 4 - price tag - 918 PlayStation 4 - features - 1028 PlayStation VR - gaming - 1092 PlayStation 4 - games perspective - 1194 Sony - hardware revision - 2163 Xbox - weight - 2267 Xbox - capabilities - 2314 Xbox - video playback - 2372 PlayStation 4 - tricks - 2508 PlayStation 4 - hardware design refinement - 2530 PlayStation 4 - Neo - 2919 PlayStation 4 - launch titles - 3104

Figure 10: Performance of algorithm now contacting the NE-NE relation graph to distinguish in multiple entity cases

B.3 Example of Model Learning

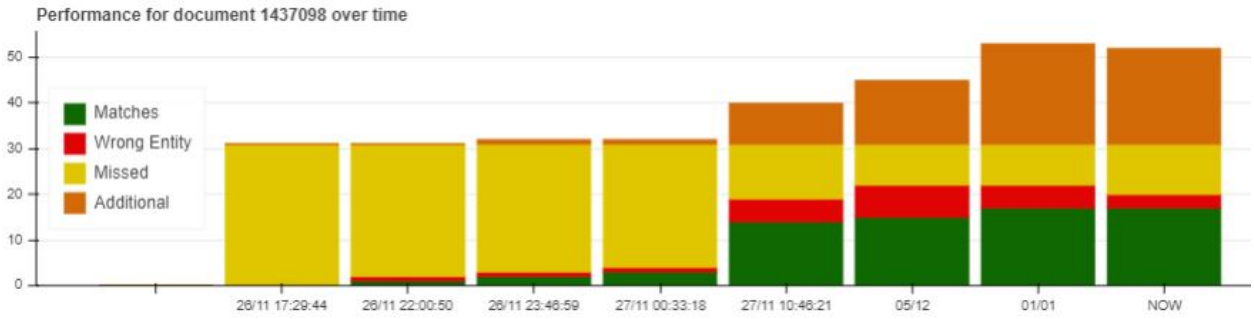


Figure 11: Graph generated within Validation UI showing performance improvement as model learns more about the named entities in the document

C Macro-Level Validation Data

Below are the complete processed results for the output for the named entity relation portion of the pipeline. Discussion for these results can be found in section 6.3 - Named Entity Relations Validation.

Key:
Correct — Incorrect — ~~Erroneously Tagged~~ (omitted from score)

Apple Inc

[‘~~American~~’, ‘**App Store**’, ‘**Apple Computer**’, ‘**Apple II**’, ‘**Apple Music**’, ‘**Apple Store**’, ‘**Apple TV**’, ‘**Apple Watch**’, ‘~~Central~~’, ‘~~Computer~~’, ‘~~Consumer electronics~~’, ‘~~Data~~’, ‘~~Desktop~~’, ‘~~Digital~~’, ‘EU’, ‘Fortune’, ‘~~Future~~’, ‘~~Glass~~’, ‘Google’, ‘~~Green~~’, ‘Greenpeace’, ‘**HomePod**’, ‘**ICloud**’, ‘**IOS**’, ‘**IPad**’, ‘**IPad Mini**’, ‘**IPad Pro**’, ‘**IPhone**’, ‘**IPhone 4S**’, ‘**IPhone XS**’, ‘**IPod**’, ‘**IPod Touch**’, ‘~~International~~’, ‘~~Irish~~’, ‘**Keynote**’, ‘~~LCD~~’, ‘**Mac**’, ‘**MacBook Pro**’, ‘**MacOS**’, ‘**Macintosh**’, ‘Martin Luther King Jr.’, ‘Microsoft’, ‘~~Mobile~~’, ‘~~Patent~~’, ‘~~Power~~’, ‘**PowerBook**’, ‘~~Pro~~’, ‘**Retina Display**’, ‘Samsung’, ‘~~Smartphone~~’, ‘~~Solar~~’, ‘~~State~~’, ‘**Steve Jobs**’, ‘~~Streaming~~’, ‘**Tim Cook**’, ‘Time’, ‘~~Video~~’, ‘~~Wi-Fi~~’, ‘Windows’, ‘~~World~~’, ‘**macOS**’]

Total	61
Non-Erroneously Tagged	32
Correct	28
Accuracy (%)	87.5

Microsoft

[‘~~Antitrust~~’, ‘~~Application~~’, ‘**Bill Gates**’, ‘~~Board of~~’, ‘~~Cloud~~’, ‘~~Computer~~’, ‘~~Digital~~’, ‘~~Explorer~~’, ‘~~Financial~~’, ‘~~Fish~~’, ‘~~Game~~’, ‘Google’, ‘IBM’, ‘~~Information~~’, ‘~~Interpreter~~’, ‘Linux’,

'**MS-DOS**', '**Microsoft Azure**', '**Microsoft Office**', '~~Mobile~~', '**Office**', '~~Operating~~', '**Paul Allen**', '**Redmond**', '**Satya Nadella**', '~~Servers~~', '~~Software~~', '**Steve Ballmer**', '~~Store~~', '**Surface**', '~~Tablet~~', '~~Video~~', '**Windows**', '**Windows 8**', '**Windows 95**', '**Windows Phone**', '**Windows Server**', '**Windows XP**', '~~World~~']

Total	39
Non-Erroneously Tagged	19
Correct	16
Accuracy (%)	84.21

Sony

['**Akio Morita**', 'Apple', '**BMG**', '**Betamax**', '~~Biobattery~~', '**Columbia Pictures**', '~~Compact~~', '~~Compact Disc~~', '~~Digital~~', '~~Digital Audio Tape~~', '**Discman**', '**Dolby Digital**', '**Ericsson**', '~~Film~~', '~~Financial~~', '~~Floppy~~', '**Gracenote**', '~~Home~~', '~~Japanese~~', '~~Laser~~', '~~Loss~~', '**Masaru Ibuka**', '~~Media~~', '**MiniDisc**', '**Mitsui**', '~~Multinational~~', '~~Network~~', 'Nintendo', 'Philips', '**PlayStation**', '**PlayStation 3**', '**PlayStation 4**', '**PlayStation Portable**', '**PlayStation VR**', 'Samsung', 'Samsung Electronics', '~~Semiconductor~~', '**Sony Bank**', '**Sony Corporation**', '**Sony Corporation of America**', '**Sony Entertainment**', '**Sony Financial**', '**Sony Financial Holdings**', '**Sony Interactive Entertainment**', '**Sony Mobile**', '**Sony Mobile Communications**', '**Sony Music**', '**Sony Music Entertainment Japan**', '**Sony Pictures**', '**Sony/ATV Music Publishing**', '**Spider-Man**', '~~Stick~~', '~~Studios~~', '~~Tablet~~', '~~VHS~~', '~~Video~~', '**Walkman**', 'Xbox 360', '**Xperia**', '~~mm~~']

Total	60
Non-Erroneously Tagged	38
Correct	32
Accuracy (%)	84.21

Samsung

['~~Aircraft~~', '~~Alpha~~', '~~Asian~~', '~~BP~~', '~~Biogen~~', 'CJ Group', '~~Canadian~~', '~~Conglomerate~~', '~~Construction~~', 'DGB Financial Group', '~~Daegu~~', 'Doosan', '**FUBU**', '**Hotel Shilla**', 'Hyosung', 'Hyundai', '~~Intelligent~~', '**Korea Aerospace Industries**', 'Korea Electric Power Corporation', 'Korea Exchange', '~~Korean~~', '**Lee Byung-chul**', '**Lee Kun-hee**', '~~Liquefied~~', '~~Lithium-ion~~', '~~Marine~~', '~~Multinational~~', 'Renault', '**Renault Samsung**', 'Royal Dutch Shell', '**S-LCD**', '**Samsung Aerospace**', '**Samsung C&T**', '**Samsung C&T Corporation**', '**Samsung Electro-Mechanics**', '**Samsung Electronics**', '**Samsung Engineering**', '**Samsung Everland**', '**Samsung Fire & Marine Insurance**', '**Samsung Heavy Industries**', '**Samsung Life Insurance**', '**Samsung Medical Center**', '**Samsung SDS**', '**Samsung Securities**', '**Samsung Techwin**', '**Samsung Total**', 'Seagate Technology', '~~Semiconductor~~', 'Sharp Corporation', 'Shinsegae', '~~Solar~~', 'Sony', '~~South Korean~~', '~~Suwon~~', '**Taylor Energy**', '~~Telecommunications~~', '~~Times~~', '**Toray**', 'Toshiba', '~~Wafer~~']

Total	60
Non-Erroneously Tagged	38
Correct	24
Accuracy (%)	63.16

The Coca-Cola Company

['**Coca-Cola**', '**Coke**', '~~Cola~~', '~~Diet~~', '~~Soft~~', '~~Syrup~~']

Total	6
Non-Erroneously Tagged	2
Correct	2
Accuracy (%)	100

Proctor & Gamble

['Amway', '~~Animal~~', '~~Brand~~', 'Cartel', '~~Child~~', '**Crest**', 'Duracell', '~~Fabric~~', '**Folgers**', '**Gillette**', '**James Gamble**', 'Johnson & Johnson', 'Olestra', '**P&G**', '~~Palm~~', '**Pampers**', '~~Rely~~', 'Sony Pictures Tele '**Tide**', '~~Toxic~~', '~~Tyne~~', 'Unilever', '**William Procter**', 'the World Turns']

Total	24
Non-Erroneously Tagged	16
Correct	8
Accuracy (%)	50

Walmart

['**Advent International**', '~~African Americans~~', 'Aldi', 'Amazon', '~~American~~', '~~Big~~', 'BlackRock', '~~Bruno~~', '~~Canadian~~', '~~Center~~', '~~Christian~~', '~~Data~~', '~~David~~', '**David Glass**', '~~Digital~~', '~~Discount~~', '~~Don~~', '**Doug McMillon**', 'Dukes', 'Equal Employment Opportunity Commission', '**Flipkart**', 'Forbes', '~~Free~~', '~~Frozen~~', '~~Grocery~~', 'Hypermart USA', 'IBM', '~~Institutional~~', 'Iowa State University', '**Jet.com**', '~~Joint~~', 'Kmart', 'Kroger', '~~Labor~~', '~~Layaway~~', '**Lee Scott**', '~~Leeds~~', '~~Low Price~~', '~~Marc~~', '**Marc Lore**', 'McDonald's', '**Moosejaw**', '~~Multinational~~', 'National Labor Relations Board', 'New York Stock Exchange', '~~Organic~~', 'Paraná', '~~Pet~~', '~~Pharmacy~~', '~~Retail~~', 'Rio Grande', '~~Sam~~', '**Sam Walton**', 'Shinsegae', '~~Slogan~~', '~~Solar~~', '**Sonae**', 'Sul', '~~Supermarkets~~', '**The Jensen Project**', 'The New York Times', '**Vudu**', '**Wal-Mart**', '**Walmart Brasil**', '**Walmart Canada**', '**Walmart Labs**', '**Walmart Neighborhood Market**', '**Walton**', '~~Warehouse~~', 'Whole Foods']

Total	70
Non-Erroneously Tagged	38
Correct	18
Accuracy (%)	47.37

Unilever

['**Alberto-Culver**', 'Asia Pulp & Paper', '**Batchelors**', '**Ben & Jerry's**', '**Best Foods**', '**Brooke Bond**', '~~Can't Believe~~', '~~Cartel~~', '~~Certified~~', '~~Cornflakes~~', '~~Deforestation~~', '**Flora**', '~~Food~~', '~~Gelato~~', 'Greenpeace', '**Hellmann's**', '**Hindustan Unilever**', '~~House~~', '**Knorr**', 'Kodaikanal', '**Lever Brothers**', '**Maille**', '**Margarine Unie**', '**PG Tips**', '~~Palm~~', '**Paul Polman**', '~~Plantations~~', '**Pond's**', 'Procter & Gamble', '**Pukka Herbs**', '~~Rotterdam~~', '**Sunsilk**', '**TRESemmé**', '~~Tahini~~', '**Talenti**', '~~Thermometer~~', '**Vaseline**', '~~Wall~~', '**Wilmar International**']

Total	39
Non-Erroneously Tagged	25
Correct	21
Accuracy (%)	84

Disney

['**Disney**', 'Fox', '~~Walt~~', '**Walt Disney**']

Total	4
Non-Erroneously Tagged	3
Correct	2
Accuracy (%)	66.67

Ford

['**Ford**', '~~Fuel~~', '~~Model~~', '~~Vehicle~~']

Total	4
Non-Erroneously Tagged	1
Correct	1
Accuracy (%)	100

Amazon

['**A9.com**', '~~Air~~', '**Amazon Go**', '**Amazon Marketplace**', '**Amazon.com**', 'Apple', '**Audible**', '**Audible.com**', '~~Audio~~', '~~Audiobooks~~', 'CIA', '~~Cloud~~', '~~Comic~~', '~~Domain~~', 'EBay', 'Ebay',

'Federal Aviation Administration', 'Federal Trade Commission', 'Fox News', '~~German~~', '~~Imprint~~', '~~Inc.~~', '**Jeff Bezos**', '~~Jewelry~~', '~~Long~~', 'McDonald's', '~~Mobile~~', '~~Platform~~', '**Shelfari**', '~~Social~~', '~~Software~~', '~~Text~~', '**The Washington Post**', 'Time', '~~Toys~~', '**United States Postal Service**', '~~Video~~', '**Whole Foods**', '**Whole Foods Market**', '~~Wind farm~~', '~~World~~']

Total	41
Non-Erroneously Tagged	21
Correct	12
Accuracy (%)	57.14

D Demonstration

D.1 Main page

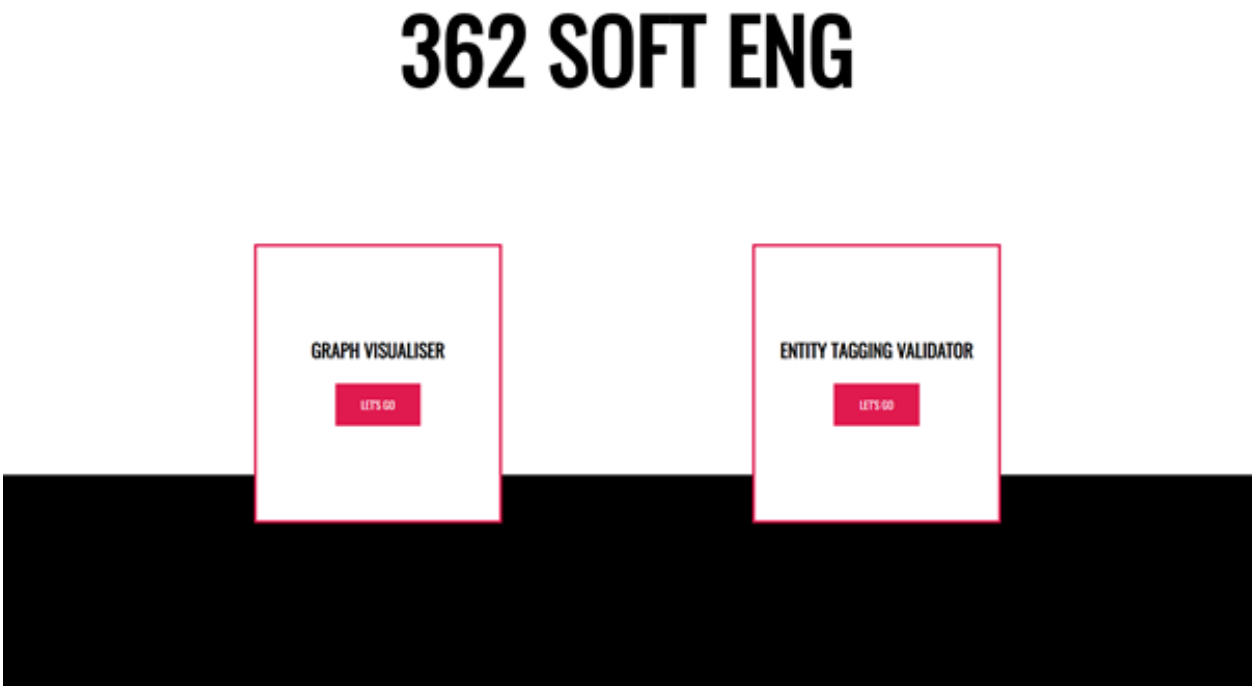


Figure 12: The index page of the web app, with links to both the Holoviews graph visualiser, and the entity tagging validator.

D.2 Launch page

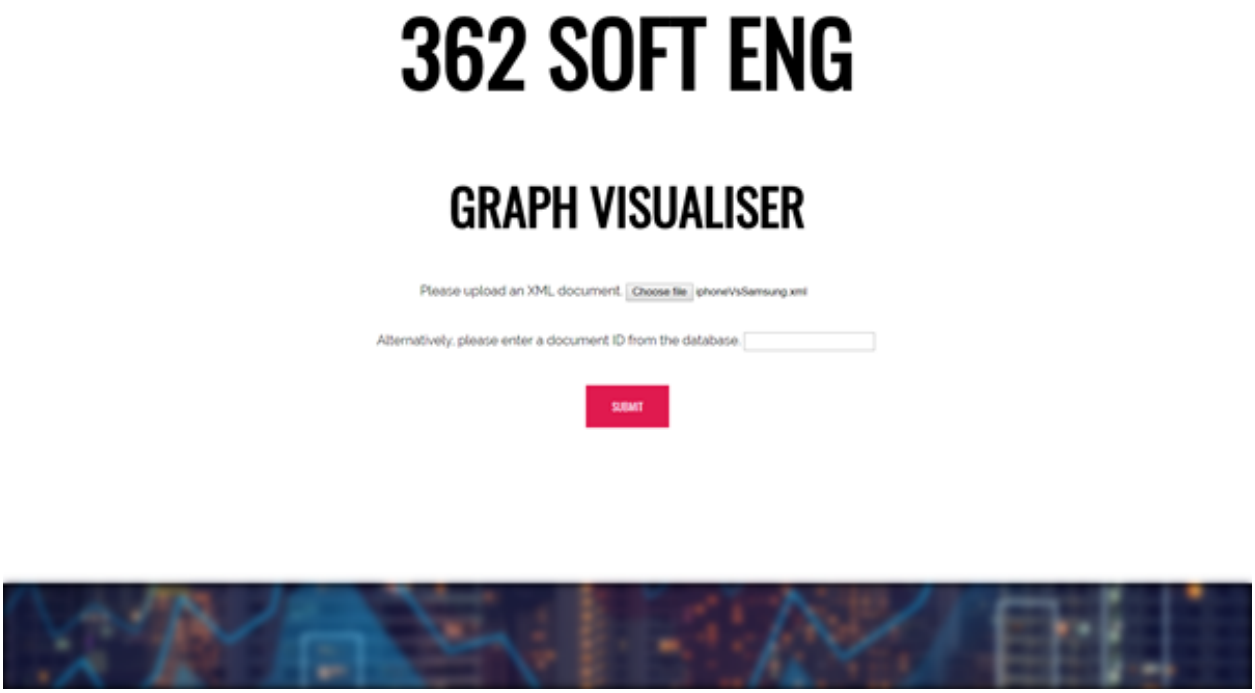


Figure 13: The launch page for the graph visualiser. Users are able to either upload an XML document of their choice, or select a document by ID from the database of previously-processed documents.

D.3 Output using Holoviews

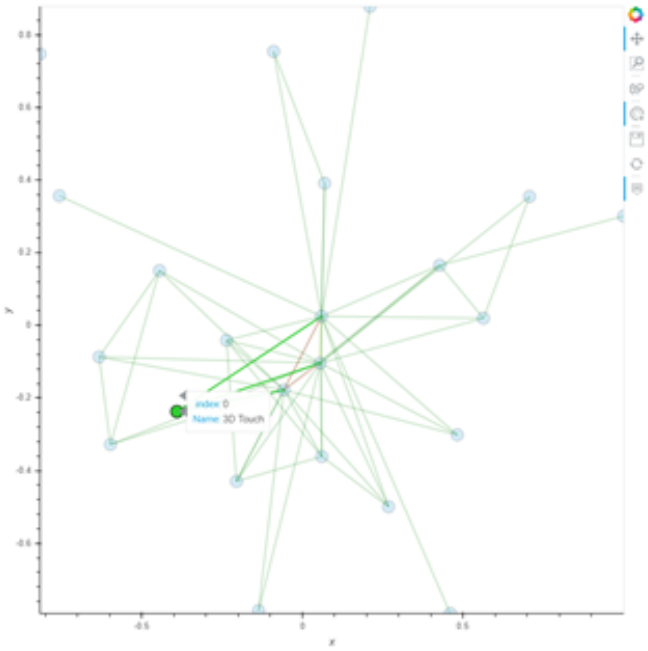


Figure 14: The Holoviews output of a sample XML file, with one of the nodes highlighted to show additional information. The arcs between nodes are coloured depending on their weights.

D.4 Entity Aspect Tagging

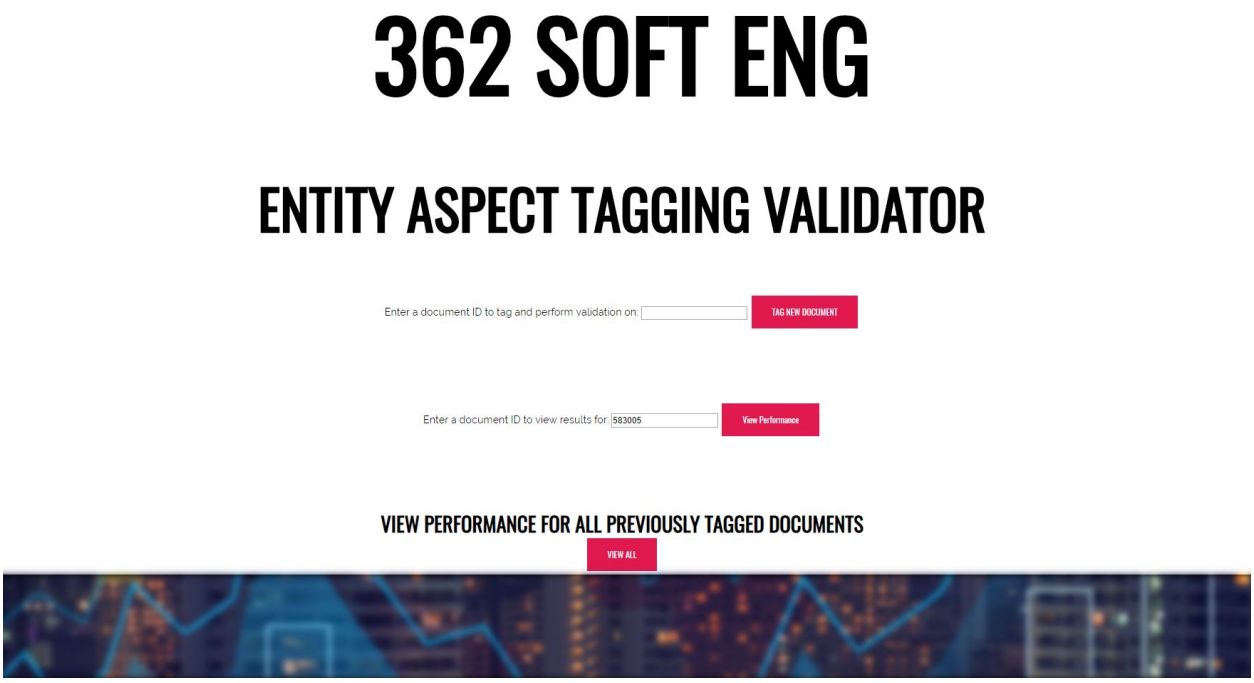


Figure 15: Screenshot of NE - A relation validation home section of UI, with links to tagging and performance review

D.4.1 Document Tagging

Figure 16: Screenshot of User Interface used to tag NE - A relations in the document

D.4.2 Viewing Entity Aspect Tagging Results

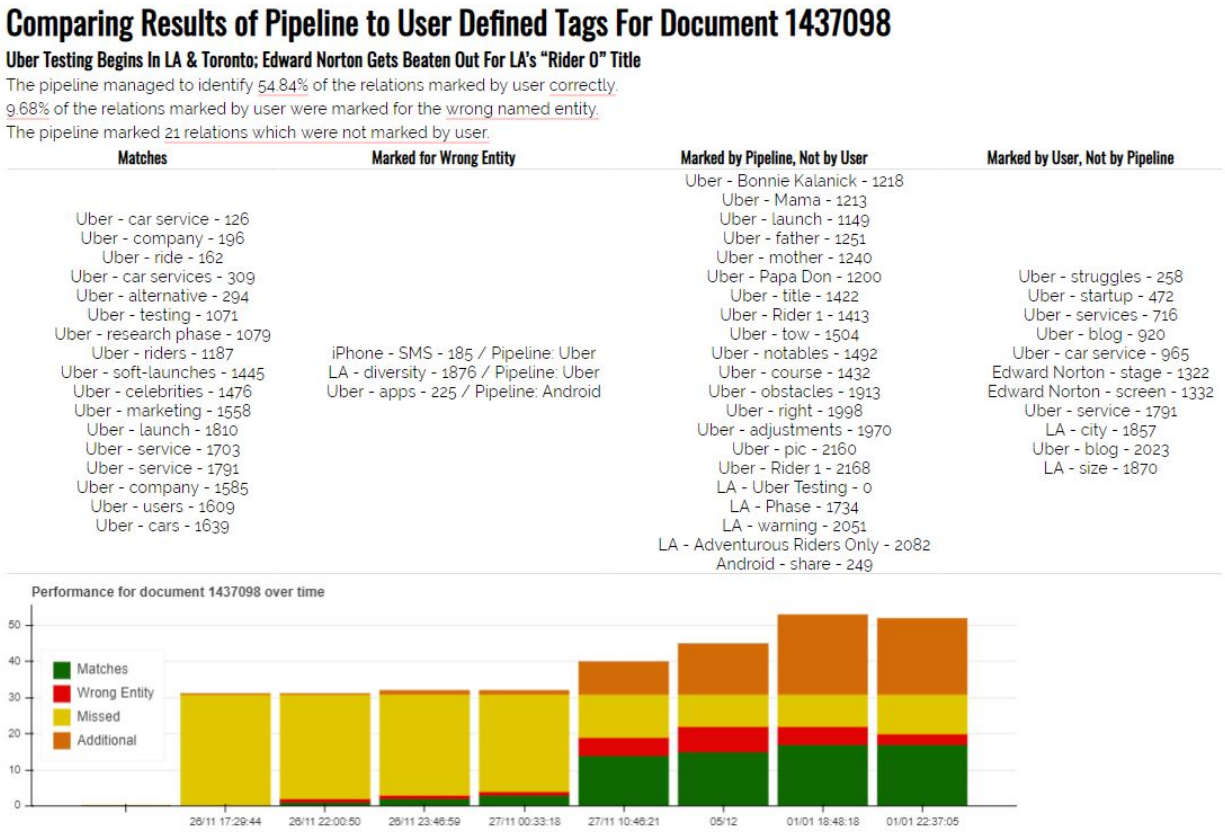


Figure 17: Screenshot of User Interface showing results of system against user defined tags