Sommelier: A Recommender System

By Peter Chamberlin
May 3, 2013

BSc Information Systems and Management Project Report,
Birkbeck College, University of London.

Abstract

Recommender systems are prevalent on the Web, and an important selling point for businesses such as Amazon and Netflix, who trade on the quality of their recommendations to users. Academic research, thanks partly to exceptional interest in the field precipitated by the Netflix Prize, has tended to concentrate research in the field of movie recommendations. Service oriented architectures have grown in popularity on the Web in recent years. Such architectures lend themselves to scalability and maintainability, and in particular satisfy the needs of organizations who deliver the same data and content to diverse set of clients, such as websites and mobile apps. In this project I implement a recommender system as a web service for wine recommendations. Using as a starting point the wines, authors and ratings data from wine website Decanter.com I implement a web API serving JSON data using the Python language, MySQL, the Flask framework, and a number of Python libraries. Implementing a primitive collaborative filtering system but finding my source data to be exceptionally sparse for recommendation, I explore and implement imputation techniques using matrix factorization and singular value decomposition in an attempt to boost the system's ability to make good recommendations. I find that both matrix factorization and singular value decomposition significantly improve the recommendation quality of the system, but due to time contraints I am forced to leave some avenues of inquiry unexplored.

Contents

1 Introduction

1.1 Online Recommender Systems

Since their origin in the mid-1990s with systems such as Tapestry [16] and GroupLens [33], recommender systems have become ubiquitous on the World Wide Web, being employed by some of the worlds largest online businesses as core parts of their offering.

The growth of the Web has given companies the ability to gather unprecedented amounts of data about their users' preferences, both explicitly collected and inferred from their behaviour, while at the same time enabling them to reach users for less cost more often than ever before.

Amazon's system of product recommendations using item-to-item collaborative filtering is regarded as a "killer feature"[14], and is one of the defining features of the Amazon website. The importance of recommendations to Amazon is reflected in their stated mission, "to delight our customers by allowing them to serendipitously discover great products"[14].

Another company which, like Amazon, is synonymous with recommender systems is Netflix, an "Internet television network" [22]. In October 2006 Netflix launched "The Netflix Prize", a competition with a $1,000,000 Grand Prize on offer for any team which could beat their own Cinematch recommender system by "at least 10%" accuracy over a fixed set of data [24]. In 2009 the prize was awarded to the BellKor's Pragmatic Chaos team, who had improved on Netflix's own system by 10.06% [23].

As well as online stores using recommender systems to recommend products a user might be interested in, there are systems ...

1.2 Aims and Objectives

My interest in recommender systems is founded in their variety and ubiquity. It occurred to me that I encounter, and am the subject of, dozens of these systems in my everyday life. Whether it's Twitter or Facebook recommending interesting interesting people or content to me, Amazon recommending me a book or film, or even a supermarket targeting special offers to me, I interact with recommender systems all the time. I am fascinated both by how these systems work theoretically and by how they are implementated in practice. As such the core objective of my project is to implement a recommender system for a web or mobile application.

I have kindly been permitted to freely use data from Decanter.com's wine reviews database [9] for my project. The sphere of wine recommendations is particularly interesting; wine is at first glance a narrow subject, but it is a nuanced one. Among oenophiles there is an emphasis on personal taste and a strong tradition of rating and grading.

In this project I aim to build a recommender system based on Decanter.com's wine database, identifying and overcoming the challenges associated with the implementation of a real-world recommender system.

The most important aspect of any recommender system may be considered to be the quality of its recommendations, and I intend to focus very strongly on recommendation quality. Nevertheless I do not want to overlook the practical implementation of the system. I aim to produce a system satisfactory in both its recommendations

and its performance as a web service, with a robust and elegant implementation in code.

I will not build any kind of graphical interface for the system, but will instead provide a machine readable API, and where necessary batch scripts and command line tools for preparing and manipulating data.

I aim to satisfy several of the most common use cases for wine recommendation, including user-item, item-item and user-user recommendations.

Overall I hope this project will be a strong learning experience for me...

2 Literature Review and Context

2.1 Recommender Systems

Although the term *recommender system* was not coined until 1997 by Resnick and Varian (Resnick and Varian, 1997 [34]), the Tapestry system of 1992 (Goldberg et al., 1992 [16]) is widely recognised as the first of the kind (Su and Khoshoftaar, 2009 [40]). The creators of Tapestry coined the term *collaborative filtering* to describe their method of recommendation, which is based on the principle that if two users rate a number of the same items in a similar manner, then it can be assumed that they will rate other new items similarly (Su and Koshgoftaar, 2009[40]).

Su and Khoshgoftaar (2009 [40]) point out that although collaborative filtering has been widely adopted as a general term to describe systems making recommendations, many such systems do not explicitly collaborate with users or exclusively filter items for recommendation. In fact the term recommender system itself was coined by Resnick and Varian in response to the inadequacy of collaborative filtering for describing the plurality of techniques that were beginning to become associated with it. Recommender system is intentionally a broader term, describing any system that "assists and augments [the] natural social process" of recommendation (Resnick and Varian, 1997 [34]).

In his 2002 survey of the state of the art in recommender systems, Robin Burke presents five categories of recommender (Burke, 2002 [5]). I have reproduced his table of recommendation techniques in Table 1. Burke presents five main categories of filtering technique: collaborative, *content-based*, *demographic*, *utility-based* and *knowledge-based* (Burke, 2002 [5]).

These five kinds of system are classified using three properties: *background data*, *input data* and *process* (Burke, 2002 [5]). Background data is that which exists before and independant of the recommendation, such as previous stated preferences of a group of users U for a set of items I. Input data is that which is considered by the system when making recommendations, such as the ratings of an individual u of items in I. Process is the method by which recommendations are arrived at by application of the input data and the background data (Burke, 2002 [5]). These three aspects provide a good lens through which to compare the different approaches.

2.1.1 Collaborative Filtering

Collaborative filtering is "the technique of using peer opinions to predict the interest of others" (Claypool et al., 1999 [6]), and uses the ratings of a set of users U over a set of items I as background data, and the ratings of each individual user u of items in I as input data. The process of recommendation is to identify similar users to u in U, and then to infer their preferences for items in I based on the preferences of those similar users (Burke, 2002 [5]).

In 2002 Burke described collaborative filtering as the most widely used and mature of these types (Burke, 2002 [5]), citing GroupLens (Resnick, 1994 [33]) and Tapestry (Goldberg, 1992 [16]) as important examples of such systems. From my observations of more recent literature that remains the case. Collaborative filtering is still certainly among the most widely used of these techniques, with Su and Khoshgoftaar describing a large number of collaborative filtering-based systems in their 2009 survey (Su and Koshgoftaar, 2009 [40]).

Table 1: Recommendation Techniques, reproduced from Burke, 2002 [5]

Technique	Backgroud	Input	Process
Collaborative	Ratings from U of items in I.	Ratings from u of items in I.	Identify users in U similar to u, and extrapolate from their ratings of i.
Content-based	Features of items in I.	u's ratings of items in I.	Generate a classifier that fits u's rating behaviour and use it on i.
Demographic	Demographic information about U and their ratings of items in I.	Demographic information about u.	Identify users that are demographically similar to u, and extrapolate from their ratings of i.
Utility-based	Features of items in I.	A utility function over items in I that describes u's preferences.	Apply the function to the items and determine i's rank.
Knowledge-based	Features of items in I. Knowledge of how these items meet a user's needs.	A description of u's needs or interests.	Infer a match between i and u's need.

Even in its most basic form there are many potential methods for measuring similarity between users in a collaborative filtering system. The most simple are such distance metrics such as Manhattan distance or Euclidean distance (Segaran, 2007 Ch.2 [36]). One of the most commonly used and relatively simple measures of similarity is the Pearson correlation coefficient, which is even used in quite advanced systems (Segaran, 2007 Ch.2 [36]).

There are a number of issues associated with the application of pure collaborative filtering, however, which hamper its application in many instances:

- Early rater problem, whereby an item entering the system with no ratings has no chance of being recommended (Claypool et al., 1999 [6]).
- Sparsity problem. Where there is a high ratio of items to ratings it may be difficult to find items which have been rated by enough users to use as the basis for recommendation (Claypool et al., 1999 [6])(Su and Koshgoftaar, 2009 [40]).
- Grey sheep, which are users who neither conform nor disagree with any other group in a significant way, making it very difficult to recommend items for them (Claypool et al., 1999 [6])(Su and Koshgoftaar, 2009 [40]).
- Synonymy, whereby identical items have different names or entries. In this case the collaborative filtering systems are unable to detect that they are the same item (Su and Koshgoftaar, 2009 [40]).

- Vulnerability to shilling, where a user may submit a very large number of ratings to manipulate the recommendation of items (Su and Koshgoftaar, 2009 [40]).

Much of the variation between collaborative filtering techniques described by Su and Khoshgoftaar (2009 [40]) can be attributed to efforts by system developers to minimise the impact of one or more of these problems by introducing auxiliary methods.

2.1.2 Content-based

In content-based filtering systems the features of items in I form the background data, and the user u's ratings serve as the input data. The process of recommendation depends on building a classifier that can predict u's rating behaviour in respect of an item i based on u's previous ratings of items in I (Burke, 2002 [5]).

Content-based, like collaborative filtering, builds up a long term profile of a user's interests and preferences (Burke, 2002 [5]). Researchers have developed systems which successfully combined content-based and collaborative filtering systems to produce better recommendations. Claypool et al. describe such a system which predicts ratings based on a weighted average of results from each system, with the weighting varying per user in order to achieve optimal results (Claypool et al., 1999 [6]).

2.1.3 Demographic

Demographic recommender systems use demographic information about users U and their ratings in I as background data, with demographic information about u as the input data. The recommendation process depends on matching u with other demographically similar users in U (Burke, 2002 [5]).

2.1.4 Utility-based

Utility-based systems use features of items in I as their background data, and depend on a utility function representing u's preferences in order to arrive at recommendations. The process is the application of the function for u to the items I (Burke, 2002 [5]).

2.1.5 Knowledge-based

Knowledge-based systems, like utility-based systems, draw on the features of items in I as their background data. As input data they require information about u's needs. The process is to infer a need for one or more items in I (Burke, 2002 [5]).

Knowledge-based systems can also benefit from being combined with collaborative filtering systems, in a way that may improve ann under-performing knowledge-based system, but without realising pure collaborative filtering's ability to identify niche groups (Burke, 1999 [3]).

2.2 Recommending Wines.

Recommender systems for wines are not a new idea, being typical of the kind of item many systems are designed to recommend. Burke developed the VintageExchange FindMe recommender system in 1999 (Burke, 1999 [2]), and there is at least one patent pending with the WIPO for a wine recommender system as an aid to salespeople or waiting staff (Ward et al., 2012 [42]).

Burke's FindMe, a knowledge-based recommender system, "required approximately one person-month of knowledge engineering effort" (Burke, 1999b [3]) in order to perform well. Such knowledge-based systems are required to recognise the importance of given product features, and so require a great deal of priming (Burke, 1999b [3]).

Another wine recommender system is the Tetherless World Wine Agent (TWWA) (Patton, 2010 [26]). The TWWA project is primarily concerned with knowledge representaion and the Semantic Web, presenting a common and collaborative ontology for wine with which users can share wine recommendations across their social networks (TWWA, 2013 [41]). The system does not automatically tailor recommendations to users, although this is stated as a target for future work (TWWA, 2013 [41]).

2.3 Evaluating Recommender Systems

Shani and Gunawardana (2011 [37]) describe several approaches to the evaluation of recommender systems, including different experimental settings and a number of different statistical methods. They highlight the different aspects of recommender systems that can be evaluated, including prediction accuracy; how accurately does the system predict ratings or preferences, item-space coverage; what proportion of the items in the system are recommendable, and user-space coverage; the proportion of users or interactions the system can provide recommendations for (Shani and Gunawardana, 2011 [37]).

In terms of prediction accuracy mean absolute error (MAE), and root mean squared error (RMSE) are the most popular measures of prediction accuracy (Shani and Gunawardana, 2011 [37]), which Su and Khoshgoftaar (2009 [40]) call "predictive accuracy metrics".

There is some criticism of accuract metrics applied to recommender systems. McNee, Riedl and Konstan (2006 [19]) cite Amazon.com as a case in point, where on the page for a book by a given author you will find recommendations for other of the same author's books. They argue that this is not interesting for the user, and that there is a need for recommender systems developers to look beyond simply the ratability of systems, concluding, "It is now time to also study recommenders from a user-centric perspective to make them not only accurate and helpful, but also a pleasure to use".

2.4 Web Services and Service Oriented Architecture

Service oriented architecture (SOA) is an increasingly widely used paradigm in enterprise applications, enabling such benefits as modularity, distribution and reuse of services (Sheikh et al., 2011 [38]). SOA systems are loosely coupled, and lend themselves to supporting heterogeneous applications (Benatallah and Nezhad, 2008

[1]), such as would be the case for a service providing data for a variety of websites or mobile applications. Web SOAs encompass traditional WS-* web services approaches, such as SOAP and XML, as well as the more recently emerging RESTful (REpresentational State Transfer) approach, which takes advantage of the existing communications protocol of HTTP, and associated protocols such as SSL (Benatallah and Nezhad, 2008 [1]). RESTful and WS-* have their benefits and weaknesses, and the choice of either would be dependant on the needs of any given service.

Pautasso et al. (2008 [27]) critically compare WS-* and RESTful services, concluding that although REST is limited by the contraints of the HTTP protocol, its restrictive feature set is at the same time its strength, "choosing REST removes the need for making a series of further architectural decisions related to the various layers of the WS-* stack and makes such complexity appear as superfluous" (Pautasso et al., 2008 [27]).

2.5 Implications

There is extensive literature on recommender systems, so much so that it is very difficult to assess the pros and cons of any given approach in respect of my project. One theme that is very strong in recommender systems is that of collaborative filtering, which is central to the majority of systems I have looked at. In many cases collaborative filtering is used in conjunction with other methods that boost or tune the results.

With regard to the web application component of my system, it seems clear that a RESTful web service would be suitable. My system will have a small number of API endpoints, so I will be able to reap the benefits the simplicity of the RESTful web stack without suffering the difficulties associated with its limited customisability.

3 Method

3.1 Methodology

Given the exploratory nature of this project I elected to take an incremental and iterative approach, developing small parts of the system at any time (increments), and iterating over those parts with improvements as necessity dicated and time allowed. This approach is based on that laid out by Cockburn (2008 [7]).

3.1.1 Phases

The main phases of development would be:

1. Clean up and migrate data
2. Create initial API app
3. Connect API with database
4. Implement routes for API access to wines and authors
5. Augment API routes for wines and authors with recommendations
6. Iterate on recommendation methods, evaluating and improving quality

My reason for taking this approach, rather than following a formal development methodology such as the waterfall model, was that my aims and objectives were unbounded, it the sense that there would not be a point at which my system was complete, only a point at which it was minimally complete, followed by a succession of points at which is was improved.

3.1.2 Minimum Viable Product

It was clear that I would be doing a large amount of experimental programming, and given my lack of prior experience in the problem domain I felt it inappropriate to attempt to quantify my expectations for the system in terms of detailed requirements. Nevertheless there were very clear minimum objectives for the system, without which it would not be possible to claim any degree of success.

The system should at least:

- Provide an HTTP API for accessing wine and user information from the Decanter.com tastings database.
- Augment the API results for wines and authors with appropriate recommendations of other similar or interesting wines and author.
- Provide API results suitable for machine interpretation by web or mobile applications.
- Provide a mechanism by which to evaluate recommendation quality.

These requirements in the least should be fulfilled by the system. With this having been done the focus of the project will be on maximising the quality of recommendations.

3.2 Technologies and Tools

3.2.1 Python

As the main progamming language for my project I chose to use Python. There were several candidate languages, not least Java, but I decided on Python because it has a number of attributes which lent themselves particularly to this project:

- Extensive mathematical and scientific libraries, such as numpy [25] and scipy [35].
- Extensive detailed documentation [30].
- Widely used in web development, such as by Google and YouTube [32].
- Interactive interpreter, allowing command line interaction and supporting scripting on Unix-like systems [29].

One deciding factor was that my first enquiry into recommender systems was reading Segaran's code examples in Chapter 2 of Collective Intelligence (2007, Ch.2 [36]), where the language he uses for his code examples is Python.

In addition to its suitability for tasks around recommender systems, Python has a solid heritage of web application frameworks, such as Django [10] and Flask [11]. Django is a fully featured website building framework, and as such carries many features unnecessary for my project, whereas Flask, a "micro-framework" [11], appeared to be more lightweight and simple to implement. Therefore I chose to implement my API using Flask [11].

For the most part I considered that my system would suit the stateless, non-persistent nature of a Python web application. The only concern in this regard would be that I would need to recreate objects in memory from scratch with each request rather than persist them as I might using another language, such as using Java with the JPA[17]. It was reasonable to suppose that in generating recommendations I would potentially be creating large objects in memory, and that there may be a performance deficit incurred by having to rebuild such object on a per request basis. I resolved that should the lack of persistence prove problematic down the line I would be able to use a persistence mechanism such as Memcached [31] to serve this purpose, and found that there is wide support for such a solution using Python and Flask [13].

3.2.2 MySQL

Originally I had envisaged a system backed by a PostgreSQL [28] RDBMS, but having received the Decanter.com data as a MySQL [21] database it did not seem, comparing the two systems, that there would be any significant benefit migrating the data to PostgreSQL. Both are widely used in production, and have similar feature sets. For a short time I considered using a NoSQL database such as MongoDB [20] for my project, but decided against such a solution, recognising that such document-oriented systems are not ideal when joining between tables in the way that I would need to for my wines and tasting notes. It seemed that an RDBMS was ideally suited to the purpose, and there was no reason why that shouldn't be MySQL.

3.2.3 GitHub

Given the iterative nature of my development process I envisaged a need to be able to easily version my source code, possibly running several different versions at once, with the ability to revert changes back to any previous state. I also wanted a remote backup of my system in case of problems with my own development computer. In order to do be able to do these things I chose to store my code as a project in GitHub [15], which is a web service providing Git version control. I chose to use GitHub for my notes and project files also, so that my entire project was stored, versioned and backed up together.

3.3 Approach to Recommendations

Wine attributes, vintage, grape variety, region etc., are of limited use for making recommendations. Take a user who has rated a red Bordeaux wine highly. It is not interesting to that user for a system to simply recommend them other highly-rated red Bordeaux wines, which is what a system will do if it looks for items with similar attributes. It is likely that any drinker who enjoys Bordeaux wines will recognise that such wines are of a type, with geography, grape varieties and production processes in common. Thus the user will be capable of "recommending" Bordeaux wines to themselves, and will not have very much difficulty sourcing ones which are highly rated. They do not need a recommender system for that.

It is recommendations in spite of the attributes of the subject item that are of real interest. Recommending a Syrah from Chile's Colchagua Valley to someone who rated a red Bordeaux wine highly might be of more interest. It is relatively more likely that a user is unaware of the fine Syrah wines from that region, and that makes it a much more interesting recommendation; potentially a good one.

Similarly a recommendation of another Bordeaux red wine might be good, as long as it were possible to establish a commonality of appeal for that particular drinker other than that the wine is similar in attributes. For example there are many delicious Pauillacs from 2005, made of the same grape varieties in the same manner; finding the one which is most interesting to a given user is not aided by looking at its grape variety or appellation as they are identical. The appeal and interest of a wine recommendation lies in qualities beyond the item's attribute profile.

So, by disregarding the attributes of the wines it may be possible to make more interesting recommendations.

A corrollary benefit of this disregard for wine attributes is that I am able to make use of far more of the Decanter.com data, many of the wines within which have incomplete attributes.

...

4 System

4.1 Development Environment

I have developed my project to run on a Linux platform, and have used version 12.04 of the Ubuntu operating system during the system's development (Ubuntu, 2013 [**?**]). Interaction with the system is via the standard terminal. Appendix A details installation instructions for the necessary software packages and libraries for the project. The instructions assume the user is using an Internet-connected computer running a recent version of the Ubuntu operating system (v12.04 or above).

System-level software requirements include:

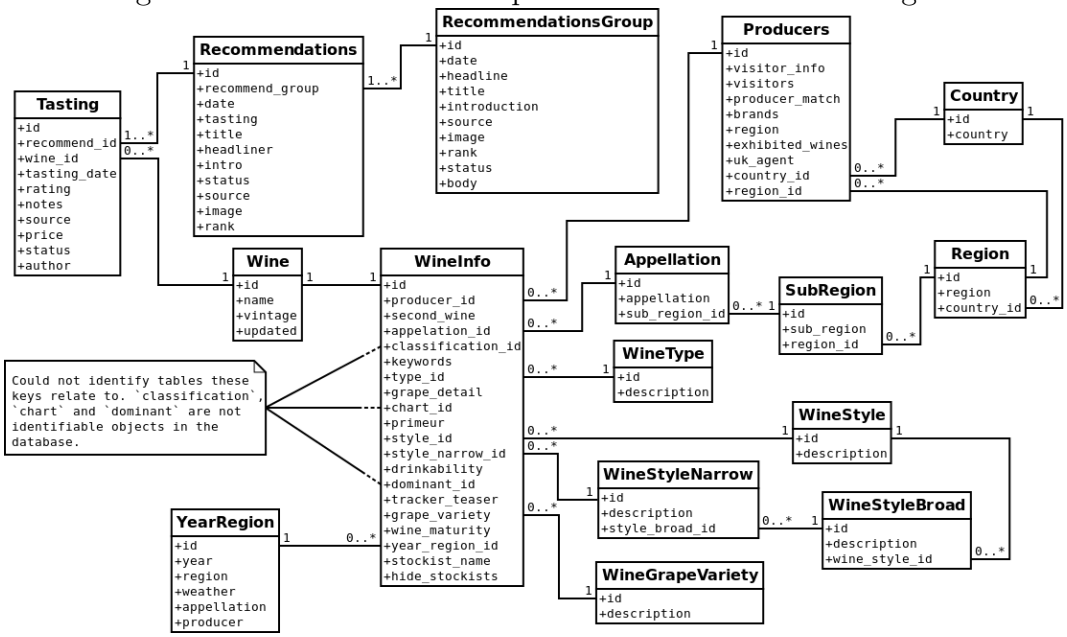
- MySQL v5.5, Python v2.7, Virtualenv, NumPy, SciPy

Other dependencies are installed locally to the project.

4.2 Data Migration

4.2.1 The Original Decanter Wines Database

Figure 1: Wines Database Represented as UML Class Diagram



The data source I have used for my project is the wines database belonging to Decanter.com [9]. The database contains nearly 40,000 professional ratings and tasting notes for wines from as far back as 1986, featuring vintages as far back as 1917.

I had hoped that the data would be fairly usable as it was received, as the data is currently used on the Decanter.com website, but on examination it was clear that there would be quite a lot of work to do in order to repurpose the data for my system.

I have modelled the database as a class diagram (Figure 1). This diagram represents the 16 tables in the original database, the columns present in those tables, and the relationships between them.

One thing immediately apparent about the database is that there are circular relationship between some of the tables. For example, the table WineInfo contains foreign key fields for both WineStyleNarrow and WineStyle, but it is the case that a association with WineStyleNarrow already encompasses an association with WineStyle, via WineStyleNarrow's association with WineStyleBroad, which associates with WineStyle. Duplicating the association in this way makes the database difficult to maintain as there is a dependency between the two foreign keys style_id and style_narrow_id on the WineInfo table which means that both must be updated any time that one is. There is a similar situation between the keys appellation_id and producer_id in the same table.

In addition to unnecessary duplication of associations in the WineInfo table there are foreign keys for missing, or unidentifiable, tables: classification_id, chart_id and dominant_id. It is likely that these keys relate to tables removed from the database at one time or other.

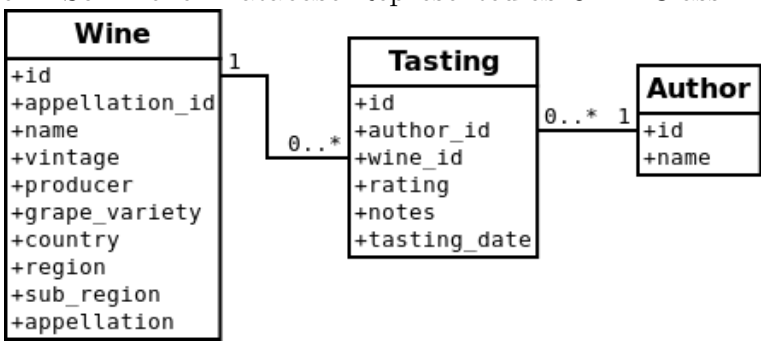
The WineInfo table is a mixture of foreign keys joining to very small tables, such as type_id joining to WineType, where WineType is a table holding only a single non-key field. This approach, of factoring out any non-integer fields to separate tables, is inconsistent with the fact that the same table also has several text fields, including second_wine and tracker_teaser. second_wine only holds data in 450 of the 38762 entries in the table, and is an empty string by default in all others.

The problems with the database structure are consistent with the fact that the database has been developed over a long period of time in an ad hoc fashion. There have probably been many different developers, and the database, being many years old, has probably supported many incarnations of Decanter's website.

Despite the problems with the database I considered there to be a great deal of useful and interesting information in the database, with it to contain usable ratings and/or tastings for over 33,000 wines, but decided that it would be best to port it to a much more simple schema, and to strip out as much redundant data as possible, for the purposes of my project.

4.2.2 The Cleaned-up Sommelier Development Database

Figure 2: Sommelier Database Represented as UML Class Diagram



For the new Sommelier database (Figure 2), I decided to minimise the complexity of the data, denormalizing the tables to make querying it as simple as possible. This would make updates to the data more complex, as any update to a field shared among many wines, for example a producer's name, would need to be replicated across a high number of records. As I did not intend to perform any writes on the data as

part of my project I felt able to overlook this shortcoming of the design. If this system were to be applied in the real world, then there may be a need to refactor the database so that updates may be made more easily.

As can be seen in Figure 2, I disregarded entirely much of the data from the original database. In many cases this was because I did not feel the data would be beneficial to my system. The Recommendations and RecommendationsGroup tables, for example, may have been useful, but the data in the tables was often incomplete and I did not reckon them to enrich the tasting notes enough to warrant migration. The fact that belonging to a certain group of recommendations may be used to infer similarities between wines seemed a tenuous reason to retain data in my development database.

The tables WineStyleNarrow and WineStyleBroad contained generic text descriptions for wine, such as "rich and creamy" and "crisp and tangy". I initially considered this to have potential for migration into tag data which I could reuse as part of my filtering. Unfortunately less than 6435 of the records in WineInfo had non-null values for their 'style_narrow_id' field, and only 3397 of these had corresponding records in the Tasting table. This figure was only around 10% of the number of wines I expected the Sommelier database to contain so I decided that the WineStyleNarrow and WineStyleBroad tables were not worth migrating.

The WineType table was ignored because no wines corresponded to it, there were values for 'type_id' in some WineInfo records, but none of those values corresponded with values in the 'id' field in WineType.

Whatever the situation with the data in other tables, it was clear that the Tastings table would need to be at the centre of my system, as it is where user, item and rating are associated. Those are the core data points for most recommender systems, so I resolved that I would migrate all tastings which had both their 'notes' and 'rating' field populated, along with the associated authors and wines, to a new database using my simplified schema.

4.2.3 The Author Problem

The biggest shortcoming of the dataset is that the author of a tasting note is often not recorded. The number of tastings with known authors is only 1411, with there being only 18 named authors on the system. There are 25812 tastings with no author associated. I had reckoned on there being a much higher number of wine tastings with identifiable authors, given that each note and rating was always by a particular person. It appeared that this information had either not been entered, or had been lost.

Table 2 shows the distribution of tastings amongst authors, only 5 of which have tasted and rated more than 100 wines in the database. The table also shows how many of the wines tasted by each author have also been tasted and rated by other authors. Table 4 shows the number of wines tasted in common for each of the authors (authors who have not tasted any wines in common with another author are omitted). It is clear that the data is extremely sparse. When migrating the data I was careful to ensure that I maintained all author to tasting associations, but rather than an oversight during migration, this shortcoming is inherent in the data.

Measuring sparsity as the percentage of empty values in a matrix $\mathbf{M}$ of size $|A| \times |W|$, where A is the vector of authors and W is the vector of wines, we find that the database is 94% sparse (Appendix B.2). This is a worryingly sparse

Table 2: Number of wines rated by each author

Author	Wines tasted	Wines also tasted by other authors
Amy Wislocki	28	1
Andrew Jefford	105	38
Beverley Blanning MW	13	-
Carolyn Holmes	1	-
Christelle Guibert	119	9
Clive Coates MW	6	-
David Peppercorn	44	-
Gerald D Boyd	7	-
Harriet Waugh	250	23
James Lawther MW	226	21
John Radford	2	-
Josephine Butchart	24	1
Norm Roby	4	-
Rosemary George MW	6	-
Serena Sutcliffe	31	15
Stephen Brook	19	3
Steven Spurrier	497	53

dataset for collaborative filtering. Su (2006 [39]) reports a "fast degradation of performance" for the collaborative filtering algorithms they tested when the rate of sparsity exceeded 90%. In addition there is the problem that ratings are not evenly distributed among authors, with a very small proportion of the authors accounting for the majority of ratings.

In a few cases there are tastings with no author associated where an author's initials or full name are recorded within the text of a tasting note, but unfortunately extracting and making use of these initials has been impractical given the time constraints of this project.

4.3 Experimentation

4.3.1 Recommending Wines With Python

In Chapter 2 of Collective Intelligence (Segaran, 2007 [36]), Segaran details basic methods for user- and item-based collaborative filtering. Following the guidelines from this chapter I recreated Segaran's recommendation methods and applied them to my dataset, utilizing the Python command line interpreter (See code in Appendix C.1).

Table 4 shows the Pearson correlation between the authors. In cases where there are fewer than 3 items rated in common it is not possible for Pearson correlation to produce a useful result, and for such relationships my code returns a value of 0.0. Table 4 shows that Pearson correlation was only able to produce similarity scores among five of the authors. For 18 authors in the system, recommendations could not be made at all for 13. This performance was consistent with my expectations given the sparseness of the data.

Table 3: Matrix of authors with wines tasted in common

Author	SS	JL	JB	SB	CG	SS	HW	AJ	AW
Steven Spurrier (SS)	-	6	1	1	7	0	15	30	1
James Lawther MW (JL)	6	-	0	0	0	15	0	0	0
Josephine Butchart (JB)	1	0	-	0	0	0	0	0	0
Stephen Brook (SB)	1	0	0	-	0	0	1	1	0
Christelle Guibert (CG)	7	0	0	0	-	0	1	5	0
Serena Sutcliffe (SS)	0	15	0	0	0	-	0	0	0
Harriet Waugh (HW)	15	0	0	1	1	0	-	10	0
Andrew Jefford (AJ)	30	0	0	1	5	0	10	-	0
Amy Wislocki (AW)	1	0	0	0	0	0	0	0	-

Table 4: Pearson Similarity of Authors (Appendix B.3)

Author	SS	JL	JB	SB	CG	SS	HW	AJ	AW
Steven Spurrier (SS)	-	0.58	0.0	0.0	0.0	-	0.67	0.49	0.0
James Lawther MW (JL)	0.58	-	-	-	-	0.22	-	-	-
Josephine Butchart (JB)	0.0	-	-	-	-	-	-	-	-
Stephen Brook (SB)	0.0	-	-	-	-	-	0.0	0.0	-
Christelle Guibert (CG)	0.0	-	-	-	-	-	0.0	0.0	-
Serena Sutcliffe (SS)	-	0.22	-	-	-	-	-	-	-
Harriet Waugh (HW)	0.67	-	-	0.0	0.0	-	-	0.71	-
Andrew Jefford (AJ)	0.49	-	-	0.0	0.0	-	0.71	-	-
Amy Wislocki (AW)	0.0	-	-	-	-	-	-	-	-

4.4 API Interface

4.4.1 Interchange Format

I chose to use JSON (JavaScript Object Notation) as my response format, which is a commonly used by RESTful APIs. It is designed to be natively handled by Javascript, but also has a sophisticated level of support in Python and other languages.

4.4.2 Routes

For the API I conceived a very simple set of routes, based on providing accesss to two the different object types, authors and wines. with an additional index route at the root to aid discoverability. Figure 3 shows the routing structure for the API.

Each route has a very simple url structure, with the plural item type and page number accessing a paged list of items, and the item type and id accessing any individual item.

Figure 3: API Routes. See code in Appendix **??**

```
/
/author/<id>
/authors/<page number>
/wine/<id>
/wine/<page number>
```

4.4.3 Response Specification

In principle RESTful services should favour discoverability, so as such the reponses contain links to other documents available on the API which are relevant to them. The objects are also designed to be consistent with each other as far as possible, using the same attribute for comparable properties wherever possible.

Every response is an object with the attributes "type" and "self". The "type" attribute is to aid the client in interpreting the object and for this API can take the values "author", "wine", or "list". "self" is a sub-object containing at least the fields "title" and "link", with additional fields depending on the type of the object.

"wine"- and "author"-type objects have the attribute "related_content", which is where different kinds of recommended content will be inserted by the API. The "list"-type objects have the attribute "list", which is a list of items with "title" and "link" attributes.

By structuring the responses this way I hope to make it as simple as possible for any client of the system to handle the different routes and navigate the API, as they should be able to ascertain the expected format of any object very quickly.

This example represents the template of an object with the minimum required fields, where %s represents a placeholder for a string value:

```
{
    ''type'': %s
    ''self'': {
        ''title'': %s
        ''link'': %s
    }
}
```

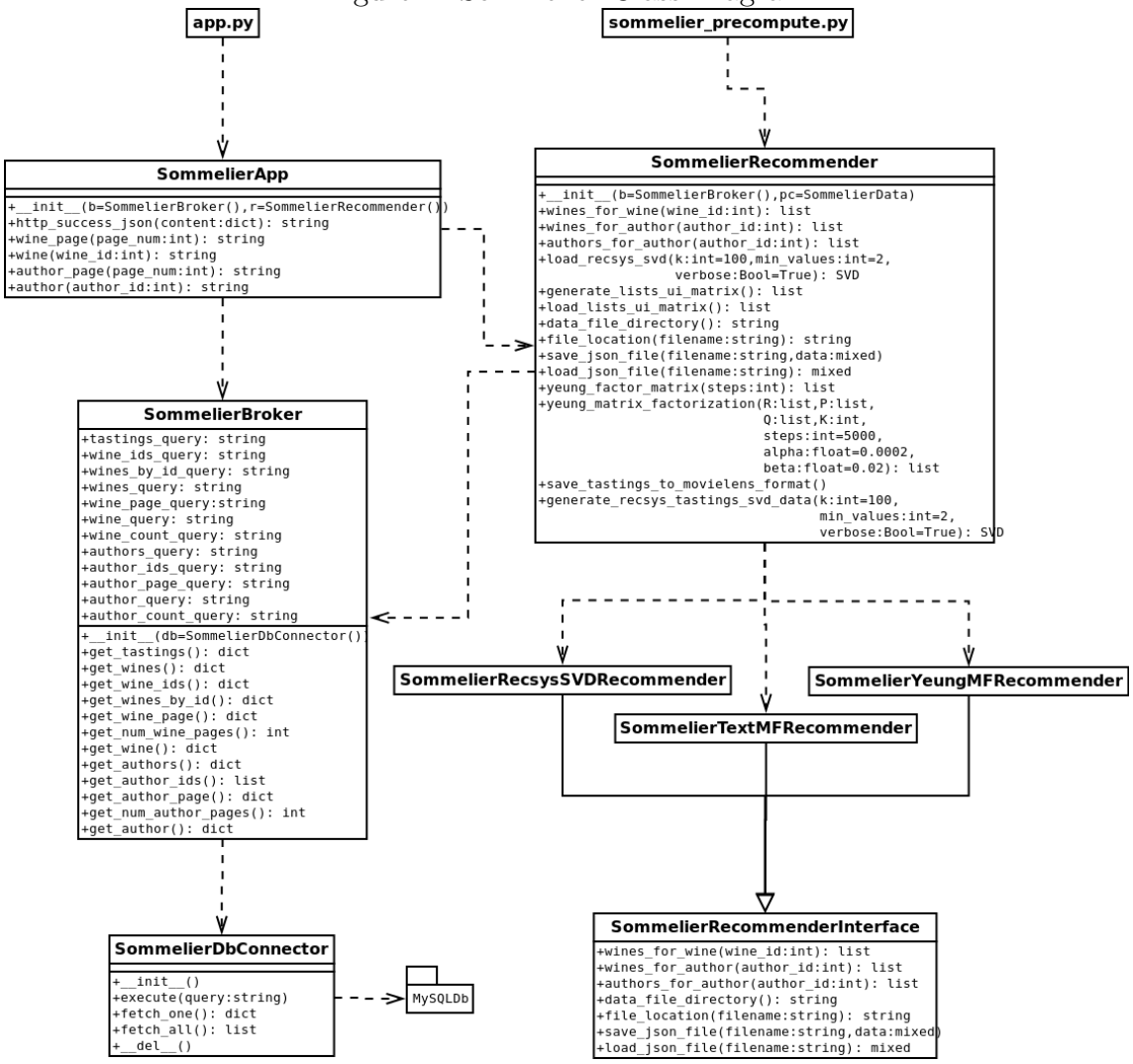
Full example responses for each content type are shown in Appendices E.1.1, E.1.2, E.1.3, E.1.4, and E.1.5. Responses are in the JSON format as specified by JSON.org (2013 [18]).

4.5 System Implementation

Figure 4 is a UML class diagram representing the complete Sommelier system.

The system is comprised of five main classes: Sommelier, SommelierBroker, SommelierDbConnector, SommelierRecommender, and SommelierRecommenderInterface. Additionally there are classes for different recommender implementations, and interactions with the files app.py and the sommelier_precompute.py are indicated.

Figure 4: Sommelier Class Diagram



4.5.1 Flask

The Flask framework is initialized by the file app.py, and this file is where all HTTP requests to the API are handled. For each route there is configuration in this file that calls the corresponding method on Sommelier. In this way the workings of the Sommelier application are completely unknown to Flask, and similarly Sommelier does not need to manage requests or routing in any way. Flask simply receives a request and routes it appropriately, passing Sommelier's return values to its Response class, which formats the HTTP response in a standard manner.

The following example is the routing in app.py for the route /authors, and shows how the only coupling between the Flask and Sommelier systems is the one-to-one relationship of Flask routes to methods on Sommelier (line 6):

```
@app.route('/authors', defaults = {'page_num': 1}, methods = ['GET'])
@app.route('/authors/<int:page_num>', methods = ['GET'])
def sommelier_authors(page_num):
    response_body, keyed_args_dict = sommelier.author_page(page_num)
    return Response(response_body, **keyed_args_dict)
```

Flask has a built-in development server, which launches the app.py application on port 5000 of localhost when executed. In a production environment there are a number of options for web servers (Flask Deployment Options, 2013 [12]).

4.5.2 Database Access

Maintaining a connection to MySQL-python comes with the attendant complexity of managing of the lifecycle of a database connection object and cursor. To avoid having to recreate these objects many times in the code I wrote the wrapper SommelierDbConnector (see Figure 4), which manages the connection and cursor in the minimum fashion, while exposing the minimum functionality to the application for executing on and fetching from the database. The full code is in Appendix **??**.

To further abstract the database from the application logic of the system I created a broker class, SommelierBroker, which holds a hard coded set of queries that are performed on the database. I a new query is to be made on the system it is to be written into the broker. This approach has serious drawbacks, such as that it would be potentially unwieldy to scale. However, as I developed the system I found that there was only a very small set of queries that I was using, and decided that it was easier in the short term for one broker to own all of them.

4.5.3 Recommenders

The system gets recommendations via the facade class SommelierRecommender (see Figure 4. This pattern abstracts away the complexity of making recommendations from the rest of the system. For the

In order to enable development of multiple potential recommender systems in parallel I needed to

4.5.4 Precomputation

For various recommendation approaches it is necessary to precompute data. I have not modelled this behaviour in Figure **??** as there are many potential ways of doing so. I have assumed that arbitrary scripts would be created and run on a scheduled basis to carry out precomputation for methods such as matrix factorization or SVD, where imputing matrices cannot practically be done at runtime.

5 Testing and Evaluation

How well does the system work? Details of testing and evaluation of the system...

6 Conclusion

Was the project successful?

7 Review

Review / reflections of the project on a personal level. What has been achieved? What were the problems, and how were they overcome?

Lessons learnt. . . - Data cleanup very time consuming - Literature vast -¿ plural techniques for recommendation: very difficult to work out what strategy is best for given situation.

References

[1] Benatallah, B., Nezhad, M., Hamid, R.. (2008). Service Oriented Architecture: Overview and Directions. In: B“orger, Egon and Cisternino, Antonio Advances in Software Engineering. Berlin, Heidelberg: Springer, Verlag. 116-130.

[2] Burke, R.. (1999). *The Wasabi Personal Shopper: A Case-Based Recommender System*. Submitted to the 11th Annual Conference on Innovative Applications of Artificial Intelligence.

[3] Burke, R.. (1999). *Integrating Knowledge-Based and Collaborative-Filtering Recommender Systems*. In: Artificial Intelligence for Electronic Commerce: Papers from the AAAI Workshop (AAAI Technical Report WS-99-0 1), pp.69-72.

[4] Burke, R.. (2000). *Knowledge-Based Recommender Systems*, Encyclopedia of Library and Information Systems. Marcel Dekker.

[5] Burke, R.. (2002). *Hybrid Recommender Systems: Survey and Experiments*, User Modeling and User-Adapted Interaction, Volume 12 Issue 4, November 2002, Pages 331 - 370. Kluwer Academic Publishers: Hingham, MA, USA

[6] Claypool, M., Gokhale, A., Miranda, T., Murnikov, P., Netes, D., Sartin, M.. (1999). *Combining Content-Based and Collaborative Filters in an Online Newspaper*, Recommender Systems - Implementation and Evaluation, ACM SIGIR Workshop On, August 1999.

[7] Cockburn, A.. (2008). *Using both incremental and iterative development* STSC CrossTalk, Vol. 21, No. 5.

[8] Debnath, S., Ganguly, N., Mitra, P.. (2008). *Feature weighting in content based recommendation system using social network analysis*, Proceedings of the 17th international conference on World Wide Web, WWW '08, 2008, Beijing, China, Pages 1041 - 1042. ACM: New York, NY, USA,

[9] Decanter.com, *Wine Reviews*. Retrieved May 3, 2013 from http://www.decanter.com/wine/reviews/1

[10] Django Project Homepage. Retrieved May 3, 2013 from http://www.djangoproject.com

[11] Flask (A Python Microframework), *Welcome*. Retrieved May 3, 2013 from http://flask.pocoo.org/

[12] Flask, *Deployment Options*. Retrieved May 3, 2013 from http://flask.pocoo.org/docs/deploying/

[13] Flask Documentation *Caching*. Retrieved May 3, 2013 from http://flask.pocoo.org/docs/patterns/caching/

[14] Mangalindan, J. P.. (2012). *Amazon's Recommendation Secret*, July 2012. Retrieved May 3, 2013 from http://tech.fortune.cnn.com/2012/07/30/amazon-5/

[15] Github Website, *About*. Retrieved May 3, 2013 from https://github.com/about

[16] Goldberg, D. Nichols, D., Oki, B. M., and Terry, D.. (1992). *Using collaborative filtering to weave an information tapestry*, Commun. ACM 35, 12 (Dec. 1992), 61–70.

[17] The Java EE 6 Tutorial, *Introduction to the Java Persistence API*. Retrieved May 3, 2013 from http://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html

[18] JSON Homepage. Retrived May 3, 2013 from http://www.json.org

[19] McNee, S. M., Riedl, J., Konstan, J. A.. (2006). *Accuracy is not always good: how accuracy metrics have hurt recommender systems*, Extended Abstracts of the 2006 ACM Conference on Human Factors in Computing Systems CHI 2006

[20] MongoDB. Retrieved May 3, 2013 from http://www.mongodb.org/

[21] MySQL, *The world's most popular open source database*. Retrieved May 3, 2013 from http://www.mysql.com/

[22] Netflix, *About us*. Retrieved May 3, 2013 from https://signup.netflix.com/MediaCenter

[23] Netflix Prize Website, *Index*. Retrieved May 3, 2013 from http://www.netflixprize.com//index

[24] Netflix Prize Website, *Rules*. Retrieved May 3, 2013 from http://www.netflixprize.com//rules

[25] Numpy, *Scientific Computing Tools for Python*. Retrieved May 3, 2013 from http://www.numpy.org/

[26] Patton, E., McGuinness, D.. (2010). *Scaling the Wall: Experiences Adapting a Semantic Web Application to Utilize Social Networks on Mobile Devices*, 2010. In: Proceedings of the WebSci10: Extending the Frontiers of Society On-Line, April 26-27th, 2010, Raleigh, NC: US.

[27] Pautasso, C., Zimmermann, O., Leymann, F.. (2008). *RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision*, 17th International World Wide Web Conference (WWW2008). SOA4All, Enabling the SOA Revolution on a World Wide Scale. Proceedings of the 2nd IEEE International Conference on Semantic Computing ICSCIEEE Computer Society. IEEE 2008

[28] PostgreSQL *The world's most advanced open source database*. Retrieved May 3, 2013 from http://www.postgresql.org/

[29] Python Documentation, *Using the Python Interpreter*. Retrieved May 3, 2013 from http://docs.python.org/2/tutorial/interpreter.html

[30] Python Documentation, *Overview*. Retrieved May 3, 2013 from http://docs.python.org/2/index.html

[31] Python Website: *python-memcached*. Retrieved May 3, 2013 from https://pypi.python.org/pypi/python-memcached/

[32] Python Website: *Quotes about Python*. Retrieved May 3, 2013 from http://www.python.org/about/quotes/

[33] Resnick, P., Iacovou, N., Sushak, M., Bergstrom, P., Riedl, J.. (1994). *GroupLens: An open architecture for collaborative filtering of netnews*, 1994 ACM Conference on Computer Supported Collaborative Work, 1994. Association of Computing Machinery, Chapel Hill, NC.

[34] Resnick, P., Varian, H. R.. (1997). *Recommender Systems*, 1997. Communications of the ACM, 40 (3), 56-58. Association of Computing Machinery, Chapel Hill, NC.

[35] Scipy, - -. Retrieved May 3, 2013 from http://www.scipy.org/

[36] Segaran, T.. (2007). *Programming Collective Intelligence*. O'Reilly, CA, US.

[37] Shani, G., Gunawardana, A.. (2011). *Evaluating Recommender Systems*, Recommender Systems Handbook, pp257-297. Springer, US

[38] Sheikh, M. A. A., Aboalsamh, H. A., Albarrak, A.. (2011). *Migration of legacy applications and services to Service-Oriented Architecture (SOA)*, Current Trends in Information Technology (CTIT), 2011 International Conference and Workshop on, pp.137,142, 26-27 Oct. 2011

[39] Su, X., Khoshgoftaar, T.M.. (2006). *Collaborative Filtering for Multi-class Data Using Belief Nets Algorithms*, Tools with Artificial Intelligence, 2006. ICTAI '06. 18th IEEE International Conference on, pp.497,504, Nov. 2006

[40] Su, X., Khoshgoftaar, T. M.. (2009). *A Survey of Collaborative Filtering Techniques*, Advances in Artificial Intelligence, vol. 2009, Article ID 421425, 19 pages, 2009.

[41] http://wineagent.tw.rpi.edu/index.php

[42] Ward, R., Towne, D., Stannard, D. H., LaChappelle, P.. (2012). *Wine Recommendation System and Method*. International Application Number PCT/US2012/046480, filed 12/07/2012. Retrieved May 3, 2013 from http://patentscope.wipo.int/search/en/detail.jsf?docId=WO2013009990

A Installation

These dependecies should be installed from the command line, having navigated to the project directory.

```
# We need pip to be installed to manage python packages
sudo apt-get install python-pip
pip install -U pip

# vitualenv enables us to install modules and packages local to our
# project, so we dont need to expose our system-level python
# installation to incompatible or otherwise obnoxious packages
# that might destabilize our other project, or OS in general
pip install virtualenv
virtualenv flask

# MySQL-python will enable us to query data (very useful!)
# Advice retrieved from http://codeinthehole.com/writing/how-to-set-up-mysql-for-python-on-ubuntu/
# there are some additional dependencies with MySQL-python that need
# to be installed at a system level
sudo apt-get install libmysqlclient-dev python-dev
# Documentation on working with MySQLdb is at: http://mysql-python.sourceforge.net/MySQLdb.html

# now we can install MySQL-python in our virtualenv using a local pip
./flask/bin/pip install MySQL-python

# flask itself is our web framework of choice
./flask/bin/pip install flask

# mock will be useful for unit testing
./flask/bin/pip install mock

# nose will serve as our test runner
# https://nose.readthedocs.org/en/latest/ Source: https://github.com/nose-devs/nose
./flask/bin/pip install nose

# gonna need numpy a lot probably!
# scipy is a pain. The following installation method is courtesy of:
# http://www.scipy.org/Installing_SciPy/Linux#head-d437bf93b9d428c6efeb08575f631ddf398374ea
# This installs rather a lot of stuff :-|
sudo apt-get build-dep python-numpy
# the following command does a big build and throws all sorts of errors, which are apparently fine to ignore.
sudo apt-get -b source python-numpy
./flask/bin/pip install scipy

# install dependencies for python-recsys
flask/bin/pip install csc-pysparse networkx divisi2
# clone python-recsys from Git and set it up in virtualenv
git clone http://github.com/ocelma/python-recsys
cd python-recsys
../flask/bin/python setup.py install

# install matplotlib for graphing test results
# n.b. this is not subsequently installed in the virtualenv, but
# that isnt a problem as the graphing can be done without
# invoking any of our virtualenv
sudo apt-get install python-matplotlib
```

B Source Data

B.1 Migration

These commands recreate the migration of data from the original Decanter.com wines database to the Sommelier database.
File./notes/queries.md. Modify the tasting notes database to remove nonsensical author names from the tasting table:

```
USE wines;
UPDATE tasting SET author = 'Christelle Guibert' where author = 'C hristelle Guibert';
UPDATE tasting SET author = '' WHERE author IN (
    'Rising stars',
    'New releases',
```

```
    'Great wine buys',
    'Panel Tasting',
    'Hot tip',
    'Wine of the month',
    'Wine of the week',
    'Connoisseur\'s choice',
    'Decanter choice',
    'Decanter Fine Wine Encounter 2002',
    'In the Decanter tasting room',
    'Christmas choice',
    ''
) OR author IS NULL;
```

Select count of wines tasted by each taster in author column of \‘tasting\‘:

```
    select t.author, count(*)
    from tasting as t
    where author is not NULL
      and author <> ''
      and author not in (
        'Rising stars',
        'New releases',
        'Great wine buys',
        'Panel Tasting',
        'Hot tip',
        'Wine of the month',
        'Wine of the week',
        'Connoisseur\'s choice',
        'Decanter choice',
        'Decanter Fine Wine Encounter 2002',
        'In the Decanter tasting room',
        'Christmas choice'
      )
      group by t.author;
```

```
    +----------------------+----------+
    | author               | count(*) |
    +----------------------+----------+
    | Alan Spencer         |       19 |
    | Amy Wislocki         |       29 |
    | Andrew Jefford       |      105 |
    | Beverley Blanning MW |       13 |
    | Carolyn Holmes       |        1 |
    | Christelle Guibert   |      120 |
    | Clive Coates MW      |        6 |
    | David Peppercorn     |       45 |
    | Gerald D Boyd        |        7 |
    | Harriet Waugh        |      253 |
    | James Lawther MW     |      238 |
    | John Radford         |        2 |
    | Josephine Butchart   |       24 |
    | Norm Roby            |        4 |
    | Richard Mayson       |       14 |
    | Rosemary George MW   |        6 |
    | Serena Sutcliffe     |       31 |
    | Stephen Brook        |      491 |
    | Steven Spurrier      |      510 |
    +----------------------+----------+
    19 rows in set (0.03 sec)
```

Investigation of prices in tasting notes:

```
    select price from tasting
    where price not like '%'
      and price not like '$%'
      and price != ''
      and price is not null
      and price not like 'n/a%'
      and price not like 'POA'
      and price not like '%TBC%'
      and price not like 'na%'
      and price not like '%Nicholas%'
      and price not like '%old out%'
```

```
  and price not like 'n./a%'
  and price not like 'tcb%'
  and price not like 'n/ a%'
  and price not like '%request%'
  and price not like '%poa%'
  and price not like '%Howard Ripley%'
  and price not like '%N\'/A%'
  and price not like '%N/A%'
  and price not like '%ut of%'
  and price not like '%tba%'
  and price not like '%trade%'
  and price not like '%undefined%'
  and price not like '%unreleased%'
  and price not like '%N/UK%'
  and price not like '%not released%'
  and price not like '%limited avail%'
  and price not like '%on reques%'
  and price not like '%not in st%'
  and price not like '%ice on ap%'
  and price not like '%JkN%'
  and price not like '%autumn%'
  and price not like '%#316'
  and price not like '%not ye%'
  and price not like '%Lib%';
```

How many wines are there which have been tasted by ¿ 1 named author?

```
mysql> select count(*) from wine w where 1 < ( select count(*) from tasting t2 where t2.author <> '' and t2.wine_id =
+----------+
| count(*) |
+----------+
|      104 |
+----------+
1 row in set (0.52 sec)
```

How many wines have been tasted by > 1 author, named or ''?

```
mysql> select count(*) from wine w where 1 < ( select count(*) from tasting t2 where t2.wine_id = w.id);
+----------+
| count(*) |
+----------+
|     3225 |
+----------+
1 row in set (0.32 sec)
```

How many wines in total?

```
mysql> select count(*) from wine;
+----------+
| count(*) |
+----------+
|    50539 |
+----------+
1 row in set (0.00 sec)
```

Generally speaking:

```
1918   Wines in denormalised_quick_search_data table associated with tastings with authors
1411   ... as above with rating > 0 (i.e. a valid rating)
31704  Wines in denormalised_quick_search_data with join to tasting
29185  ... as above with rating > 0
29732  ... as above with notes <> '' (i.e. written tasting note)
27232  ... as above with intersection of rating > 0 and notes <> ''‘
Wine data completeness...
```

```
SELECT COUNT(*) FROM sommelier WHERE (grape_variety IS NULL OR appellation IS NULL OR sub_region IS NULL OR region IS
+----------+
| COUNT(*) |
+----------+
|    21516 |
+----------+
```

```
    1 row in set (0.04 sec)


    SELECT COUNT(*) FROM sommelier WHERE (grape_variety IS NULL OR appellation IS NULL OR sub_region IS NULL OR region IS
    +----------+
    | COUNT(*) |
    +----------+
    |    14830 |
    +----------+
    1 row in set (0.00 sec)

    SELECT COUNT(*) FROM sommelier WHERE (grape_variety IS NULL OR appellation IS NULL);
    +----------+
    | COUNT(*) |
    +----------+
    |    14222 |
    +----------+
    1 row in set (0.00 sec)

    mysql> SELECT COUNT(*) FROM sommelier WHERE (grape_variety IS NULL);
    +----------+
    | COUNT(*) |
    +----------+
    |    13523 |
    +----------+
    1 row in set (0.00 sec)

    mysql> SELECT COUNT(*) FROM sommelier WHERE (grape_variety IS NOT NULL AND appellation IS NOT NULL AND sub_region IS
    +----------+
    | COUNT(*) |
    +----------+
    |    10169 |
    +----------+
    1 row in set (0.05 sec)

## Conversion from latin1 to utf8:

Based on advice from: http://en.gentoo-wiki.com/wiki/Convert_latin1_to_UTF-8_in_MySQL

    From the Bash shell:
    $ mysqldump -uroot -p -hlocalhost --default-character-set=latin1 -c --insert-ignore --skip-set-charset -r wine_dump.s
    $ file wine_dump.sql
    > wine_dump.sql: Non-ISO extended-ASCII English text, with very long lines
    $ iconv -f ISO8859-1 -t UTF-8 wine_dump.sql > wine_dump_utf8.sql
    $ sed -i 's/latin1/utf8/g' wine_dump_utf8.sql

    Now, from the MySQL command line:
    mysql> CREATE DATABASE sommelier CHARACTER SET utf8 COLLATE utf8_general_ci;

    And finally, back in the Bash shell:
    $ mysql -uroot --max_allowed_packet=16M -p --default-character-set=utf8 sommelier < wine_dump_utf8.sql

-------------------------------------------------------

mysql> select count(*) from tasting t join wine w on w.id = t.wine_id join wine_info wi on w.id = wi.id left join produce
*************************** 1. row ***************************
count(*): 14273
1 row in set (0.20 sec)

-------------------------------------------------------

Content for sommelier.wine:

CREATE TABLE `sommelier_wine` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) NOT NULL DEFAULT '',
  `vintage` int(4) NOT NULL DEFAULT '0',
  `grape_variety` varchar(255) NOT NULL DEFAULT '',
  `producer` varchar(255) NOT NULL DEFAULT '',
  `country` varchar(255) NOT NULL DEFAULT '',
  `region` varchar(255) NOT NULL DEFAULT '',
  `sub_region` varchar(255) NOT NULL DEFAULT '',
  `appellation` varchar(255) NOT NULL DEFAULT '',
```

```
  PRIMARY KEY (‘id‘)
) ENGINE=MyISAM AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;

INSERT INTO sommelier_wine SELECT
  w.id,
  w.name AS name,
  w.vintage AS vintage,
  p.producer_match AS producer,
  gv.description AS description,
  c.country AS country,
  r.region AS region,
  sr.sub_region AS sub_region,
  a.appellation AS appellation
FROM
  wine w
JOIN wine_info wi ON w.id = wi.id
LEFT JOIN producers p ON p.id = wi.producer_id
LEFT JOIN wine_grape_variety gv ON gv.id = wi.grape_variety
LEFT JOIN appellation a ON a.id = wi.appellation_id
LEFT JOIN sub_region sr ON sr.id = a.sub_region_id
LEFT JOIN region r ON r.id = sr.region_id
LEFT JOIN country c ON c.id = r.country_id
ORDER BY w.id ASC;
```

Content for sommelier.author:

```
CREATE TABLE ‘sommelier_author‘ (
  ‘id‘ int(11) NOT NULL AUTO_INCREMENT,
  ‘name‘ varchar(255) NOT NULL DEFAULT ’’,
  PRIMARY KEY (‘id‘)
) ENGINE=MyISAM AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;

INSERT INTO sommelier_author SELECT DISTINCT
  NULL,
  t.author as name
FROM
  tasting t
WHERE t.author <> ’’;
```

Content for sommelier.tasting:

```
CREATE TABLE ‘sommelier_tasting‘ (
  ‘id‘ int(11) NOT NULL AUTO_INCREMENT,
  ‘wine_id‘ int(11) NOT NULL,
  ‘author_id‘ int(11) NOT NULL,
  ‘rating‘ int(11) NOT NULL,
  ‘notes‘ TEXT NOT NULL,
  ‘tasting_date‘ datetime NOT NULL DEFAULT ’0000-00-00 00:00:00’,
  PRIMARY KEY (‘id‘),
  KEY ‘wine_idx‘ (‘wine_id‘),
  KEY ‘author_idx‘ (‘author_id‘)
) ENGINE=MyISAM AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;

INSERT INTO sommelier_tasting SELECT
  NULL,
  t.wine_id AS wine_id,
  a.id AS author_id,
  t.rating AS rating,
  t.notes AS notes,
  t.tasting_date AS tasting_date
FROM
  tasting t
JOIN wine w ON w.id = t.wine_id
JOIN wine_info wi ON w.id = wi.id
LEFT JOIN sommelier_author a ON t.author = a.name
WHERE t.rating > 0
  AND t.notes <> ’’
ORDER BY w.id ASC;
```

Finally, delete all wines without tasting records:

```
DELETE FROM sommelier_wine WHERE id NOT IN ( SELECT wine_id FROM sommelier_tasting );
```

```
DROP TABLE wine;
DROP TABLE tasting;
DROP TABLE author;

RENAME TABLE sommelier_wine TO wine;

RENAME TABLE sommelier_tasting TO tasting;

RENAME TABLE sommelier_author TO author;

/*
mysql> show tables;
+---------------------+
| Tables_in_sommelier |
+---------------------+
| author              |
| tasting             |
| wine                |
+---------------------+
3 rows in set (0.00 sec)
*/


// Author ids...

+----+----------------------+
| id | name                 |
+----+----------------------+
|  1 | Steven Spurrier      |
|  2 | Beverley Blanning MW |
|  3 | James Lawther MW     |
|  4 | Josephine Butchart   |
|  5 | Rosemary George MW   |
|  6 | Norm Roby            |
|  7 | Clive Coates MW      |
|  8 | John Radford         |
|  9 | Gerald D Boyd        |
| 10 | Stephen Brook        |
| 11 | Christelle Guibert   |
| 12 | Alan Spencer         |
| 13 | Serena Sutcliffe     |
| 14 | Harriet Waugh        |
| 15 | Andrew Jefford       |
| 16 | David Peppercorn     |
| 17 | Richard Mayson       |
| 18 | Carolyn Holmes       |
| 19 | Amy Wislocki         |
+----+----------------------+
```

B.2 Sparsity

Sparsity of 94% calculated by:
Distinct number of known authors in tasting table: 18
Total number of tastings by known authors: 1411
Distinct number of wine ids in tasting table with known author: 1307
Sparsity percentage = $100 - ((1411 \div (1307 \times 18)) \times 100)$

B.3 Author Similarity

Table data obtained in the Python interactive interpreter by the following commands (see also C.1):

```
import recommendations
recommendations.getAuthorSimilarities()
```

C Experimental Code

C.1 Code derived from Segaran, 2007

This code was copied largely from Segaran's exercises in Ch.2 of Collective Intelligence (2007). I have truncated part of the data fixture for brevity.

I have adapted Segaran's code, implementing new methods to query data from the Sommelier database and apply his methods to that data.

```
# a dictionary of critics and their ratings of a small
# set of movies
# copied from Segaran: Collective Intelligence (2006) Ch.2
critics={
    'Lisa Rose': {
        'Lady in the Water': 2.5,
        'Snakes on a Plane': 3.5,
        'Just My Luck': 3.0,
        'Superman Returns': 3.5,
        'You, Me and Dupree': 2.5,
        'The Night Listener': 3.0
    },
#### TRUNCATED ####
    'Toby': {
        'Snakes on a Plane': 4.5,
        'Superman Returns': 4.0,
        'You, Me and Dupree': 1.0
    }
}

# Method copied from Segaran: Collective Intelligence (2006) Ch.2
from math import sqrt
def sim_distance(prefs,person1,person2):
    si={}
    for item in prefs[person1]:
        if item in prefs[person2]:
            si[item]=1
    if len(si)==0: return 0
    sum_of_squares=sum([pow(prefs[person1][item]-prefs[person2][item],2)
    return 1/(1+sum_of_squares)

# This method is equivalent to sim_distance() above, uses scipy's sqeuclidean method
import scipy.spatial
def euclidean_distance(prefs,person1,person2):
    vector1=[]
    vector2=[]
    for item in prefs[person1]:
        if item in prefs[person2]:
            vector1.append(prefs[person1][item])
            vector2.append(prefs[person2][item])
    if len(vector1)==0: return 0
    euclidean_distance=scipy.spatial.distance.sqeuclidean(vector1,vector2)
    return 1 / (1 + euclidean_distance)

# Method copied from Segaran: Collective Intelligence (2006) Ch.2
def sim_pearson(prefs,p1,p2):
    si={}
    for item in prefs[p1]:
        if item in prefs[p2]: si[item]=1
    n=len(si)
    if n==0: return 0
    sum1=sum([prefs[p1][it] for it in si])
    sum2=sum([prefs[p2][it] for it in si])
    sum1Sq=sum([pow(prefs[p1][it],2) for it in si])
    sum2Sq=sum([pow(prefs[p2][it],2) for it in si])
    pSum=sum([prefs[p1][it]*prefs[p2][it] for it in si])
    # calculate Pearson score:
    num=pSum-(sum1*sum2/n)
    den=sqrt((sum1Sq-pow(sum1,2)/n)*(sum2Sq-pow(sum2,2)/n))
    if den==0: return 0
    r=num/den
    return r

# Method copied from Segaran: Collective Intelligence (2006) Ch.2
def topMatches(prefs,person,n=5,similarity=sim_pearson):
    scores=[(similarity(prefs,person,other), other)
            for other in prefs if other!=person]
    scores.sort()
    scores.reverse()
    return scores[0:n]
```

```
# Method copied from Segaran: Collective Intelligence (2006) Ch.2
# Gets recommendations for a person by using weighted average
# of every other user's rankings
def getRecommendations(prefs,person,similarity=sim_pearson):
    totals={}
    simSums={}
    for other in prefs:
        if other==person: continue
        sim=similarity(prefs,person,other)
        if sim<=0: continue
        for item in prefs[other]:
            # only score movies 'person' hasn't seen
            if item not in prefs[person] or prefs[person][item]==0:
                # similarity*score
                totals.setdefault(item,0)
                totals[item]+=prefs[other][item]*sim
                # sum of similarities
                simSums.setdefault(item,0)
                simSums[item]+=sim
    rankings=[(total/simSums[item],item) for item,total in totals.items()]
    rankings.sort()
    rankings.reverse()
    return rankings

# Method copied from Segaran: Collective Intelligence (2006) Ch.2
def transformPrefs(prefs):
    result={}
    for person in prefs:
        for item in prefs[person]:
            result.setdefault(item,{})
            result[item][person]=prefs[person][item]
    return result

# Method copied from Segaran: Collective Intelligence (2006) Ch.2
def calculateSimilarItems(prefs,n=10,similarity=sim_distance):
    result={}
    itemPrefs=transformPrefs(prefs)
    c=0
    for item in itemPrefs:
        c+=1
        if c%100==0: print "%d / %d" % (c,len(itemPrefs))
        scores=topMatches(itemPrefs,item,n=n,similarity=sim_distance)
        result[item]=scores
    return result

# Method copied from Segaran: Collective Intelligence (2006) Ch.2
def getRecommendedItems(prefs,itemMatch,user):
    userRatings=prefs[user]
    scores={}
    totalSim={}
    for (item,rating) in userRatings.items():
        for (similarity, item2) in itemMatch[item]:
            if item2 in userRatings: continue
            # Weighted sum of rating times similarity
            scores.setdefault(item2,0)
            scores[item2]+=similarity*rating
            # Sum of all the similarities
            totalSim.setdefault(item2,0)
            totalSim[item2]+=similarity
    # Divide each total score by total weighting to give an average
    rankings=[(score/totalSim[item],item) for item,score in scores.items()]
    rankings.sort()
    rankings.reverse()
    return rankings

# Method copied from Segaran: Collective Intelligence (2006) Ch.2
def loadMovieLens(path='../data/ml-100k'):
    movies={}
    for line in open(path+'/u.item'):
        (id,title)=line.split('|')[0:2]
        movies[id]=title
    prefs={}
```

```
    for line in open(path+'/u.data'):
        (user,movieid,rating,ts)=line.split('\t')
        prefs.setdefault(user,{})
        prefs[user][movies[movieid]]=float(rating)
    return prefs

def loadSommelierWines(comparator='rating'):
    import MySQLdb
    from MySQLdb.constants import FIELD_TYPE
    from MySQLdb.cursors import DictCursor
    converter = { FIELD_TYPE.LONG: int }
    connection = MySQLdb.connect(user="sommelier",db="sommelier",passwd="vinorosso",conv=converter)
    connection.set_character_set('utf8')
    cursor = connection.cursor(DictCursor)
    cursor.execute('SET NAMES utf8;')
    cursor.execute('SET CHARACTER SET utf8;')
    cursor.execute('SET character_set_connection=utf8;')
    cursor.execute("""
select w.name as wine, w.vintage, a.name as author, t.rating, t.notes
from wine w join tasting t on t.wine_id = w.id join author a on a.id = t.author_id
    """)
    results = cursor.fetchall()
    prefs={}
    for row in results:
        user = row['author']
        wine = row['wine']
        vintage = row['vintage']
        rating = row['rating']
        notes = row['notes']
        prefs.setdefault(user,{})
        if comparator == 'notes':
            comp = row['notes']
        else:
            comp = row['rating'] + 0.0
        prefs[user][''.join([wine,str(vintage)])] = comp
    cursor.close()
    connection.close()
    return prefs

def loadSommelierAuthors():
    import MySQLdb
    from MySQLdb.constants import FIELD_TYPE
    from MySQLdb.cursors import DictCursor
    converter = { FIELD_TYPE.LONG: int }
    connection = MySQLdb.connect(user="sommelier",db="sommelier",passwd="vinorosso",conv=converter)
    connection.set_character_set('utf8')
    cursor = connection.cursor(DictCursor)
    cursor.execute('SET NAMES utf8;')
    cursor.execute('SET CHARACTER SET utf8;')
    cursor.execute('SET character_set_connection=utf8;')
    cursor.execute("""
select w.name as wine, w.vintage as vintage, a.name as author, t.rating as rating from wine w join tasting t on t.wine_id
    """)
    results = cursor.fetchall()
    authors = {}
    for row in results:
        author = row['author']
        wine = ' '.join([row['wine'], str(row['vintage'])])
        rating = row['rating']
        authors.setdefault(author,{})
        authors[author][wine] = rating;
    cursor.close()
    connection.close()
    return authors

def getAuthorSimilarities(similarity=sim_pearson):
    authors = loadSommelierAuthors()
    sims = {}
    for author1 in authors.keys():
        sims.setdefault(author1, {})
        for author2 in authors.keys():
            if author1 == author2:
                continue
```

```
            sim = similarity(authors, author1, author2)
            if sim != 0:
                sims[author1][author2] = sim
    return sims
```

D Data Access

D.1 Sommelier Database Connector

A class managing database queries for the Sommelier application. This class implements only the minimum interface with the MySQLDB library, simply managing the lifecycle of a single cursor and exposing the cursor's execute(), fetchone() and fetchall() methods.

```
#!python

import math
import MySQLdb
from MySQLdb.constants import FIELD_TYPE
from MySQLdb.cursors import DictCursor

class SommelierDbConnector:

    cursor = None
    connection = None

    def __init__(self):
        converter = { FIELD_TYPE.LONG: int }
        self.connection = MySQLdb.connect(
                            user="sommelier",
                            db="sommelier",
                            passwd="vinorosso",
                            conv=converter)
        self.connection.set_character_set('utf8')
        self.cursor = self.connection.cursor(DictCursor)
        self.cursor.execute('SET NAMES utf8;')
        self.cursor.execute('SET CHARACTER SET utf8;')
        self.cursor.execute('SET character_set_connection=utf8;')

    def execute(self, query):
        return self.cursor.execute(query)

    def fetch_one(self):
        return self.cursor.fetchone()

    def fetch_all(self):
        return self.cursor.fetchall()

    def __del__(self):
        if self.cursor is not None:
            self.cursor.close()
        if self.connection is not None:
            self.connection.close()
```

E API Design

E.1 API Routes

E.1.1 API Response: Index

```
{
    ``type'': ``list'',
    ``self'': {
        ``title'': ``Sommelier API'',
        ``link'': ``/''
    },
    ``list'': [
        {
        ``title'': ``All Authors'',
        ``link'': ``/authors/1''
```

```
        },
        {
        ``title'': ``All Wines'',
        ``link'': ``/wines/1''
        }
    ]
}
```

E.1.2 API Response: Authors

```
{
    ``type'': ``list'',
    ``self'': {
        ``title'': ``Authors, Page 1''
        ``link'': ``/authors/1''
    },
    ``list'': [
        /* maximum 50 links per page */
        {
        ``title'': ``Mr. Author'',
        ``link'': ``/author/123''
        },
        {
        ``title'': ``A.N. Other'',
        ``link'': ``/author/234''
        }
    }
}
```

E.1.3 API Response: Author

```
{
    ``type'': ``author'',
    ``self'': {
        ``title'': ``Mr. Author'',
        ``name'': ``Mr. Author'',
        ``tastings'': [
            {
            ``rating'': 5,
            ``notes'': ``Tasting notes about the wine'',
            ``tasting_date'': ``2003-04-01 00:00:00'',
            ``wine'': {
                ``title'': ``Wine Name 1990'',
                ``link'': ``/wine/123''
            }
        ],
        ``link'': ``/author/123''
    },
    ``related_content'': {
        /* maximum of 5 wines */
        ``recommended_wines'': [
            {
            ``title'': ``Chateau du Vin 1996'',
            ``link'': ``/wine/234''
            }
        ],
        /* maximum of 5 other authors */
        ``similar_authors'': [
            {
            ``title'': ``A.N. Other'',
            ``link'': ``/author/234''
            }
        ]
    }
}
```

E.1.4 API Response: Wines

```
{
    ``type'': ``list''
    ``self'': {
        ``title'': ``Wines, Page 1''
```

```
        ``link'': ``/wines/1''
    },
    ``list'': [
        /* maximum 50 links per page */
        {
        ``title'': ``Wine Name 1990'',
        ``link'': ``/wine/123''
        },
        {
        ``title'': ``Chateau du Vin 1996'',
        ``link'': ``/wine/234''
        }
    }
}
```

E.1.5 API Response: Wine

```
{
    ``type'': ``wine'',
    ``self'': {
        ``title'': ``Wine Name 1990'',
        ``name'': ``Wine Name'',
        ``vintage'': 1990,
        ``producer'': ``Wine Producer Name'',
        ``grape_variety'': ``Cabernet Sauvignon'',
        ``appellation'': ``Pessac-Leognan'',
        ``country'': ``France'',
        ``region'': ``Bordeaux'',
        ``sub_region'': ``Graves''
        ``tastings'': [
            {
            ``rating'': 5,
            ``notes'': ``Tasting notes about the wine'',
            ``tasting_date'': ``2003-04-01 00:00:00'',
            ``author'': {
                ``title'': ``Mr. Author'',
                ``link'': ``/author/123''
            }
        ],
        ``link'': ``/wine/123''
    },
    ``related_content'': {
        /* maximum of 5 wines */
        ``similar_wines'': [
            {
            ``title'': ``Chateau du Vin 1996'',
            ``link'': ``/wine/2345''
            }
        ]
    }
}
```