Imperial College London

MEng Individual Project

Imperial College London

Department of Computing

fAshIon: Personal Shopper

Author:
Donald Lau

Supervisor:
Prof. Francesca Toni

Second Marker:
Prof. Alessio Lomuscio

2018 - 2019

Abstract

Many recommender systems are used to enhance the effectiveness of online shopping. However, they tend to use similarity as the main basis for predictions and overlook compatibility and semantics of items. In addition, they rarely give explanations behind recommendations, which could be used to improve the system and make learning of user preferences more efficient.

In this project, we have developed **fAshIon**, a recommender that provides explainable recommendations and show how explainability can improve recommendation quality. Two key features of **fAshIon** are: (1) provide reasoning for each recommendation and (2) incorporate user interactions with such reasoning, which is used to improve predictions.

Acknowledgement

I would like to thank my supervisors Prof. Francesca Toni for her guidance and support throughout the project. Additionally, I wish to express my gratitude to Oana Cocarascu, for her insights and advice over the past months.

A massive thank you to all my friends who have supported me during the past four years. To my course mates and friends, Si Wei Tan, Jeremy Ling, Sunghun Jung and Nimesh Subedi for being exceptional people to make it through last year with. Special thanks to Lewis Luk, Bei Zhang and Sue Yee for your friendships and continuous emotional supports.

Of course, I am especially grateful to my family for the continuous support over the years. Most of what I do would not be possible nor meaningful without you.

Contents

Chapter 1

Introduction

1.1 Recommenders in fashion industry

The online shopping market has been rapidly outgrowing the traditional physical shopping market in recent years. However, the online shopping task itself can often be a tedious process as customers often have to browse through large inventories before finding items that fit their preferences and/or style. In this regard, various kinds of recommender systems[1] have been employed to enhance the effectiveness of online shopping.

1.2 Motivation

The most common types of approaches in recommender systems are "collaborative filtering" (determining recommendations based on the preferences of other users with similar profile), "content-based filtering" (recommending items close to the user's usual taste and past selections) and "hybrid" approaches (combining both filters)[2]. While they refine searches, they primarily focus on the notion of similarity[3] (items that are interchangeable without impacting the overall feel) and pay less attention to the semantics or "compatibility" (items that go well together in a whole outfit), which is just as important, if not more[4].

Consider a male user who is getting an outfit for a formal occasion. If he picked a suit as input, recommenders may output a pair of brogue shoe to complete the outfit. However, the reasoning behind suggesting the brogue shoes is because the current user or other users with similar user profiles recognized the two items as having previously fitted together in a formal context. The recommenders themselves do not understand these items as having a formal semantic context. In some cases like this particular example, the recommender could output favourable suggestions. However, not addressing the semantics properly could yield poor performance for a user that possesses a more niche taste than the general public, or when adapting to

new fashion trends that most users are not yet aware of.

Another common problem of recommenders is the lack of explanations behind each recommendation. This hinders the improvement on the accuracy of general recommendations and the process of learning user's preferences to give a tailored experience. If a recommender has learned a wrong connection, such as leather being for casual occasions only from biker jacket, but not considering that brogue shoes are also commonly made of leather, it will then usually require a lot of new correct data to rectify the connection. Also, without any explanations, users are not given the rationale behind the explanations. Consequently, users are unable to spot the untold reasoning, and unable to input specific feedbacks to improve the system. The same reasoning applies for learning user preferences.

1.3 Objectives

The primary objective of this project is to build a recommender that is capable of learning the semantics of different fashion attributes other than simple similarities. We will be using different Natural Language Processing and Machine Learning techniques, in an attempt to achieve this. Thanks to a variety of past research, there is a large amount of fashion data collated, which makes training a well-formed machine learning model possible.

Our secondary objective is to provide explainable recommendations and allow users to interact with the explanations. Implementing the interactions should be relatively straightforward once we supply the explanations. These two objectives should show how learning the subtlety of fashion attributes and enabling user feedback allows a potential increase in performance for recommenders.

1.4 Contributions

This project makes several contributions to the field of explainable recommendations and lays the groundwork for future research. In particular, we provide:

- We devise our own ontology framework to feed into our prediction model. This framework contains four major classes, context (formal, semi-formal, casual), item, color and material.

- We create **fAshIon**, a specialized recommender on auto-completing outfit which provides explainable recommendations and uses user feedback to improve predictions. **fAshIon** has a fully-functional backend with a graphical user interface (GUI) frontend.

- We provide our own prediction algorithm that uses ontological features to make recommendations, which consider similarity, compatibility and semantics of fashion attributes.
- An evaluation of our model's performance against benchmark methods and user surveys. We also discuss how recommendation explainability can improve recommendation quality.

Chapter 2

Background

In this chapter, we first introduce some fundamental concepts in natural language processing and artificial neural network, which will be relevant throughout the project. Then, we present several representations and features models we have adopted in our recommender system, showing how they capture the essence of the subjectivity and cultural element of fashion items in a more mature way, compared them to simply recommending similar items. Further we discuss the encoding of fashion items with ontology to identify the subset of items of interest to the current context. Following this, we go through the intuition of explaining the recommendation output.

2.1 Natural Language Processing

Natural Language Processing (NLP) describes the technology a computer uses to understand the natural language of humans. Computers are machines functioning in precision with binary codes, while human languages are often filled with ambiguities and subtlety. For example, *"The chicken is ready for lunch"* could be interpreted as the chicken being cooked and ready to be served as lunch, or it could mean that the chicken is ready to eat its lunch. NLP serves as the bridging component between human languages and computer languages, often transforming words into numerical representations.

Syntactic analysis and semantic analysis are major components in NLP techniques. Syntactic analysis addresses how natural languages fits with grammatical rules. Popular syntactic analysis techniques include the following:

- **word-segmentation**: tokenizing sentences or decompounding words, such as transforming *wouldn't* into *would* and *not*.
- **lemmatization**:reducing variant forms of a word back into its base form, like turning *played, playing* back to *play*.

- **morphological analysis**: studying their forms and relationship with other words, such as Part of Speech tagging (POS tagging) and pre/suffixes analysis. POS tagging identify the category of words, i.e. *dogs* will be noun and *run* will be verb. Pre/suffixes analysis identify the prefixes and suffixes of words, like *"un"* in *unhappy* and *"un", "ness"* in *unhappiness*.

Semantic analysis explores the meaning and proper interpretation of words and sentences. Here are some examples of such techniques:

- **named entity recognition**: extracting texts in pre-defined categories or database in unstructured text, for instance getting person names, brands, professional terms.

- **word sense disambiguation**: understanding words in context, for example in *I just bought a new mouse for my laptop.* the word *mouse* will refer to the electronic device instead of the creature rodent.

To apply this analysis, natural languages have to be fed into computers in some representations. Various choices include raw words themselves, mappings to some lexical databases, one-hot vectors and word embeddings.

Raw words are unstructured and tend to have irregularities and ambiguities, making it hard for computers to understand. Mappings to databases simply means that each word is linked from a mapping to some databases whereby the computers can extract the meaning and attributes from the database. Co-occurrence vectors and word embeddings are vector space models which can represent words in a frequency vector form or continuous form. The former would typically be a very sparse vector, with the dimension size being the vocabulary size, and each position representing a single word. The latter maps similar words to nearby points, such as *cat* being near with *kitten* while far from *dog*.

The above representations are capable of representing the words themselves, but they lack the ability to represent words in context. Consider these three sentences, *"Today is rainy.", "Yesterday was sunny.", "This car is beautiful."*, the first two sentences have a closer resemblance and this is due to their context. More complicated models like continuous bag of words (CBOW) and skip-gram model are used to achieve this context awareness. CBOW predicts target words using context words before and after, while skip-gram predicts context words from the target word. Figure 2.1 shows the intuitions behind CBOW and skip-gram model.

2.2 Artificial Neural Networks

Artificial Neural networks, sometimes shortened to Neural Networks (NNs) are one of the most popular techniques used in machine learning. They are systems that

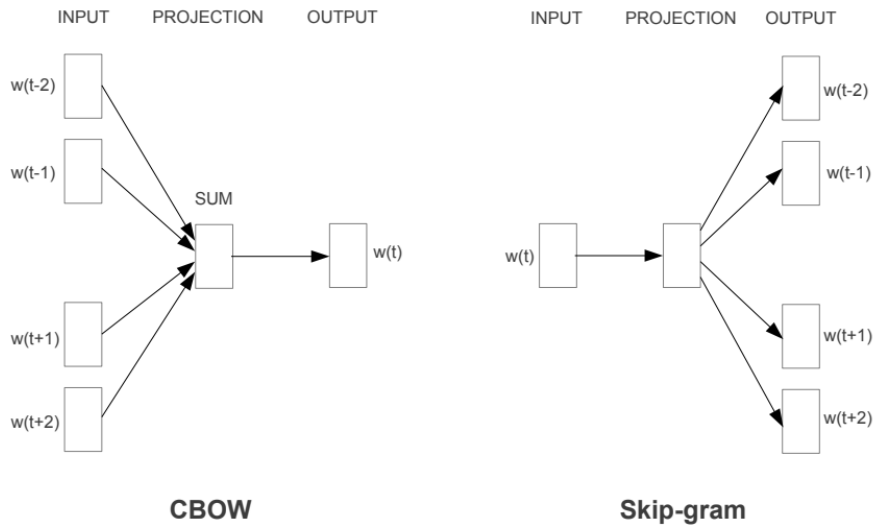

Figure 2.1: The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word. [5]

are inspired by the brain and consist of an input, output and layers of nodes that are usually hidden (neurons) that transform the input into data the next layer can process. They are well suited for learning complex patterns and features from a large amount of data. Figure 2.2 shows the architecture of a neural network with one hidden layer.

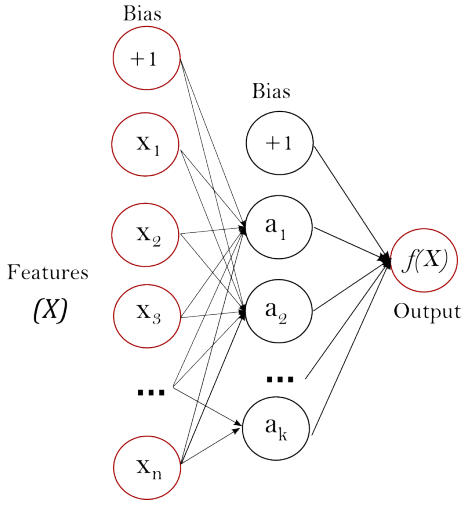

Figure 2.2: Neural Network diagram. [6]

Two neurons are connected through an weighted edge which dictates the significance of the input neuron to the output neuron. Neurons usually have an activation function to determine whether it provides an output, and a common activation is the sigmoid function which is illustrated in figure 2.3.
Training NNs mainly involves learning the correct weight between neurons. Back-

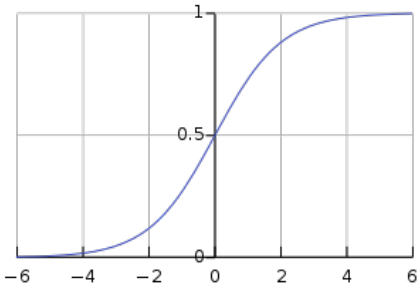

Figure 2.3: The logistic curve of a sigmoid function. [7]

propagation algorithms like gradient descents are widely used for such purposes. These algorithms use loss function (functions that measure the error between the model prediction and the actual output value) to evaluate the performance of weights and update them accordingly from the output layer back to the input layer.

2.3 Literature Review

2.3.1 Similarity and Compatibility

In most of the popular online fashion stores, their recommender systems still rely heavily on *visual similarity*. For example, through making recommendations based on preferences of other users with similar profiles or based on previous purchases made by the user with a good rating. This dependence poses challenges in correctly identifying the true need of users' future purchase [3].

Firstly, the same user may have purchased two items with completely different styles for different contexts, e.g. a tuxedo for a formal event and a leather biker jacket for casual-wear over a period of time. It is extremely hard to identify features from these items that build a particular profile of the user's liking and style. Also, suggesting similar items may not always fit well with the previously chosen item. For example, having green trousers with a green shirt and green shoes may not typically be an ideal outfit and make the user looks like a frog.

Instead, the user may want a different bottom to go with the green shirt which makes a compatible outfit, e.g. black denim trousers. Related works have been trying to overcome these challenges, and one of the notions is to incorporate both *similarity* and *compatibility* within an outfit.

Vector representation model

Style2Vec [3] is a vector representation model for fashion items which focuses on similarity on sets and is context sensitive. The fact that Style2Vec is training on

sets makes it more applicable to this project from comparable recommender systems [8, 9], as these recommender systems use discrete item attributes or low level image features instead.

It borrows the idea from distributed word representation models in the natural language community, using words to represent single fashion items (e.g. a shirt, a dress, etc) and sentences to represent a style set (a collection of garments and accessories that make up a single outfit). The assumption in this model is that words occurring in similar contexts tend to have similar semantics, so fashion items in similar style sets share a coherent style as their representative words share similar meanings.

Two types of vectors are used to calculate the similarity, specifically the representation vector of the target fashion item and the weight vectors of other context items in their corresponding style set. The inner products of the representation vector and each context vector are inputted to a softmax function for a probabilistic value and to maximize the sum of the log probability.

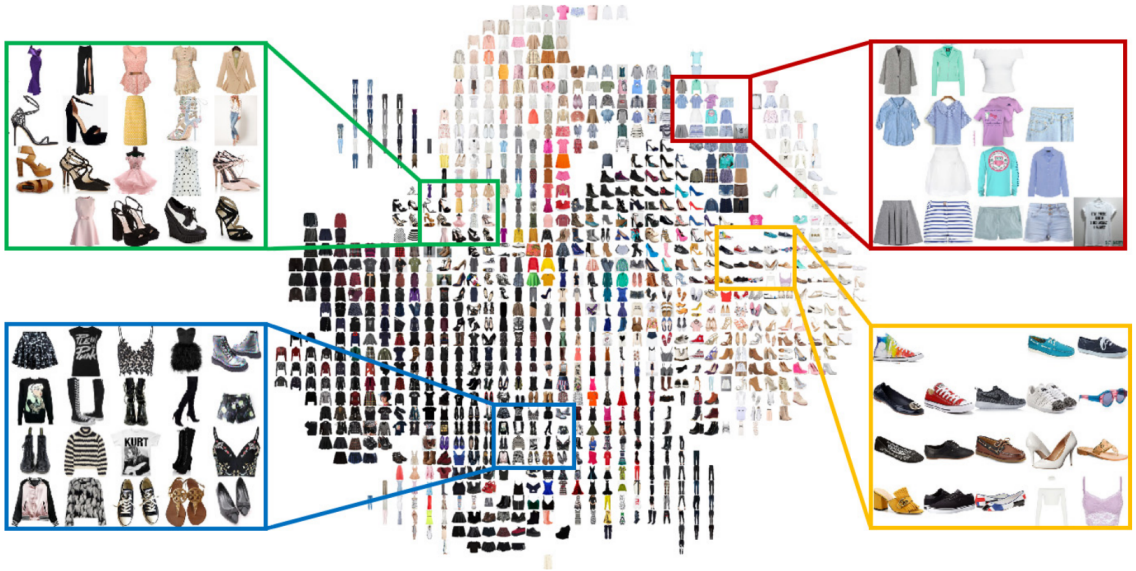

Figure 2.4: Visualization of the learned style space in 2D. [3]

Formally, let F be a set of all fashion items and u_i, v_c, v_j be the vector representation of input item image i and context item image c, j respectively (two neural networks are used to map input images and context item images to vector respectively). Style2Vec uses the assumption that all items in each style set S in the training dataset $D \subset P(F)$ are in coherent style. Therefore, the model aims to maximize

the average log probability for each S in D[1]:

$$\frac{1}{|S|}\sum_{i\in S}\sum_{c\in S\setminus\{i\}}\log\left(\frac{\exp u_i^\intercal v_c}{\sum_{j\in F}\exp(u_i^\intercal v_j)}\right)$$

The resulting latent space groups items together based on similarity is shown in figure 2.4, which is both visually-based as well as context-based (i.e. they have been matched with similar context items). For instance, in the upper right red box in figure 2.4 we observe items that are not all of similar shape or are not comprised of the same features (e.g. high collar, long sleeve, etc), yet since they share subtle pastel tones, it makes them complement each other.

Type-aware Embedding

While Style2Vec recovers the similarities between items in a context-based manner, it does not explicitly respect the item types (e.g. shoes and hats are of different types). This leads to potential mismatch of items due to the issue that being close by is naturally a transitive property, but compatibility is not. To illustrate: if a hat matches with a pair of shoes, and that pair of shoes matches with a pair of jeans, then it is likely that the hat is forced to match with that jeans, and this is known as the improper triangle [10]. This could happen because the similarity is encoded according to distance, so if the hat is projected near to the shoes, and the shoes near to the jeans, then the hat is forced to be nearby with the jeans.

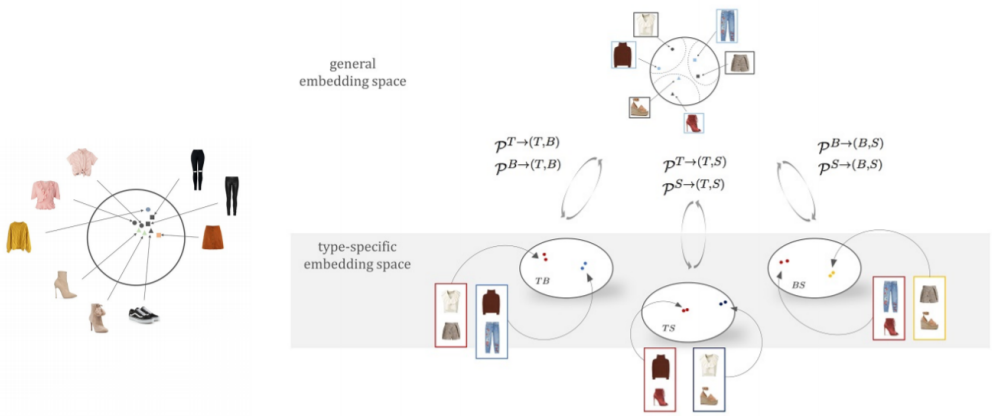

Figure 2.5: Comparison between general embedding strategy (left) and Fashion-Compatibility strategy (right). The top right shows the first stage where items are projected to the similarity embedding space and bottom right is the type-specific embedding space. [10]

[1] A negative sampling approach is applied to this formula to reduce computational cost, see [3] for further details

Fashion-compatibility project [10] aims to tackle this problem by using multiple embeddings. A general embedding is trained in the first stage to learn the semantic similarity in items. Then it maps the general embedding to different secondary embedding spaces that consider the compatibility between two item types. Extending from the above example, hats, shoes and jeans will be in 3 distinct embeddings, namely the hat-shoes, shoes-jeans and hat-jeans pairs. The notion of respecting types is important to this project, because this intuition is reflected in the concept of ontology which we will discuss in further detail later. Therefore this work is nearer to the scope of the project comparing to similar work. [11, 12]

The intuition of the compatibility function is to use the Euclidean distance to reflect compatibility between items, with a smaller distance in between reflecting a greater compatibility and vice versa. This happens within each type-respecting embedding subspace so that the improper triangle problem will not arise. Formally, let $x_i^{(\tau)}$ represent item i with type τ, Fashion-Compatibility computes a nonlinear feature embedding $y_i = f(x_i; \theta) \in \mathbb{R}^d$ with the parameter θ learned from a neural network. To conclude data items $x_i^{(u)}, x_j^{(v)}$ are compatible, the corresponding embeddings y_i, y_j are projected into a type-specific space with the projection $\mathcal{P}^{u\to(u,v)}$ and the distance $||\mathcal{P}^{u\to(u,v)}(y_i) - \mathcal{P}^{u\to(u,v)}(y_j)||$ have to be small. In here $\mathcal{M}^{(u,v)}$ represent items of types u and v are matched. We observe that this does not require the embedding vectors y_i, y_j in general (similarity) embedding space to be similar (close distance), it is only required in the kernel of $\mathcal{P}^{u\to(u,v)}$.

The computational cost to calculate the distance is high because it requires learning a $2(d \times d)$ matrix for every pair of types in a d-dimensional general embedding. Two assumptions are made to simplify this process: (1) assume the diagonal projection matrices $\mathcal{P}^{u\to(u,v)} = \mathcal{P}^{v\to(u,v)} = \text{diag}(\mathbf{w}^{(u,v)})$ and $\mathbf{w} \in \mathbb{R}^d$ is a vector of learned weights; (2) similarly, but $\mathbf{w}$ being a predefined vector instead of a learned one. Then the compatibility function for items x_i, x_j and their corresponding embedding vector y_i, y_j is rewritten as:

$$d_{ij}^{uv} = d(x_i^{(u)}, x_j^{(v)}, \mathbf{w}^{(u,v)} = ||y_i \odot \mathbf{w}^{(u,v)} - y_j \odot \mathbf{w}^{(u,v)}||_2^2$$

where $\odot$ denotes component0wise multiplication and learned with a modified triplet loss function[2].

Temporal property in forming outfit

Another related work is the jointly learned visual-semantic embedding and compatibility bidirectional LSTM (Bi-LSTM) model [4]. This model learns the compatibility of items through their roles and dependence within the outfit. On top of that, the

[2]the details about triplet loss is not included as it is out of the scope of this project, please refer to [10] for more details on it and the modified version

items order of the outfits in the dataset are mostly fixed: tops to bottoms to shoes and then accessories. The order within the categories of tops and accessories are also ordered: (1) for tops they are shirts/t-shirts then outerwears and (2) for accessories are usually in the order of handbags, hats, glasses, watches, necklaces, earrings, etc. The ordering adds an interesting "temporal" property in the calculation of recommendations. The model is able to utilize the query items and generate a corresponding outfit via finding the nearest neighbour of the "next" query item in either or both directions. An example of finding next query items in both directions is shown in fig 2.6.

For an image item in an outfit, where the set $S = w_1, w_2, ..., w_M$ represents its annotation with w_i denotes each word in the annotation for $1 \leqslant i \leqslant M$. The model first phrases w_i into a one-hot vector $\mathbf{e}_i$, then projects it into the embedding space using $\mathbf{v}_i = \mathbf{W_T} \cdot \mathbf{e}_i$, where $\mathbf{W_T}$ is the word embedding matrix. The embedded vector is then encoded with bag-of-words $\mathbf{v} = \frac{1}{M} \sum_i \mathbf{v}_i$. Similarly, let $\mathbf{W_I}$ be the image embedding matrix, the image representation $\mathbf{x}$ is projected to the embedding space as $\mathbf{f} = \mathbf{W_I} \cdot \mathbf{x}$.

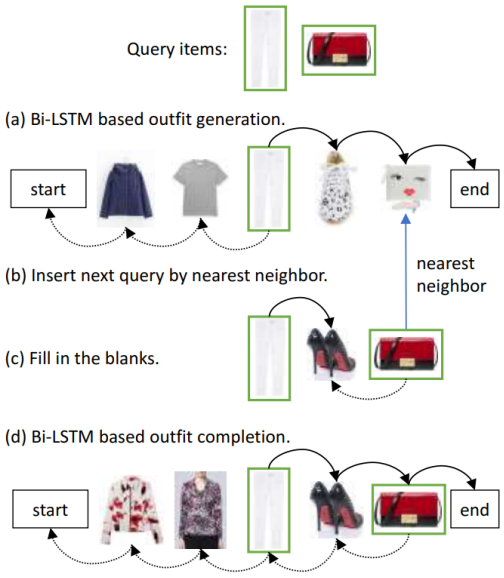

Figure 2.6: Given query items, the model can generate a compatible outfit by finding neighbouring items in both directions. [4]

The similarity of an image $\mathbf{x}$ and its description S is estimated in the visual-semantic space, using cosine distance of the corresponding vectors in normalized form. In the join space, the images and descriptions are embedded by minimizing the contrastive loss:

$$E_e(\Theta_e) = \sum_{\mathbf{f}} \sum_{k} \max(0, m - d(\mathbf{f}, \mathbf{v}) + d(\mathbf{f}, \mathbf{v}_k)) + \max(0, m - d(\mathbf{v}, \mathbf{f}) + d(\mathbf{v}, \mathbf{f}_k))$$

where $\Theta_e = \{\mathbf{W_I}, \mathbf{W_T}\}$ and $\mathbf{v}_k$ are the non-matching descriptions for image $\mathbf{f}$ and $\mathbf{f}_k$ are the non-matching images for description $\mathbf{v}$. This ensures the distance between matching $\mathbf{v}, \mathbf{f}$ is learned to be smaller than the non-matching pairs.

2.3.2 Ontology

An important aspect of our recommender system is the application of fashion ontology. A fashion ontology encodes the visual-semantic relationships between fashion items, the users and the context. Finding similar and compatible items for the query items could be an expensive process, especially when the search space is large, which is typically true for the fashion domain with rich varieties in item types and features. To make this process more efficient and provide further flexibility to incorporate the user preferences, using ontology in the selection process can reduce the search space into a subset of items that fit into the given context and follows the user taste.

Multimedia Web Ontology Language

Multimedia Web Ontology Language (MOWL) [13], is an ontology language that is capable of analyzing visual properties of garments while respecting fashion concepts. MOWL is constructed based on a causal model of the world. In this model, patterns and features are extracted from abstract concept and projected into specific media properties in multimedia instances. Through these instances, it is possible to link the media features to different media forms in various level of abstraction. The feature/pattern extraction is not deterministic and such uncertainties is combated by giving the linking a probabilistic nature - the uncertainty in the media properties is denoted by a set of Conditional Probability Tables (CPT's).

The ontology representation also allows reasoning of properties in concepts, usually through inheritance. For example, a garment made of silk should inherit the shininess property of silk. Using this reasoning, an Observation Model (OM) for a concept can be derived through a Bayesian network, such model will contain a large set of media property descriptors for the corresponding concept. Furthermore, MOWL supports formal representation of spatio-temporal relations which is useful in formulating a relationship between media features. MOWL is independent of the media types and application domains since the ontology representation and reasoning schemes are generic.

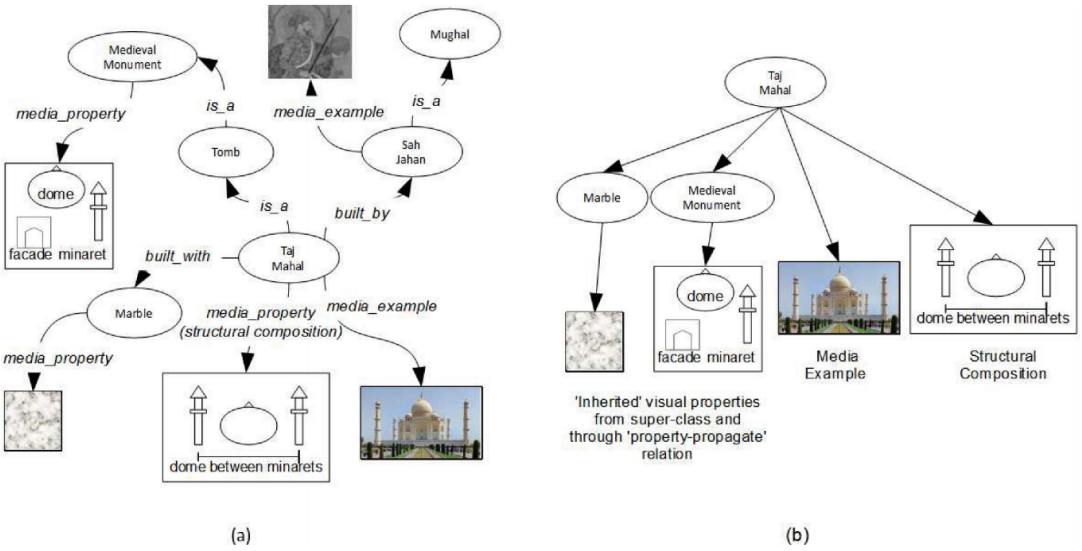

Figure 2.7: (a) a domain ontology and (b) a possible corresponding observation model. MOWL is independent of the media types and application domains. [13]

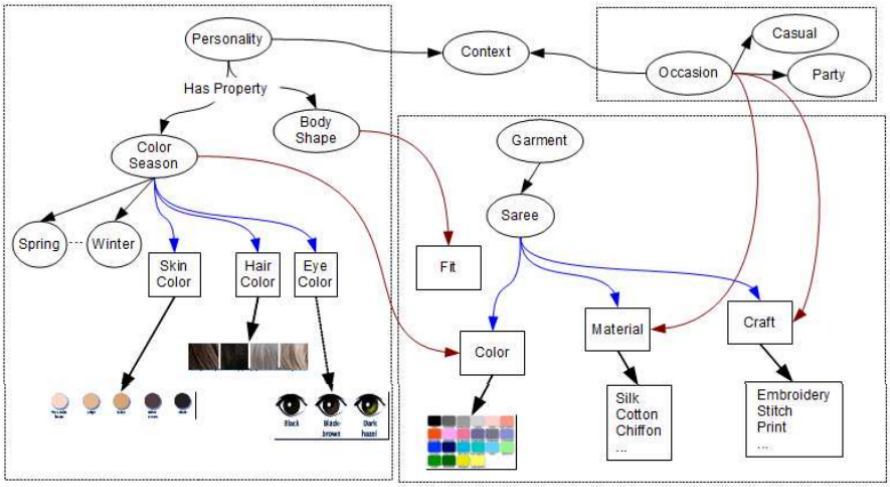

Figure 2.8: figure
A section of fashion ontology. [14]

Ontology based modeling

One of the applications of MOWL is the prototype recommender system [14] which employs content-based filtering on the Sarees[3] domain. In this prototype, the fashion ontology is adapted to the Sarees domain. For example, general garment ontology may include fit, color, material and craft as its entities, but for Sarees the *fit* property is not applicable because Sarees can be wrapped around in different ways for different body shapes. Figure 2.8 illustrate this in greater detail, with the brown

[3] *Saree* is an ethnic garment typically worn by women in the Indian subcontinent.

arrows representing expert views for associating item properties with personality and occasion attributes.

Figure 2.9 visualizing the learned OM for the reasoning of a Saree recommendation. The OM has a root node of *context* X with two children nodes P, for *personality* and O, for *occasion*. The personality node is further linked with the *user color season profile* (UC) and *user body shape profile* (UB). The *user color season profile* provides compatible colors ($G1, G2, ...$) for the user. Similarly, *occasion* includes properties like material and craft, which may infer silk-made garments for parties and cotton-made items for casual wear.

An alternative fashion ontology is Vetivoc [15], an industrialized project that provides generalized modular ontology for the fashion, textile and clothing (FTC) domain. While MOWL and other ontology frameworks are based on language processing or statistical analysis, VetiVoc integrates a social reading of the domain definition by considering the ecosystem, practices and the diversity of the domain.

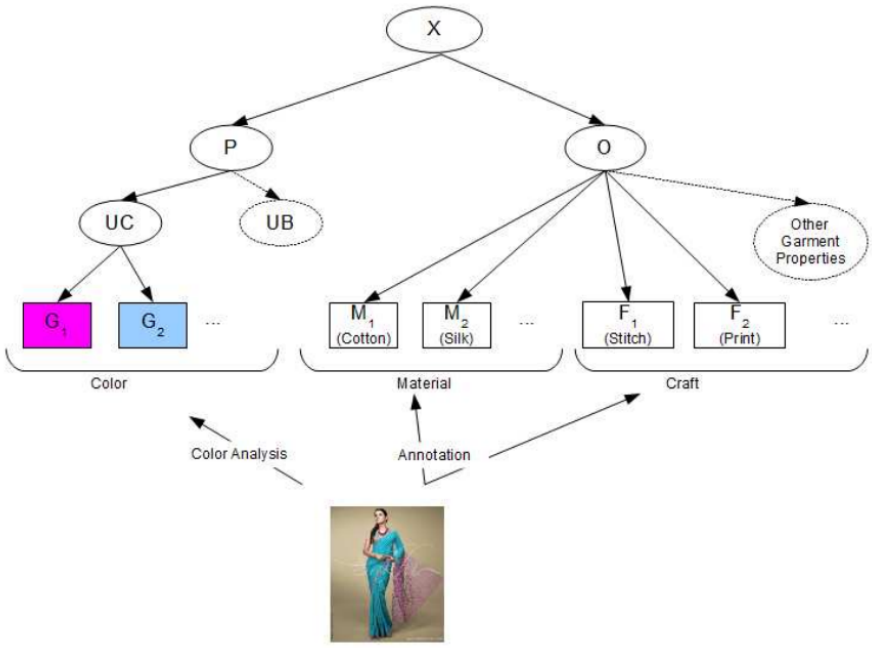

Figure 2.9: figure
Reasoning for Sarees recommendation. [14]

VetiVoc recovers an interesting structural property of vocabularies used to describe items in the FTC domain. It denotes the vocabularies into three consecutive levels: (1) *Object*, (2) *Support* and (3) *Variation*. Here, the signification of a fashion item is determined by the support variation. For instance, in figure 2.10 the *shirt* object supports Italian *collar* is classified as an elegant shirt because of the variation

Italian. If the variation is *American* then the semantic of the shirt would be relaxed.

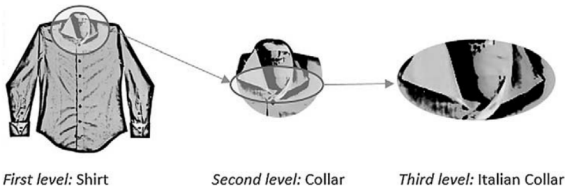

Figure 2.10: A shirt with an Italian collar. [15]

More formally, the object level represents the most generic approach to describe the item. For example, *jacket* describes every kind of clothing that dresses trunk and arms. The *support* level provides refined details about the shape: (1) *volume*, e.g. the length, width, degree of fullness, etc, (2) *parts*, e.g. the type of collar, heel, etc (3) *effects*, e.g. puffed, pleated, etc, (4) *color* and (5) *closing*, e.g. button, zip, etc. The *variations* level is more abstract, relating less to the visual impact and more towards the concept of the items: (1) the *cut* or type of design, e.g. tailored or sportswear, (2) *material* and (3) *context* of usage, e.g. for sport, work, parties, etc.

VetiVoc is built with modules distributed over four levels corresponding to four abstraction degrees, as seen in figure 2.11. The top-ontology contains high-level concepts and relations; the core-ontology consolidates and structures the main domain concepts; below that is a set of specific modules and at the bottom is a set of consolidation modules.

The top-ontology contains generic concepts to cover any domain. The core-ontology inherits from top-ontology and defines the set of concepts structuring the FTC domain. Fashion items are categorized into five parts: *Clothing, Underwear, Shoes, Accessories, Headwear*. A specific module is dedicated to each of these parts. Specific module inherits directly from core-ontology and indirectly from top-ontology. They express opinions to items, for example it defines the feature of having collar in trunk clothing or the concept of collar being Italian. Lastly, consolidation modules are used for aggregation and single entity operation.

2.3.3 Reasoning recommendation

As our proposed recommender system uses similarity, compatibility and ontology for recommendations, using distance between items as explanation [3, 4] is insufficient since it does not cover the ontology aspect. Argumentation-based recommendations [2] inspires the use of argumentation framework [16, 17] to justify our recommendations. In this work, the application domain is film but the underlying concept is transferable to the fashion domain.

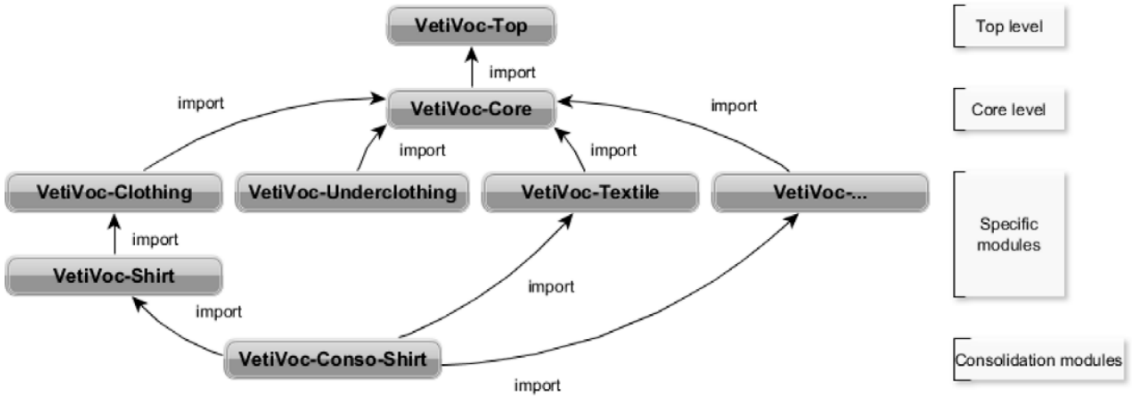

Figure 2.11: Implementation structure of VetiVoc. [15]

In this Aspect-Item recommender system, the ontology defines *items* (e.g. movies) are associated with *aspects* (e.g. horror), where *aspects* have the property *types* (e.g. genre), and *users* can provide *ratings* on *items* and/or *aspects*. The formal definitions are:

- An Aspect-Item framework (A-I) is a 6-tuple $< \mathcal{I}, \mathcal{A}, \mathcal{T}, \mathcal{L}, \mathcal{U}, \mathcal{R} >$ such that:
- $\mathcal{I}$ is a finite, non-empty set of items
- $\mathcal{A}$ is a finite, non-empty set of aspects and $\mathcal{T}$ is a finite, non-empty set of types, where each aspect in $\mathcal{A}$ has a unique type in $\mathcal{T}$; for any $t \in \mathcal{A}$, we use $\mathcal{A}_t$ to denote $\{a \in \mathcal{A} \mid \text{type of } a \text{ i s} t\}$;
- the sets $\mathcal{I}$ and $\mathcal{A}$ are pairwise disjoint; we use $\mathcal{X}$ to denote $\mathcal{I} \cup \mathcal{A}$, and refer to it as the set of item-aspects
- $\mathcal{L} \subseteq (\mathcal{I} \times \mathcal{A})$:= is a symmetrical binary relation
- $\mathcal{U}$ is a finite, non-empty set of users
- $\mathcal{R} : \mathcal{U} \times \mathcal{X} \to [-1, 1]$ is a partial function of ratings

Before the system can make recommendations, an additional *user profile* is required. The profile π_u of user $u \in \mathcal{U}$ is made up of: (1) a collaborative filtering constant $U^u_{c.f.} \in [0, 1]$, (2) $\forall t \in \mathcal{T}$ a type importance constant $U^u_t \in [0, 1]$ and (3) $\forall v \in \mathcal{U}$ such that $u \neq v$, a similarity constant $W_{u,v} \in [0, 1]$. With the above definitions, for any $u \in \mathcal{U}$ the predicted item rating $\mathcal{P}^{\square}_{\mathcal{I}}$"$\mathcal{I} \to [-1, 1]$ is obtained as follows:

if $\mathcal{R}(u,i)$ is defined then $\mathcal{P}^u_{\mathcal{I}}(i) = \mathcal{R}(u,i)$; else

if $\rho^u(i)$ is defined and $\sum_{t\in\mathcal{T}} \mu_t = 0$ then $\mathcal{P}^u_{\mathcal{I}}(i) = \rho^u(i)$; else

if $\rho^u(i)$ is undefined and $\sum_{t\in\mathcal{T}} \mu_t > 0$ then

$$\mathcal{P}^u_{\mathcal{I}}(i) = \frac{\sum_{t\in\mathcal{T}} \mu_t [\sum_{a\in\mathcal{L}_t(i)} \mathcal{P}^u_{\mathcal{A}}(a)]/|\mathcal{L}_t(i)|}{\sum_{t\in\mathcal{T}} \mu_t}; \text{ else}$$

if $\rho^u(i)$ is defined and $\mu^u_{c.f.} + \sum_{t\in\mathcal{T}} \mu_t > 0$ then

$$\mathcal{P}^u_{\mathcal{I}}(i) = \frac{\mu^u_{c.f}\rho^u(i) + \sum_{t\in\mathcal{T}} \mu_t [\sum_{a\in\mathcal{L}_t(i)} \mathcal{P}^u_{\mathcal{A}}(a)]/|\mathcal{L}_t(i)|}{\mu^u_{c.f.} + \sum_{t\in\mathcal{T}} \mu_t}; \text{else}$$

$\mathcal{P}^u_{\mathcal{I}}(i) = 0$

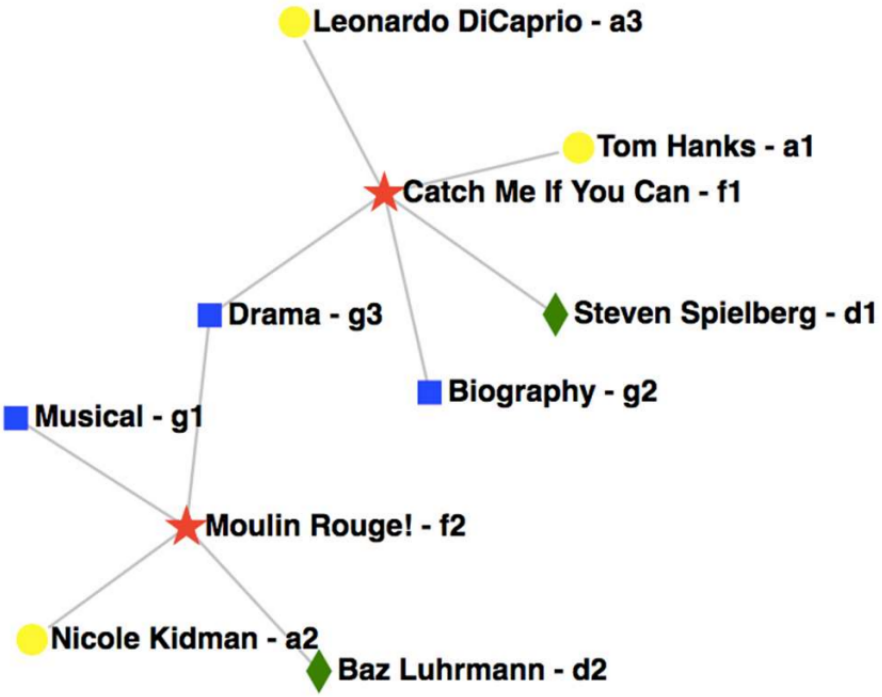

Figure 2.12: Example components of an A-I visualized as a graph, with items given by stars and types: *genres* (whose aspects are squares), *actors* (whose aspects are circles) and *directors* (whose aspects are diamonds). Each node's label is of the form (Name - $x, \mathcal{R}(u,x), \mathcal{R}(v,x), \mathcal{P}^u_{\mathcal{X}}(x)$), with $\mathcal{U} = u, v$ and _ as 'undefined'. [2]

Chapter 3

Data

Three different datasets are used as the building blocks for the recommender application and underlying model. They are (1) the training dataset for the neural network to perform topic modelling for the ontology framework, (2) the definition of the aforementioned ontology framework and (3) the sample user dataset to enable the recommender to adapt to user preferences.

3.1 Training data for topic modelling

Topic modelling on ontology keywords is one of the backbones of the recommender, with each keyword serve as a ontological feature. Similarities and compatibilities between these features are used when producing recommendations, and these metrics are calculated based on this clustering model. We have obtained the dataset from a past research [4] to feed into a tensorflow neural network in learning this clustering model.

The dataset is collected from Polyvore (www.polyvore.com), a popular fashion website where users create and upload outfit/fashion ideas as shown in figure 3.1.[1] The dataset contains the following multimedia data:

- over 17000 uploaded fashion sets with metadata such as upload timestamp, numbers of view and like, each fashion set contains multiple fashion items
- over 110000 unique fashion items, each fashion item is associated with its number of like
- both fashion sets and items are paired with description (includes branding, color, material, style etc) and image
- the fashion sets and items combine to a vocabulary of size over 3000 words

[1] www.polyvore.com became inaccessible in 2018 after its acquisition by Ssense.

Figure 3.1: A sample outfit upload by users to Polyvore. A typical outfit contains a fashion item list, i.e., pairs of fashion images and corresponding descriptions [4]

However, within all the uploaded fashion sets, some of them are not outfit ideas but rather interior designs that mainly focus on furniture and decoration or cosmetic ideas focusing only on accessories such as lipsticks and foundations. These fashion sets were filtered to reduce noise in the data. Besides that, we also dropped fashion sets that has less than 300 views or 30 likes, which is 10% of the average number of views and likes respectively. We believe they were out-of-fashion and should not be accounted for a good measure in training the model. Lastly, we disregarded the title description of the fashion outfit as a significant amount of them are unrelated to the items in the set, for example *"Kenzie's first day of school"*. This leave us with an usable dataset of:

- more than 14000 fashion outfits and 100000 fashion items with detail metadata
- a vocabulary of 2400 words

3.1.1 Data pre-processing

A crucial step before building our model is to analyze and standardize the data to make sure that it is in an usable format for further training and testing process in later stages. Several pre-processing techniques were applied for this purpose:

- Firstly we transformed the raw data from `Json` format to `pandas.DataFrame` representation for easier processing and manipulation in future stages. An example fashion outfit in the raw data in `Json` is shown below:

```
{
  "name": "Work Wear",
  "views": 335,
  "items": [
    {
```

```
      "index": 1,
      "name": "river island white twist wrap blouse",
      "price": 42.0,
      "likes": 5583,
      "image": "http://img1.polyvoreimg.com/cgi/...",
      "categoryid": 17
    },
    .
    .
    .
    {
      "index": 6,
      "name": "pre-owned hermes silver bracelet",
      "price": 599.0,
      "likes": 320,
      "image": "http://img1.polyvoreimg.com/cgi/...",
      "categoryid": 106
    }
  ],
  "image": "http://ak2.polyvoreimg.com/cgi/img-set/...",
  "likes": 42,
  "date": "One year",
  "set_url": "http://www.polyvore.com/work_wear/...",
  "set_id": "152064873",
  "desc": "#princeofwalescheck"
}
```


- Common NLP pre-processing procedures were used on the description texts, such as lower-casing, removing stop words, lemmatization, etc. Standardizing the description reduced noises and sparsity of vocabularies, hence enabling a clearer definition between clusters in the model.

- Finally, we calculated a `dup_score` per outfit based on their number of likes and views to represent their "correctness", this was used in the training process of the clustering model with neural network. The formula to generate `dup_score` is:

$$[if\ (x\_likes > 10)\ then\ (x\_likes - 10)^{0.9}\ else\ x\_likes] +$$
$$[if\ (x\_views > 10)\ then\ (x\_views - 10)^{0.7}\ else\ x\_views],$$
$$where\ x\_likes = \left\lfloor \frac{likes}{avg\_likes} * 2 \right\rfloor, x_views = \left\lfloor \frac{views}{avg\_views} * 2 \right\rfloor$$

  Further details of its usage and training process are discussed in section 4.2.2.

	s_id	s_likes	s_views	i_category_id	i_name	dup_score
0	214181831	394	8743	4495	{embroidery, neck, mock, sweatshirt, suede}	7
1	214181831	394	8743	25	{hooded, luxe, double, zip, jacket}	7
2	214181831	394	8743	27	{high, hem, citizen, rise, humanity, jean, roc...	7
3	214181831	394	8743	261	{suede, boot, short, tie}	7
4	214181831	394	8743	259	{school, cloth, travel, backpack}	7
5	214181831	394	8743	1967	{}	7
6	214181831	394	8743	2	{polyvore}	7
7	120161271	9	188	21	{tshirt, distressed, nirvana}	0
8	120161271	9	188	237	{rock, black, skinny, rag, jean, w, bone}	0
9	120161271	9	188	49	{van, black, authentic, trainer, mono}	0

Figure 3.2: An snippet of the pre-processed data in pandas.Dataframe.

3.1.2 Data features

An overview of the training data structure and features is shown in figure 3.2, the fields of each records are as follows:

- s_id: unique identifier of the fashion set
- s_likes: number of likes on the fashion set
- s_views: number of views on the fashion set
- i_category: a look up identifier for the item to its category, e.g. long skirts
- i_name: pre-processed tokens from description text
- dup_score: indicator of the fashion set "correctness"

3.2 Ontology data

As mentioned above, an ontology framework is integrated into our recommender. There exists some sophisticated fashion ontology framework from past research [12, 14], however the code was not made public so we created a minimalistic one using vocabulary from the polyvore.

We began building the framework by evaluating frequently used words in the polyvore data, then we cross-referenced popular fashion brands such as Zara, ASOS, Burberry, etc and two experts [18] in the fashion industry to decide their ontology class and context score. Three ontology classes were implemented into the definition, which are `item, color` and `material`.

`Item` class refers to the product type of the item and has sub-classes `top, bottom, footwear, full-wear, outerwear` and `accessory`. `Color` class has `root color` sub-classes to group more refined color variation together, for example the color wine is associated with root colors pink, red and purple. `Material` class has no sub-classes and is self-explanatory. These three ontology classes have 102, 73 and

```
{
 "def_": ["item", "color", "material", "context"]
},
{
  "def_": ["id","class" , "context_score"],
  "def_topic_": ["top", "bottom", "foot", "accessory", "fullwear", "outerwear"],
  "def_id_range": ["0-199", "200-399", "400-599", "600-799", "800-899", "900-999"],
  "item": {
    "top": [0, 0, [1, 1, 1]],
    "shirt": [1, 0, [1, 1, 1]],
    "blouse": [2, 0, [1, 1, 1]],
    "tshirt": [3, 0, [0, 1, 4]],
    "knit": [4, 0, [0, 1, 1]],
    "tank": [5, 0, [1, 1, 1]],
    "cami": [6, 0, [1, 2, 2]],
    "jersey": [7, 0, [0, 1, 4]],
    "bra": [8, 0, [0, 0, 1]],
    "hood": [9, 0, [0, 1, 1]],
    "polo": [10, 0, [1, 1, 1]],
    "tunic": [11, 0, [1, 3, 3]],
```


Figure 3.3: An snippet of the ontology framework definition.

44 keywords respectively. Apart from these three classes, `size, gender` were also considered but abandoned at the end because the related word had a low occurrence counts and low unique vocabulary size.

The context score is represented as an array with three values, with each value illustrating the correlation to a formal, semi-formal and casual setting respectively. The values are in the form of ratios, for example, a tuxedo is designed for formal settings only so it has a context score of `[1, 0, 0]` while a suit can be worn in both a formal and a semi-formal setting so its context score would be `[1, 1, 0]`. A code snippet of the ontology framework definition is shown in figure 3.3.

3.2.1 User profile data

User profiles are mainly made up of two major components, saved outfits and user preferences. We kept the data stored in user profiles as minimal and generic as possible to accommodate future updates to the ontology framework.

Saved outfits are represented by a list of dictionaries, where each dictionary contains the information of one outfit. In each outfit, items, materials, colors and images are stored in their own list respectively and using the position to correlate. Consider an outfit that has the following lists for items, materials and colors: ["polo", "jeans",

"trainers"], ["", "denim", ""], ["white", "", "black"], this means the outfit is composed by a white polo, a pair of denim jeans and a pair of black trainers. A unique id is also assigned to each outfit.

User preferences are also represented by a list of dictionaries describing each preference. The dictionary stores the preference id, preference type, preference value (e.g. actual bias), modifier value (e.g. user rating on the learned preference), applicable ontology categories and applicable ontological features (ids of the features within the categories). The rest of the user profile contains information such as user name, user id, current max outfit id, etc. Figure 3.4 visualize this data structure.

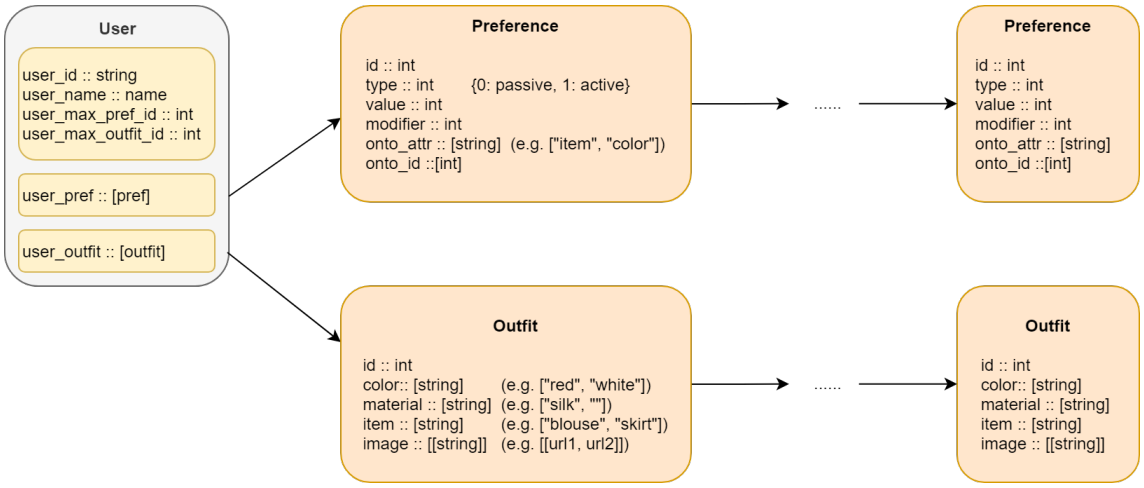

Figure 3.4: Architecture of the user profile data.

In this chapter we described the data we use in this project, how we represent the data to be used by our system and also give examples of the data for better visualizations. In the next chapter, we describe the system design and implementation.

Chapter 4

fAshIon Model

In this chapter we present the general design of **fAshIon: Personal Shopper** and justifications to the model decisions we have made.

4.1 Application design

fAshIon is a specialized recommender on auto-completing outfit using semantics of fashion items and user preferences, with functionality to (1) provide reasoning for each recommendation and (2) incorporate user interactions with such reasoning, enabling a continuous feedback loop to improve future recommendations. These key features differentiate **fAshIon** from other generic recommenders that recommend items for the input item based on similarity, or recommenders which provide only results but no explanations.

There are two problems we aim to tackle with **fAshIon**. The first problem is the oversight of significance of fashion item's semantic when making recommendations. Recommenders which rely on collaborative filtering (determining recommendations based on the preferences of other users with similar profiles), content-based filtering (recommending items close to user's usual taste and past selections) or a mixture of the two filterings may not necessarily understand the semantics of fashion items, as the semantics are "modelled" as a side effect.

For example, given a suit as the input, these recommenders may suggest a pair of brogue shoes. However, this is because the current user or other users with similar user profiles recognized the two items as fitting together in a formal/semi-formal context before. The recommenders themselves do not have the notion of these items having a formal/semi-formal semantic context, therefore we regard the matching of suit and brogue shoes in a formal context as a side effect from the filters. Our approach to this problem is to introduce a context score for each item, which will be evaluated both when defining the overall context of the target outfit and when deciding individual item recommendations.

The second problem we are trying to solve is that common recommenders do not provide explainable recommendations, which hinders the improvement on the accuracy of general recommendations and the process of learning user's preferences. Consider a naive recommender that learned leather is a material that tends to be for casual occasions from items such as leather jacket and biker boots. It would restrain itself from recommending leather on items in formal context like leather brogue shoes, which is in fact a common choice. Without explanations, users would not be able to discover nor reflect this incorrect knowledge to the recommender.

This means that the recommender has to re-learn the correct semantics of leather passively through new formal outfits containing leather, which is typically a slow process and requires a large number of new data since the learning rate is usually small to minimize overfitting. In addition, with explanations and user interactions on them, **fAshIon** is capable of going beyond the initial learning rate and establish the user's own fashion rules much quicker dynamically.

Noted that **fAshIon** remains a proof of concept at this stage and does not support unseen words in the release build. This is because we associate handcrafted multimedia attributes to vocabularies according to their nature, and attempting to automatically generate these attributes for unseen words would result in high uncertainties and potential inaccuracies. Instead, we focus on fine tuning the attributes in seen words for a better model to showcase a clearer picture in how our approaches tackle the aforementioned problems.

4.1.1 Application pipeline

Figure 4.1 illustrates how data flows between components of our application in a typical use case. The different components and their roles are as follows:

- **User profile**:
  Data storing saved outfits from user and user preferences that are learned from the saved outfit or from user's feedback, as discussed in section 3.2.1.
- **User management**:
  A interactive panel allows users to manage their saved outfits and learned preferences.
- **User query**:
  This interface lets users to input items they have already chosen for the target outfit and send these items to the backend algorithm, which will return a corresponding list of recommendations.
- **Generic semantic module**:
  This component's objective is to output the baseline semantics for range of

items and attributes (color and material) it has learned from the training data. The output semantics are fed into the prediction module.

- **User-specific semantic module**:
  This component's objective is to output the learned targeted user's preference based on the user's saved outfits and user feedback if the latter exists. The output semantics are fed into the prediction module.

- **Prediction module**:
  The core of the application, where similarity between items, context scores of items and user preferences are taken into calculations to produce recommendations with explanations.

- **Recommendations, Explanations & Feedback**:
  Users are able to add the desired recommendations along with the input outfits and store the entire set in the user profile. Users can also rate the explanations given, which will be updated in user profile and used in future prediction accordingly.

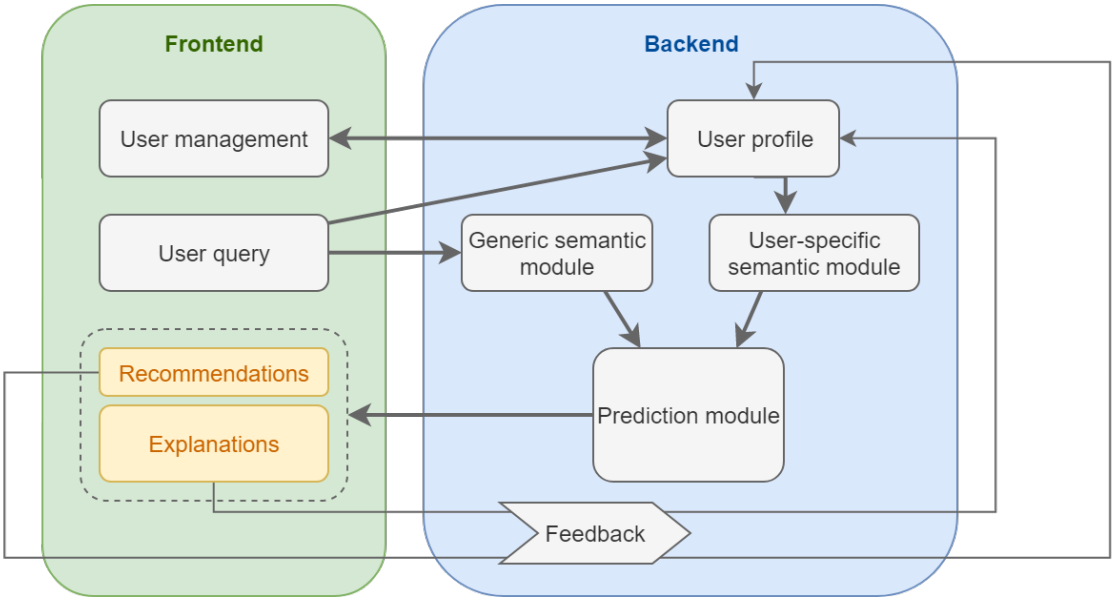

Figure 4.1: Data flow between components in **fAshion** application model.

4.1.2 Front-end

We have decided to keep our frontend minimalistic and as a mock up interface on localhost. Since our project focus is on exploring explainable recommendations and user interactions with the explanations, the frontend's main purpose is exposing this core mechanisms to users.

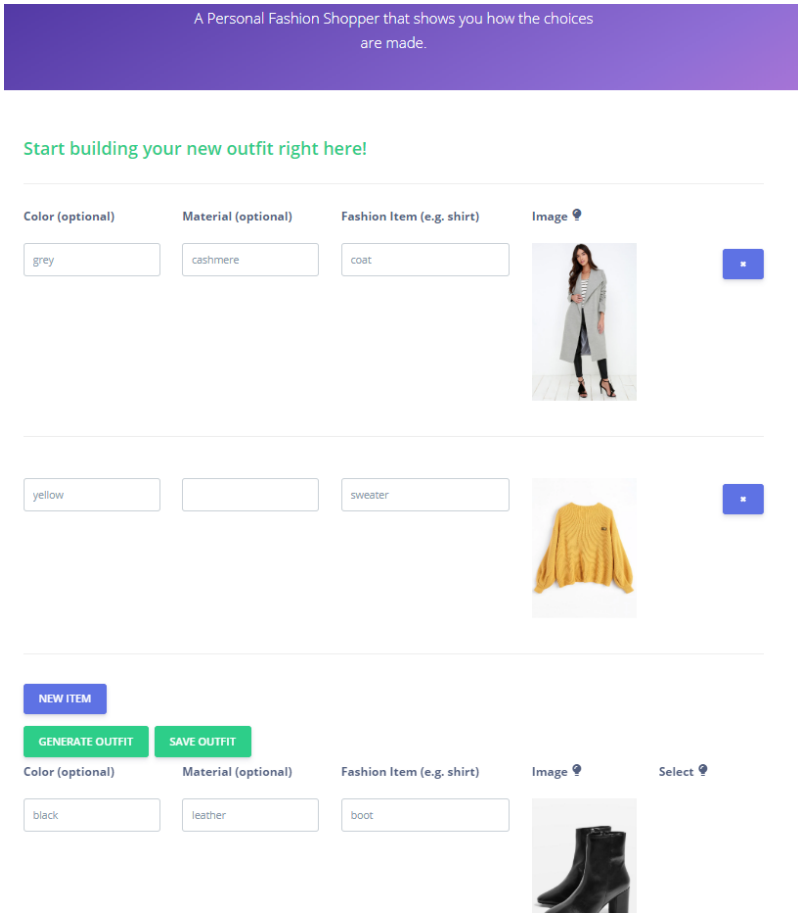

Figure 4.2: Prediction view in the frontend interface.

We built three views in our frontend for the users, the prediction view, saved outfits/rules view and statistics view. Figure 4.2 shows the prediction view, it provides a simple input/output layout, with the upper section being the input panel for the users to enter the already chosen items for the target outfit. The bottom half will be rendered after the users clicked the `Generate Outfit` button and after the webview received the recommendations response from the backend server. The users will then be able to review and interact with the explanations of individual recommendation, including increasing and decreasing the correctness of each explanation or even completely ruling it out or setting it as a fact that should always be considered.

Other than that, the user can flag the recommendations they like and save the entire outfit into their profiles. Alternatively, users may input an already well-formed outfit and save the outfit directly without using the predict function. To provide a clearer illustration of the outfit, each item is associated with a list of images found using a Google search engine with the corresponding description. Users are able to cycle through these images and the displaying image will be saved as part of the outfit data.
Saved outfits/rules view is shown in figure 4.3, in this view users can review and

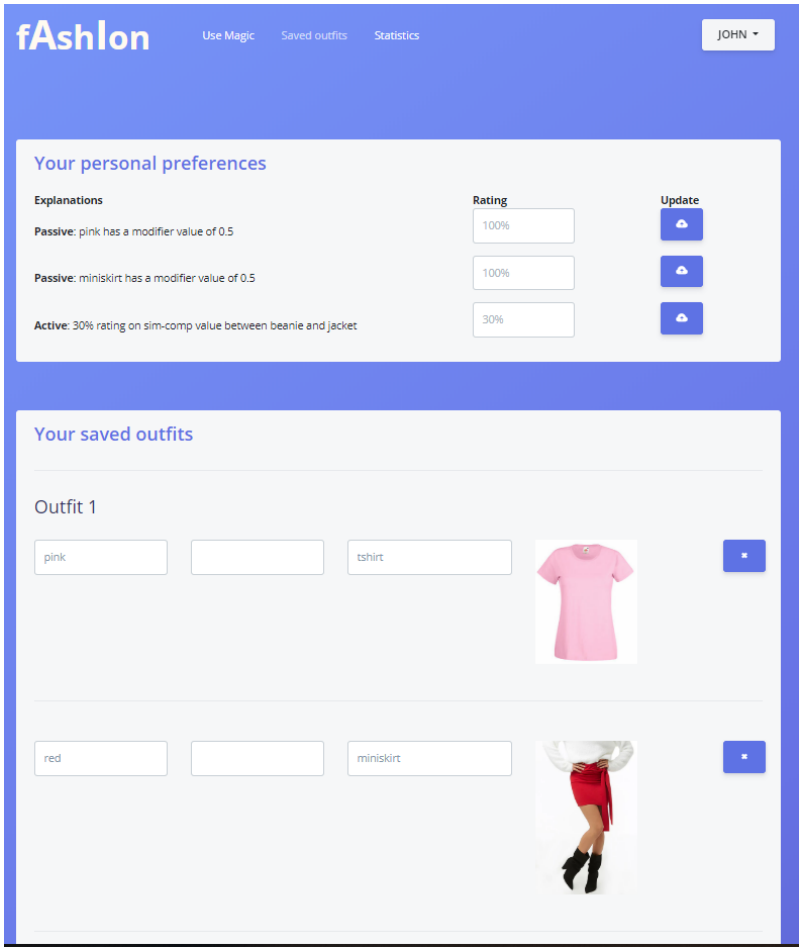

Figure 4.3: Outfit view in the frontend interface.

Figure 4.4: Statistic view in the frontend interface.

edit their previously saved outfit and established user preferences. Outfits and established user preferences are not associated, so deleting one will not remove the other and vice versa. Statistics view as shown in figure 4.4 allows the user to see visualizations of the base clustering of ontological features, which can be in same space (e.g. item to item), two spaces (e.g. item to color) or three spaces (i.e. item to color to material). Statistics view also visualize the user's outfit statistics, including top five selected materials, normalized distribution of user context score, etc. Together with the saved outfit view, they form the user's management component in the application pipeline.

4.1.3 Semantic modules and user feedback

These are the underlying models in providing explainable recommendations. The explanations are focused in two argumentative styles: similarity/compatibility between ontological features and user subjectivity.

Generic semantic modules

This module is the main source of outputting the similarity/compatibility value (sim-comp value) between ontological features. For better performance, we have stored all the values in the relative lookup matrices, for instance a 2D matrix to represent the sim-comp values between items and colors and a 3D matrix for sim-comp values among items, colors and materials. After these matrices are generated they are stored as part of the artefacts and read directly in future sessions.

Upon initiation, this module extracts ontology keywords (in item, color and material categories) from unique outfits in the training data and phrases a corresponding skip-gram model. This skip-gram model is used as input to a neural network model to train a topic model on different combinations of ontological features (i.e. items to items, colors to colors, items to colors, items to materials and all three together). An example of clustering within the item ontology is shown in figure 4.5.

With the clusters representing the topic models, each keyword is represented as a word embedding vector. We then compute the cosine similarity among these vectors for their corresponding sim-comp value. When calculating the values, we use the assumption that ontological features appearing within the same outfit tend to be compatible and distances between features in the topic model would reflect similarities. Improper triangles caused by the non-transitivity of compatibility [10] is handled using a weighted influence graphs between items, which is explained in further details in section 4.2.4.

Figure 4.5: Visualization of a snippet in the clustering in items-to-items topic model.

User-specific semantic modules

This module is responsible for outputting two types of user preferences on different ontological features, a passive one which is obtained from the user's previous outfits and an active one which is perceived via user feedback.

The passive user preference takes into account the number of occurrences of ontological features from all saved outfits. Repeating features will be marked and an corresponding bias is added, for example, if a user saved a lot of black and white outfit, then the color black and white will create an additional modifier when calculating their sim-comp values in future predictions. Modifiers on context are added to user profile with similar approach, for user submitting mostly business attire, the module will mark him/her as a formal person and tend to identify new outfit from that user to be in formal context.

For active user preferences, it requires the user to mark and input a correctness score for the explanation after a prediction is done. A modifier based on the correctness score will be given to the ontological feature associated with the marked explanation, which will be used in future predictions. For example, if a recommendation item heels is predicted and the user marked the explanation: *"...because heels*

has a similarity/compatibility score 0.341096 with dress" as incorrect, a negative modifier will be added between heels and dress.

4.1.4 Prediction module

The core module of the application include three different procedures that are used to calculate the color, material and item in a recommendation respectively, each procedure also has a variant sub-procedure when user preferences need to be taken into account. In this section we explain the intuition behind the predictions, implementation details are listed in section 4.2.4.

Item prediction

We will first go through the procedure when there is no user preference. The module will first decide on the outfit's context using the input items and materials. Then for each input items it obtains the list of sim-comp values for each pair of current input item and nearby item, after that the sim-comp values are modified by an item-to-item influence modifier. The intuition of this influence modifier is to model the case where different types of item have different amount of influence in selecting other items in an outfit. For instance, a bottom wear typically has more impact on the choice of the footwear than an accessory would have.

The module will proceed to aggregate the influenced sim-comp value for all items before generating the final sim-comp values. The final sim-comp values are calculated using the aggregated influenced sim-comp value, the context scores of the non-input items in the pair and the outfit context.

Finally, it ranks the items on final sim-comp value in descending order and populate the gaps in the outfit definition. An outfit should have a top and a bottom item or a fullwear as the first requirement, then a pair of footwear as the second requirement and optionally an outerwear and any number of accessories. The module will fill the gaps missing from input items, discarding items with item type that has already been chosen.

Two additional steps are taken when there is user preference to be considered. For passive preferences, an extra bias is used in calculating the outfit's context in the first step. As for active preferences, they are added in calculation before applying the influence modifier.

Material prediction

Material predictions are carried out after item predictions are completed and computed with respect to individual recommended item. The module obtains nearby

materials for that particular item and picks the top material using both sim-comp values, material context scores and the outfit context calculated earlier. With user preferences available, extra biases are used to update the sim-comp value and context scores.

Color prediction

Color predictions are also carried out after item predictions and computed respectively to each recommended item. However, unlike items and materials, we did not assign context score to color, instead we make use of the attribute root color. The module first gets root colors from input items, then combines with the root colors of nearby colors to the recommended item. A distributed probability of root colors will then be calculated and used to return a non-deterministic choice of recommended color. If applicable, the user's preferred color scheme will be added in the stage of gathering root colors.

4.1.5 Accommodating unseen words

Our original design to handle unseen words is to make use of an neural network that given a English word as input will output a corresponding word-embedding vector. We could train such a neural network with our vocabularies and corresponding embedding vectors from the topic model. After getting the embedding vector of the new word, we can obtain its semantics such as context scores and root colors from the nearby ontological features. A simple scenario we have envisioned is learning new word *orange*, which would gives a vector close to keywords *red* and *yellow*. Calculating new sim-comp value is trivial and we could obtain the root colors from red and yellow, preform some calculations and assign it to orange. Future predictions can then be carried out with the new word orange.

However, after some iterations of implementation, we discovered that there were subtle semantics that could not be learned simply by having a embedding vector learned from nearby items. For the ontological class items, while the above example logic was applicable to the context scores, learning the item type yields poor performance. This was mainly caused by the clustering in the topic model not being solely based on similarity but also compatibility, so compatible items with different item types would then became noises in the computation. Because of this compatibility noise issue, obtaining sensible root colors was also situational. As for materials, since the vocabulary size of material is too small for a material-to-material topic model, while the logic was theoretically correct, we lack the topic model to execute it. If we use a cross space topic model such as item-to-material one, then we would circulate back to the compatibility noises.

Besides that, we also had to solve the problem of a small sample size for unseen

words and we rely heavily on the users giving appropriate training target. Together with the obstacles mentioned above, we decided handling these gracefully was out of this project scope and instead we should focus in improving the quality of explanations for recommendations and the feedback loop. Therefore we abandoned the support for unseen words in this project, but this functionality remains as a potential extension for future work.

4.2 Program implementation

In this section we will discuss the technicalities of **fAshIon**, elaborating on the formulas, models and more. Figure 4.6 illustrates the system architecture of the application model.

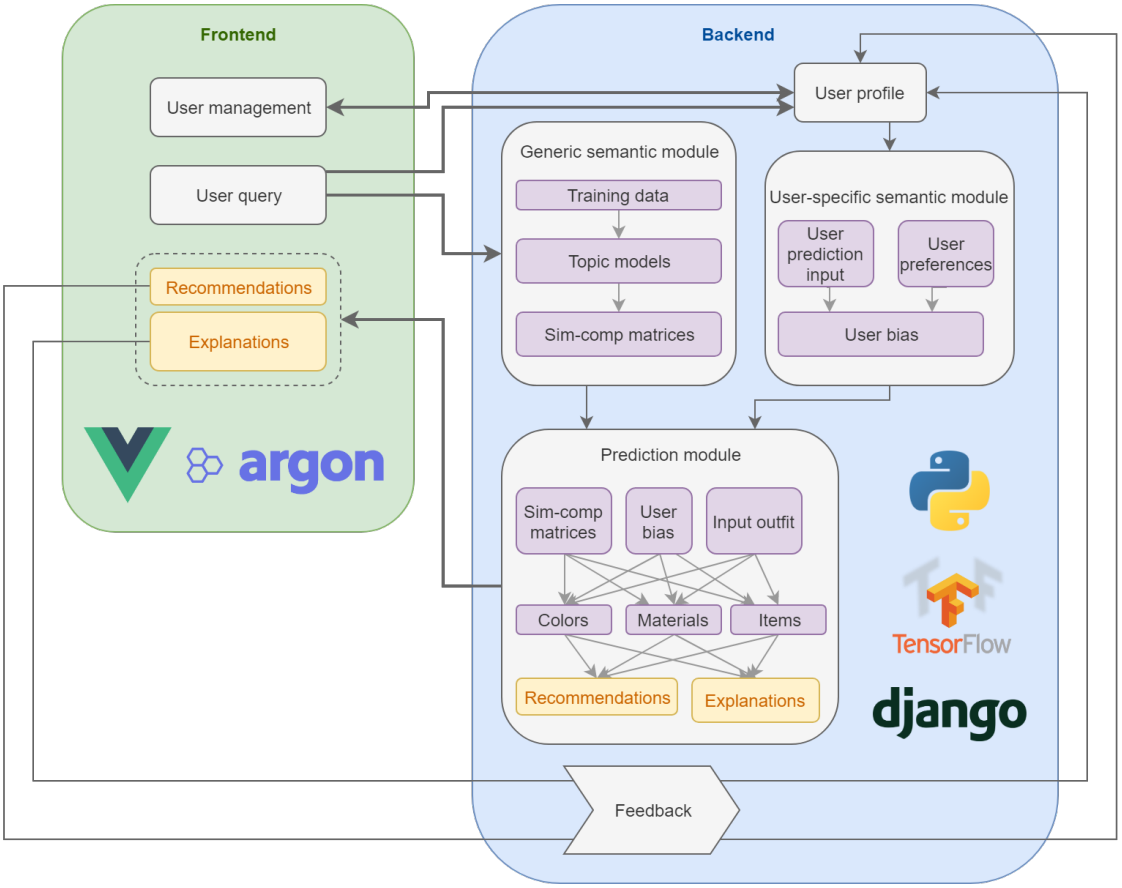

Figure 4.6: Architecture diagram for **fAshIon**.

4.2.1 Frontend

The frontend is built using Vue.js framework on a template design system [19]. We removed most of the components in the original template, keeping only the general

layout and CSS files. The frontend does not carry out computation for any values, all the JavaScript code are responsible for phrasing data only.

4.2.2 Generic semantic module

The follow steps are taken to transform the pre-processed training data to matrices containing the sim-comp values between ontological features:

- The module extracts ontology keywords for each item, putting these keywords in one of the three columns corresponding to their categories (item, color or material).
- Keywords in the chosen categories are then grouped together within the same outfit, for example when training for an item-to-item topic model, keywords in the color and material categories are ignored.
- Each group is duplicated N times where N is the outfit's dup_score. The duplicated set of words are shuffled and phrased into a skip-gram model with a windows size of 2 (2 context words before and 2 after) and no padding.
- The skip-gram model is used as the input to a tensorflow neural network with a loss function of noise-contrastive estimation training loss and a gradient descent optimizer. This neural takes pairs of target words and context words from the skip-gram model to use as the inputs and labels, outputting a word embedding vector for each unique input ontology word. We chose a common word embedding dimension of 128 for simplicity. We also employed mini-batch training and the use of negative examples to enhance the training performance.
- To obtain the sim-comp matrices from the learned topic model clusters, we first checked what ontology categories are in use. For same-category or paired-categories topic models, we use the cosine similarity between vectors to represent the sim-comp value. For the topic model with all three ontology categories, the sum of cosine similarities between each pair is used instead. The sim-comp value is stored within the matrix with the corresponding definition id of the ontology keywords.
- The definition ids may differ from the cluster ids used by the neural network itself, because in our ontology framework we reserved a range of ids for each item types while we would squash the unused identifier range when training for the neural network. Consider the top item type, it has the id range 0 - 199 but only 10 top items are declared in the framework, meaning the ids in used are 0 - 10. When we input keywords in the item category, ids from 11 - 199 are used for the next item type items. For example, the item jean has a definition id of 200 but a cluster id of 10, since it is the first entry in bottom-typed items after 10 entries of top items.

4.2.3 User feedback

After the users have been saving three or more outfits, the module will start to learn the passive user preferences. There are in total four type of semantics that can be learned in a passive preference: item preferences, material preferences, color preferences, and context preferences.

For item preferences, any item that is presented more than 3 times across all outfits will be assigned a sim-comp value modifier. The modifier value is calculated by

$$\text{value} = min\Big(\frac{\text{number of item occurrence}}{\text{number of outfit}}, 0.5\Big)$$

The material preferences are computed in a similar fashion. Color preferences are obtained slightly differently, unless the target color has appeared more than 70% among all the items, the color modifiers are assigned to the root colors instead, in order to find a color theme the user prefers and recommend variant colors under that color scheme. For instance, the colors amber, ruby and wine are often picked by the target user, but none of the individual color is dominantly selected for over 70% of the times. From these colors the module would conclude the user favors a red color scheme and add bias to all children color of red in future prediction.

Lastly, context preferences are obtained by normalizing the sum of the modified context scores for all items, with context modifiers based on the item type. The context modifier for each item type is as follows: {top: 4, bottom: 4, fullwear: 8, footwear: 4, outerwear: 3, accessory: 1}. Also recall that a context score is in the form of an array of three values (e.g. [0.2, 0.2, 0.6]), representing the likelihood of the item or material fitting in a formal, a semi-formal and a casual context. In the aggregated context score, any likelihood that has a value over 0.5 will result in an extra bias of value 0.35 being attach to that context. Therefore, if a final context score of [0.2, 0.2, 0.6] is calculated, an extra bias will be attached to the casual context.

On the other hand, there is no minimum outfit numbers before users can update their active preferences. An active preference is established whenever the user provides feedback on an explanation, which includes both reasoning with sim-comp value in generic semantics module and passive user preferences. Figure 4.7 demonstrate how users can interact with explanations, users can rate the explanation for correctness from -200% to 400%. A corresponding modifier will be added, if the explanation is the sim-comp value between two ontology keywords, then the modifier is linked between them, but if the explanation is on passive user preference, the modifier is attached to the target ontological feature in the passive preference. Further details on how this is used in calculation is stated in section 4.2.4.

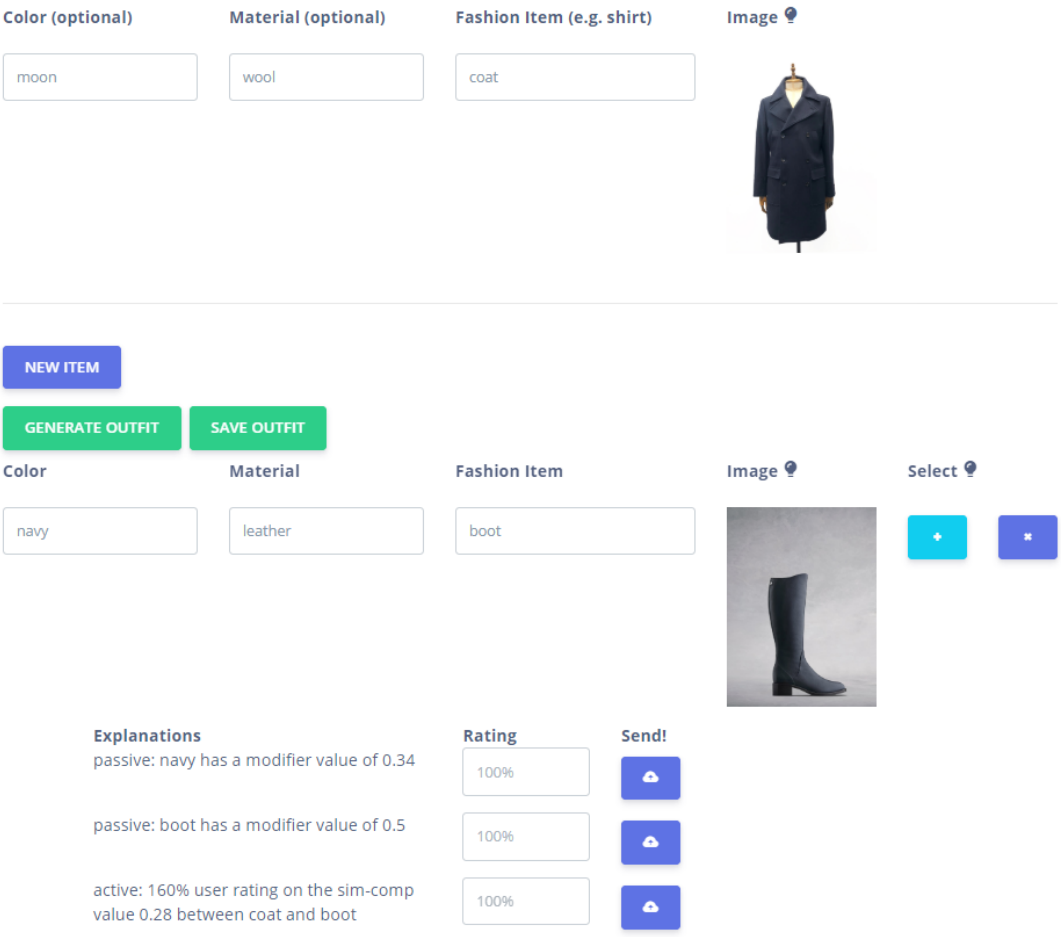

Figure 4.7: The recommendation of navy leather boot with its explanations.

4.2.4 Prediction module

We will first talk about how the generic procedure works, then proceed to explain how user preferences are integrated and finally go through an example for a clearer illustration.

Item prediction

- The module will first attempt to identify the context nature of the desired outfit, which can be formal, semi-formal or casual. It iterates through each input item and extracts the corresponding context score from the item itself (and material if possible). Then normalize and multiply these scores with the context modifier introduced in section 4.2.3. The result is aggregated and after examining all the inputs, the aggregated score is normalized again and used as a probabilistic model for the outfit context.

- For example, if the final aggregated score is normalized to [0.75, 0.2, 0.05], it means with the given inputs the module determines that the user has a 75% probability of creating an outfit for a formal context, and 20% and 5% probability of creating a semi-formal and casual outfit respectively.

- In the next step the module iterates through the input items again and computes an *influenced sim-comp value* between the current input items and the all other items. The influenced sim-comp value is calculated by multiplying the original sim-comp value by an influence modifier. The influence modifiers between each item type is illustrated in figure 4.8.

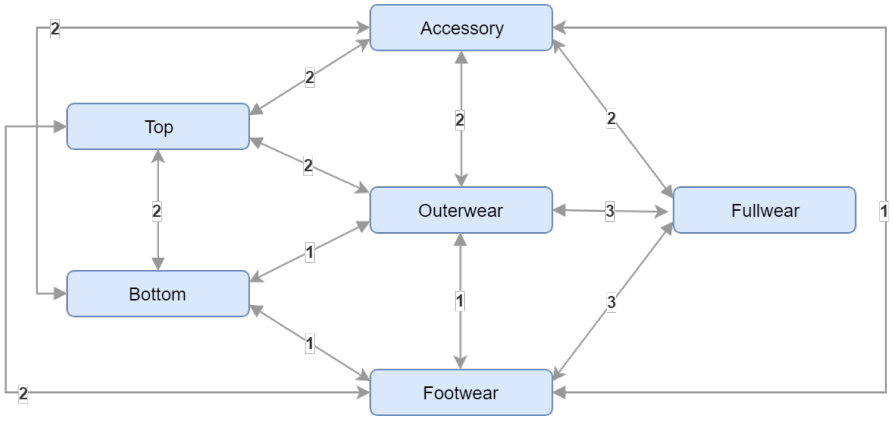

Figure 4.8: Influence modifiers between different item types, note that the modifiers are bidirectional. And same item type has an influence of 1 which is hidden for visual clarity.

- With both the influenced sim-comp value (ISCV) and outfit context nature calculated, the module can use them to produce a final sim-comp value (FSCV)

with the following formula:

$$FSCV_{item} = \sum_{\substack{input \in inputs, \\ item \in all\_items}} FSCV_{input,item}$$

$$FSCV_{input,item} = ISCV * [\text{item context}]^{\top}[\text{outfit context}]$$

where the item context is the base context score defined for every item. This FSC value is calculated between every input item and the remaining items that are not the currently selected input item.

- All items are then sorted in descending order on their final sim-comp value, then selecting the top 10 items that do not have repeating item types within an outfit. At any time the recommendation on a most popular item type can only have more than 3 items that one with the least populated item types. Recall our definition on a fashion outfit introduced in section 3.2, an outfit should have a top and a bottom item or a fullwear as the first requirement, then a pair of footwear as the second requirement and optionally an outerwear and any number of accessories.

- Consider a user who inputs a dress and handbags, recommendations will be made on footwear, outerwear and accessory. Fullwear items are not in recommendations because the user has already chosen a fullwear item. Top and bottom items are not recommended because they also act as duplicate items since typically people would not be wearing a t-shirt over a dress. There are two exceptions to the no-duplicating item types rule, firstly accessories is always included in recommendations and secondly if both top and fullwear items are presented in the input, then bottom items will be recommended regardless of the presence of a fullwear input, similar case for both bottom and fullwear items in input.

Material prediction

- Material prediction is started after item prediction is completed and also make use of the outfit context obtained during item prediction.

- The module iterates through all items in the recommendation queue and computes the influenced sim-comp value between the items and all materials. Then the material with the top sim-comp value will be selected.

Color prediction

- Color prediction is started after material prediction. First compute the sim-comp values of all colors using the color in input as the input sim-comp values, i.e. color-to-color pairs. From the input sim-comp values the module extracts

a color theme from the 10 colors with highest sim-comp value, by aggregating the occurrence count of root colors in these colors then normalizing the counts. If only less than two colors are assigned for non-accessory item-type input, then the color theme will be initiated with uniform likelihood for all colors.

- Then for each item in the recommendation queue, find the sim-comp values of all colors and item/material, i.e. color-to-item and color-to-material pairs. Add the two sim-comp values and multiply by the likelihood sum of the root colors of the current color. Meaning when finding sim-comp value between the item/material and the color luna, which has the root color white and grey, the sim-comp value will be multiplied by the sum of white's and grey's likelihood.

4.2.5 Integrating user preferences

To integrate user preferences into prediction, the following variations in procedure are made:

- Passive preferences:
  - **Context preference**:
    After identifying the outfit context nature in item prediction, a bias of value 0.35 will be added to the corresponding context likelihood. A new final context scores will be calculated by normalizing original context score with the added bias. Consider an original outfit context score of [0.2, 0.4, 0.4] is obtained and the user has a preference on casual outfit, a bias of 0.35 is added to the casual likelihood changing the context score to [0.2, 0.4, 0.65]. The final context score is therefore updated to be [0.16, 0.32, 0.52].
  - **Item, material preferences**:
    After finding the sim-comp value between the preferred item/material and any other ontological feature, the original sim-comp value is updated by adding in the modifier value. The formula to obtain the value is presented in section 4.2.3.
  - **Color preferences**: As the modifier value is attached to root colors, the modifier is applied to all colors that has the preferred root color as one of their potential many parent colors. Therefore, if blue is the preferred color, then colors such as cobalt, navy, teal will be marked as preferred as well. Other than that, same method in applying item preference is used to apply color preference.
- Active preferences:
  - **Context, item, material and color preferences**:
    Same procedure as with the passive preference's, except the modifier value

is multiplied by the percentage of user rating. For example, if a user rate 30% for a preference of value 0.25 on leather then the value of bias added to leather will be $0.25 * 0.3 = 0.075$. The final sim-comp value on the items, materials and color can be negative, but context likelihood would be limited between 0 and 1.

– **Sim-comp value rating**:
After finding the sim-comp value between the two targeted ontological features, the sim-comp value is modified by the percentage of user rating. Therefore if a user marked 150% that heels and dress fit together, in future recommendation the obtained sim-comp value between heels and dress will be multiply by 150%. This will not affect the sim-comp values between heels/dress and any other ontological features whatsoever.

Example prediction

Now we present an in-depth example to demonstrate how this algorithm works in practice. Some data has modified values for demonstration purpose.

Consider a user who has two passive preferences on formal context and the item heels (with the modifier value of 0.2) respectively, and also two active preferences, one being a 200% rating on the sim-comp value between skirt and heels and the other being a 0% rating on the root color blue.

Now the user inputs a white skirt and a necklace and wants a completed outfit. We first go through the input items (skirt and necklace) and their materials (none in this case), obtaining the context scores [1, 1, 0] and [1, 1, 1] respectively, which are summed and normalized to [0.4, 0.4, 0.2]. Then it detects the user has a passive preference on formal context, so it adds 0.35 to the formal likelihood, which updates the score from [0.4 + 0.35, 0.4, 0.2] to a normalized [0.55, 0.30, 0.15].

After identifying the outfit context with user preferences, we now proceeds to generate items for recommendations. It iterates through the input items skirt and necklace, for each of them it obtains the corresponding list of items with their sim-comp value, i.e. *(skirt, shirt, value), (skirt, vest, value), ..., (necklace, hightop, value), ..., (necklace, knitwear, value)* and apply the influence modifier accordingly.

Next, after obtaining the two lists from the input items, it checks the user preferences and takes note that the user has a passive preference for the item heels and an active preference for the sim-comp value between skirt and heels. Therefore, it finds the sim-comp value between skirt and heels and multiplies it by modifiers of 200%. Also, the modifier value for heels is added to the sim-comp values between skirt/necklaces and heels. To further illustrate, the sim-comp values between

skirt/necklace and heels are computed as follows:

- We first pivots skirt as anchor item and obtain the corresponding list of sim-comp values between skirt and all items in the generic topic model, i.e. *(skirt-top: 0.3), (skirt-shirt: 0.35), ..., (skirt-heels: 0.2), ..., (skirt-knitwear: 0.25).*
- These values are all multiplied by the corresponding influence modifier, the sim-comp values become *(skirt-top: 0.6), (skirt-shirt: 0.7), ..., (skirt-heels: 0.2), ..., (skirt-knitwear: 0.25).* Same step is repeated with necklace as the pivot item.
- Then it applies active preferences by doubling the sim-comp value between skirt and heels, updating it to *(skirt-heels: 0.4).* Following this the passive preferences are applied, extra bias of value 0.2 is added to the sim-comp values between skirt/necklaces and heels, their sim-comp values are *(skirt-heels: 0.6), (necklace-heels: 0.5)* respectively.

We now compute the final sim-comp values (FSCV) for all items, using the normalized item context and the outfit context. We will explain the computations using heels as example. To compute $FSCV_{heels}$, We first calculate $FSCV_{skirt,\ heels}$ which is $0.6 * [0.33, 0.33, 0.33]^\top [0.55, 0.30, 0.15] = 0.198$ and for $FSCV_{necklace,\ heels}$ it is $0.5 * [1, 1, 1]^\top [0.55, 0.30, 0.15] = 0.165$. Since $FSCV_{heels} = \sum_{inputs} FSCV_{inputs,\ heels}$ we obtain the FSCV for heels as 0.363.

We notice the inputs have item types bottom wear and accessory, so recommendation's potential item types are top, footwear, outerwear and accessory. Items are put in recommendation queue from highest to lowest FSCV, with a constraint that the most popular recommendation item type can only have 3 more recommendations than the least popular recommendation item type. For example, if 3 top items, 2 footwears, 0 outerwear and 0 accessory are currently in recommendation queue, then we cannot add another top item into recommendation queue even if it has the highest FSCV in the remaining item, until there is at least 1 recommendation in both outerwear and accessory type.

As we have completed recommendations on item, we then proceed to choose material for each recommending item. We calculate the FSCV as of before for all materials with respect to the current item and select the material with the highest FSCV.

Lastly, we compute recommendations on color. First we attempt to identify the desired color scheme for this outfit from user input. The only color input is white, so we compute FSCV for all colors with respect to white, also applying the user preference of 0% rating on all colors having blue as one of its root color. We keep only the top 10 colors with highest FSCV.

We then aggregate the occurrence count for root colors within these 10 colors which yields [8, 3, 4, 1, 1, 1, 1, 0, 0, 1] (for the 10 root colors [white, black, grey, pink , red, yellow, green, blue, purple, brown]). This is normalized to [0.4, 0.15, 0.2, 0.05, 0.05, 0.05, 0.05, 0, 0, 0.05]. This root colors likelihood is used during color's FSCV computation, in similar fashion to outfit context. For instance, the color ivory has root colors white and grey, so its sim-comp value is multiplied by $0.4+0.2 = 0.6$.

Figure 4.9 shows a screen shot of the recommendations made with input silk skirt and gold necklace. We can see that for each recommendation it comes with corresponding explanations and users can give ratings for every one of them.

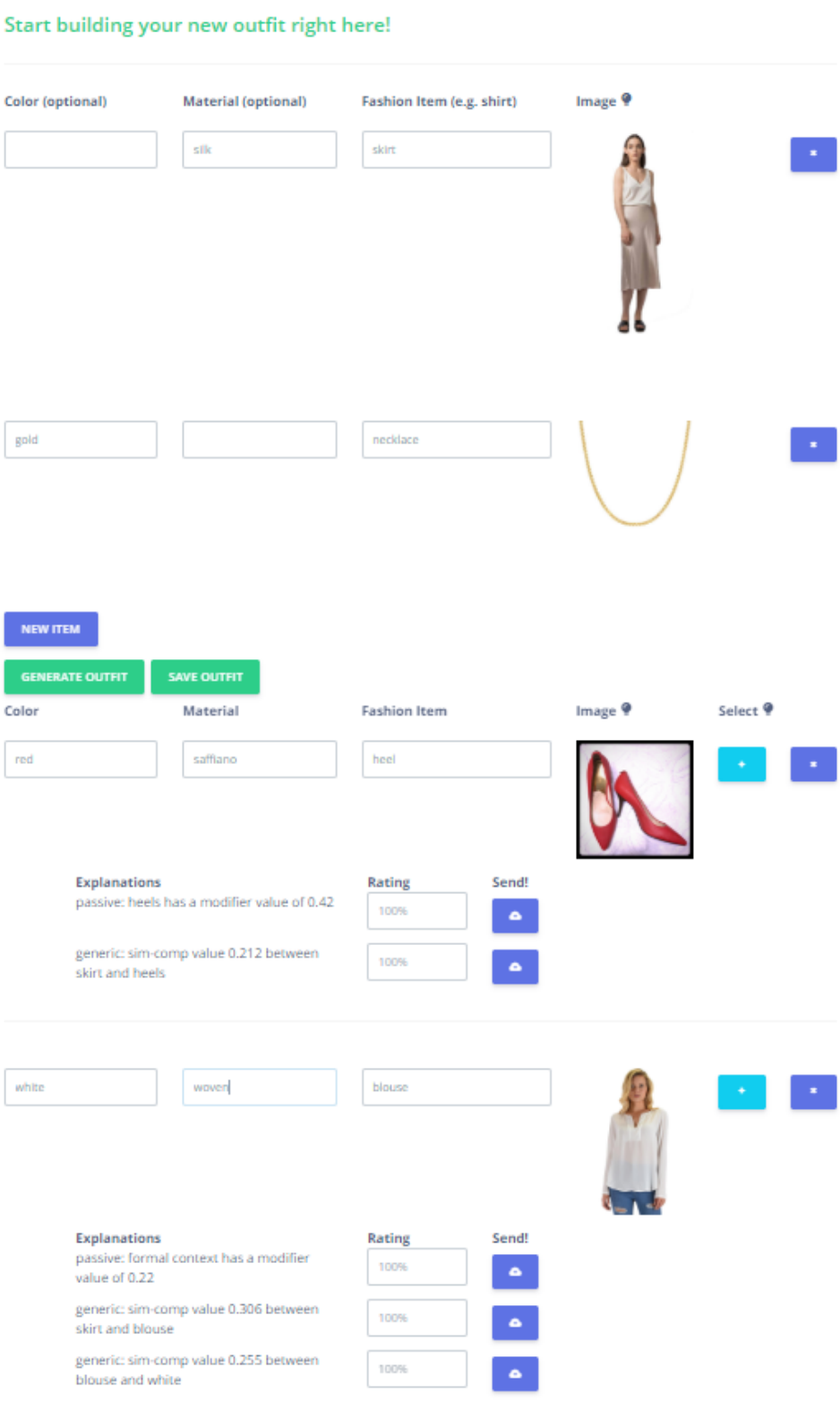

Figure 4.9: The returned explainable recommendations with the input silk skirt and gold necklace.

4.2.6 Alternative design choices

Prior to using tensorflow to train topic models with skip-gram models, we explored an alternative approach using the library gensim [20], which specializes in topic modelling. An interesting functionality it provides is `wv.most_similar()`, which is capable to give top-N similar target words using positive and negative input words. Given ["male", "king"] as positive words and ["woman"] as negative words, `wv.most_similar()` can output "queen". This ability in understanding the analogical difference deems to be useful, as we can explain to users we recommend item X due to their input Y using items A to B in their saved outfit.

However, gensim focuses in providing a high level API and its ideal training input in word2vec model is raw text. After some basic implementation, we observed that it was not intuitive to integrate compatibility into gensim's metrics. This motivated us to look for alternative library leading to the current implementation decision.

Throughout iterations, we have also experimented with different prediction algorithms, including using the 3 spaces topic model, i.e. item-color-material, to cross predict ontological features. Instead of having item as the pivot where material and colors are derived from recommending item. However, during evaluation this 3 spaces algorithm yielded a poorer result than the current algorithm. We suspected having more constraints overshadowed the user preferences. For instance, using both leather and jacket to determine the color returns black and brown as the top choices, even when user has a 273% rating on the color red. This only changes when user ratings went beyond 273% and we believed the recommender should reflect the user preference without requiring these kind of relatively strong bias. Therefore we kept our current prediction algorithm.

Chapter 5

Evaluation

We will evaluate our model's predictive performance in absolute terms and in qualitative perspectives over the explanation and feedback features. We will also compare our model against some state-of-the-art models in past research. Lastly, we look at the strengths and limitations of our algorithms and implementation.

5.1 Quantitative and qualitative analysis

As we are aiming to obtain a proof of concept to demonstrate the impact of explainable recommendations, there is no target accuracy for our model to attain to be successful in this project. These benchmarks serve mainly to ensure our model is operating in a reasonable manner. We will aim to answer the more interesting questions with qualitative analysis: (1) is the model giving sensible explanations to recommendations? (2) is the model learning user preference correctly and effectively? (3) can the model make use of the user feedback to output better recommendations?

5.1.1 Fill-in-the-blank (FITB) tests

From the training dataset [4] we used for topic modelling, it came with a fill-in-the-blank test set with over 3000 test cases. Each test contains a question of a fashion items sequence, which is randomly obtained from a random fashion set in the original training dataset. One item in this sequence is extracted and put together with other random items and the model is expected to pick the original item in the outfit as recommendation. Note that the FITB test is conducted without user preferences in our model.

Our model has achieved a predictive accuracy of 28.4%, which is similar to random guessing the answer out of the four choices, $\frac{1}{4} = 25\%$. This could be seen as our model is performing inadequately, however it is worth noticing that the inputs in FITB questions differ from the model's expected input and would impact the predictive performance. In the FITB inputs, raw description of items are used, such

as *"unisex hollow mesh top hat casual braid fedora beach sun panama hat"* while our model is expecting a single item ontology keyword with optionally a color and a material ontology keyword.

To feed suitable input to our model, we first filter the questions and potential answers in leaving only the recognized ontology keywords. Then we populate the item, color and material input slot with the recognized ontology sequence in reverse. In addition, if an ontology keyword occurs multiple times, then we would select it over other words with the same ontology type. Reusing the above example, hat is filled into the item slot rather than top, i.e. *"unisex hollow mesh top hat casual braid fedora beach sun panama hat"*. Nonetheless, since we cannot model the semantics of non-ontology keywords and also some FITB test case refers to fashion set that is not outfit idea (as discussed in section 3.1), we are expecting our model not to have excellent performance.

Figure 5.1: Examples of the fill-in-the-blank task, green bounding boxes indicate the correct answers [4].

5.1.2 FITB - ontology variants

We define a new metric revolving around ontology keywords in an attempt to better understand the predictive performance of our model. In this metric we compute the ratio of x correct ontology keywords occurring in the top y recommendations. In which correct ontology keyword is defined as keywords appearing in the original item. We also filter out test cases with less than 3 ontology keywords in total (combining item, color and material). Under this evaluation, we observe that our accuracy performance as:

- **Finding 100% of correct ontology keywords**:
  31.2% with top 3 recommendations, 33.5% with top 5 recommendations, 36.3% with top 10 recommendations.

- **Finding 50%+ of correct ontology keywords**:
  33.4% with top 3 recommendations, 35.0% with top 5 recommendations, 38.6% with top 10 recommendations.

- **Finding 33%+ of correct ontology keywords**:
  37.7% with top 3 recommendations, 40.5% with top 5 recommendations, 45.8% with top 10 recommendations.

We select values 100%, 50%+ and 33%+ to represent cases of our model getting all 3 keywords, 2 out of 3 and 1 out of 2 keywords, and 1 out of 3 keywords respectively.

5.1.3 Evaluation on recommendation explainability

We used two approaches to evaluate our explainable recommendations. In the first approach we have created 30 sample users via surveying colleagues, friends and ourselves. Each user has some obvious preferences and more than 10 saved outfits. For example, we have user who only choose black or white on their outfits, user who only wear formal items and user who strongly favors on street style.

On top of the newly introduced metric, we add the z saved outfits into the equation. For every user with a total outfits and different values of z, we have $(a-z) * \binom{a}{z} = (a-z) * \frac{a!}{z!(a-z)!}$ numbers of FITB test, by selecting z out of a outfits to learn the user preferences and each remaining outfit serve as a correct answer for a separate question. We repeat this until all combinations of z outfits are chosen. For each questions the corresponding user profile is loaded. At this stage all users only passive preferences learned from saved outfits and we obtain the following results:

- **Finding 100% of correct ontology keywords**:
  35.1% with top 3 recommendations, 37.2% with top 5 recommendations, 39.9% with top 10 recommendations.

- **Finding 50%+ of correct ontology keywords**:
  38.6% with top 3 recommendations, 41.8% with top 5 recommendations, 47.6% with top 10 recommendations.

- **Finding 33%+ of correct ontology keywords**:
  48.1% with top 3 recommendations, 55.5% with top 5 recommendations, 67.8% with top 10 recommendations.

We notice that the accuracy with user preference shows a noticeable increase comparing to the FITB result on generic predictions. Although the sample size is much smaller, the increase in accuracy remains visible even during the iteration where we use 3 random outfits out of the total 10+ outfits to learn the user preferences. Hence, we regard there are plausible evidences that system is learning user preference correctly and effectively.

Extending from this, we add in active preferences accordingly and repeat the FITB test:

- **Finding 100% of correct ontology keywords**:
  36.2% with top 3 recommendations, 39.1% with top 5 recommendations, 44.6% with top 10 recommendations.

- **Finding 50%+ of correct ontology keywords**:
  42.4% with top 3 recommendations, 46.6% with top 5 recommendations, 57.7% with top 10 recommendations.

- **Finding 33%+ of correct ontology keywords**:
  55.8% with top 3 recommendations, 62.2% with top 5 recommendations, 73.6% with top 10 recommendations.

The results show another noticeable increase in accuracy and we believe this clearly shows that firstly the model is able to use user feedback to output better recommendations, and secondly explainable recommendations and user feedback can have a positive impact in the recommendation quality. However, we would like to make a remark that some of the users have exaggerate user preferences. Hence a larger sample size of user profiles is advised for further evaluation on the model's performance if time and resources allow.

In the second approach, we demonstrated our model to 25 users and ask for feedback regarding the explanations and prediction quality after receiving feedback. The majority of the feedback are positive, including "Good to see why a clutch is chosen over a purse", "If only ASOS has this then I can see why I am getting ridiculous results" and "Oh, it is learning faster than I expected". The only two area of critics are firstly the request to separate reasoning between similarity and compatibility and secondly the ability to manually create user preference.

The decision of separating similarity and compatibility has been considered in the implementation phase. However, for simplicity and to save more resources on exploring other explanations like context scores, we decided to combine them in current implementation. This remains as an extension on this topic and it would be interesting to see how it impacts on prediction accuracy. As for the functionality to create arbitrary rule, it is a simple extension that can be completed via an exposed API to some internal functions. We did not do this in the final phase because of the submission deadline, as we were fine-tuning the modifier values used in prediction, which we believed have a higher priority and more impact to the actual prediction

Overall, we consider our model quality to be successful as it has a passable performance on generic prediction, capable of giving sensible explanations of recommendations and able to show explainable recommendations have noticeable positive impacts on prediction quality.

5.2 Comparing to related work

Method	FITB accuracy
SetRNN	29.6%
SiameseNet	52.0%
VSE	29.2%
F-LSTM + VSE	63.7%
B-LSTM + VSE	61.2%
Bi-LSTM	66.7%
Bi-RNN + VSE	63.7%
Bi-GRU + VSE	67.1%
Bi-LSTM + VSE	68.6%

Figure 5.2: Performance of some state-of-the-art method in the FITB test [4].

From figure 5.2, we make the following observations:

- SetRNN [21] and Visual-semantic Embedding (VSE) perform similarly to random guessing, which is 25%. SetRNN makes prediction based on outfit popularity, but high popularity does not always reflect good compatibility. VSE has a poor performance due to the noisy labels and failure in modelling relationships among items within a same outfit.

- SiameseNet [12] has a better performance than VSE and SetRNN but still works worse than Long short-term memory (LSTM) based methods, since it focuses on pairwise relationships instead of the compatibility of the whole outfit. Thereby, it sometimes picks items with item type already in the outfit, because their styles are similar.

- LSTM based methods have a noticeable better performance than the rest of the methods, as they make use of item features and sequential information to reflects item compatibilities within the same outfit.

- Looking at our model's performance, we see that our general prediction performs alike SetRNN and VSE. However, our predictive performance with user preferences is comparable to some of the stat-of-the-art methods using our alternative metric.

5.3 Strengths and limitations

5.3.1 Strengths

- **Explainable recommendations**:
  We have built a model which supplies intuitive explanations (similarity/compatibility and past occurrences) and allows user feedback on these explanations to improve future recommendations.

- **Easy updates on ontology classes**:
  The model is generic enough that extensions on current ontology classes (item, color, material) and addition of new ontology class can be easily achieved. This provides good reusability and lays groundwork for future projects.

- **Promising predictive performance**:
  We achieved a classification accuracy comparable to state-of-the-art methods such as Bi-LSTM by integrating user preferences into our predictions. Our baseline prediction also falls into a reasonable range that is higher than random guessing.

5.3.2 Limitations

- **Simple ontology framework**:
  As discussed before we do not have access to existing handcrafted ontology frameworks [13] and hence we designed our own framework. We believe that using a more well-developed framework can provide a greater degree of explainability. In particular, the attributes (item, color, material, root color, context score) we have included in our framework are relatively basic. In reality, a wider range of subtle features would affect the compatibility between items, for example the style of item can greatly vary the overall presentation of the same item. Consider the item sweater, a mock turtleneck sweater has a very different style compared to a Korean style loose-fit sweater.

- **Limited work on heuristics and optimizations**:
  Currently, our algorithm rely mainly on predicting the item then on predicting

color and material accordingly. We believe there is room for improving this approach. For example, we have abandoned using the 3 spaces topic model due to its poorer performance, but we have not investigated in depth whether we can improve this by optimizing various parameters.

- **Limited vocabulary**:
  Given the timescale of the project, our model does not support unseen words that are not defined in our ontology framework. To have this functionality would help make the model more robust and closer to be deployable in real life scenario.

Chapter 6

Conclusion

6.1 Summary

In this project, we have built a recommender, **fAshIon**, as a proof of concept to provide explainable recommendations and show how the explainability can improve recommendation quality.

Our model uses a custom-defined ontology framework as one of the backbones in learning the semantics of different fashion attributes and it also make use of various topic models to obtain similarity and compatibility between ontological features. In addition, we have devised our own prediction algorithm that utilizes the ontological features. Other than the model itself, we create a GUI frontend for the model where the users can interact with the model and the explainable recommendations.

To evaluate our model, we have compared our predictions to other state-of-the-art methods in order to better understand our model's predictive performance. Also, we have used user surveys to access our functionalities of explaining recommendations and user interactions with such reasoning.

6.2 Future extensions

There are multiple directions in which this project could be taken further with more time and resources.

6.2.1 More sophisticated ontology framework

In this project, we have used our own ontology framework which is rather simple when comparing to some of the frameworks [13] built on Web Ontology Language. We believe using this kind of professionally crafted framework would increase the degrees of explainability and prediction accuracy.

There are few key features that we have considered useful but have not included in the project, due to the lack of time and resources. For instance, integrating fashion brands would introduce new semantics: items in luxury brands like Chanel and brands like Primark may not go well together even if the generic items themselves and their item types are compatible. Another features that would be interesting to include is learning the variations within the same fashion item. Consider the item shirt, a shirt with Italian collar may be classified as an elegant shirt while one with American collar may be more relaxed. Hence, the former may lean towards a formal context and the latter towards a semi-formal or a casual context.

6.2.2 Greater degree of explanation

Currently, our explanations on recommendations focus on similarity, compatibility and context scores. While these elements are sufficient to improve the system, a wider range of explanation angles can be explored. For instance, using variations within same fashion item as discussed above could be a sensible explanation.

6.2.3 Accommodate unseen words

The current model in this project does not support input with unseen words. Although we implemented support for this at iterations, we abandoned this feature at a later stage to focus on refining the ontology framework and prediction algorithm.

To upscale this project to be used in production environment, supporting unseen words would be a crucial feature, as it aids the system to adapt to new fashion trends and be applicable in wider spectrum.

6.2.4 Integrating imaging techniques

Sometimes, subtle features in fashion items like Italian collar on a shirt may not be noticed by the user as well. In scenario like this, it would be beneficial if the system can extract these kinds of features from the image corresponding to input items and add as additional input. Extending further from this, supporting pure image input is another possible future direction.

Bibliography

[1] P. Resnick and H. R. Varian, "Recommender systems," *Communications of the ACM*, vol. 40, no. 3, pp. 56–58, 1997.

[2] A. Rago, O. Cocarascu, and F. Toni, "Argumentation-based recommendations: Fantastic explanations and how to find them.," in *IJCAI*, pp. 1949–1955, 2018.

[3] H. Lee, J. Seol, and S.-g. Lee, "Style2vec: Representation learning for fashion items from style sets," *arXiv preprint arXiv:1708.04014*, 2017.

[4] X. Han, Z. Wu, Y.-G. Jiang, and L. S. Davis, "Learning fashion compatibility with bidirectional lstms," in *Proceedings of the 2017 ACM on Multimedia Conference*, pp. 1078–1086, ACM, 2017.

[5] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[6] S. Learn, "Neural network models (supervised)," 2019.

[7] Wikipedia, "Sigmoid function," 2019.

[8] H. Lee and S.-g. Lee, "Style recommendation for fashion items using heterogeneous information network.," *RecSys Posters*, vol. 2, 2015.

[9] Y. Hu, X. Yi, and L. S. Davis, "Collaborative fashion recommendation: A functional tensor factorization approach," in *Proceedings of the 23rd ACM international conference on Multimedia*, pp. 129–138, ACM, 2015.

[10] M. I. Vasileva, B. A. Plummer, K. Dusad, S. Rajpal, R. Kumar, and D. Forsyth, "Learning type-aware embeddings for fashion compatibility," *arXiv preprint arXiv:1803.09196*, 2018.

[11] R. He, C. Packer, and J. McAuley, "Learning compatibility across categories for heterogeneous item recommendation," in *Data Mining (ICDM), 2016 IEEE 16th International Conference on*, pp. 937–942, IEEE, 2016.

[12] A. Veit, B. Kovacs, S. Bell, J. McAuley, K. Bala, and S. Belongie, "Learning visual clothing style with heterogeneous dyadic co-occurrences," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4642–4650, 2015.

[13] A. Mallik, H. Ghosh, S. Chaudhury, and G. Harit, "Mowl: An ontology representation language for web-based multimedia applications," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 10, no. 1, p. 8, 2013.

[14] S. Ajmani, H. Ghosh, A. Mallik, and S. Chaudhury, "An ontology based personalized garment recommendation system," in *Proceedings of the 2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)-Volume 03*, pp. 17–20, IEEE Computer Society, 2013.

[15] X. Aimé, S. George, and J. Hornung, "Vetivoc: a modular ontology for the fashion, textile and clothing domain," *Applied Ontology*, vol. 11, no. 1, pp. 1–28, 2016.

[16] D. M. Gabbay, "Logical foundations for bipolar and tripolar argumentation networks: preliminary results," *Journal of Logic and Computation*, vol. 26, no. 1, pp. 247–292, 2016.

[17] P. Baroni, G. Comini, A. Rago, and F. Toni, "Abstract games of argumentation strategy and game-theoretical argument strength," in *International Conference on Principles and Practice of Multi-Agent Systems*, pp. 403–419, Springer, 2017.

[18] E. L. Frankie Wong, "Interview with founders of trim design," 2019.

[19] C. Tim, "Argon design system.," 2018.

[20] R. Řehůřek, "Gensim - topic modelling for humans," 2019.

[21] Y. Li, L. Cao, J. Zhu, and J. Luo, "Mining fashion outfit composition using an end-to-end deep learning approach on set data," *IEEE Transactions on Multimedia*, vol. 19, no. 8, pp. 1946–1955, 2017.