Beyond Metadata for BBC iPlayer: an autoencoder-driven approach for embeddings generation in content similarity recommendation

Simone Spaccarotella

September 2024

Contents

1 Introduction and background

I am a Software Engineer at the BBC, Team Lead for the Sounds web team, and I have been training as a Data Scientist, working in attachment with the iPlayer Recommendation team.

I built a machine learning model pipeline that produces content-to-content (C2C) similarity recommendations of video on-demand (VOD), for the ”More Like This” section on BBC iPlayer [2]. This project is relevant to me because I have been crossing paths with the world of recommendations multiple times during my career at the BBC, and it sparked an interest. I had a tangent encounter back in 2015 while working for a team that was building an initial recommender for BBC News, and an API to provide recommendations using 3rd party engines. During a Hack Day some time later, I produced and presented a talk called ”Recommendation Assumptions” [13], which was about types of recommendations and external factors affecting them, which are contextual to the consumption of the content itself. Until these days, where I was able to finally put my knowledge into practice with an actual project on real data.

The BBC is a well-known British broadcaster, and it is always evolving to remain relevant to its audience. Its mission is to inform, educate and entertain, and it operates within the boundaries set by the Royal Charter [6]. The current media landscape requires the BBC to deliver digital-first content that is relevant to the audience, and this involves investements in data and personalised services, not to mention a certain revolution in machine learning that is keeping everyone busy.

2 Outline of the issue or opportunity and the business problem to be solved

The BBC produces and stores a vast amount of data for its content, and this data is produced and surfaced by countless services and APIs. One of the priority for the BBC is to increae the usage across the business of **Passport** [7] an internal BBC system that provides a richer set of metadata annotations for multi modal content (audio, video and text). The usage in production is very low, and its BBC-wide adoption would make the access to metadata consistent, removing duplications and reducing effort and costs.

Furthermore, the similarity score of the current C2C recommender is directly proportional to the number of values in common between any pairs of items on a per-feature basis. But the commonality is calculated with an exact string equality, ignoring any relationship between different categorical values expressing a similar concept (e.g. ”comedy”, ”stand-up comedy”).

Moreover, the number and types of tags are not enough to sufficiently describe the content, while the data distribution is severely skewed towards the most popular annotations and no pre-processing is applied.

Lastly, each similarity score is multiplied by a hardcoded weight that modulates the importance of a feature, but it doesn't solve the polarising effect of a skewed distribution. Unfortunately, because these are hyperparameters and not learned weights, the model can't improve its performances by minimising them against a cost function.

To address these issues, the aim of this project was:

- **To improve the quality of the C2C similarity recommendations**. The hypothesis was that by using in input a richer set of metadata that better describes the content, and by reducing the high-dimensional data to a lower-dimensional latent manifold, the model would be able to generate embeddings that could improve the quality of the recommendations, by mapping the item similarity problem to a geometric distance calculation between vectors in a multi-dimensional Euclidean space.

- **To build a general solution that can be applied once, reducing the costs**. I used Passport tags as input data to build a general solution that could be used for any BBC product and applied to any type of content, because they all share the same set of common tags.

- **To build a foundational item-embeddings generator**. Content-based recommenders use item metadata. This project provided an immediate solution for unpersonalised C2C recommendations that solely relies on them and it could provide a foundational approach for personalised recommender that combine content metadata with other signals like user interactions and contextual data.

3 Methods and justification

3.1 Data pre-processing

I used **one-hot encoding** to transform the categorical features (i.e. the metadata annotations) into a numerical vector. It's a simple yet effective encoding method and it's perfect for the transformation of nominal categoricals, because it doesn't introduce any ranking and/or arithmetic relationship among the encoded values. The downside of this approach is that it generates high-dimensional sparse arrays, introducing the so called *curse of dimensionality* problem. Nonetheless, this was an accepted drawback that was managed in the modelling phase.

3.2 Modeling and regularisation

I trained an undercomplete **autoencoder** to improve the the quality of the recommendations, and to reduce the cost of calculating them. Autoencoders are self-supervised models, capable of capturing non-linearity from the data. This type of encoder-decoder limits the number of nodes in the hidden layers, and creates a "bottleneck" of information flow through the neural network. This bottleneck structure is a form of *regularisation* that forces the model to learn latent attributes from the input, while reconstructing it with minimal loss. Ultimately, it prevents the model from *overfitting* the training dataset, by indexing it like a caching layer.

The *encoder* part of the trained encoder-decoder is used to compress the one-hot encoded high-dimensional sparse vectors into so-called *embeddings*, a lower-dimensional dense representation of the initial content metadata. This technique solved the curse of dimensionality and the data sparsity problems, and improved the calculation complexity and the quality of the recommendations at inference time.

To improve and assess the ability of the model to *generalise* on unseen data, I randomly shuffled the dataset and split it into 3 chunks: training, validation and test. I run **hyperparameters tuning** to find the best set of model parameters that minimised the objective function and I used the validation set to regularise the model with **early stopping**. This technique monitored the reconstruction loss on an out-of-sample dataset, allowing the model to stop training within a certain "patience" threshold, after reaching a local minima on the validation error.

I used **dropout** to further regularise the model and make it robust to small changes in the input, and **data augmentation**, by including in the training data the episodes of a programme that share the same tags with their parent container.

Weight decay and **batch normalisation** were tested during hyperparameter tuning and discarded for poor performances.

3.3 Inference

Item similarity was calculated with the **cosine** of the angle θ between each pair of embeddings. This metric is insensitive to the magnitude of the vectors, and because popular values tend to have a larger magnitude, it mitigates the impact of popularity in the similarity calculation. In addition, multi-hot encoded pairs can only have a finite number of angles. They represent unit vectors bound in the "positive quadrant" of a multi-dimensional Euclidean space. This forces the cosine similarity to also assume a finite number of discrete values between 0 and 1, leading to an information loss. Ideally, we would expect the similarity score to assume a continuous value bound between -1 and 1, and this is only possible if the angle θ of any vector pair can assume a continuous value between 0 and 360 (i.e. 0π and 2π), hence the use of embeddings.

3.4 Tools and frameworks

The entire project was written in **Python**. It is the *de facto* programming language for data science and machine learning tasks. Python has an established, diverse and well-documented ecosystem of external libraries and frameworks that facilitated the job, and it is also the language of choice at the BBC.

I used **Pandas** only for tabular data manipulation, to generate and store the one-hot encoded vectors. Unfortunately it wasn't possible to use it for exploratory data analysis (EDA), because the iPlayer catalogue used in development had roughly one-year worth of data and it didn't fit in memory, causing Pandas to crash. For this reason I used **Dask**. It is a library capable of running out-of-memory and parallel execution for faster processing on single-node machines and distributed computing on multi-nodes machines, while using the familiar Pandas API.

I used **TensorFlow** and **Keras** for modelling, to build and train the encoder-decoder neural network architecture, and **Keras Tuner** for hyperparameters tuning. I also used **Scikit-learn** but not for modelling. It provided utility functions for the dataset splitting and the cosine similarity calculation, and I was already familiar with its API.

For data visualisation I used a combination of **Matplotlib** and **Seaborn**, while I used **rdflib** to fetch and parse the RDF documents from the BBC

Ontology. These documents represented the metadata values and contained the actual entity label. This label was needed to hydrate the set of metadata for each item, to visualise the recommendations for testing purposes. Worth also mentioning the use of **pytest** for unit testing and **black** for PEP 8 code compliance and formatting. Finally, I used **Jupyter Lab** to edit the project, **git** for code versioning, **GitHub** as a remote code repository and for collaboration, and **AWS Sagemaker** to run the pipeline on more capable virtual machines, especially during hyperparameter tuning.

4 Scope of the project and Key Performance Indicators

The scope for this project was to build an end-to-end machine learning solution able to produce unpersonalised content-to-content similarity recommendations, using Passport metadata tags as input.

The minimum viable outcome for this solution was to produce a set of recommendations comparable with the ones currently in production and as a desired outcome to increase user engagement. The ability to do so using Passport tags (resulting in a general solution applicable to multimodal BBC content) was also accounted as key performance indicators (KPIs), a boolean success/failure one.

I didn't set a KPI to track the reduction of costs, because I didn't have access to the actual information for each AWS account used for C2C across the BBC. And I didn't need to, because if the solution uses Passport tags, it can be adopted to all products. If we assume that there are N products that use different solutions and the global cost is C, by simply measuring the Passport tags KPI - assuming the solution is valid and viable and it gets adopted - the cost would be reduced by a factor of $\frac{C}{N}$.

Comparability was a qualitative and subjective KPI that served as a compass, indicating whether the project was progressing towards the right direction. It was evaluated by several technical and non-technical stakeholders, with a diverse domain knowledge and background. I built a rudimentary visualisation tool that rendered the title, image, description and metadata of the seed programme and the top-K C2C recommendations. People involved could give their subjective feedback on what was their perceived level of similarity of the output, testing edge cases and common use cases with expected outcome, sensitive recommendations (i.e. content recommendations for children accounts), and discussing whether there were anomalies and/or surprising results.

To measure user engagement though, the system needed to be A/B tested in production with real data. Unfortunately, there were too many moving parts outside my control that needed to happen first, for me to be able to measure this KPI. It would have delayed the project, increasing the chance of failure. To mitigate this risk, I defined a hypothesis supported by a KPI that could be measured offline within my area of control, moving the testing of this hypothesis outside the scope of the project.

The hypothesis stated that long-term user engagement is impacted by the degree of diversity of recommended content. It went on sayng that a diverse set of recommendations can generate new and unexpected results, which can increase surprise and serendipity, pushing the user away from boredom. This was an untested hypothesis, but grounded in active research on the topic, such as [10] and [8], that supported the initial statement. Although the hypothesis needed validation, it was ok for it to be pushed out of scope, because it wasn't too far fetched afterall. For this reason, I defined a proxy metric for diversity that was used to measure an approximation of it offline, pending a future A/B test validation.

5 Data selection, collection and pre-processing

News and Sports articles, iPlayer videos on-demand, Sounds podcasts etc. are annotated with the so-called Passport tags. These tags describe the content produced by the BBC and can be used for retrieval (search) and filtering (recommendations). They can be applied either manually by an editorial team with domain knowledge, or semi-automatically by machine learning algorithms with human supervision.

Passport tags are distributed across the BBC via the universal content exposure and delivery (UCED) system, a self-service metadata delivery platform that exposes data as a document stream for products to integrate with. This platform provides different types of "consumers" such as REST API, AWS S3 bucket, etc. Passport documents are JSON objects that contain a property called `taggings`, an array of objects representing the metadata annotations. These objects in turn contain two properties: `predicate` and `value`. They represent the name and the value of a tag, and are expressed as URL-formatted strings, with the exception of dates. The predicate is a class of the BBC Ontology [1] while the value can be a date or an entity defined as an RDF [16, 14] document, accessible in Turtle format [15] via the BBC Things API [3, 4, 5]. These entities are linked to each other and/or to external resources, and are described by attributes and relationships, giving the data a graph structure.

For the development of the project, I decided not to integrate with UCED but to use batches of Passport files, manually collected and stored on a local folder. This was a tradeoff that allowed me to develop the project with real data while keeping the costs down, given that the resources needed to be set on two AWS accounts. Moreover, I didn't want to pass the burden of maintenance to the team that owned the accounts, without having tested the feasibility of the solution first.

Content metadata is not classified as personally identifiable information (PII), according to the UK GDPR [9]. Nonetheless, this data is regulated by internal BBC data governance policies and as such, it is encrypted at rest and in transit by default. For this reason, no further precautions were required during storage and processing.

I chose to use Passport because this data provides a set of tags shared across all content produced by the BBC, making this a general solution that reduces duplications and ultimately costs. Passport provides a flexible and rich set of tags to describe any content. Annotations can describe canonical information such as *genre* and *format*, but also things like the "contributor" that features in, the "narrative theme", the "editorial tone", what the content is "about" or what relevant *entities* are mentioned, etc.

During pre-processing, a list of JSON files were loaded into the pipeline and the tags extrapolated into a dictionary data structure. The key of the dictionary was the programme ID (called *PID*) and the value was another dictionary describing the annotations. A programme can be tagged with the same predicate multiple times, as long as it has different values, while the same value (e.g. "Music") can be used by multiple predicates (e.g. "about" or "genre").

The dictionary was transformed in a Pandas *Dataframe*, where the rows represented the programmes and the columns the tags. I used a MultiIndex [12] for the columns because I needed to keep the duplicate values (2nd-level index) across the predicates (1st-level index). I then populated the cells of the *Dataframe* with the value `"1"` if the programme was annotated with the corresponding tag, or `"0"` otherwise. This process generated one-hot encoded arrays. I initially started with a vectorisation approach known to be performant, using the `"get_dummies"` Pandas function. Surprisingly, it was slower and less scalable of the solution that I adopted in the end.

A source of bias in the dataset was the usage of the `"mentions"` tag. This tag is automatically generated by an algorithm that extracts terms deemed important, appearing in the text of an article or the transcript of an audio/video content. Something "mentioned" doesn't necessarily describe what the content is about, because of the intrinsic ambiguities of natural languages. Figure of speech devices such as metaphors, analogies, allegories, etc., alter the meaning of a sentence for stylistic effect and can affect the representation of what the content was about. For example, if the idiom "being over the moon" is mentioned referring to something unrelated with space, and the term "moon" is extracted as a descriptor, it certainly increases the chances of misrepresentation. To mitigate this source of bias I dropped the tag in favour of `"about"`, a tag that describes what the content is really about. This tag is manually annotated by editorial teams who are trained to only annotate with relevant and topical entities.

A source of error was the encoding of unseen tags. I decided to drop the new tags and encode the data with the existing ones only. Also, a subset of programmes didn't have any annotations. Creating an entry for them would have generated a minority of one-hot encoded arrays with all zeros, representing a characteristic class of uninformative observations. I decided to drop these programmes and include the only the ones with at least one annotation.

6 Survey of potential alternatives

Given a numerical vector representing the Passport tags of a programme in input, the model needed to return a list of the top-k programmes, sorted in a descending similarity order. The geometric interpretation of the similarity between two items, is the distance between the two vectors representing them.

I considered using a Clustering technique to group "similar" items together, so that given an item in input, the model would return the ones belonging to the same cluster. I didn't know how many cluster there were in the data, and I didn't need to know. I could have used a density-based technique to autodiscover them, but even in that case, some of the clusters could have had less than K items in a top-K similarity scenario. I could have returned the items belonging to the nearest cluster if needed, but at that point, what's was the meaning of clusters in that context anyway? In content similarity, any item has a degree of similarity with all the others and clustering was just a coarse-grained discretisation of that concept. I needed a more granular approach where every item could be compared with anyone else. In geometric terms it meant that I had to calculate the pairwise distance between any two pairs of vectors, so I didn't need to use clustering in the first place.

Both clustering and pairwise distance use the concept of gemoterical distance between any two pairs, and this calculation is computationally expensive, and doesn't scale on big datasets, especially if it involves high-dimensional sparse arrays. Not only that, the one-hot encoding processing doesn't use any spatial proximity information to transform the categorical features into ones and zeros. It's a plain transformation that pivots the unique values of a given variable as features, and sets 1 to the corresponding annotations. If we take this raw vector as is, and interpret it as a multi-dimensional coordinate system and then calculate similarity, we are out of luck.

From a computational point of view, I needed to handle the curse of dimensionality, by reducing the dimension of the numerical vectors. Not only that, the new vectors needed to be in a denser and lower dimensional manifold embedded into the original high-dimensional ambient space, for the "local proximity" to have a meaning in terms of similarity.

Dimensionality reduction techniques such as Principal Component Analysis (PCA) (which I'm more familiar with), or Independent Component Analysis (ICA), Linear Discriminant Analysis (LDA), etc. were next on the list, but there was a problem with them too. Their job is to find a linear projection of the data and this is a strong assumption that - more ofthen than not - misses important non-linear structures in the data. So, I discarded the idea of using PCA, but not the idea of using dimensionality reduction. I just needed a non-linear aproach, so I turned my attantion to manifold learning.

Before discussing that, I'd like to point out that I also considered changing the pre-processing step to generate a different set of vectors that didn't suffer from the curse of dimensionality.

Hashing is a well-known non-invertible technique that can be used in machine learning for feature reduction. The reason why I didn't adopt this technique was because hashing has more hyperparameters than one-hot encoding, which increase its complexity. Settings like the size of the hash could have impacted the descriptive power of the vector even before starting the embedding learning phase. But most importantly, hashing functions introduce the collision problem, were two distinct entities can be mapped to the same index in the target domain. Because this is an issue that can't be removed, only mitigated, I tested the latest and strongest algorithm to reduce the likelihood of collisions, but the trade-off was too computetionally expensive during training.

I talked about why I adopted one-hot encoding, let's now talk about my chosen approach for manifold learning, pros and cons and its justification.

7 Implementation and performance metrics

I used an undercomplete autoencoder to compress the high-dimensional one-hot encoded vectors to a lower dimensional embedding representation. This technique captured non-linearity by learning the underlying latent structure of the data, which was key to represent the original Passport tags in a geometrical space, exploiting local proximity as a measure of similarity.

Because the autoencoder has a symmetrical architecture between the *encoder* and the *decoder*, the input and output layers had the same dimensions: 8982 nodes. This number corresponded to the size of the ohe-hot encoded array.

Some of the hyperparameters were decided based on the nature of the problem. The input data was a tensor of zeros and ones, and the main objective of the network was to reconstruct the output with minimal loss. To achieve this, I used the *Sigmoid* as activation function for the output layer, and *binary cross-entropy* as loss function, to allow the network to push the reconstructed output values as close as possible to 0 and 1. I didn't use the *SoftMax* activation function for the output layer, because each single node needed to be able to assume a value between 0 and 1. I didn't use any residual-based loss function such as MSE or MAE, because the sum of the square (or absolute value) of the residuals for 0/1 values doesn't have a very steep slope to allow the gradient to move fast enough. Which is why the negative log loss in cross entropy is the best choice, because it massively penalises huge differences between $\hat{y}$ and y with a logarithmic progression.

Some other hyperparameters were set after few rounds of training. For example, I used the *Rectified Linear Unit (ReLU)* as activation function for the hidden layers. I also tested other variants like *Leaky ReLU (LReLU)* and *Parametric ReLU (PReLU)* with some isolated hyperparameters tuning tests, but without any relevant improvements in performance. These hyperparameters were:

- the optimizer: Adam
- number of epochs: 100
- batch size: 300
- dropout rate: 0.2
- early stopping patience: 10
- early stopping monitoring: binary cross entropy on validation set

- data split: 80% training, 10% validation, 10% test

The remaining hyperparameters were selected through a final round of optimisations. I decided to use a "Bandit-based" approach called *Hyperband* [11], which improves upon *Random Search* by running fewer epochs on the randomly sampled set of parameters, and move on to the next stage by only testing the best performing ones, returning a ranked list of the best hyperparameter sets. The hyperparameters considered by Hyperband were:

- the number of hidden layers
- the embedding size
- the learning rate
- whether to use dropout
- whether to use batch normalisation

The number of hidden layers was a positive integer $N > 0$ where $N - 1$ layers were used for the encoder and the same amount For the decoder - because the autoencoder has a symmetrical structure - and one layer was instead used for the bottleneck in the middle. If the hyperparameters was set to 1, the network would only have the bottleneck.

The number of nodes per layer was also a positive integer N. Unfortunately, the number of combinations was too big to be meaningfully optimised. To reduce the complexity, I decided to couple the number of nodes per layer as a progression of integer divisions by 2. Each layer in the encoder had half the amount of nodes compared to the previous one and double the amount of the successive one (and viceversa in the decoder). For example, because I had 8982 nodes in input, the progression of layer dimensions was: 4491, 2245, 1122, 561, 280, 140 and so on. As a consequence, the embedding size was also bound to assume one of these values, greatly reducing the number of choices, hence the runtime of the hyperparameter optimisation.

The strengths of this approach were the efficient use of space and the fast inference time. Training took longer during hyperparameter tuning. This drawback was compensated by the fact that I adopted an offline-training approach, together with a "stale-while-revalidate" policy to return the recommendations at inference time, in addition to the similarity scores being cached. The space scaled linearly with respect to the size of the input to store the embeddings, and quadratically to store the similarity scores. The drawback was the model's interpretability, or in this case, the lack thereof. The autoencoder is a *black box* by definition and the use of embeddings to calculate the similarity just made the problem worse. It is difficult if not impossible to interpret which of the tags influenced the ranking in the top-K list by untangling the weights and biases of the neural network. But it's also difficult to explain what is going on, even using model-agnostic techniques like SHAP. The autoencoder's output is the reconstruction of the input, but the actual ououtput of the entire model are the cosines of each pairs of vector embeddings. So, no explainability either, or at least, localised to the neural network.

To mitigate this problem, I considered "the model" being the entirety of the pipeline. Given a programme in input which is annotated with a set of tags, I get in output a list of the top-K most similar programmes, which are also annotated with a set of tags. This meant that I could analyse the tags and use their frequency and composition as proxy metrics for feature importance. I could calculate the intersection between the input tags and the recommended onse, giving a measure of coverage that can approximate the strenght of the annotations of the input programme, conditioning the recommendations. I could also calculate the difference between the recommended tags minus the one in input, to calculate the distribution of the new tags introduced by the recommended programmes, to approximate diversity.

For this reason, and because the Sigmoid function can lead to loss saturation (i.e. the function plateau at the +/- infinity) that could prevent gradient-based learning algorithms from making progress, I'm using Binary Cross Entropy as a loss function. Binary Cross Entropy not only helps pushing the values closer to 0 or to 1 depending on the input value (to minimise the reconstruction loss), but also counteracts the effects of the exponential in the Sigmoid with the use of the log.

8 Results

9 Discussion & conclusions/recommendations

10 Summary of findings

11 Implications

12 Caveats & limitations

13 Appendices

References

[1] BBC. BBC Ontologies. https://www.bbc.co.uk/ontologies/.

[2] BBC. Bluey - "more like this" tab. https://www.bbc.co.uk/iplayer/episodes/m000vbrk/bluey?seriesId=more-like-this.

[3] BBC. Things. https://www.bbc.co.uk/things.

[4] BBC. Things - About. https://www.bbc.co.uk/things/about.

[5] BBC. Things - API. https://www.bbc.co.uk/things/api.

[6] BBC. The Royal Charter. https://www.bbc.com/aboutthebbc/governance/charter, 2017. Valid until 31 December 2027.

[7] BBC. How metadata will drive content discovery for the bbc online. https://www.bbc.co.uk/webarchive/https%3A%2F%2Fwww.bbc.co.uk%2Fblogs%2Finternet%2Fentries%2Feacbb071-d471-4d85-ba9d-938c0c800d0b, 2020. This page was archived on 1st August 2023 and is no longer updated.

[8] Tomislav Duricic, Dominik Kowald, Emanuel Lacic, and Elisabeth Lex. Beyond-accuracy: A review on diversity, serendipity and fairness in recommender systems based on graph neural networks, 2023.

[9] UK Government. UK GDPR. https://www.legislation.gov.uk/eur/2016/679/contents, 2016. Regulation (EU) 2016/679 of the European Parliament and of the Council.

[10] Marius Kaminskas and Derek G. Bridge. Diversity, serendipity, novelty, and coverage. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 7:1 – 42, 2016.

[11] Lisha Li, Kevin G. Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *CoRR*, abs/1603.06560, 2016.

[12] pandas via NumFOCUS, Inc. MultiIndex / advanced indexing. `https://pandas.pydata.org/docs/user_guide/advanced.html`, 2024.

[13] Simone Spaccarotella. Recommendation assumptions. `https://www.slideshare.net/slideshow/recommendations-assumptions/236291920`, 2015. Presented at Prototyping Day @ Mozilla London on 3 September 2015, at Engineering Summit @ BBC on 7 March 2018, uploaded on SlideShare on 27 June 2020.

[14] W3C. RDF 1.1 Concepts and Abstract Syntax. `https://www.w3.org/TR/rdf11-concepts`, 2014. W3C Recommendation 25 February 2014.

[15] W3C. RDF 1.1 Turtle - Terse RDF Triple Language. `https://www.w3.org/TR/turtle/`, 2014. W3C Recommendation 25 February 2014.

[16] W3C. Resource Description Framework (RDF). `https://www.w3.org/RDF`, 2014. Publication date: 2014-02-25 (with a previous version published at: 2004-02-10).