Graph-powered Machine Learning for Movie Predictions

Nathan Smith
Kansas City Graph Databases Meetup
December 10, 2022

Explore the dataset with Bloom

Data from https://www.themoviedb.org/

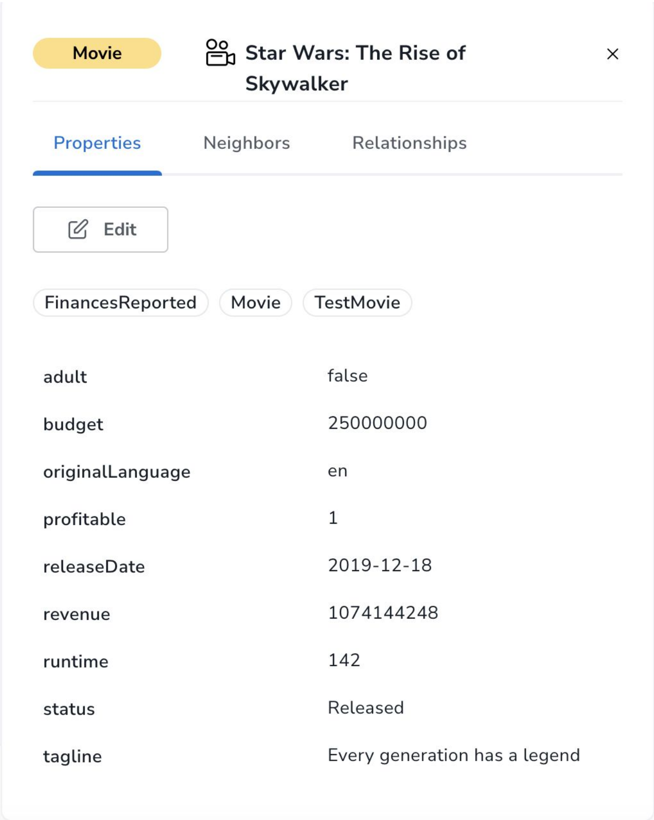
Predicting profitability

What do we know about a movie?

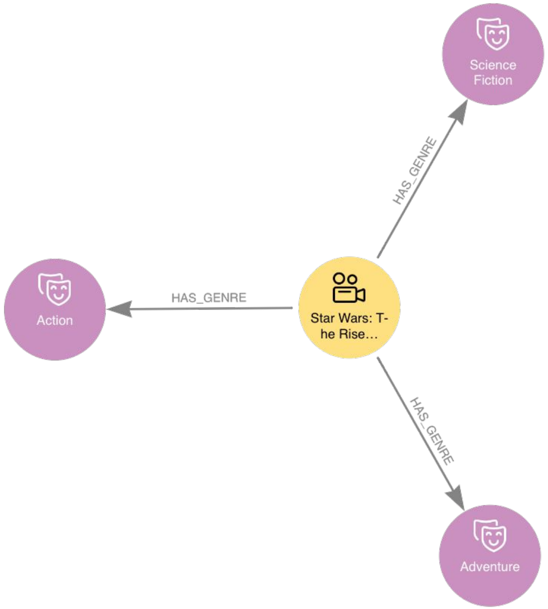
- Budget
- Genre(s)
- Cast/crew
- Profitability

Properties on movie nodes are easy to use

They are already at the right grain to add to a data frame

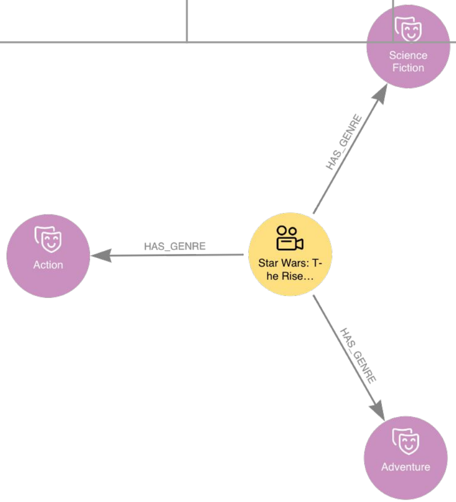


Genres need a little manipulation

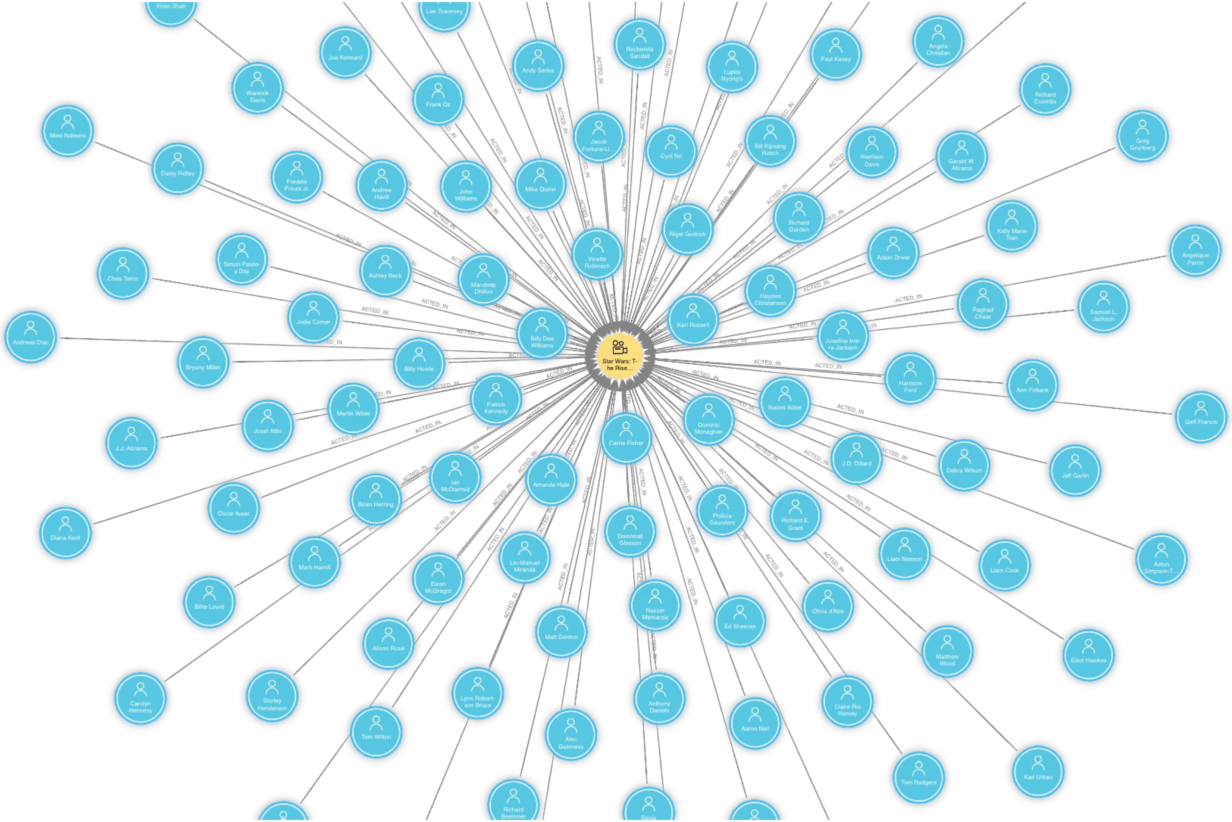


Represent them with one-hot encoding

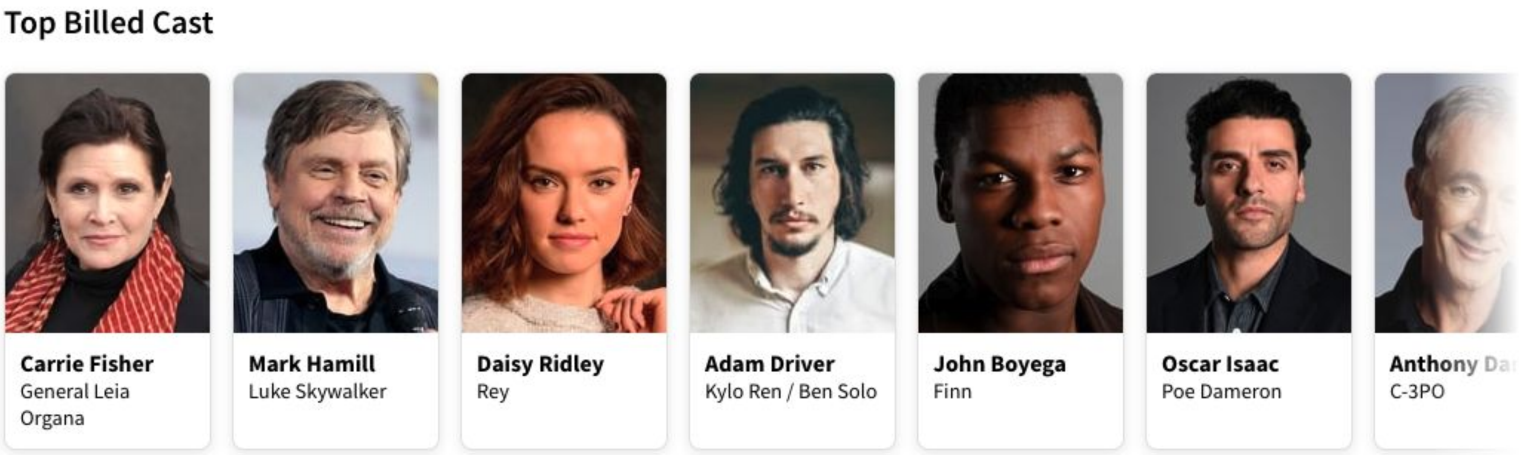
Movie	Action	Adventure	Animation	Comedy	Crime	Documentary	Drama
Star Wars: The Rise of Skywalker	1	1	0	0	0	0	0



Cast and crew aren't a good fit for one-hot encoding



Use experience of the star with top billing



How many films had Carrie Fisher appeared in before Star Wars: Rise of Skywalker?

Our final dataframe

Movie	Budget	Star Experience	Action	Adventure	Animation	Comedy	…	Profitable
Star Wars: The Rise of Skywalker	250,000,000	79	1	1	0	0		1

Split data

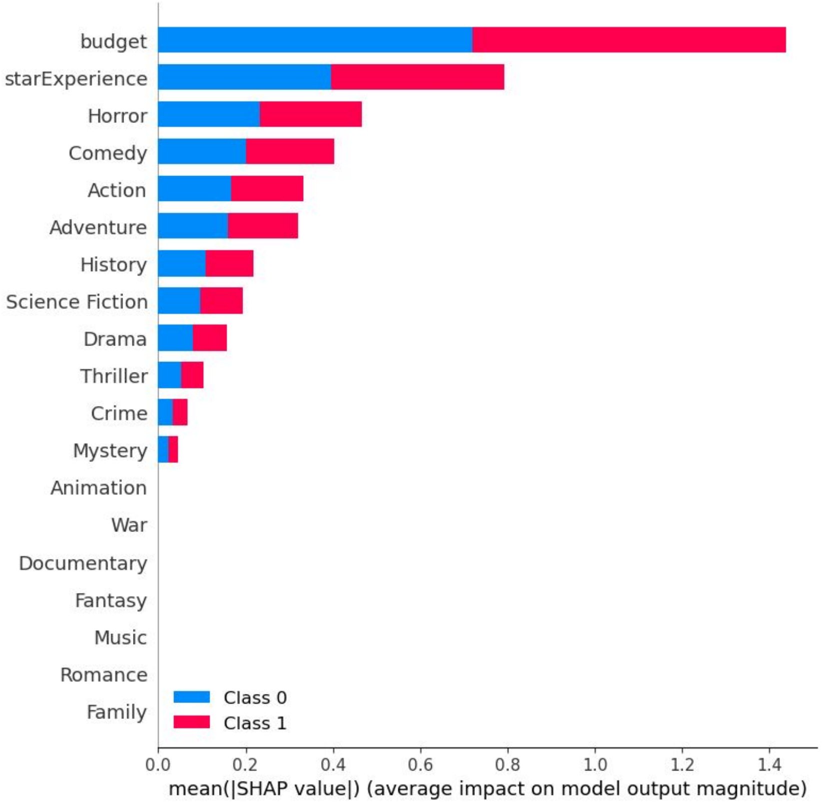
- Use films released between 1980 and 2016 as context for the “star experience” feature
- Use films released in 2016-2017 as training data
- Use films released in 2018 as validation data for tuning hyperparameters
- Use films released in 2019 as test data
- Deliberately avoided 2020 data because of COVID disruptions to the movie industry

Baseline model results

- Used LightGBM (Light Gradient Boosting Machine) library for model
- Used Optuna library to tune LightGBM hyperparameters
- Results:
  - Test F1: 0.70
  - Test AUC: 0.65
  - Test confusion matrix:

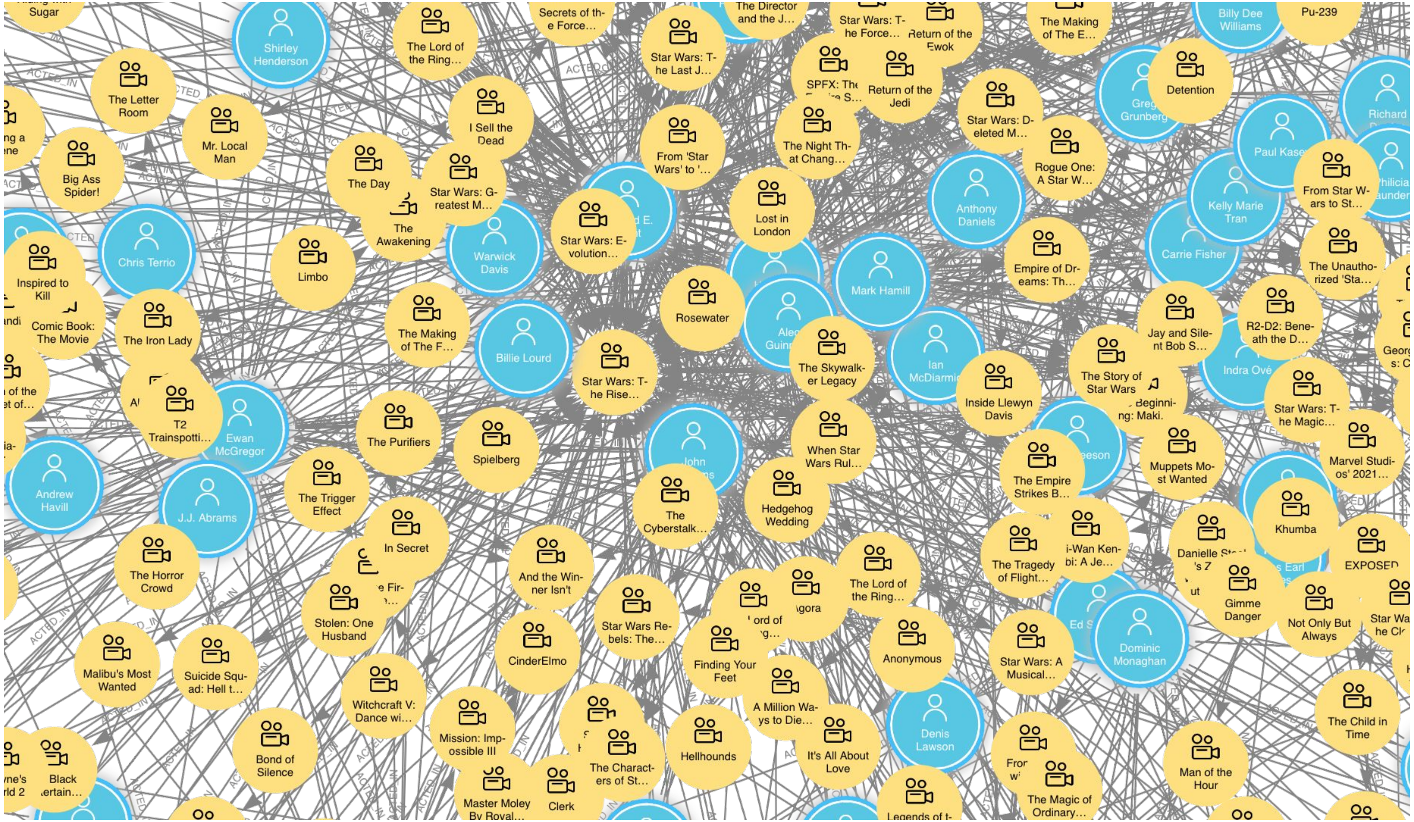
	Predicted unprofitable	Predicted profitable
True unprofitable	25	23
True profitable	37	70

Feature importances



A look at the Python code

Can we do something more with the cast and crew?



neo4j

Use Fast RP to represent the cast as a vector

Movie	Embedding
Star Wars: The Rise of Skywalker	[0.0318, 0.4215, -0.2183, 0.7352, … 0.9389]

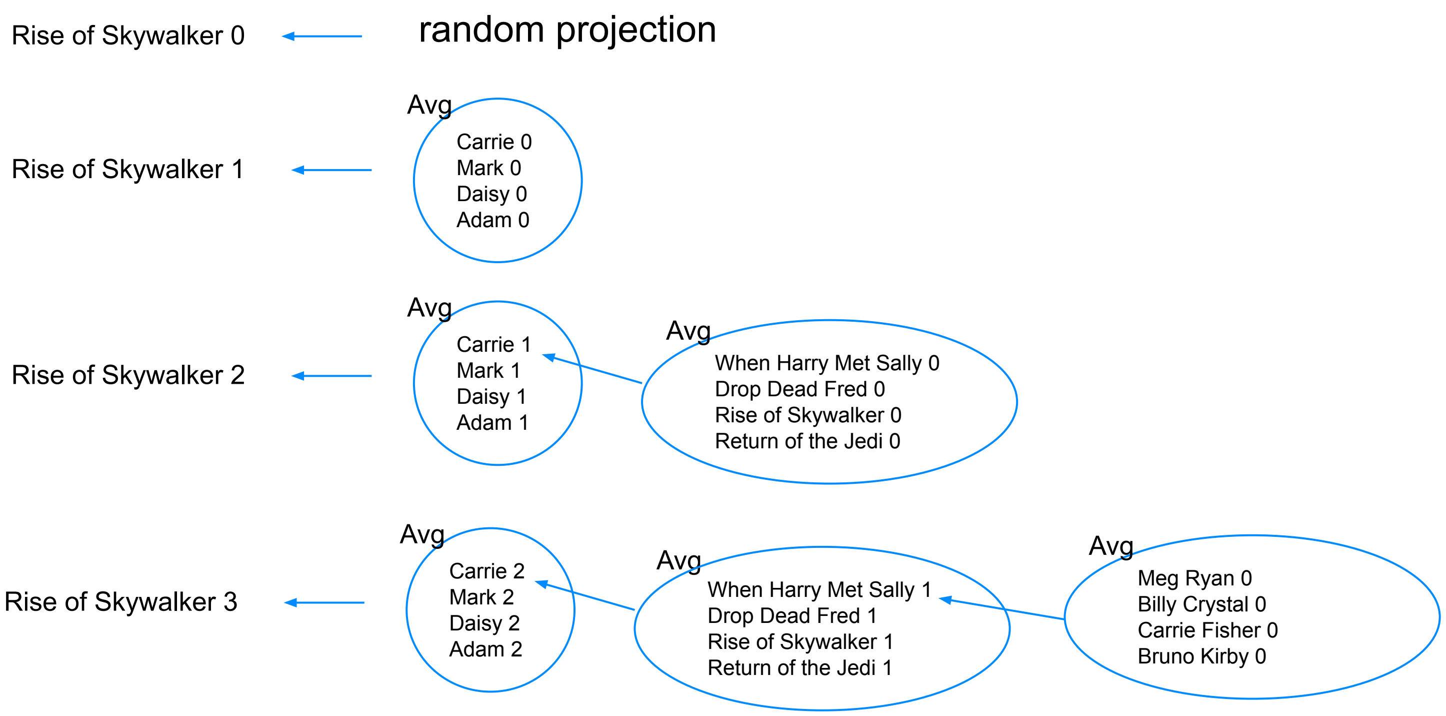
How does FastRP work?

- FastRP stands for
  Fast **R**andom **P**rojection
- Choose an embedding dimension
- Assign each node a random position in the vector space
- Think of it like a signature for the node that gets broadcast across the graph

Create a new vector representation of each node

- The new vector is located at the average coordinates of each node's neighboring nodes in the graph
- Nodes with common neighbors end up closer together in the new vector space

Repeat the aggregation process to generate more intermediate representations of each node



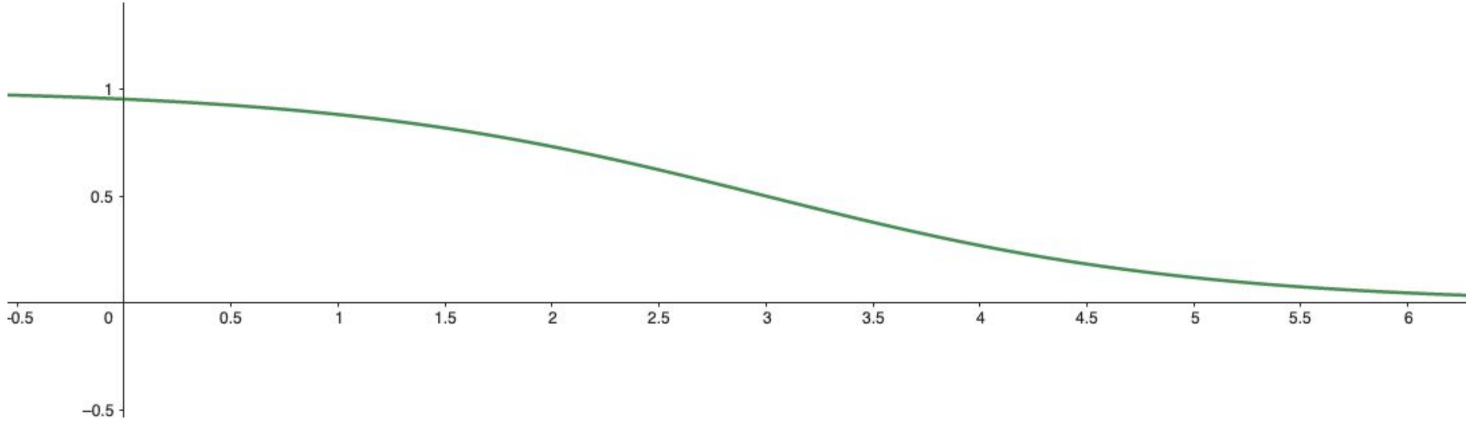
Combine intermediate projections to get a final embedding

- Normalize each intermediate vector to preserve direction but make the length 1 (L2 norm).
- Multiply each intermediate vector by iteration weights specified by the data scientist.
- Sum the weighted intermediate vectors to get the final embedding for the node.

FastRP nuances

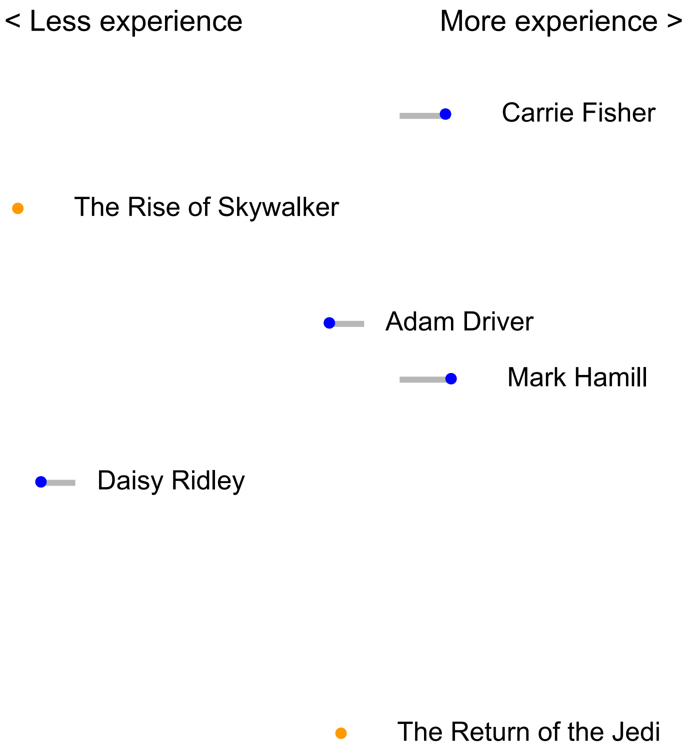
Not all actors are equally influential in the movie's success

- Use a logistic function to taper the effects of the minor characters
- Makes aggregation calculation a weighted average of the nodes neighbors
- How fast to taper becomes a hyperparameter of the model
- Use the tapered value as the relationship weight parameter for FastRP



We can incorporate node properties in the projection

- Use node properties to apply a linear transformation to the random projection
- Nudges nodes with similar property values in the same direction
- Use the property ratio parameter to control the strength of the nudge
- We used properties “budget,” “actor experience,” and “profitable” in the final embeddings



We can include the node’s initial random projection in the embedding formula

- By default, the initial random projection is not included when you sum the intermediate projections to get the embedding
- Why would you want to include it?
  - You might have some nodes with no relationships.
    - With no self-influence, these nodes would all end up with embedding [0, 0, 0…0]
    - They would end up in the same place in the embedding space even though they might not really be very similar
  - You want to give extra emphasis to the node properties that are baked into the initial vector
- Use the selfInfluence parameter to determine how much each node’s initial random projection impacts the final embedding

Normalization strength changes the influence of high degree nodes

- Node degree tends to be distributed unevenly in graphs
- Some nodes will be sending their randomly projected “signature” out across the graph many more times than other nodes
- We can use a positive normalization strength to dampen the influence of high degree nodes
- In some situations, you might use a negative normalization strength to boost the influence of high degree nodes

Train the embedding-based model

Our embedding data frame

Movie	Budget	Action	Adventure	…	War	Embedding1	Embedding2	…	Profitable
Star Wars: The Rise of Skywalker	250,000,000	1	1		0	0.938	-0.213		1

neo4j

Split data

- Use the same train-validation-test cut dates as the baseline model
- Make sure that test movies are excluded from the graph projection when calculating embeddings for training, otherwise you will get data leakage
- Only use profitability values from previous periods when generating embeddings
    - Train embeddings can see profitability for context movies only
    - Validation embeddings can see profitability for context and train movies
    - Test embeddings can see profitability for contest, train, and validation movies
    - Impute 0.5 for movies in the period you are making predictions for

Use Optuna to control LGB and FastRP parameters

- Rather than using one train and one validation set, we will generate new train and validation sets with different embedding parameters for each model run
- Dramatically increased hyperparameter search space requires more trials to explore
- It’s easier for Optuna to get stuck in a local minimum, so you might try different initial parameter values

neo4j

The results!

Baseline model

- Test F1: 0.70
- Test AUC: 0.65
- Confusion matrix:

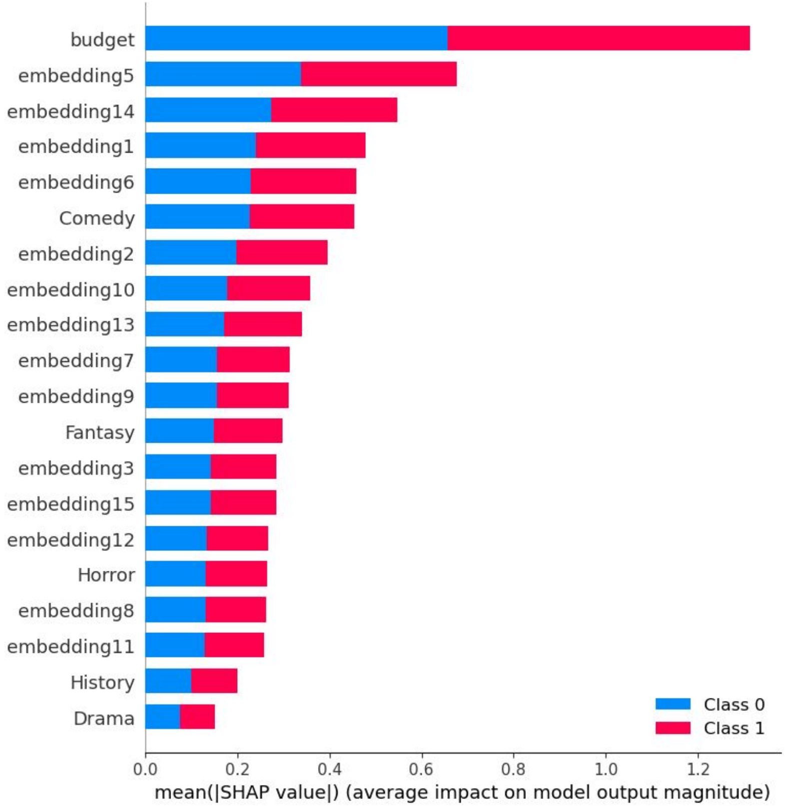
	Pred -	Pred +
True -	25	23
True +	37	70

Embedding model

- Test F1: 0.81
- Test AUC: 0.73
- Confusion matrix:

	Pred -	Pred +
True -	23	25
True +	18	89

Feature importances



A look at the Python code

Thank you!

nathan.smith@neo4j.com
@nsmith_piano
https://www.linkedin.com/in/nathansmithit/