Experiment setup
We used the following pipeline to collect the performance metrics for the categorization task:

1. Create embeddings
2. Filter articles
3. Create training and validation set
4. Create journal embeddings
5. Categorize validation articles
6. Calculate performance metrics

Create embeddings
For this research, we used word-embeddings (created using the word2vec model) and TF-IDF embeddings. The word-embeddings were created with the word2vec model from PySparks MlLib library[**?**]. The embeddings were pre-trained and have a vector size of 300, which is an industry default. The TF-IDF embeddings were created with the TF-IDF model from pyspark's MlLib library[**?**], in combination with a token hasher (HashingTF) from the same library. We used multiple hashing dimensions and multiple vocabulary sizes. Since the TF-IDF uses the output of the term hasher, the TF-IDF model produces the same dimensions.

TF-IDF selection
To limit computational expenses, we used only the top performing which can be kept in RAM.
aaaaaaaaaaaa

Embeddings
=== DESCRIBE: === - default embedding - TF-IDF embedding - 5K embedding - 1K 5K embedding

Filter articles
We create our initial set of articles by collecting all articles from the journals that were published in the year 2017, and have at least 200 publications in 2017. This reduces the journal set to 3.7 thousand journals $(3,759)$, resulting in a set of 1.3 million articles $(1,391,543)$.

Create training and validation set
We split our initial set in an 80% (79.95) - 20% (20.05) split, we use the 80% set as the training set for the journal representations, and the 20% set as the validation set for the journal representations.

Create journal embeddings
From our training set we create the journal embeddings by averaging all title embeddings as the journal title embedding, and by averaging all abstract embeddings as the journal abstract embedding. We also normalized both embeddings.

Categorize validation articles

To categorize the articles, we calculate the distance between the title- and abstract embedding of each article, from the validation set, to the title- and abstract embedding of each journal. To calculate the distance, we use cosine similarity (as provided by the SciPy library[**?**]). During this process we keep track of:

- Title-based-rank of the actual journal
- Abstract-based-rank of the actual journal
- Best scored journal on the abstract similarity
- Best scored journal on the title similarity
- Abstract similarity between the actual journal and the article
- Title similarity between the actual journal and the article

Performance measurement
We use multiple metrics to indicate the performance of the embeddings on a categorization task. These metrics are:

1. F1-score
2. Median abstract rank
3. Median title rank

- F1-score
We define the positive & negative metrics as follows:
$TruePositive$ = Articles that are correctly matched to the current journal
$FalsePositive$ = Articles that are incorrectly matched to other journals
$FalseNegative$ = Articles that are incorrectly matched to the current journal

We used these metrics to calculate the Recall, Precision & F1 as follows:

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative}$$

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive}$$

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall}$$

- Median abstract rank
We use the median abstract rank to indicate around which rank the 'standard' article would be ranked, based on its abstract. We do this by taking the median of the abstract rank from each article.

- Median title rank
We use the median title rank to indicate around which rank the 'standard' article would be ranked, based on its title. We do this by taking the median of the title rank from each article.