Electronic supplementary materials *Mendeleev Commun.*, 2025, **35**, 224–227

How to stop worrying and love multiple citation experimental data

**Yaroslav V. Timofeev, Amir M. Mrasov, Maria V. Panova,
Fedor N. Novikov and Igor V. Svitanko**

Table of contents

Appendix 1. Data extraction

ChEMBL

For 77,932 documents from version 32 of the ChEMBL database, datasets were loaded consisting of ChEMBL ID, PubMed ID, DOI, article title, abstract, journal title where the article was published, first and last page, year of publication, and authors. The number of references was extracted using the Crossref API. MEDLINE publication types were loaded from the PubMed database. For 5,357 documents, a unified publication type was present, and for 48,908 (62.76%), there was a mention of at least one MEDLINE type. ChEMBL data are regularly extracted from seven major journals: ACS Med Chem Lett, Bioorg Med Chem, Bioorg Med Chem Lett, Eur J Med Chem, J Med Chem, J Nat Prod, and MedChemComm. The share of publications from these journals in version 32 of ChEMBL is 94.30%. The remaining 5.7% of data were extracted from more than 200 journals.

PubMed

To supplement the ChEMBL data, a set of documents was loaded, including 8,689 publications from PubMed, extracted from the seven main journals used by ChEMBL, and having a similar set of MEDLINE publication types as the 5,357 documents from the previous chapter (the full list of publication types is presented in all_pubtupes.xlsx). Additionally, data on the number of references were loaded using the Crossref API (pubmed_reference.csv). The merging of annotated data from ChEMBL and PubMed contained in the seven main journals, followed by the removal of duplicates, left about 8,676 documents in the training set.

Appendix 2. Data preprocessing

Text preprocessing

To train models on texts (where a text refers to the combination of the publication title and abstract), we used various natural language processing approaches. The TF-IDF method requires thorough preprocessing of texts. Similar to the bag-of-words model, this model represents a document as a set of words, ignoring their order. However, unlike the bag-of-words model, TF-IDF considers the frequency of word occurrences in documents, allowing for more accurate weighting of their significance in the context of the entire corpus. Therefore, to use this approach, stop words (prepositions, conjunctions, articles, etc.) should be removed. Additionally, various HTML tags, accented characters, punctuation marks, and other symbols, numbers, extra spaces should be removed, words should be converted to lowercase, and lemmatization should be performed.

The stop words set was taken from the nltk.corpus package. HTML tags were removed using the BeautifulSoup library. Accented characters were replaced using the unidecode library. Punctuation marks were removed using the string library. Numbers were removed using regular expressions and the re library. Text lemmatization was performed using WordNetLemmatizer from the nltk library. When using the FastText model, text preprocessing included the same steps.

For BERT models, empty texts were replaced with the "[MASK]" token. The procedure for obtaining embeddings using BERT models was the same. The tokenizer was set up using AutoTokenizer from the transformers library, and the pre-trained model was set up using AutoModel. All models and tokenizers are available on the Hugging Face platform. The DistilBERT model was set up from distilbert/distilbert-base-uncased, PubmedBERT from neuml/pubmedbert-base-embeddings, BioBERT from dmis-lab/biobert-base-cased-v1.1, and BioMed-RoBERTa from allenai/biomed_roberta_base. Each text was split into sentences, after which the tokenizer processed the array of sentences, adding special tokens indicating the beginning and end of a sentence, as well as padding tokens.

Then, using the pre-trained model, CLS tokens of the sentences were extracted from the last hidden layer with a length of 768. To obtain the text embedding, the CLS tokens were averaged.

Categorical data preprocessing

Since the ChEMBL data are mainly extracted from seven journals, with only 5.7% of documents obtained from approximately 200 other journals, the journals were represented as a vector of length 7 using one-hot encoding. If an article did not belong to any of the seven journals, the one-hot vector for that article was a zero vector.

Numerical data preprocessing

Page Transformation

In the page data, the issue number was often indicated. In such cases, the difference between the first and last pages was zero. These values were replaced with empty values. After replacement, the number of pages was transformed to the natural logarithm of the number of pages plus one.

Author Count Transformation

The number of authors was transformed to the natural logarithm of the number of authors plus one.

Reference Count Transformation

After extracting the data, zero reference values were replaced with empty values, as most publications with zero references actually had references. The number of references was transformed to the natural logarithm of the number of references plus one.

Appendix 3. Machine learning techniques

SVM (Support Vector Machine)

A support vector machine (SVM) is a supervised machine learning algorithm designed to classify data by identifying an optimal hyperplane or decision boundary that maximizes the margin between distinct classes within an N-dimensional feature space.

We used the SVM method for classifying text embeddings, as well as for classifying the entire dataset, which includes numerical and categorical parameters. Since the embeddings were obtained in various ways, the approaches to their classification also differed.

The SVM method classifies standardized data better[1], so for classifying embeddings obtained using BERT, the SVM method in the form of the SVC class from the sklearn.svm module was used within a pipeline containing the StandardScaler class from the sklearn.preprocessing module.

The model's hyperparameters were optimized using the RandomizedSearchCV class from the sklearn.model_selection module. For the SVM model, the regularization parameter C, the type, and the coefficient for the kernel were optimized (see parameters in repository[2]).

Adding numerical and categorical data caused missing values in the training set. To solve this problem, the KNNImputer class from the sklearn.impute module was included in the pipeline, and the number of neighbors for KNNImputer was added as an additional hyperparameter (see parameters in repository[2]).

The TF-IDF method was used in combination with the SVM method within a single pipeline. Since the embedding vectors obtained using TF-IDF are already in the range from 0 to 1, additional standardization was not required. During the hyperparameter optimization process using RandomizedSearchCV, the embedding length, regularization parameter (C), and the type and coefficient for the kernel were adjusted (see parameters in repository[2]).

When adding numerical and categorical data, new columns were filled using the KNNImputer method and additionally standardized. The embedding vector length, the number of neighbors for the KNN method, the regularization parameter (C), and the type and coefficient for the kernel were optimized as hyperparameters (see parameters in repository[2]).

The process of extracting embeddings using the FastText method was carried out independently of the subsequent text classification. The hyperparameters of the FastText model were manually optimized (see FastText model parameters in repository[2]). The embedding vectors were pre-standardized using the StandardScaler class within a single pipeline that included the classifier. The classifier's hyperparameters were optimized using RandomizedSearchCV on a fixed validation dataset to prevent data leakage from the test set into the training set. The regularization parameter (C), as well as the type and coefficient for the kernel, were optimized as hyperparameters (see parameters in repository[2]).

When adding numerical and categorical data, new columns were filled using the KNNImputer method and additionally standardized. The number of neighbors for the KNN method, the regularization parameter (C), and the type and coefficient for the kernel were optimized as hyperparameters (see parameters in repository[2]).

Random Forest

The random forest method is a machine learning algorithm based on the use of an ensemble of decision trees. This algorithm integrates two key concepts: the bagging method proposed by Breiman and the method of random subspaces.

Since the random forest method is based on decision trees, which do not use the distance between objects, additional data standardization was not mandatory. For classifying embeddings obtained using BERT models, the RandomForestClassifier class from the sklearn.ensemble module was used. Hyperparameter optimization was performed using the RandomizedSearchCV class. During the hyperparameter optimization process for the random forest, parameters such as the number of decision trees, tree depth, split quality criterion, and the maximum number of samples from the training dataset used to train each tree were adjusted (see parameters in repository[2]).

When integrating numerical and categorical data, missing values were not filled using the KNNImputer method, as the random forest method can handle such missing values correctly. Hyperparameter optimization was performed using the RandomizedSearchCV method, with similar hyperparameters optimized (see parameters in repository[2]).

The TF-IDF method was used in combination with the random forest model within a single pipeline. During the hyperparameter optimization process using RandomizedSearchCV, the embedding length, number of decision trees, tree depth, split quality criterion, and the maximum number of samples from the training dataset used to train each tree were adjusted (see parameters in repository[2]).

Integrating numerical and categorical data did not change the pipeline stages, as it was not necessary to include procedures for filling missing values and standardization. Hyperparameter optimization was performed using the RandomizedSearchCV method, with the same hyperparameters optimized as for the classifier of embeddings obtained using the TF-IDF model (see parameters in repository[2]).

Embeddings obtained using the FastText model were classified using the RandomForestClassifier. During the hyperparameter optimization process using the RandomizedSearchCV method, parameters such as the number of decision trees, tree depth, split quality criterion, and the maximum number of samples from the training dataset used to train each tree were adjusted (see parameters in repository[2]).

As before, integrating numerical and categorical data did not change the pipeline stages, as it was not necessary to include procedures for filling missing values and standardization. Hyperparameter optimization was performed using the RandomizedSearchCV method, with the same hyperparameters optimized as for the classifier of embeddings obtained using the FastText model (see parameters in repository[2]).

eXtreme Gradient Boosting

Extreme Gradient Boosting, also known as XGBoost, is a scalable and optimized computer science algorithm that significantly improves the speed and efficiency of predictions performed by Gradient Boosting Machines (GBMs). This is achieved by introducing a novel tree-based learning algorithm, as well as parallel and distributed computing, which accelerates the process of finding optimal models.

For gradient boosting, as with the random forest method, it is not necessary to fill in missing values and standardize data beforehand. For classifying embeddings obtained using BERT models, the XGBClassifier class from the xgboost library was used. Hyperparameter optimization was performed using the RandomizedSearchCV class. During the hyperparameter optimization process for gradient boosting, parameters such as the number of decision trees, tree depth, learning rate, and the maximum number of samples from the training dataset used to train each tree were adjusted (see parameters in repository[2]).

Integrating numerical and categorical data did not change the pipeline stages, as it was not necessary to include procedures for filling missing values and standardization. Hyperparameter optimization was performed using the RandomizedSearchCV method, with the same hyperparameters optimized as for the classifier of embeddings obtained using the BERT model (see parameters in repository[2]).

The TF-IDF method was integrated with the gradient boosting model within a single pipeline. During the hyperparameter optimization process using RandomizedSearchCV, parameters such as embedding length, number of decision trees, tree depth, learning rate, and the maximum number of samples from the training dataset used to train each tree were adjusted (see parameters in repository[2]).

Adding numerical and categorical data did not change the pipeline stages, as it was not necessary to include procedures for filling missing values and standardization. Hyperparameter optimization was performed using the RandomizedSearchCV method, with the same hyperparameters optimized as for the classifier of embeddings obtained using the TF-IDF model (see parameters in repository[2]).

Embeddings obtained using the FastText model were classified using the XGBClassifier. During the hyperparameter optimization process using the RandomizedSearchCV method, parameters such as the number of decision trees, tree depth, learning rate, and the maximum number of samples from the training dataset used to train each tree were adjusted (see parameters in repository[2]).

As before, integrating numerical and categorical data did not change the pipeline stages, as it was not necessary to include procedures for filling missing values and standardization. Hyperparameter optimization was performed using the RandomizedSearchCV method, with the same hyperparameters optimized as for the classifier of embeddings obtained using the FastText model (see parameters in repository[2]).

Appendix 4. Estimation of the number of expected citations in documents

At this step, the activities obtained from the 32-nd version of the ChEMBL database are grouped into «protein – ligand – activity type» systems, considering only such activity types as K_i and IC_{50}. Within each system, a complete enumeration is performed to construct its corresponding graph and identify connectivity components. Then, within each connectivity component, original and cited activities are identified. The general algorithm is as follows:

1) All activity vertices are initialized

2) If the logarithms of two activities differ by less than or equal to 0.02, or from 2.98 to 3.02, a directed edge is drawn between the two corresponding vertices, from the one published later to the one published earlier.[3]

3) The connectivity components of the graph arise naturally. For each connectivity component, the vertex(es) for which there are no outgoing edges is declared to be the original.

4) Each non-original activity is matched with the nearest original that can be reached by a directed subgraph of the connectivity component. Next, the correspondence of original activities to non-original activities is used in manual data curation.

The constructed graph for one of the systems is illustrated in Figure S5. After the algorithm has run on the entire set of considered activities, the fraction of cited activities is calculated for each document and used as features for the models described above.

Appendix 5. OOD (Out of distribution) detection

Before identifying out-of-distribution (OOD) data, we excluded points from the unlabeled dataset that did not belong to the seven main ChEMBL journals. Documents containing missing values were also excluded. OOD data was identified by calculating the Mahalanobis distance between the mean values of the labeled dataset and the points in the unlabeled dataset.

The Mahalanobis distance is the distance between the point and the distribution. Let there be a distribution X on R^N with vector of mean values of parameters $\overrightarrow{\mu} = \left(\mu_1, \mu_2, \mu_{3,} \dots, \mu_N\right)^T$ and positive semi-definite covariance matrix S. The Mahalanobis distance of a point $\overrightarrow{x} = \left(x_1, x_2, x_{3,} \dots, x_N\right)^T$ from X is

$$d_M(\overrightarrow{x}, X) = \sqrt{(\overrightarrow{x} - \overrightarrow{\mu})^T S^{-1} (\overrightarrow{x} - \overrightarrow{\mu})}$$

After calculating the distances for each point in the unlabeled dataset, data falling within the 90% confidence interval were considered in-distribution, while points outside the confidence interval were considered out-of-distribution.

From the set of OOD documents, those with two "Research Support" labels were selected, as these documents predominantly contained experimental data in each iteration of manual curation of small samples (see manual curation results in the repository[2]).

Appendix 6. Fisher test details

For each article, the number of cited activities m and the number of remaining activities $n - m$ were compared with the limiting case where there is no cited activity and n original activities. Thus, for each publication for which activities were found in ChEMBL, the probability of obtaining a distribution (m citations, $n - m$ originals) was calculated, provided that the null hypothesis is true and the original distributions are the same.

Appendix 7. Description of attached files

Files used for model training

ChEMBL_32_documents.xlsx	File with publication data in ChEMBL_32 with added MEDLINE types.
ChEMBL_reference.csv	File with data on the number of references in each publication ChEMBL_32
data_from_pubmed.xlsx	File with additional data extracted from PubMed
pubmed_reference.csv	File with the number of references for publications retrieved from PubMed
abstracts_pubmed.xlsx	File with abstracts for publications retrieved from PubMed
all_pubtypes.xlsx	File with all MEDLINE tags for ChEMBL_32 database publications
docs_citations_all_14_07.xlsx	File containing publications that present activity and estimated number of citations based on the method described in Appendix 6
title_abstract_texts_X.csv	File containing titles and abstracts for the publications originally labelled
title_abstract_texts_all.csv	File containing titles and abstracts for unlabelled publications
cls_X_matrix_pubmedbert.csv	File containing CLS vectors of labelled sample texts obtained from PubMedBERT
cls_all_matrix_pubmedbert.csv	File containing CLS vectors of unlabelled sample texts obtained from PubMedBERT
cls_X_matrix_biobert.csv	File containing CLS vectors of labelled sample texts obtained from BioBERT
cls_all_matrix_biobert.csv	File containing CLS vectors of unlabelled sample texts obtained from BioBERT
cls_X_matrix_biomed_roberta.csv	File containing CLS vectors of labelled sample texts obtained from BioMed-RoBERTa
cls_all_matrix_biomed_roberta.csv	File containing CLS vectors of unlabelled sample texts obtained from BioMed-RoBERTa
cls_X_matrix_distilbert.csv	File containing CLS vectors of labelled sample texts obtained from DistilBERT
cls_all_matrix_distilbert.csv	File containing CLS vectors of unlabelled sample texts obtained from DistilBERT
train_embeddings_fasttext.csv	File containing vectors of training set texts obtained with FastText
test_embeddings_fasttext.csv	File containing vectors of test set texts obtained with FastText
val_embeddings_fasttext.csv	File containing vectors of validation set texts obtained with FastText
classif_df_previous_30_10.csv	File containing the classification of the best model without modifications
classif_df_second_30_10.csv	File containing the classification after the first modification
classif_df_third_30_10.csv	File containing the classification after the second modification
classif_df_fourth_30_10.csv	File containing the classification after the third modification

Scripts used for model training

model_comparison.ipynb	File with a Python script in which 36 models are trained
model_modification.ipynb	File with a Python script in which the best model is modified several times

all_pubtypes_parser.py	File with a Python script that parses from PubMed all MEDLINE types for each publication
abstract_parser_pubmed_data.py	File with a Python script that parses abstracts for additional documents from PubMed
reference count parser.py	File with a Python script that parses the number of references for additional documents from PubMed
PubmedBERT_vectorisation.ipynb	File with a Python script that converts text to embeddings using the large language model PubMedBERT
BioBERT_vectorisation.ipynb	File with a Python script that converts text to embeddings using the large language model BioBERT
BioMed-RoBERTa_vectorisation.ipynb	File with a Python script that converts text to embeddings using the large language model BioMed-RoBERTa
DistilBERT_vectorisation.ipynb	File with a Python script that converts text to embeddings using the large language model DistilBERT
FastText_vectorisation.ipynb	File with a Python script that converts text to embeddings using the large language model FastText
Figures.ipynb	File with a Python script that draws Figures for the article and Appendix 7

Figures

Figure S1. Histograms of the logarithm parameter of the number of pages (classified data represent data with a specific MEDLINE type, unclassified data, respectively, without this type)

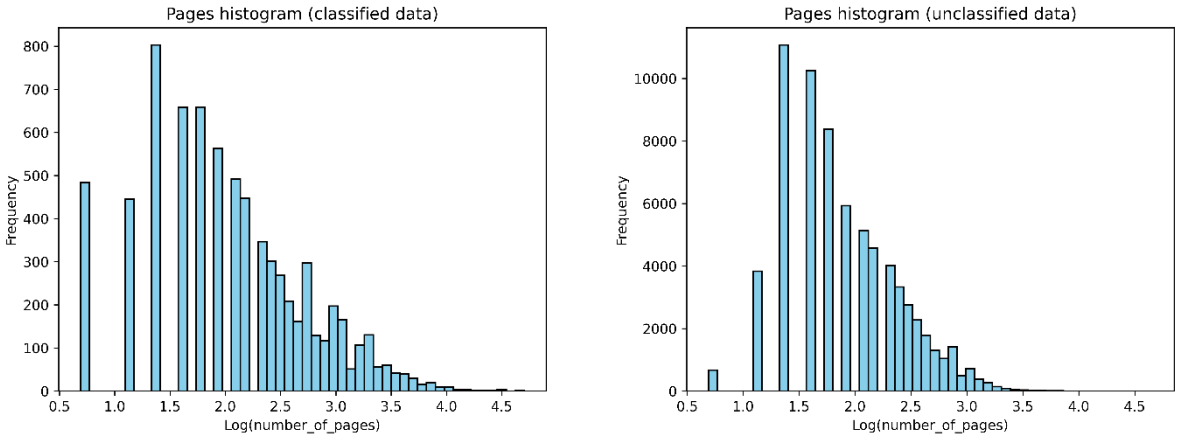
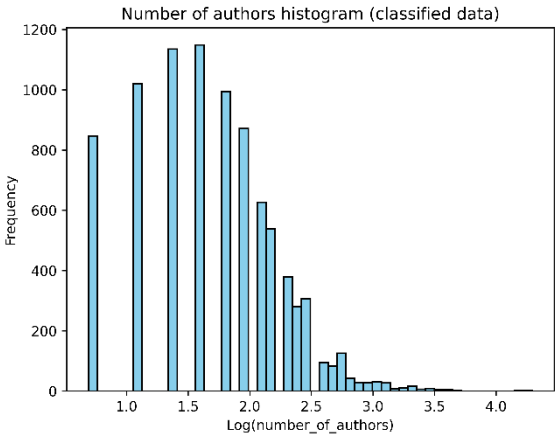


Figure S2. Histograms of the logarithm parameter of the number of authors (classified data represent data with a specific MEDLINE type, unclassified data, respectively, without this type)



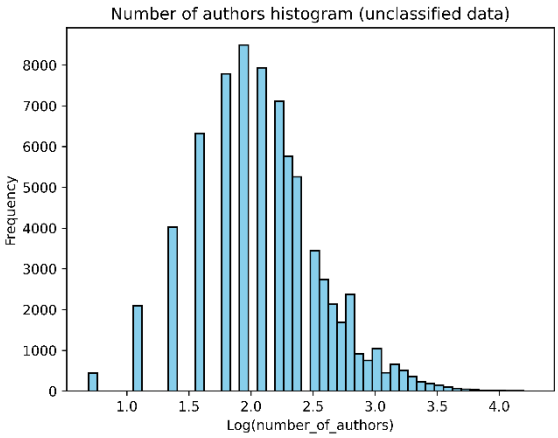


Figure S3. Histograms of the parameter of the year (classified data represent data with a specific MEDLINE type, unclassified data, respectively, without this type)

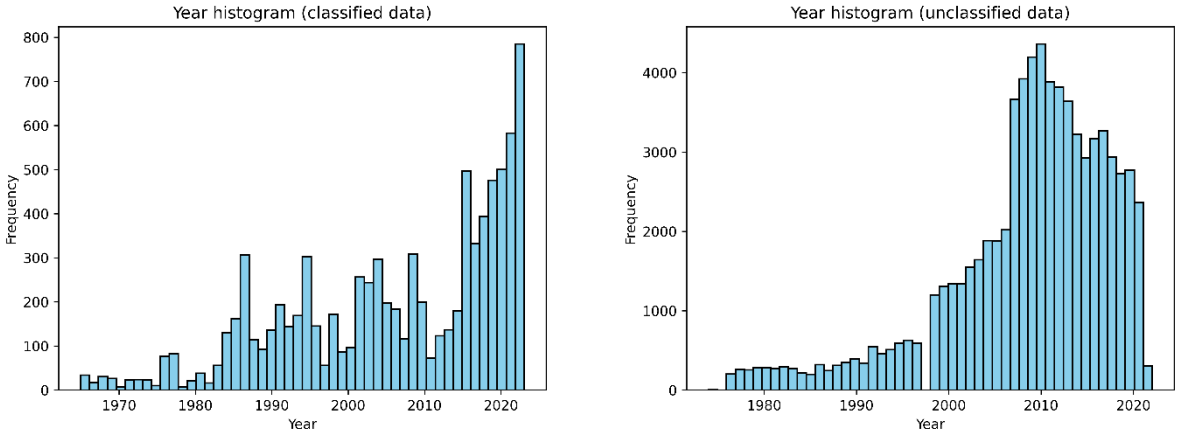


Figure S4. Histograms of the logarithm parameter of the number of reference (classified data represent data with a specific MEDLINE type, unclassified data, respectively, without this type)

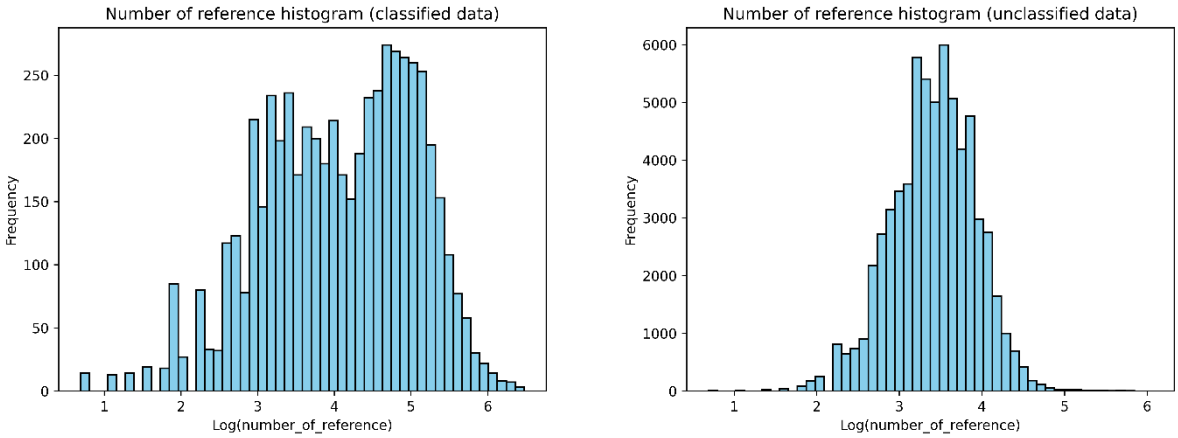
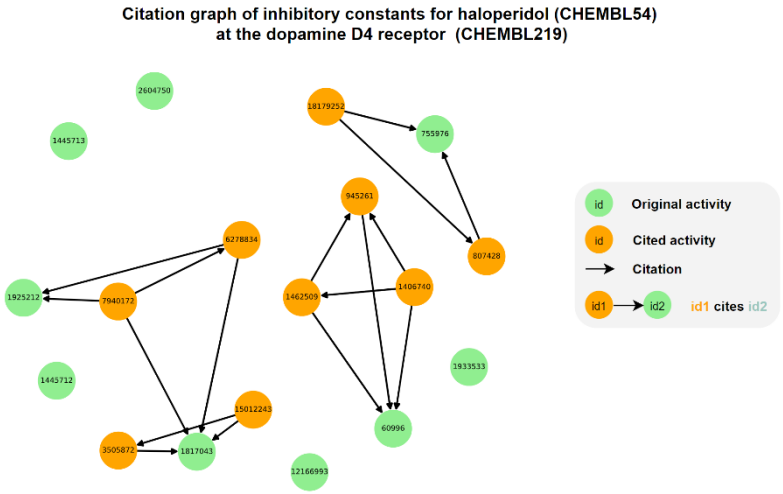


Figure S5. Citation graph constructed for a single «protein – ligand – activity type» system from a subset of ChEMBL activities**.** Each «id» represents corresponding activity ChEMBL ID.



References

S1 R. G. Brereton and G. R. Lloyd, *The Analyst*, 2010, **135**, 230–267.

S2 How-to-Stop-Worrying-and-Love-the-Multiple-Citation-Experimental-Data/README.md at main yaroslav-timofeev/How-to-Stop-Worrying-and-Love-the-Multiple-Citation-Experimental-Data, https://github.com/yaroslav-timofeev/How-to-Stop-Worrying-and-Love-the-Multiple-Citation-Experimental-Data/blob/main/README.md, (accessed December 11, 2024).

S3 C. Kramer, T. Kalliokoski, P. Gedeck and A. Vulpetti, *J. Med. Chem.*, 2012, **55**, 5165–5173.