Violin

MeLoDy Lab

Aug 05, 2021

CONTENTS:

CHAPTER

ONE

INTRODUCTION

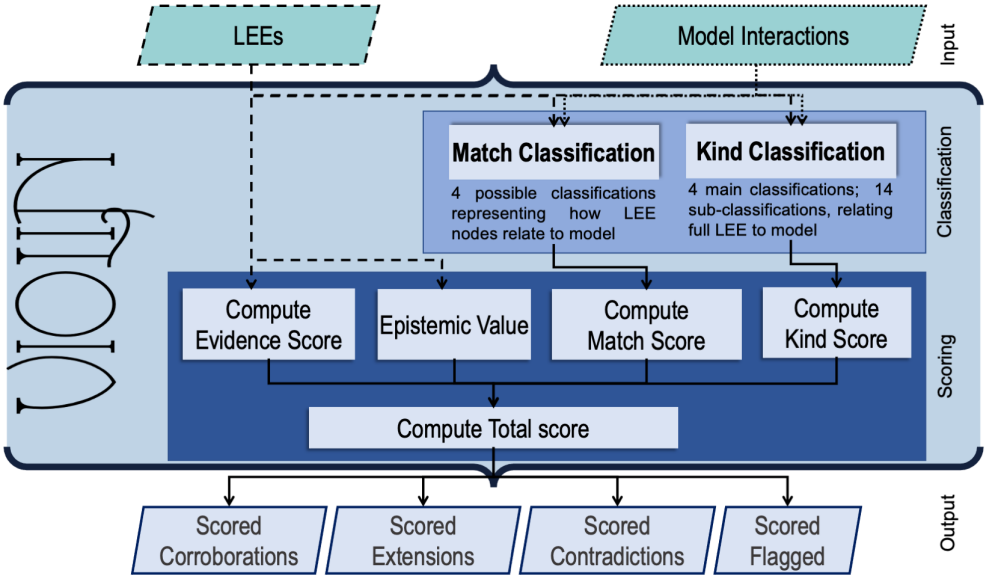
VIOLIN (Validating Interactions Of Likely Importance to the Network) is a tool used to automatically classify and judge literature-extracted interactions curated from machine readers by comparing them to existing models. This comparison can help identify key interactions for model extension.

As part of this comparison, VIOLIN assigns multiple numerical values to each literature extracted event (LEE) representing the LEEs relationship to the model. These individual scores cumulate into a Total Score, which can be used to quickly judge how relevant and useful the LEE is to a given model.

1.1 VIOLIN Objectives

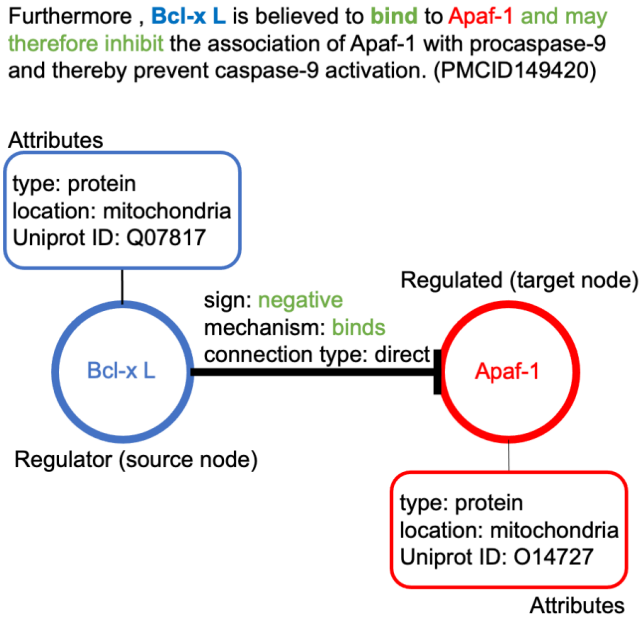
1. To classify reading output, specifically with respect to finding the most useful information for modeling
2. To compare information from the literature to an existing model using multiple metrics at varying levels of detail
3. To carry out 1. and 2. on incredibly large amounts of machine reading output very quickly

1.2 VIOLIN Methodology



VIOLIN takes spreadsheets of formatted machine reading output and static models as input and outputs the scored LEEs, separating them by their primary classification: Corroboration, Contradiction, Extension, or Flagged.

Each biological interaction can be defined by a *source* node, and *target* node, and a connecting *edge* between them. There may also be additional *attributes* that present additional details. VIOLIN takes advantage of this definiton when making its judgements.


Furthermore , Bcl-x L is believed to bind to Apaf-1 and may therefore inhibit the association of Apaf-1 with procaspase-9 and thereby prevent caspase-9 activation. (PMCID149420)
Attributes
type: protein
location: mitochondria
Uniprot ID: Q07817
Regulated (target node)
sign: negative
mechanism: binds
connection type: direct
Bcl-x L
Apaf-1
Regulator (source node)
type: protein
location: mitochondria
Uniprot ID: O14727
Attributes

CHAPTER

TWO

LICENSE AND FUNDING

Here is where the legal jargon will go. Information of Licensing, possibly a line on how to cite the tool.

This work is funded by DARPA award W911NF-17-1-0135

CHAPTER

THREE

INSTALLATION

3.1 Dependencies

VIOLIN requires Python version 3.7 or higher, as well as the pandas and NumPy libraries.

Python can be installed from the Python website for Mac or Windows OS.

For Mac OSX, Python can also be installed at the command line by:

```
>> brew install python
```

3.2 Installing VIOLIN

To install directly from github:

```
>> pip install git+https://bitbucket.org/biodesignlab/violin/src/master/
```

You will then need to run the *setup.py* file to use VIOLIN as a package:

```
>> python setup.py install
```

3.3 VIOLIN GUI

Currently in development is a user interface for VIOLIN for inputting LEEs and models for a non-specified extension comparison. Check back soon for updates.

CHAPTER

FOUR

INPUT AND OUTPUT FILES

This page details the type and format of the files VIOLIN takes as input, as well as the format of VIOLIN output

4.1 Input

As VIOLIN is developed as part of a larger modeling framework, the default files expected are model spreadsheets in the BioRECIPE format, and a machine reading output spreadsheet:

Table 1: Model input spreadsheet

Element Name	**Element ID**	**Element Type**	**Variable**	**Positive Regulators**	**Negative Regulators**
Name	ID	Type	variable name	pos regulators list	neg regulators list

The minimum required information from a model is: element/target name, element/target type, element/target ID, positive and negative regulator/source names, types, and IDs. The nomenclature does not have to be specific, as VIOLIN has the capabilities to recognize all of the commonly used vocabulary.

Table 2: LEE input spreadsheet

Element Name	**Element ID**	**Element Type**	**Positive Reg Name**	**Positive Reg ID**	**Positive Reg Type**	**Negative Reg Name**	**Negative Reg ID**	**Negative Reg Type**	**PMCID**	**Evidence**
Name	ID	Type	Pos regulator	reg ID	reg type	Neg regulator	reg ID	reg type	PMCID	evidence text

The minimum information required from reading output is: element/target name, element/target type, element/target ID, positive and negative regulator/source names, types, and IDs. The nomenclature does not have to be specific, as VIOLIN has the capabilities to recognize all of the commonly used vocabulary.

Additionally, VIOLIN has built-in functions to allow for other input file formats, found on the *Formatting (VIOLIN.formatting)* page.

Currently accepted file types are comma-separated files (**.csv**), tab-separated files (**.txt** or **.tsv**), and excel spreadsheets (**.xlsx**).

4.2 Output

VIOLIN output is sorted into five **.csv** output files:

- Total output - all interactions listed by total score
- Corroborations - all interactions classified as corroborations, listed by total score
- Extensions - all interactions classified as extensions, listed by total score
- Contradictions - all interactions classified as contradictions, listed by total score
- Questionable - all interactions classified as questionable, listed by total score

CHAPTER

FIVE

INPUT AND OUTPUT FUNCTIONS (VIOLN.IN_OUT)

This page details the functions which handle the input files and output of VIOLIN.

For more information on the types of accepted inputs, see *Input and Output Files*.

5.1 Functions

VIOLIN.in_out.**input_biorecipes**(*model*, *model_cols=['Element Name', 'Element Type', 'Element IDs', 'Variable', 'Positive Regulators', 'Positive Regulators Connection Type', 'Negative Regulators', 'Negative Regulators Connection Type']*)

This function imports a model file which is already in the BioRECIPES format, and converts all characters to lower case

Parameters

- **model** (*str*) – Directory and filename of the file containing the model spreadsheet in BioRECIPES format Accepted files: .txt, .csv, .tsv, .xlsx
- **model_cols** (*list*) – Column names of the model file. Default names found in model_columns

Returns **new_model** – Formatted model dataframe

Return type pd.DataFrame

VIOLIN.in_out.**input_reading**(*reading*, *evidence_score_cols=['Element Name', 'Element Type', 'Element ID', 'Positive Reg Name', 'Positive Reg Type', 'Positive Reg ID', 'Negative Reg Name', 'Negative Reg Type', 'Negative Reg ID', 'Connection Type']*, *atts=[]*)

This function imports the reading file into the correct mode

Parameters

- **reading** (*str*) – Directory and filename of the machine reading spreadsheet output Accepted files: .txt, .csv, .tsv, .xlsx
- **evidence_score_cols** (*list*) – Column headings used to identify identical interactions in the machine reading output
- **atts** (*list*) – List of additional attributes which are available in LEE output Default is none

Returns **new_reading** – Formatted reading dataframe, including evidence count and list of PMCIDs

Return type pd.DataFrame]

VIOLIN.in_out.**output**(*reading_df*, *file_name*, *kind_values={'att contradiction': 12, 'dir contradiction': 10, 'flagged1': 20, 'flagged2': 20, 'flagged3': 20, 'full extension': 40, 'hanging extension': 40, 'internal extension': 40, 'sign contradiction': 11, 'specification': 30, 'strong corroboration': 2, 'weak corroboration1': 1, 'weak corroboration2': 1, 'weak corroboration3': 1}*)

This function outputs the scored reading interactions. This writes output files, there are no return variables

Parameters

- **reading_df** (*pd.DataFrame*) – Dataframe of the scored reading dataframe
- **file_name** (*str*) – Directory and filename of the output suffix
- **kind_values** (*dict*) – Dictionary containing the numerical values for the Kind Score classifications Default values are found in kind_dict

5.2 Dependencies

Python: pandas and NumPy libraries, and os.path module

VIOLIN: `formatting` and `network` modules.

5.3 Defaults

Default Reading Columns

```
reading_columns = ['Element Name', 'Element Type', 'Element ID',
                   'Positive Reg Name', 'Positive Reg Type', 'Positive Reg ID',
                   'Negative Reg Name', 'Negative Reg Type', 'Negative Reg ID',
                   'Connection Type', 'Mechanism', 'Paper ID', 'Evidence']
```

Default Model Columns (From BioRECIPES format)

```
model_columns = ['Element Name', 'Element Type', 'Element IDs', 'Variable',
                 'Positive Regulators', 'Positive Regulators Connection Type',
                 'Negative Regulators', 'Negative Regulators Connection Type']
```

CHAPTER

SIX

FORMATTING (VIOLIN.FORMATTING)

This page details the formatting functions of VIOLIN, used during model and reading input.

The formatting step is important, as it:

- identifies duplicate interactions in the reading output,
- counts the number of times an interaction was found in the reading (*Evidence Score*),
- converts the variable representation of the model regulators into the common names

The formatting functions are also responsible for inputting models and machine reading output which are not in the BioRECIPES or REACH format (respectively).

6.1 Functions

VIOLIN.formatting.**evidence_score**(*reading_df*, *col_names*)

This function merges duplicate interactions and calculates evidence score of each LEE

Parameters

- **reading_df** (*pd.DataFrame*) – The dataframe of the machine reading output
- **col_names** (*list*) – Specifically the column headings used to determine if interactions are identical

Returns **counted_reading** – A new dataframe with the evidence count and PMCID list for each interaction

Return type pd.DataFrame

VIOLIN.formatting.**add_regulator_names_id**(*model_df*)

This function converts the model regulator lists from BioRECIPE variables to the common element names and database identifiers

Parameters **model_df** (*pd.DataFrame*) – The model dataframe (in BioRECIPE format)

Returns **model_df** – A new dataframe with added columns containing the positive and negative regulators listed by their Element Names and IDs

Return type pd.DataFrame

VIOLIN.formatting.**convert_to_biorecipes**(*model*, *att_list=[]*, *separate=True*)

This function imports a model which is NOT in the BioRECIPES format, such as models formatted as node-edge lists. Regulators may be represented in the REACH formatt, separated by regulator sign, or unseparated, with a speicifed column for regulator sign

Parameters

- **model** (*str*) – Directory and filename of the file containing the model BioRECIPES spreadsheet Accepted files: .txt, .csv, .tsv, .xlsx

- **model_cols** (*list*) – Column names of the model file. Default names are found in required_model
- **att_list** (*list*) – List of Element attributes (in addition to Name, ID, and Type) Default is no additional attributes
- **separate** (*Boolean*) – Whether or not the model presents regulator in separate Positive/Negative columns (True) or in a single column with Regulator Sign attribute (False) Default is True

Returns **new_model** – Formatted model dataframe

Return type pd.DataFrame

VIOLIN.formatting.**convert_reading**(*reading*, *action*, *atts=[]*)

This function formats the machine reading output, either separating regulator names and attributes into 'positive' and 'negative' columns to match REACH formatting, or combining regulator names and attributes without regulator sign distinction, and adding a 'regulator sign' column. This function can take the machine reading as either a filename or as an already uploaded dataframe.

Parameters

- **reading** (*str or pd.DataFrame*) – Machine reading output, either as file location string or dataframe
- **action** (*str*) – Action to be performed by function Accepts only 'combine' or 'separate' as input
- **atts** (*list*) – List of attributes associated with each regualtor Default list is ['Type','ID'] List should not include regulator signs (where applicable)

Returns **reading_df** – A dataframe with the specified formatting completed

Return type pd.DataFrame

6.2 Dependencies

Python: pandas and NumPy libraries, as well as the os.path module

VIOLIN: none

6.3 Usage

This module is used in during file input in the *input/output* module. For an example of using the *convert* functions, see *Tutorial 4: Alternative Input*.

CHAPTER

SEVEN

NUMERICAL FUNCTIONS (VIOLIN.NUMERIC)

This page details the numeric operators of VIOLIN.

These functions discretize qualitative operations within VIOLIN:

1. searching for an element in the reading output,
2. comparing attributes, indentifying whether a given attribute
    - matches exactly an attribute in a corresponding model interaction (MI),
    - is missing where a MI attribute is present,
    - is present where a MI attribute is missing,
    - differs from an attribute in a corresponding MI.

Both functions return numerical values to represent the outcome of the function.

7.1 Functions

VIOLIN.numeric.**find_element**(*search_type*, *element_name*, *element_type*, *model_df*)

This function finds the correct indices of an element within the model. Because elements can exists as multiple types (Protein, RNA, gene, etc.), this function checks the element name/ID along with the element type. Function may return a list, if a given element of a specific type exists with varying attributes (such as different locations)

Parameters

- **search_type** (*str*) – Whether the element is being searched for by 'name' or 'ID'
- **element_name** (*str*) – The name (or ID) of the element being searched for
- **element_type** (*str*) – The type of element searched for ('protein', 'protein family', etc.)
- **model_df** (*pd.DataFrame*) – The model dataframe

Returns **location** – All rows of the model spreadsheet in which the element is found (returns -1 if not found)

Return type list

VIOLIN.numeric.**compare**(*model_atts*, *reading_atts*)

Compares a list of model attributes to the corresponding LEE attributes, returns numeric value

Attributes are the same (strong corroboration): 0

Some or all LEE attributes are missing (weak corroboration): 1

Some or all of the model attributes are missing (specification): 2

One or more model attribue differs from the LEE attributes (contradiction): 3

Parameters

- **model_att** (*list*) – List of attributes from the model interaction
- **reading_att** (*list*) – List of attributes from the literature extracted event (LEE)

Returns **value** – Numerical representation of comparison outcome

Return type int

7.2 Dependencies

Python: pandas library

VIOLIN: none

7.3 Usage

This example searches for the index of protein family AMPK in the model spreadsheet.:

```
find_element_index("name","ampk","protein family",model_df)
>> 2
```

This example compares the *location* of the reading interaction to the *location* of its counterpart interaction in the model.:

```
reading_att = "nan"
model_att = "GO:0005737"
compare(model_att,reading_att)
>> 2
```

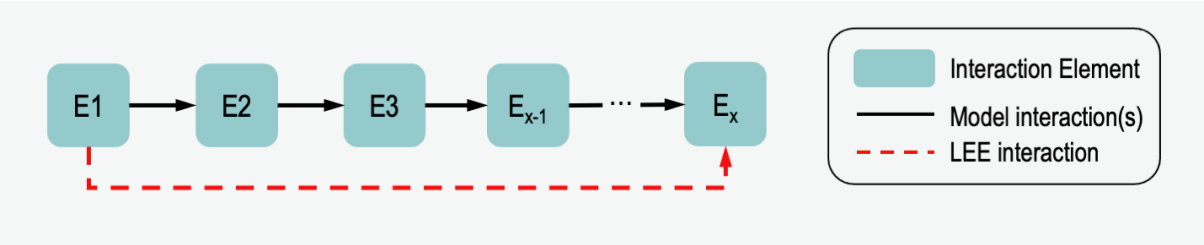
CHAPTER

EIGHT

NETWORK FUNCTIONS (VIOLIN.NETWORK)

This page details how paths are defined and found in the model in VIOLIN. Because of the compact nature of the BioRECIPES model format, the model must be converted into a node-edge list for use with the NetworkX Python package.

One special feature of VIOLIN is its ability to compare interactions from machine reading output to paths that exist in the model. For two nodes, *E1* and *Ex*, an LEE may exist with *E1* regulating *Ex*. If in the model there is a path of multiple interactions where *E1* regulates *E2* which regulates *E3* etc. to *Ex*, VIOLIN can identify this, and compare the LEE to this whole path. And indirect LEE may be a *weak corroboration* to the model interaction, or a direct LEE may be a *specification*, identifying a more direct relationship between 2 nodes than is given in the model. This functionality reduces the number of false extensions.



8.1 Functions

VIOLIN.network.**node_edge_list**(*model_df*)

This function converts the model from the BioRECIPES format into a node-edge list for use with NetworkX

Parameters **model_df** (*pd.DataFrame*) – The model dataframe, must be in BioRECIPES format

Returns **node_edge_list** – A directed graph representation of the model

Return type nx.DiGraph

VIOLIN.network.**path_finding**(*regulator*, *regulated*, *sign*, *model_df*, *graph*, *kind_values*, *reading_cxn_type*, *reading_atts*, *attributes*)

This function searches for a path between the reading regulator and regulated in the model, and calculates the kind score based on the results

Parameters

- **regulator** (*str*) – Element variable name of the regulator node
- **regulated** (*str*) – Element variable name of the regulated node
- **sign** (*str*) – Sign of regulated node
- **model_df** (*pd.DataFrame*) – Model dataframe
- **graph** (*nx.DiGraph*) – Model edgelist to create network for finding paths between elements

- **kind_values** (*dict*) – Dictionary containing the numerical values for the Kind Score classifications
- **reading_cxn_type** (*str*) – Connection Type of interaction from reading - 'i' for indirect, 'd' for direct

Returns **kind** – Kind Score value for the interaction

Return type int

8.2 Dependencies

Python: pandas and numpy libraries, NetworkX package

VIOLIN: `numeric` module

8.3 Usage

Use of the *path_finding* function in the *scoring.kind_score()* module:

```
#If model does not contain interaction - check for path
else:
    kinds.append(path_finding(model_df.loc[s_idx,'Variable'],model_df.loc[t_idx,
↪'Variable'],
                  reg_sign,model_df,graph,kind_values,lee_cxn_type,reading_atts,
↪attributes))
```

CHAPTER

NINE

SCORING (VIOLIN.SCORING)

This page details the scoring functions of VIOLIN

9.1 Match Score

The Match Score (S_M) measures how many new nodes are found in the reading with respect to the model. For an interaction from the reading $\mathbf{A} \rightarrow \mathbf{B}$, where **A** is the regulator and **B** is the regulated node, this calculation considers 4 cases which determine the scoring outcome:

1. Both **A** and **B** are in the model
2. **A** is in the model, **B** is not
3. **B** is in the model, **A** is not
4. Neither **A** nor **B** are in the model

Deafult Match Level scores are given for the assumption that the user wants to extend a given model without adding new nodes which may not be useful to the network. Thus, new regulators and new edges between model nodes are considered most important.

9.2 Kind Score

The Kind Score (S_K) measures the edges of a reading interaction (LEE) with respect to the model (MI). The Kind Score easily identifies the classification of an interaction, as well as searching for paths between nodes in the model when the reading interaction is identified as indirect. Using the same assumption from the Match Level calculation, the Kind Score represents the following scenarios:

Classification	Definition
Corroboration	LEE matches MI
Extension	LEE contains information not found in model
Contradiction	LEE disputes information in MI
Flagged	Must be judged manually

And within each classification, there are further sub-classifications. These subclassifications allow for more detailed scoring, if the user wishes.

Corroborations

Strong Corroboration: LEE matches MI exactly

Weak Corroboration Type 1: LEE matches direction, sign, connection type, and node type, of a model interaction but is missing additional attributes

Weak Corroboration Type 2: an indirect LEE matches direction and sign of direct model interaction with non-contradictory attributes

Weak Corroboration Type 3: an indrect LEE matches the direction and sign of a *path* in the model with non-contradictory attributes

Extensions

Full Extension: Neither source nor target of the LEE is in the model

Hanging Extension: The target of the LEE is in the model

Internal Extension: Both the source and target of the LEE are in the model, but there is no model interaction between them

Specification: LEE contains more information (attributes) than MI, or shows a direct relationship compared to Model Path

Contradictions

Direction Contradiction: The target and source of the LEE correspond to the source and target of the model interaction, respectively

Sign Contradiction: The regulation sign of the LEE is opposite of the corresponding model interaction (e.g. the LEE shows a positive regulation where the model interaction shows negative)

Attribute Contradiction: One or more of the LEE node attributes differs from that found in the corresponding model interaction

Flagged

Flagged Type 1: Mismatched Direction and non-contradictory Other Attributes with a Direct connection type in the model

Flagged Type 2: An LEE with a corresponding path which has one or more Mismatched Attributes

Flagged Type 3: An LEE which is a self-regulation based on the definition of model element (e.g. LEE has caspase-8 –> caspase-3, but the model considers cas-8 and cas-3 to be the same element)

9.3 Evidence Score

The Evidence Score (S_E) is a measure of how many times an LEE is found in the machine reading output. In the *VIOLIN.formatting.evidence_score()* function, column names are defined to determine how the function determines duplicates. For example, the Evidence Score can be calculated by comparing all LEE attributes and all machine readings spreadsheet columns. So only an exact match between LEEs will be counted as a duplicate. However, the user can also define fewer attributes, creating a more coarse-grained Evidence Score calculation.

9.4 Epistemic Value

In the NLP output, we sometimes receive an Epistemic Value (S_B), which is a measure of the believability of an interaction in the LEI. Zero, Low, Moderate, and High believability correspond to numerical scores of 0.0, 0.33, 0.67, and 1.0, respectively.

9.5 Total Score

The total score (S_T) is calculated by

$$S_T = [S_K + (S_E * S_M)] * S_B$$

9.6 Functions

VIOLIN.scoring.**match_score**(*x*, *reading_df*, *model_df*, *reading_cols*, *match_values={'both present': 10, 'neither present': 0.1, 'source present': 1, 'target present': 100}*)

This function calculates the Match Score for an interaction from the reading

Parameters

- **x** (*int*) – The line of the reading dataframe with the interaction to be scored
- **reading_df** (*pd.DataFrame*) – The reading dataframe
- **model_df** (*pd.DataFrame*) – The model dataframe
- **reading_cols** (*dict*) – Column Header names taken on input
- **match_values** (*dict*) – Dictionary assigning Match Score values Default values found in match_dict

Returns **match** – Match Score value

Return type int

VIOLIN.scoring.**kind_score**(*x*, *model_df*, *reading_df*, *graph*, *reading_cols*, *kind_values={'att contradiction': 10, 'dir contradiction': 10, 'flagged1': 20, 'flagged2': 20, 'flagged3': 20, 'full extension': 40, 'hanging extension': 40, 'internal extension': 40, 'sign contradiction': 10, 'specification': 30, 'strong corroboration': 2, 'weak corroboration1': 1, 'weak corroboration2': 1, 'weak corroboration3': 1}*, *attributes=[]*, *mi_cxn='d'*)

This function calculates the Kind Score for an interaction in the reading

Parameters

- **x** (*int*) – The line of the reading dataframe with the interaction to be scored
- **model_df** (*pd.DataFrame*) – The model dataframe
- **reading_df** (*pd.DataFrame*) – The reading dataframe
- **graph** (*nx.DiGraph*) – directed graph of the model,used when function calls path_finding module
- **reading_cols** (*dict*) – Column Header names taken on input
- **kind_values** (*dict*) – Dictionary assigning Kind Score values Default values found in kind_dict
- **attributes** (*list*) – List of attributes compared between the model and the machine reading output Default is None

- **mi_cxn** (*str*) – What connection type should be assigned to model interactions if not available Accepted values are “d” (direct) or “i” (indirect) Deafult is “d”

Returns **kind** – Kind Score score value

Return type int

VIOLIN.scoring.**epistemic_value**(*x*, *reading_df*)

Finds the epistemic value of the LEE (when available)

Parameters

- **x** (*int*) – The line of the reading dataframe with the interaction to be scored
- **reading_df** (*pd.DataFrame*) – The reading dataframe

Returns **e_value** – The Epistemic Value; if there is no Epistemic Value available for the reading, default is 1 for all LEEs

Return type float

VIOLIN.scoring.**score_reading**(*reading_df*, *model_df*, *graph*, *reading_cols*, *kind_values={'att contradiction': 10, 'dir contradiction': 10, 'flagged1': 20, 'flagged2': 20, 'flagged3': 20, 'full extension': 40, 'hanging extension': 40, 'internal extension': 40, 'sign contradiction': 10, 'specification': 30, 'strong corroboration': 2, 'weak corroboration1': 1, 'weak corroboration2': 1, 'weak corroboration3': 1}*, *match_values={'both present': 10, 'neither present': 0.1, 'source present': 1, 'target present': 100}*, *attributes=[]*, *mi_cxn='d'*)

Creates new columns for the Match Score, Kind Score, Epistemic Value, and Total Score. Calls scoring functions and stores the values in the approriate column.

Parameters

- **reading_df** (*pd.DataFrame*) – The reading dataframe
- **model_df** (*pd.DataFrame*) – The model dataframe
- **graph** (*nx.DiGraph*) – directed graph of the model, necessary for calling kind_score module
- **reading_cols** (*dict*) – Column Header names taken upon input
- **kind_values** (*dict*) – Dictionary assigning Kind Score values Default values found in kind_dict
- **match_values** (*dict*) – Dictionary assigning Match Score values Default values found in match_dict
- **attributes** (*list*) – List of attributes compared between the model and the machine reading output Default is None

Returns **scored = reading_df** – reading dataframe with added scores

Return type pd.DataFrame

9.7 Dependencies

Python: pandas library

VIOLIN: `network` and `numeric` modules.

9.8 Defaults

Default Match Score values

```
match_dict = {"source present" : 1,
                "target present" : 100,
                "both present" : 10,
                "neither present" : 0.1}
```

Default Kind Score values

```
kind_dict = {"strong corroboration" : 2,
                "weak corroboration1" : 1,
                "weak corroboration2" : 1,
                "weak corroboration3" : 1,
                "hanging extension" : 40,
                "full extension" : 40,
                "internal extension" : 40,
                "specification" : 30,
                "dir contradiction" : 10,
                "sign contradiction" : 10,
                "att contradiction" : 10,
                "flagged1" : 20,
                "flagged2" : 20,
                "flagged3" : 20}
```

9.9 Usage

scoring.score_reading scores the reading output in the following manner:

```
    for x in range(reading_df.shape[0]):

        scored_reading_df.at[x,'Match Score'] = match_score(x,reading_df,model_df,
↪reading_cols,match_values)
        scored_reading_df.at[x,'Kind Score'] = kind_score(x,model_df,reading_df,graph,
↪reading_cols,kind_values,attributes,mi_cxn)
        scored_reading_df.at[x,'Epistemic Value'] = epistemic_value(x,reading_df)
        scored_reading_df.at[x,'Total Score'] =  {((scored_reading_df.at[x,'Evidence␣
↪Score']*
                                                   scored_reading_df.at[x,'Match␣
↪Score'])+
                                                   scored_reading_df.at[x,'Kind Score
↪'])*
                                                   scored_reading_df.at[x,'Epistemic␣
↪Value']}
```

CHAPTER

TEN

VISUALIZATION (VIOLIN.VISUALIZE_VIOLIN)

VIOLIN's visualization function creates a visual summary of the VIOLIN output, incuding total score, evidence score, and match score distributions.

The visualization function includes a filtering option, which can help the user make choices on how to use the VIOLIN output. Visualization can be filtered by three possible metrics:

1. "%x" : Returns the top X% of LEEs, by Total Score
2. "Se>y" : Returns all LEEs with an Evidence Score greater than Y
3. "St>z" : Returns all LEEs with a Total Score grater than Z

- When visualizing the total output, this function shows the score distributions by classification, as well as the classification distribution
- When visualizing output of a single classification, the classification distribution is replaced by the number of LEEs given that classification
- When subcategories are identified in the Kind Score definition, additional plots of subcategory distribution are included

10.1 Functions

VIOLIN.visualize_violin.**visualize**(*match_values*, *kind_values*, *file_name*, *filter_opt='100%'*)

This creates graphs of the VIOLIN output: evidence score, match score, and total score, and classification breakdown

Parameters

- **match_values** (*dict*) – Dictionary assigning Match Score Values
- **kind_values** (*dict*) – Dictionary assigning Kind Score values
- **file_name** (*string*) – VIOLIN output to be visualized. Can be specific classification, or choosing 'TotalOutput' file will visualize all VIOLIN output
- **filter_opt** (*str*) – How much VIOLIN output should be visualized. Can be filtered by top % of total score, evidence score (Se) threshold, or total score (St) threshold Accepted options are 'X%','Se>Y', or 'St>Z', where X, Y, and Z, are values Default is '100%' (Total Output)

10.2 Dependencies

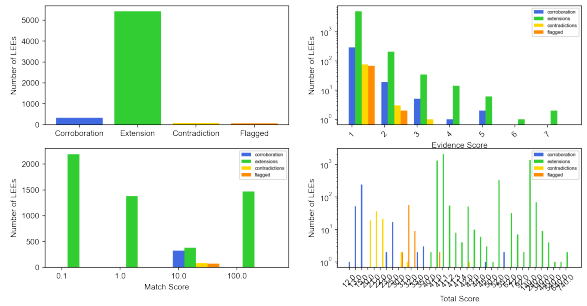
Python: pandas and matplotlib libraries

VIOLIN: none

10.3 Example output

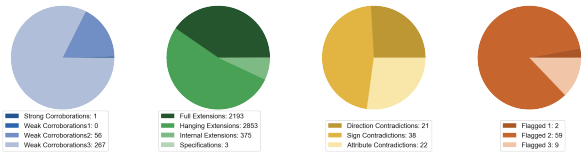
Visualizing the total output

```
visualize(match_dict, kind_dict, 'RA2_sub_TotalOutput.csv', filter_opt='100%')
```



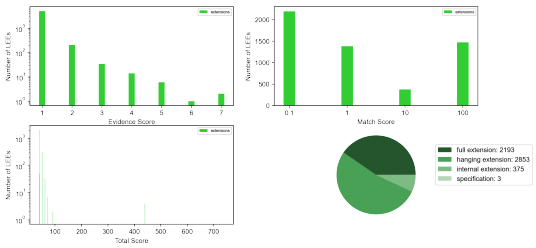
Visualizing subcategories:

```
kind_dict ={"strong corroboration" : 2,
            "weak corroboration1" : 1,
            "weak corroboration2" : 3,
            "weak corroboration3" : 5,
            "hanging extension" : 40,
            "full extension" : 41,
            "internal extension" : 42,
            "specification" : 30,
            "dir contradiction" : 10,
            "sign contradiction" : 11,
            "att contradiction" : 12,
            "flagged1" : 20,
            "flagged2" : 21,
            "flagged3" : 22}
visualize(match_dict, kind_dict, 'RA2_sub_TotalOutput.csv', filter_opt='100%')
```



Visualizing an individual category (extensions)

```
visualize(match_dict, kind_dict, 'RA2_sub_extensions.csv', filter_opt='100%')
```


extensions
Number of LEEs
Evidence Score
Match Score
Total Score
full extension: 2193
hanging extension: 2853
internal extension: 375
specification: 3

CHAPTER

ELEVEN

TUTORIALS

11.1 Test files

Input files used to create publication data can be found at https://bitbucket.org/biodesignlab/violin/src/master/

11.2 Tutorial 1: Default Use (general extension)

This tutorial assumes that the input files are a model in the standard BioRECIPES format and an LEE set from REACH.

A *violin_tutorial* folder containing an example ipy notebook, input files, and expected output is included with the VIOLIN package. This tutorial recreates some of the data presented in [Paper name and link]

11.3 Tutorial 2: Specifying Score and Attributes

If the user wants to compare attributes in addition to the default, the additional attributes must be defined before inputting the reading, and then call them in the input_reading function:

```
attributes = ['Location ID']
reading_df, reading_cols = input_reading(reading_file, evidence_score_cols,
↪atts=attributes)
```

If the user wants to use a different scoring scheme than the default (e.g. to account for the subclassifications in judgement), a scoring dictionary must be defined before calling the scoring function:

```
kind_dict = {"strong corroboration" : 2,
             "weak corroboration1" : 1,
             "weak corroboration2" : 3,
             "weak corroboration3" : 5,
             "hanging extension" : 40,
             "full extension" : 41,
             "internal extension" : 42,
             "specification" : 30,
             "dir contradiction" : 10,
             "sign contradiction" : 11,
             "att contradiction" : 12,
             "flagged1" : 20,
             "flagged2" : 21,
             "flagged3" : 22}
scored = score_reading(reading_df,model_df,graph,reading_cols,kind_values = kind_dict,
↪attributes = attributes)
```

11.4 Tutorial 3: Using VIOLIN at the terminal

This tutorial assumes that the input files are a model in the standard BioRECIPES format and an LEE set from REACH. This tutorial also assumes that the user wants to run VIOLIN for basic extension using VIOLIN's default values, and visualization is for the total output

The *use_violin.py* script is included in the *violin_tutorial* folder. The input for this script allows for four classification schemes:

1. 'extend' - default Kind and Match Score values for general extension
2. 'extend subcategories' - general extension values with subcategories specified in Kind Score values
3. 'corroborate' - Kind and Match Score values for general corroboration (preference towards strong corroborations, weak corroborations, contradictions)
4. 'corroborate subcategories' - general extension values with subcategories specified in Kind Score values

as well as the same filtering options from *Visualization (VIOLIN.visualize_violin)*

use_violin.**use_violin**(*model_file*, *lee_file*, *out_file*, *score='extend'*, *filt_opt='100%'*)

This function calculates the Match Score for an interaction from the reading

Parameters

- **model_file** (*str*) – Directory and filename of the the machine reading spreadsheet output Accepted files: .txt, .csv, .tsv, .xlsx
- **lee_file** (*str*) – Directory and filename of the model file in BioRECIPE format Accepted files: .txt, .csv, .tsv, .xlsx
- **out_file** (*str*) – Directory and filename of the output suffix
- **score** (*str*) – Scoring scheme used for classification Options are: 'extend', 'extend subcategories', 'corroborate', 'corroborate subcategories'
- **filt_opt** (*str*) – How much VIOLIN output should be visualized. Can be filtered by top % of total score, evidence score (Se) threshold, or total score (St) threshold Accepted options are 'X%','Se>Y', or 'St>Z', where X, Y, and Z, are values Default is '100%' (Total Output)

To run *use_violin.py* at the command line:

```
python use_violin.py test_input/ModelA.csv test_input/RA2_reading.xlsx output extend␣
↪50%
```

11.5 Tutorial 4: Alternative Input

Machine Reading Output

REACH has a specific output format, where the source nodes are separated by their regulation sign - positive regulators are stored separately from negative regulators. In other machine readers, source nodes are not separated, and instead the regulation sign is stored as a node attribute (e.g. increase/decrease or activate/inhibit):

The *VIOLIN.formatting.convert_reading()* can automatically separate the regulators into the specific positive/negative distinctions, along with any associated attributes.

If the user has the following machine reading output spreadsheet:

Table 1: LEE input spreadsheet

Target Name	Target ID	Target Type	Source Name	Source ID	Source Type	Source Location	Regulation	PMCID	Evidence
Name	ID	Type	regulator name	reg ID	reg type	reg location	increase/decrease	PMCID	evidence text

Running the following function:

```
reading_df = convert_reading(reading, 'separate', atts = ['location'])
```

Would produce the new spreadsheet (stored as a pandas dataframe)

Table 2: LEE input spreadsheet

Target Name	Target ID	Target Type	Positive Source Name	Positive Source ID	Positive Source Type	Positive Source Location	Negative Source Name	Negative Source ID	Negative Source Type	Negative Source Location	PMCID	Evidence
Name	ID	Type	Pos regulator	reg ID	reg type	reg location	Neg regulator	reg ID	reg type	reg location	PMCID	evidence text

Model Representation

While the BioRECIPES format is necessary for the DySE modeling framework, models are frequently presented as node-edge lists. In this case, the model will need to be converted to the BioRECIPE format using the `VIOLIN.formatting.nconvert_to_biorecipes()` function:

```
model_df,graph = formatting.convert_to_biorecipes(model, att_list=['location',
↪'organism'], separate=False)
```

In a single step, VIOLIN separates the regulators into positive and negative (if needed) and condenses interactions into lists of regulators for each element, as shown in *Input*. VIOLIN also adds the weights representing the regulation sign, necessary for the `VIOLIN.formatting.path_finding()` function, making use of the already created node-edge list instead of calling on the `VIOLIN.formatting.node_edge_list()` function.

CHAPTER

TWELVE

INDICES AND TABLES

- genindex
- search

INDEX