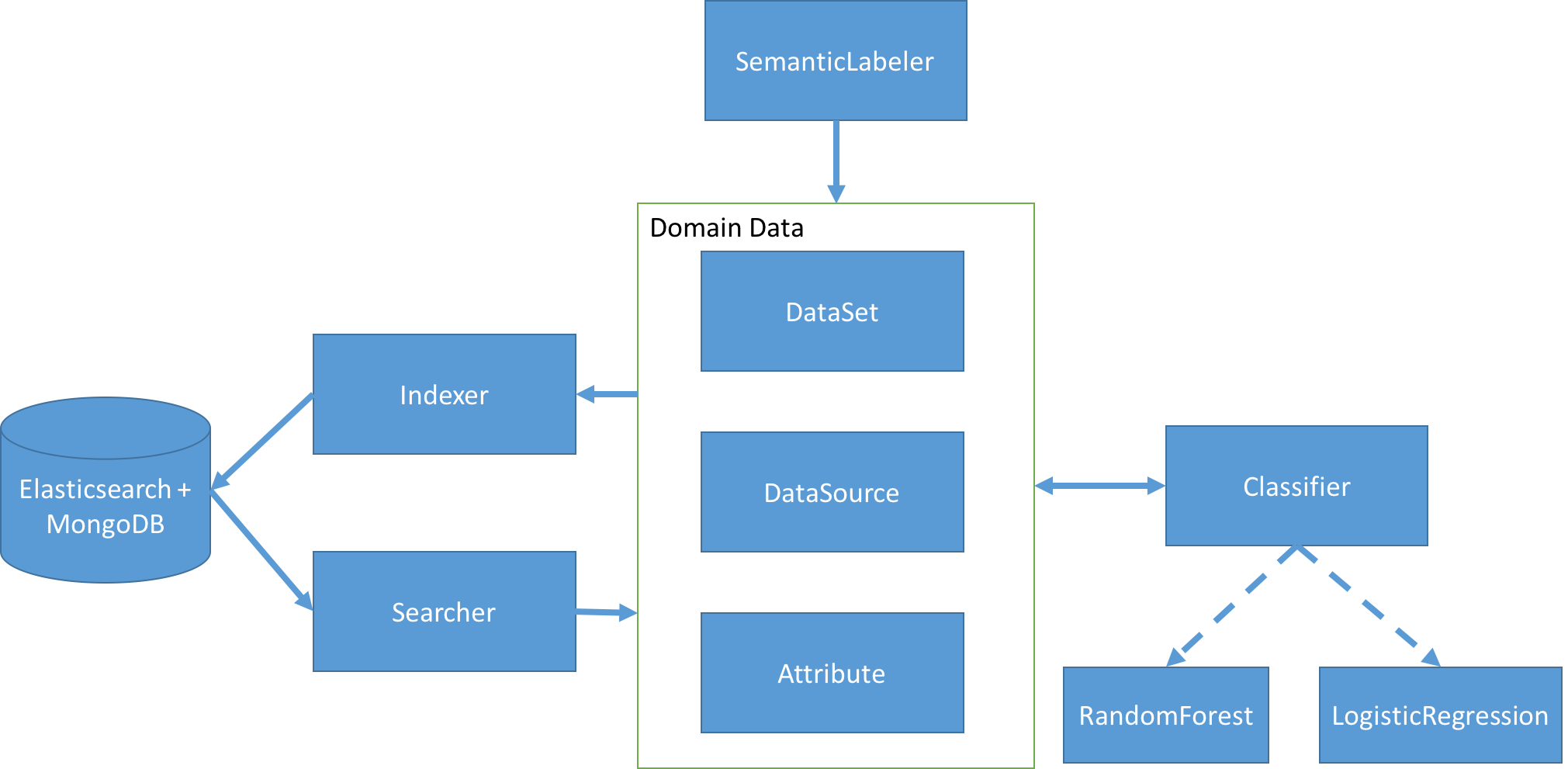
**PROGRAM MANUAL**

# OVERALL STRUCTURE

The overall structure of the program is illustrated as follows:



Rectangular shape represent for high level classes in the program. Solid arrows shows relations in invoking methods and return results in program flows. Dash arrows show inheritance relationships between superclasses and subclasses.

# TOP LEVEL CLASSES

* **SematicLabeler:** 
  + Top-level class. Contains core methods to run semantic labeling on different data sets.
  + Methods:
    - **train\_classifier(training\_set\_dict)**: train machine learning classifiers using the specific data sets
      * Params:
        + training\_set\_dict: dictionary of training data sets and the corresponding number of labeled sources for each data set
      * Output: trained classifier model (class Classifier)
      * For example: “*train\_classifier({“soccer”: [5]}”* will train the classifier with generated data when run semantic labeling on soccer data set with 5 labeled sources.
    - **store\_data\_sets(set\_list)**: store data from a list of data sets into MongoDB and Elasticsearch. Data sets should be stored seperately in different subfolders inside “/path\_to\_project/data” directory.
      * Params:
        + set\_list: list of data sets that need to be stored
      * Output: none
      * For example: “*store\_data\_sets([“soccer”, “museum”])”* will store soccer and museum data sets into MongoDB and Elasticsearch
    - **test\_semantic\_typing(labeling\_set\_dict)**: run semantic labeling on specific data sets
      * Params:
        + labeling\_set\_dict: dictionary of data sets and the corresponding numbers of labeled sources for each data set.
      * Output: dictionary of labeling sets and their Mean Reciprocal Rank (MRR) scores in the evaluation
      * For example: “*test\_semantic\_typing({“soccer”: [5]})”* will run semantic labeling on soccer data set with 5 labeled sources

# HIGH LEVEL MODULES AND CLASSES

## Module data\_source

* **DataSet**:
  + Class contains details about data set
  + Methods:
    - **read(folder\_path)**: read data set from a folder contains data source files. (Files need to be in xml, json or csv format)
      * Params:
        + folder\_path: path to the folder
      * Output: none
    - **save()**: store data set into Mongodb and ElasticSearch
      * Output: none
    - **test(source\_number\_list, classifier):** test semantic labeling with different number of labeled sources from the same data set
      * Params:
        + source\_number\_list: list of different numbers of labeled sources in semantic labeling
        + classifier: machine learning classifier used in semantic labeling
      * Output: MRR score for this test
    - **generate\_training\_data(source\_number\_list)**: generate training data for classifier with different number of labeled sources
      * Params:
        + source\_number\_list: list of different numbers of labeled sources in semantic labeling
      * Output: training data for classifier
* **DataSource**:
  + Class contains details about data source
  + Methods:
    - **read(file\_path)**: read data source from a. (File need to be in xml, json or csv format)
      * Params:
        + filde\_path: path to read file
      * Output: none
    - **save(set\_name)**: store source data into Mongodb and ElasticSearch with its data set
      * Params:
        + set\_name: name of data set that contain this source
      * Output: none
    - **label(set\_name, labeled\_data\_dict, classifier, labeled\_source\_list)**: run semantic labeling on this source with labeled sources as parameters
      * Params:
        + set\_name: name of data set that contain the data source
        + labeled\_data\_dict: dictionary that contains attributes’ data from labeled sources (Can be retrieved using Searcher.search\_attribute\_data)
        + classifier: classifier models used in semantic labeling
        + labeled\_source\_list: list of labeled sources for semantic labeling.
      * Output: dictionary that contains probability of candidate semantic types on every attribute in the source

### **Attribute**:

* + Class contains details about attribute
  + Methods:
    - **add\_value(value)**: add new value into attribute data
      * Params:
        + value: value to be added (“str” or “unicode” type)
      * Output: none
    - **to\_json()**: convert attribute data into json format for Elasticsearch and MongoDB indexing
      * Output: none
    - **compute\_features(set\_name, labeled\_sources, labeled\_data\_dict)**: compute feature vector from attribute data in semantic labeling with labeled sources as parameters
      * Params:
        + set\_name: name of data set that contain the attribute
        + labeled\_source\_list: list of labeled sources for semantic labeling.
        + labeled\_data\_dict: dictionary that contains attributes’ data from labeled sources (Can be retrieved by Searcher.search\_attribute\_data)
      * Output: list of multiple feature vectors between the attribute and other labeled attributes from labeled sources. Each feature vector is a dictionary contains the feature names and their values.
    - **predict\_type(set\_name, labeled\_sources, labeled\_data\_dict, classifier):** predict semantic type of attribute based on its data and labeled sources as parameters
      * Params:
        + set\_name: name of data set that contain the attribute
        + labeled\_source\_list: list of labeled sources for semantic labeling.
        + labeled\_data\_dict: dictionary that contains attributes’ data from labeled sources (Can be retrieved by Searcher.search\_attribute\_data)
        + classifier: classifier models used in semantic labeling
      * Output: list of predictions of semantic types for this attribute. Each prediction is a tuple contains the semantic type and its confidence score.
    - **save(set\_name)**: store attribute data into Mongodb and ElasticSearch with its data set
      * Params:
        + set\_name: name of data set contains the attribute
      * Output: none
    - **update(set\_name**): update attribute data in Mongodb and ElasticSearch with its data set
      * Params:
        + set\_name: name of data set contains the attribute
      * Output: none
    - **delete(set\_name**): delete attribute data in Mongodb and ElasticSearch with its data set
      * Params:
        + set\_name: name of data set contains the attribute
      * Output: none
    - **prepare\_data()**: preprocessing attribute data before indexing
      * Output: none

## Module machine\_learning

* **Classifier**:
  + Class contains details of classifiers that use to determine whether 2 attributes have the same semantic types(core approach used in semantic labeling)
  + Methods:
    - **train(data\_set\_dict)**: train the classifier based on generated data
      * Params:
        + data\_set\_dict: dictionary of training data sets and the corresponding number of labeled sources for each set
      * Output: none
    - **predict(feature\_vectors)**: give Yes/No result on whether the 2 attributes have the same semantic types using feature vectors
      * Params:
        + feature\_vectors: list of dictionaries using as feature vectors. Each dictionary contains names and values of different features.
      * Output: dataframe where columns are features and rows are semantic types. One new column of confidence score for semantic types is added.
    - **save(file\_path)**: save the classifier model into file
      * Params:
        + file\_path: path to the save file
      * Output: none
    - **load(file\_path)**: load the classifier model from file
      * Params:
        + file\_path: path to the load file
      * Output: none
    - **debug():** print additional debug information for the classifier
      * Output: none
* **RandomForest**:
  + Subclass of Classifier that uses RandomForests as classification method
* **LogisticRegression**:
  + Subclass of Classifier that uses Logistic Regression as classification method

## Module search\_engine

* **Indexer**:
  + Class which handles Indexing attribute data into Elasticsearch and Mongodb
  + Methods:
    - **check\_set\_indexed(set\_name)**: check whether a data set is already indexed
      * Params:
        + set\_name: name of data set
      * Output: none
    - **store\_attribute(attr\_data, source\_name, set\_name)**: store data of an attribute in Elasticsearch and Mongodb
      * Params:
        + attr\_data: data contained in attribute in json format using to\_json() method
        + source\_name: name of data source
        + set\_name: name of data set
      * Output: none
    - **delete\_attribute(attr\_name, source\_name, set\_name)**: delete data of an attribute from Elasticsearch and Mongodb
      * Params:
        + attr\_name: name of attribute
        + source\_name: name of data source
        + set\_name: name of data set
      * Output: none
* **Searcher**:
  + Class which handles querying attribute data from Elasticsearch and Mongodb
  + Methods:
    - **search\_attribute\_data(set\_name, source\_name\_list)**: query data from attributes in a list of sources contained in a data set
      * Params:
        + set\_name: name of data set
        + source\_name\_list: list of data sources’ name
      * Output: list of dictionaries that contains data from attributes in source in source\_name\_list. Each dictionary contains data of an attribute
    - **search\_similar\_text\_data(set\_name, text, source\_name\_list)**: search similar text documents in Elasticsearch with specific list of source and data name.
      * Params:
        + set\_name: name of data set
        + text: query text to be searched
        + source\_name\_list: list of data sources’ name
      * Output:
        + list of dictionaries that contains attributes with similarity text to the query text. Each dictionary contains the actual text field and the similarity score computed by TF-IDF

## Other methods

* **compute\_feature\_vectors(labeled\_data\_dict, test\_data\_dict, text\_search\_dict):** compute pairwise feature vectors between labeling attribute with all attributes from labeled sources for semantic types prediction.
  + Params:
    - labeled\_data\_dict: dictionary that contains attributes’ data from labeled sources (Can be retrieved using Searcher.search\_attribute\_data() method)
    - test\_data\_dict: dictionary that contains data from labeling attribute (using to\_json() method of class Attribute)
    - text\_search\_dict: dictionary that contains result from text similarity search in Elasticsearch (Can be retrieved Searcher.search\_similar\_text\_data() method)
  + Output: dictionary contains
* **ks\_distribution\_sim(list1, list2, num\_fraction1, num\_fraction2)**: compute Kolmogorov–Smirnov test between list of numeric values from two attributes
  + Params:
    - list1, list2: lists of numeric values
    - num\_fraction1, num\_fraction2: fraction of numeric values from original attributes (for value adjustment)
  + Output: test result in probability
* **mw\_histogram\_sim(list1, list2, num\_fraction1, num\_fraction2)**: compute Kolmogorov–Smirnov test between list of numeric values from two attributes
  + Params:
    - list1, list2: lists of numeric values
    - num\_fraction1, num\_fraction2: fraction of numeric values from original attributes (for value adjustment)
  + Output: test result in probability
* **jaccard\_name\_sim(name1, name2, num\_fraction1, num\_fraction2)**: compute Jaccard similarity between attribute names based on tri-grams generated from these names.
  + Params:
    - list1, list2: names of attributes
    - num\_fraction1, num\_fraction2: fraction of numeric values from original attributes (for value adjustment)
  + Output: similarity result in probability
* **jaccard\_str\_sim(list1, list2, num\_fraction1, num\_fraction2)**: compute Jaccard similarity between list of textual values from 2 attributes.
  + Params:
    - list1, list2: lists of textual values
    - num\_fraction1, num\_fraction2: fraction of numeric values from original attributes (for value adjustment)
  + Output: similarity result in probability
* **jaccard\_num\_sim(list1, list2, num\_fraction1, num\_fraction2)**: compute numeric Jaccard similarity between list of numeric values from two attributes.
  + Params:
    - list1, list2: lists of numeric values
    - num\_fraction1, num\_fraction2: fraction of numeric values from original attributes (for value adjustment)
  + Output: similarity result in probability
* **jaccard\_similarity(x, y)**: compute Jaccard similarity between two sets of values
  + Params:
    - x, y: two sets of values
  + Output: similarity result in probability
* **split\_number\_text(str)**: split a string into two different parts: numeric and textual
  + Params:
    - str: string to be processed
  + Output: a tuple contains numeric and textual parts of the string
* **adjust\_result**(num\_fraction1, num\_fraction2, value): adjust results from similarity test based on numeric fractions of two attributes
  + Params:
    - num\_fraction1, num\_fraction2: fraction of numeric values from original attributes
    - value: original result from similarity test
  + Output: adjusted result