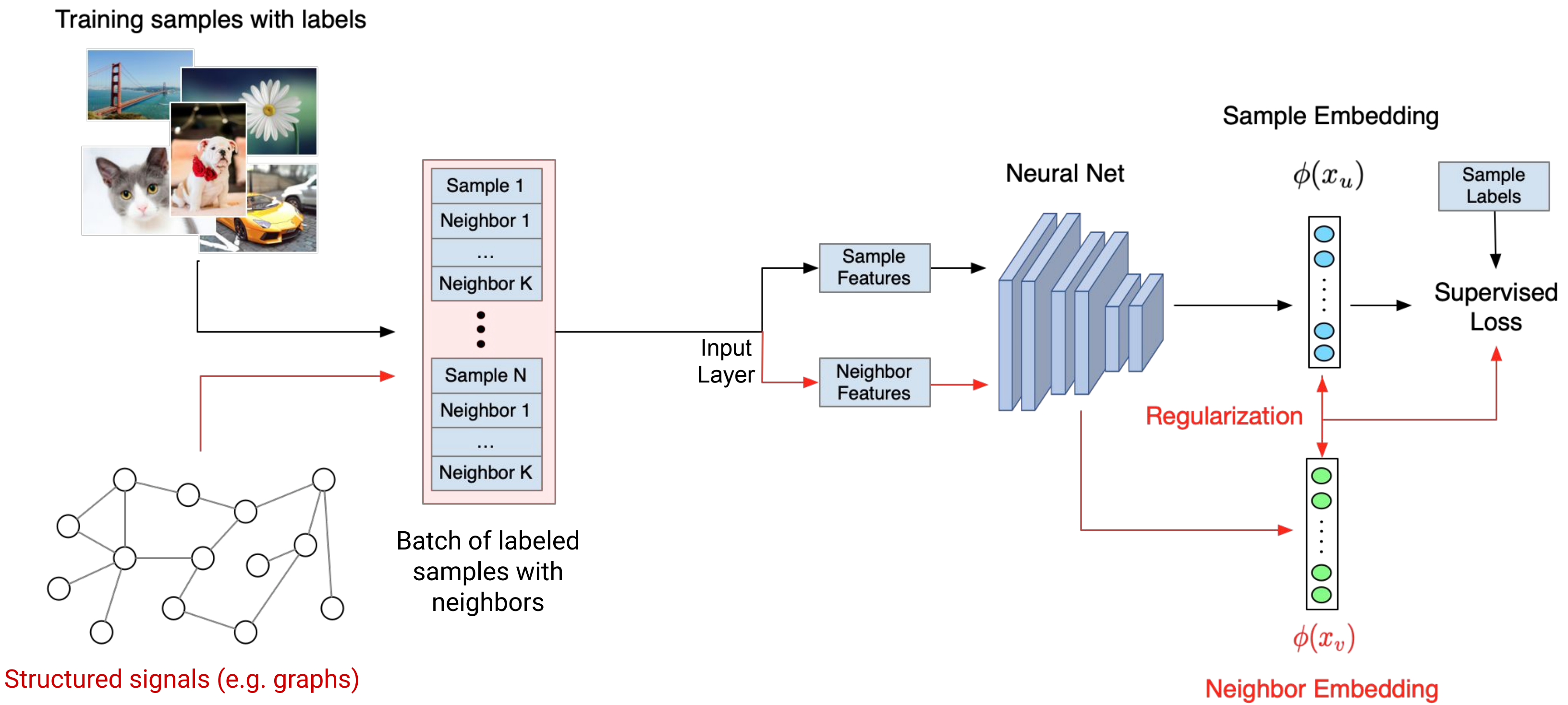
TensorFlow

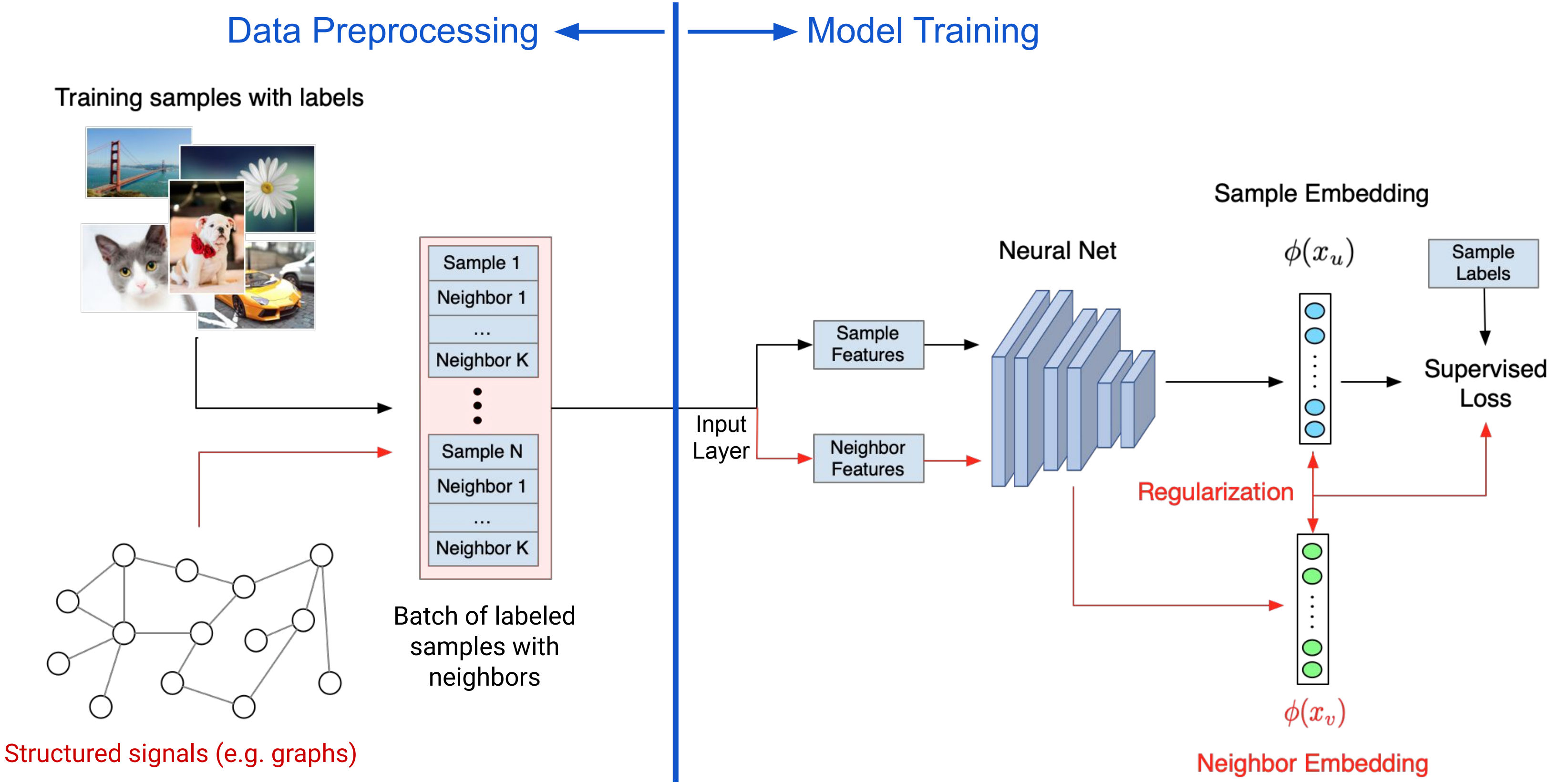
Data Preprocessing

Allan Heydon

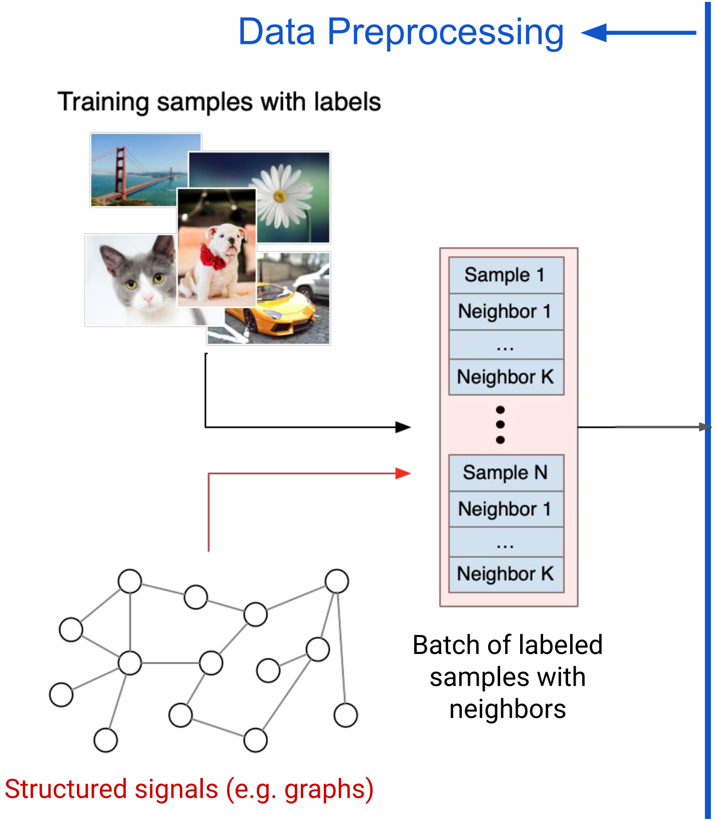
Data Preprocessing



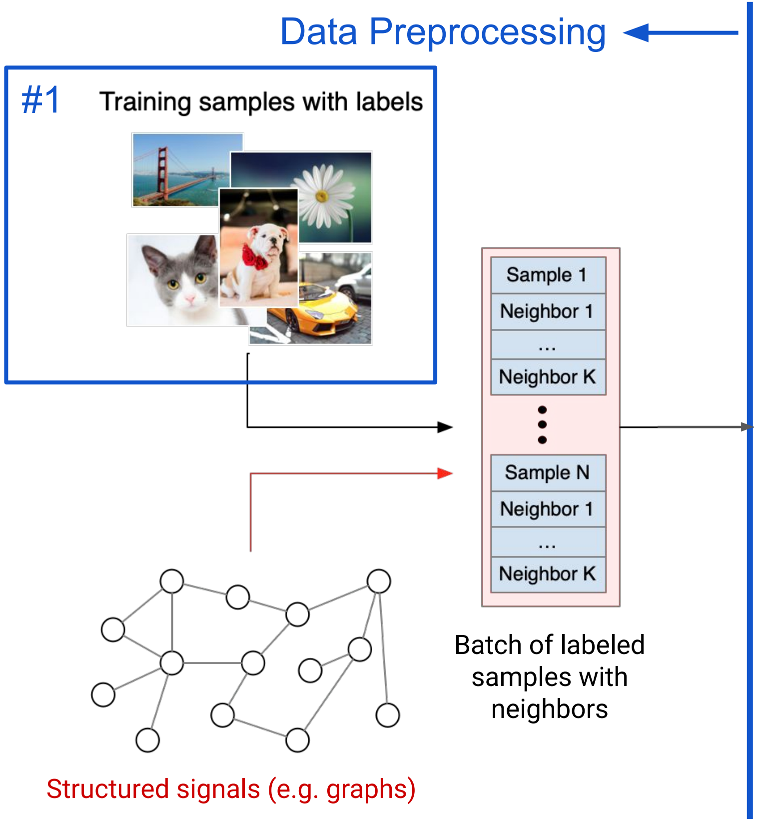
Data Preprocessing



Data Preprocessing

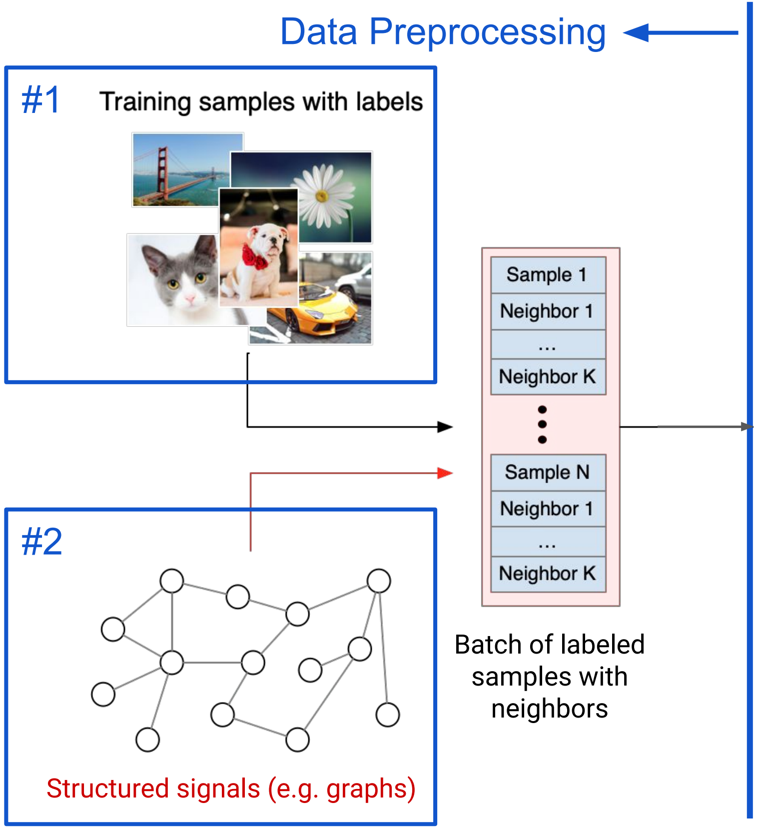


Data Preprocessing



1. Training examples
    - Most of these may be unlabeled, but a subset need to be labeled for training.

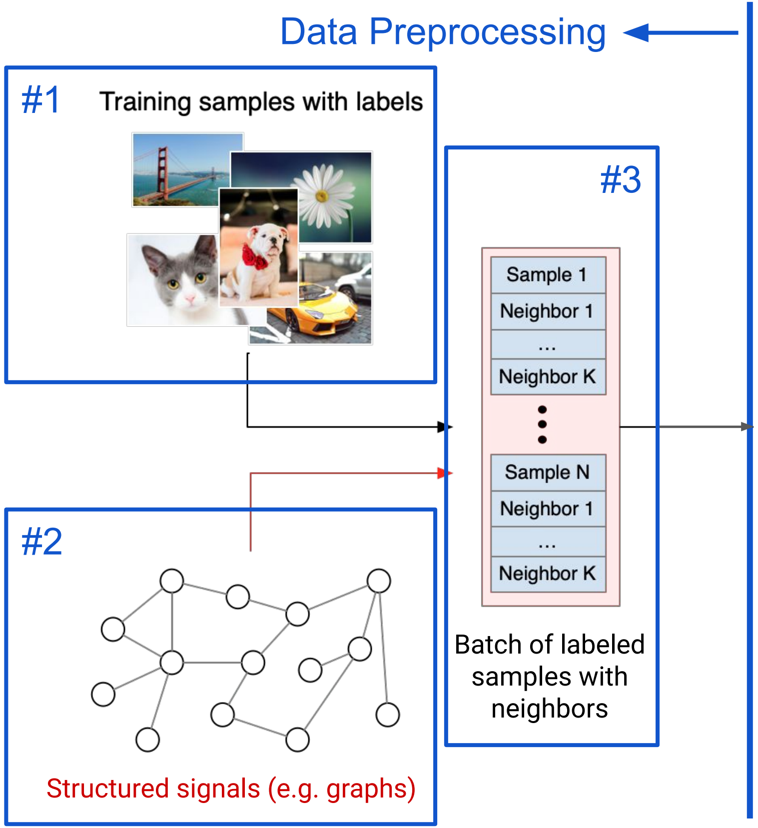
Data Preprocessing



1. Training examples
   - Most of these may be unlabeled, but a subset need to be labeled for training.

2. Similarity graph
   - Graph nodes denote examples.
   - Weighted graph edges represent degree of similarity between pairs.
   - Two forms:
     - Natural
     - Constructed

Data Preprocessing



1. Training examples
   - Most of these may be unlabeled, but a subset need to be labeled for training.

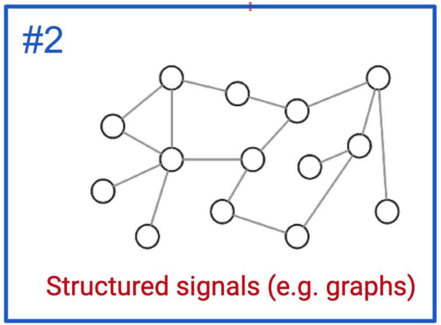
2. Similarity graph
   - Graph nodes denote examples.
   - Weighted graph edges represent degree of similarity between pairs.
   - Two forms:
     - Organic
     - Constructed

3. Combine labeled examples with their neighbors in the similarity graph

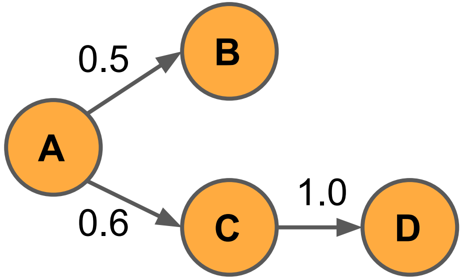
Training Examples

- Represented by `tensorflow.Example` protocol buffers.
- Stored in [TFRecord files](), which contain a sequence of Examples.
- Each example must define a string-valued feature containing its globally unique ID.
- Labeled examples are distinguished by defining a single-valued feature containing the label value.
- The NSL toolset requires that labeled and unlabeled examples are stored in separate TFRecord files.

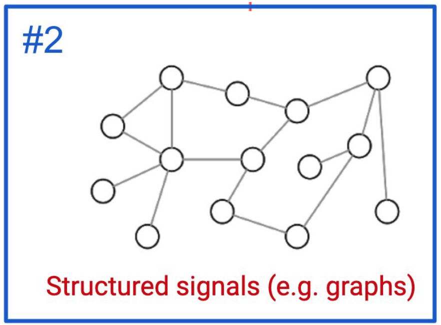
Similarity Graphs

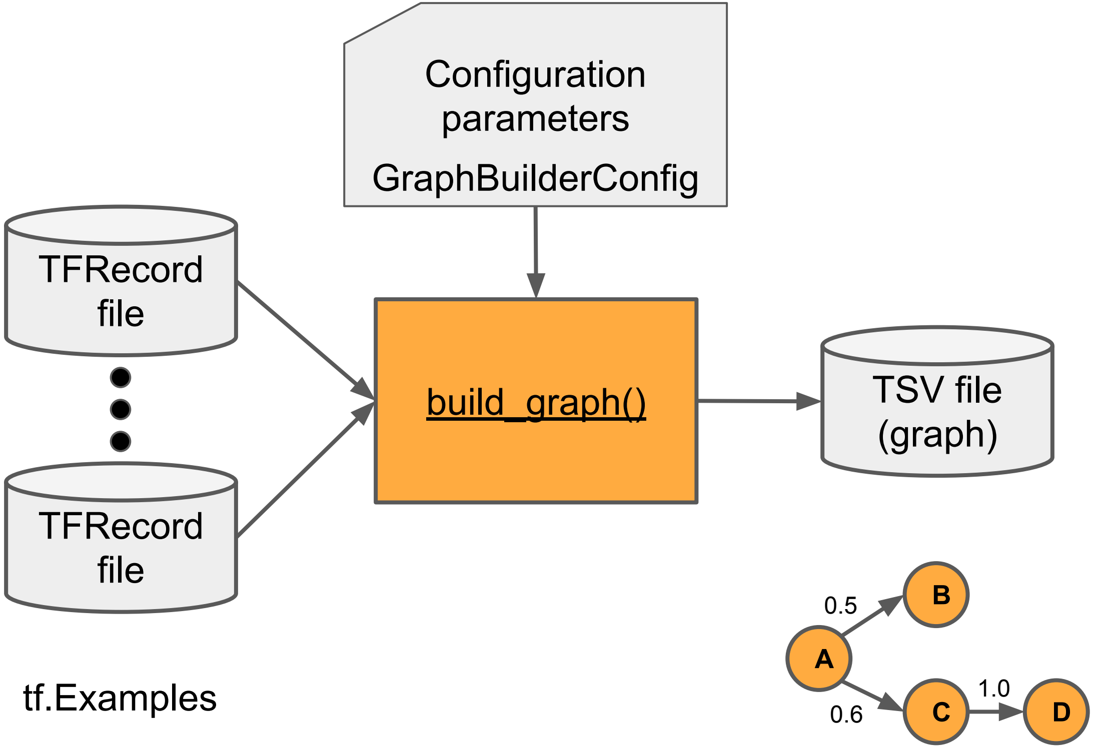


- Represented using TSV files, each with 3 columns:
  - source_id <TAB> target_id [<TAB> edge_weight]
- I/O helper functions:
  - nsl.tools.read_tsv_graph(filename): graph
  - nsl.tools.write_tsv_graph(filename, graph): None
- Python graph representation:
  - dict: source_id → (dict: target_id → edge_weight)
  - Example: { "A": { "B": 0.5, "C": 0.6 }, "C": { "D": 1.0 } }
- Graph utils:
  - nsl.tools.add_edge(graph, edge): Boolean
  - nsl.tools.add_undirected_edges(graph): None

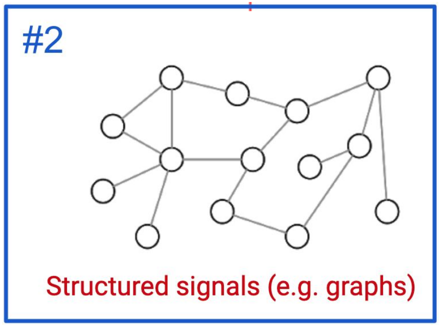


Graph Building



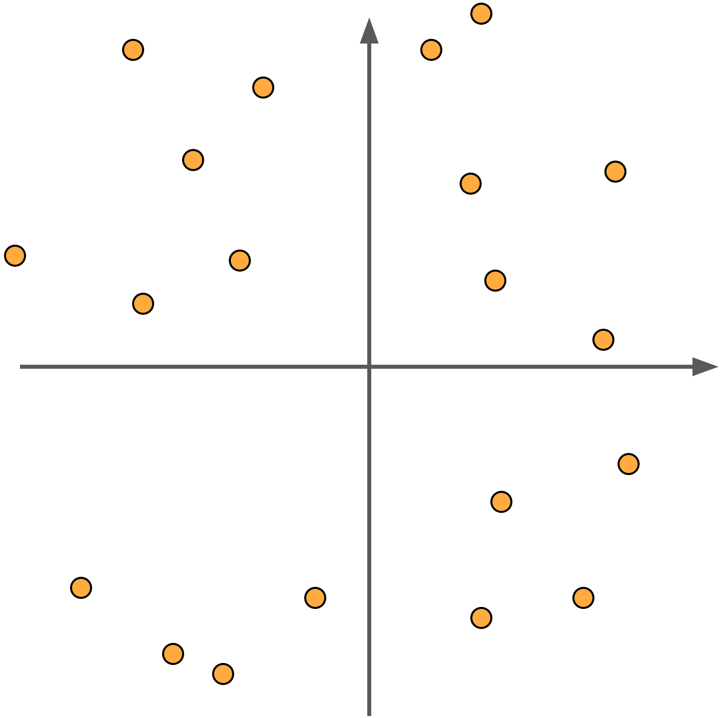


Graph Building algorithm

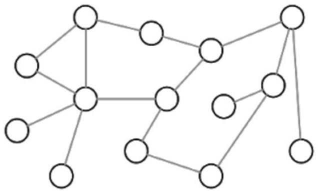


- Requires every input tf.Example to have 2 features:
  - id -- singleton `bytes_list` identifying example
  - embedding -- dense `float_list` containing an embedding
- All embeddings must have the same dimension d
- Compares all pairs of inputs for similarity
- Similarity computation:
  - edge_weight = cosine_similarity(embedding1, embedding2)
  - This is the cosine of the angle between the two embeddings when each is thought of as a vector in R^d.
- Problem: # of pairs is $O(n^2)$.

Locality-Sensitive Hashing (LSH)

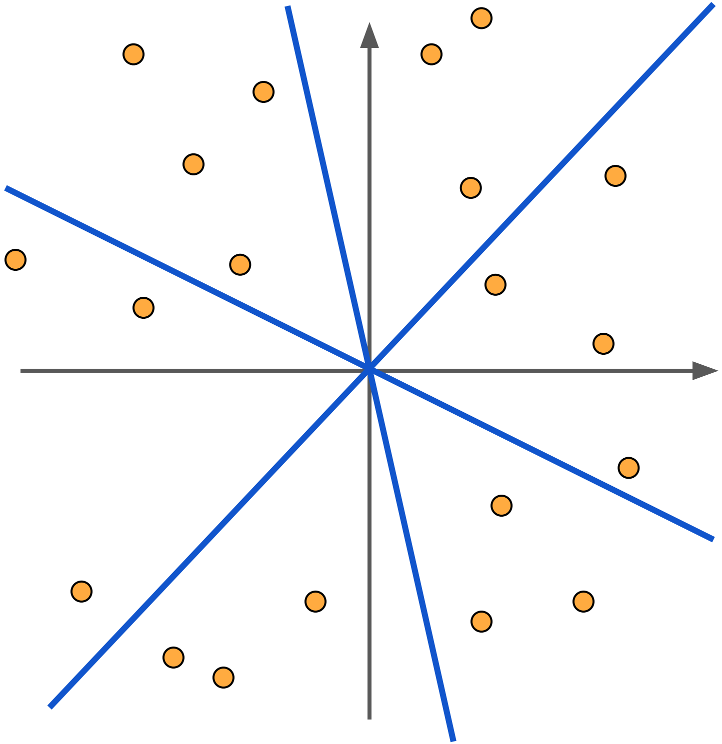


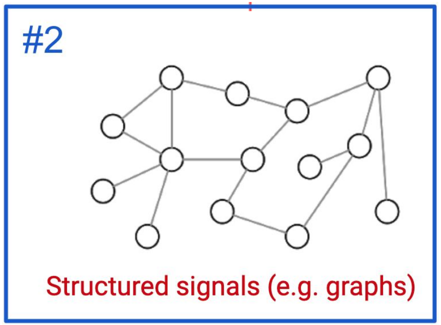
#2



Structured signals (e.g. graphs)

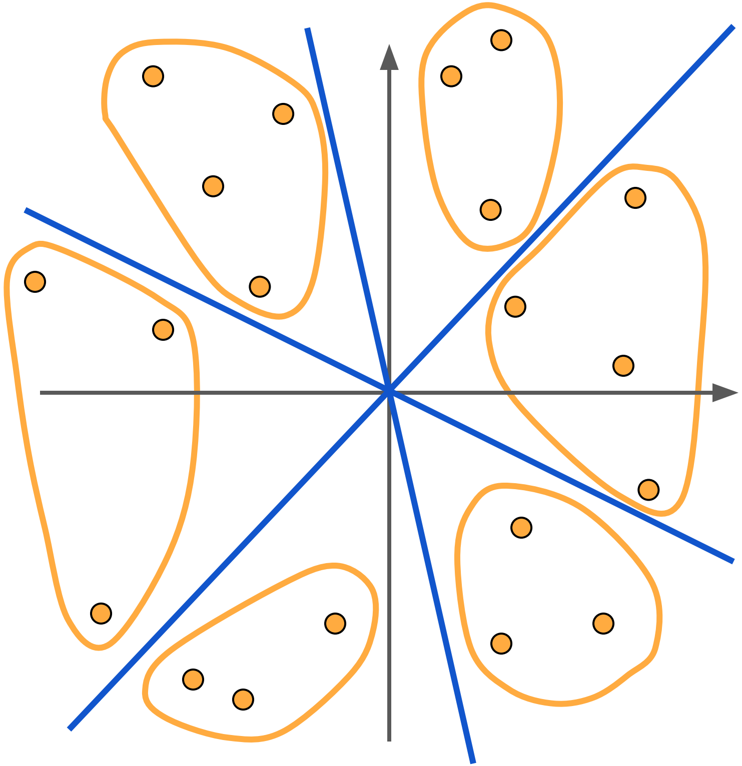
Locality-Sensitive Hashing (LSH)



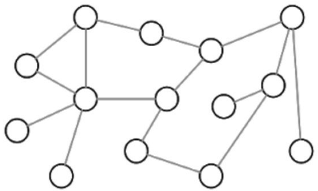


- Randomly split points into *LSH buckets*

Locality-Sensitive Hashing (LSH)



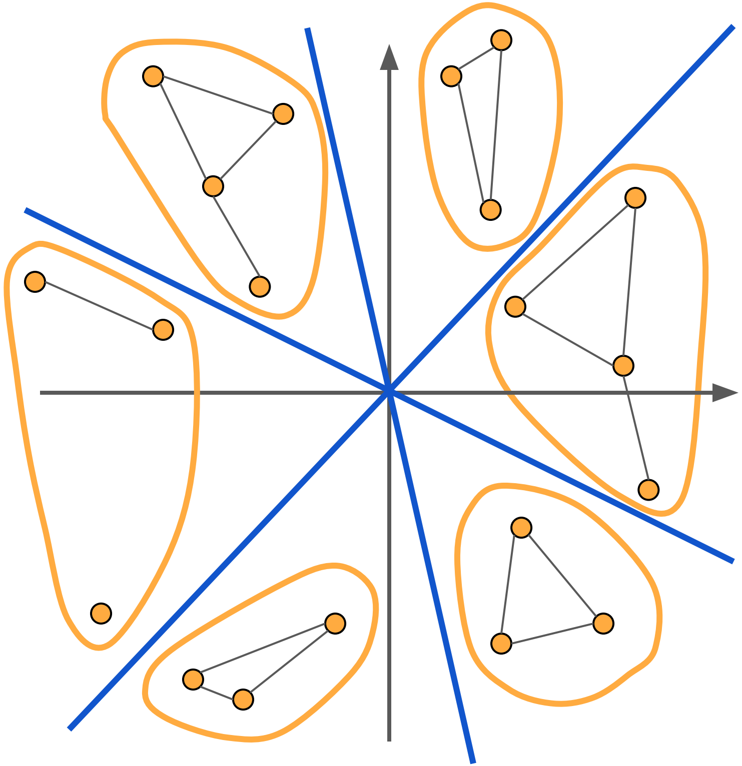
#2

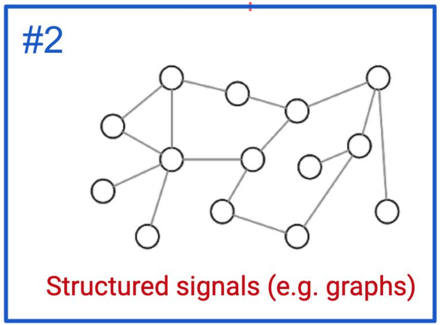


Structured signals (e.g. graphs)

- Randomly split points into *LSH buckets*
- Compare all point pairs in each bucket

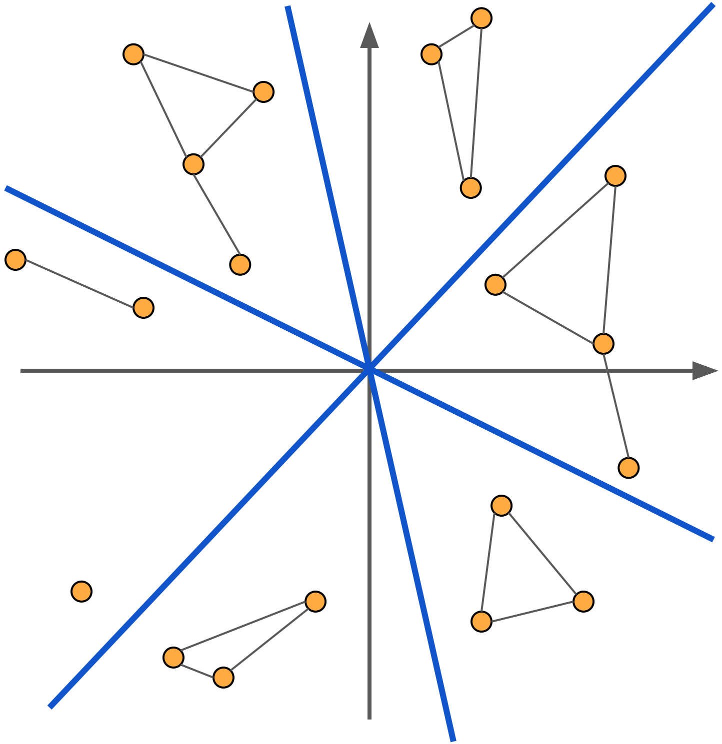
Locality-Sensitive Hashing (LSH)

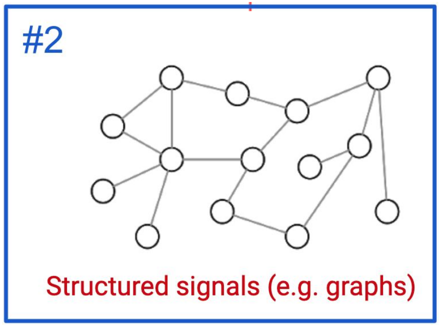




- Randomly split points into *LSH buckets*
- Compare all point pairs in each bucket
- Construct intra-bucket edges

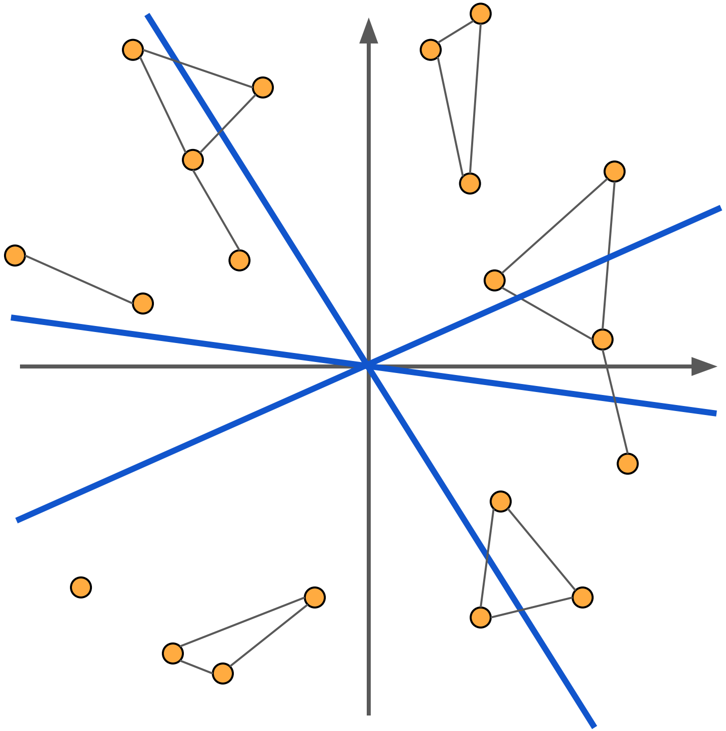
Locality-Sensitive Hashing (LSH)

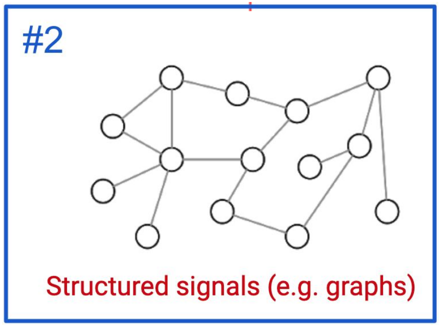




- Randomly split points into *LSH buckets*
- Compare all point pairs in each bucket
- Construct intra-bucket edges
- Note: No edges across buckets!

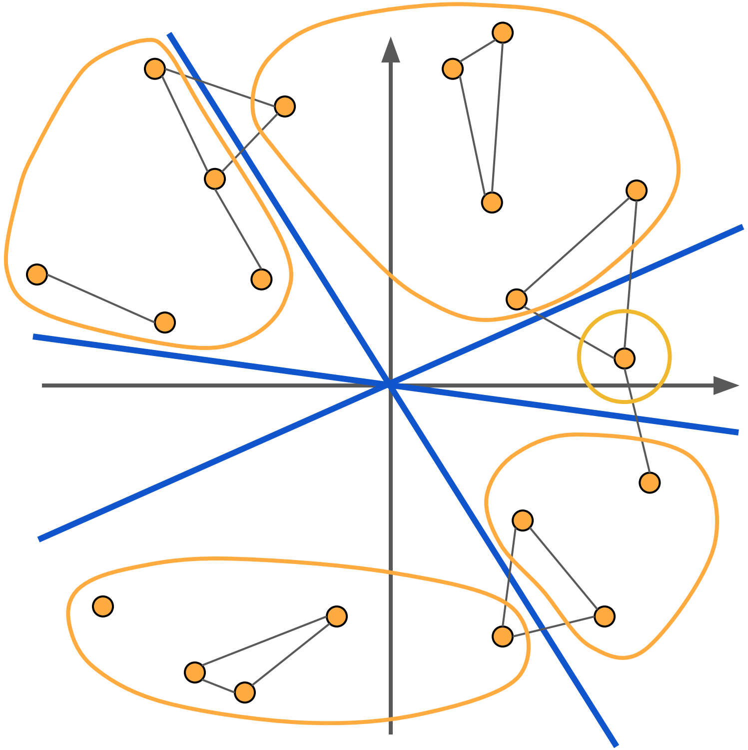
Locality-Sensitive Hashing (LSH)

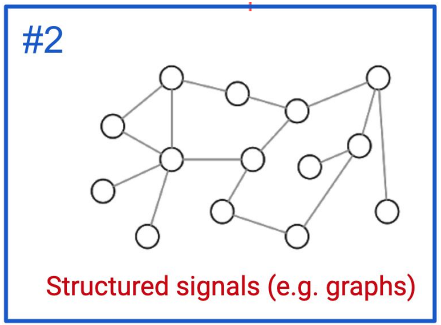




- Randomly split points into *LSH buckets*
- Compare all point pairs in each bucket
- Construct intra-bucket edges
- Note: No edges across buckets!
- Repeat this process multiple times

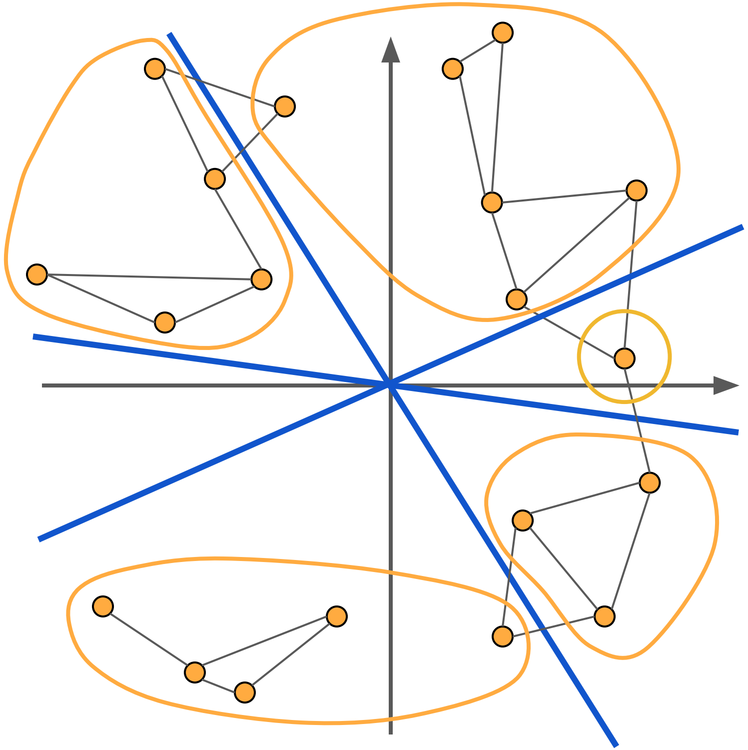
Locality-Sensitive Hashing (LSH)

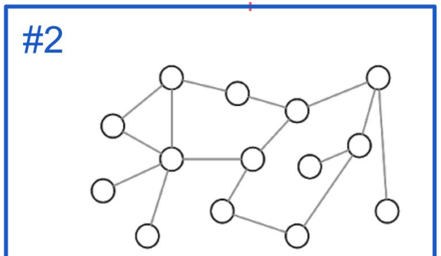




- Randomly split points into *LSH buckets*
- Compare all point pairs in each bucket
- Construct intra-bucket edges
- Note: No edges across buckets!
- Repeat this process multiple times

Locality-Sensitive Hashing (LSH)

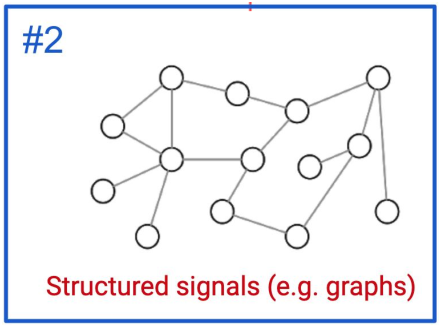




Structured signals (e.g. graphs)

- Randomly split points into *LSH buckets*
- Compare all point pairs in each bucket
- Construct intra-bucket edges
- Note: No edges across buckets!
- Repeat this process multiple times
  - Each *round* of randomized LSH bucketing finds new edges

nsl.configs.GraphBuilderConfig



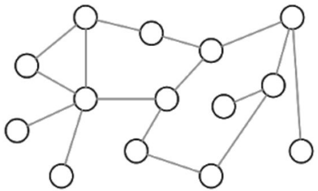
- id_feature_name: string
  - Name of the feature containing the example ID
- embedding_feature_name: string
  - Name of the feature containing the (dense) embedding
- similarity_threshold: float
  - Lower bound on cosine similarity for edge to be created
- lsh_rounds: int
  - Number of LSH bucketing rounds performed
- lsh_splits: int
  - Number of random partitions on each LSH round
  - ⇒ Maximum of 2^{lsh_splits} LSH buckets per round
- random_seed: int

LSH Splits vs. Rounds

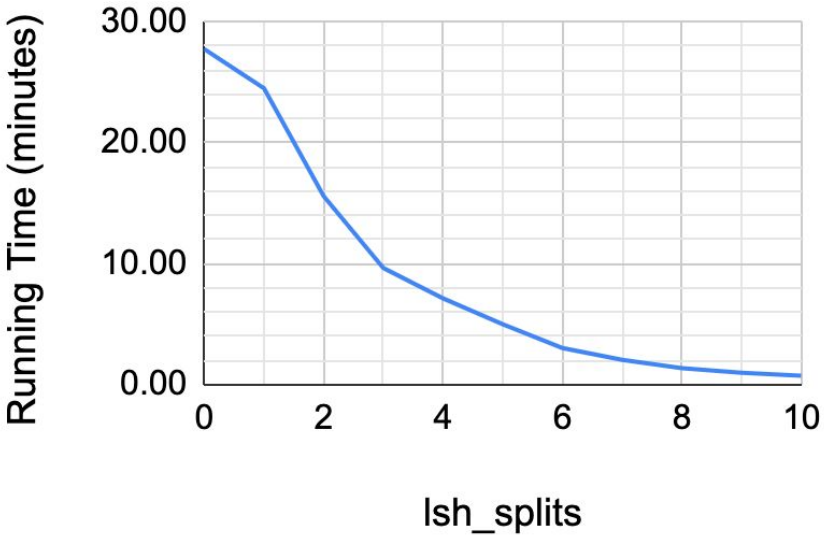
- 50K samples
- 100-D embedding vectors
- 0.9 similarity threshold

- Goal: Achieve 99.7+% recall of all edges resulting from lsh_splits = 0.

#2



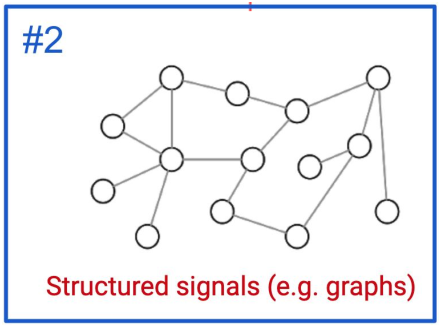
Structured signals (e.g. graphs)

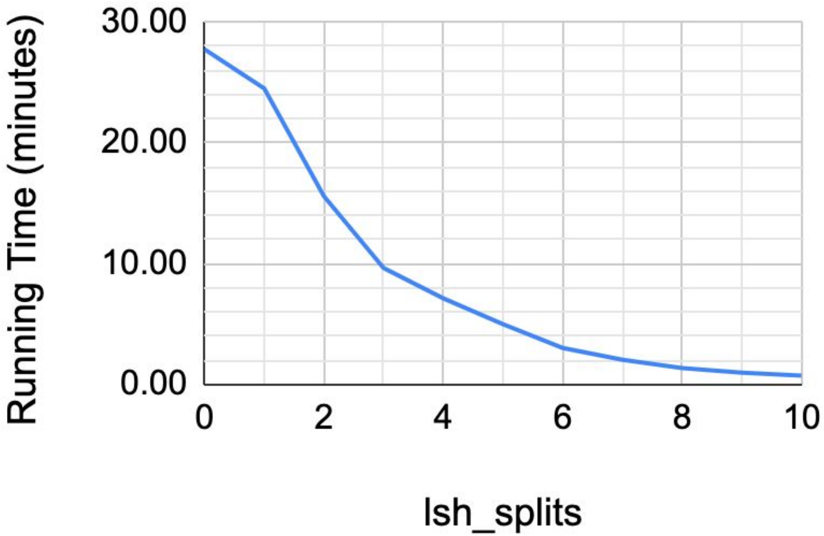


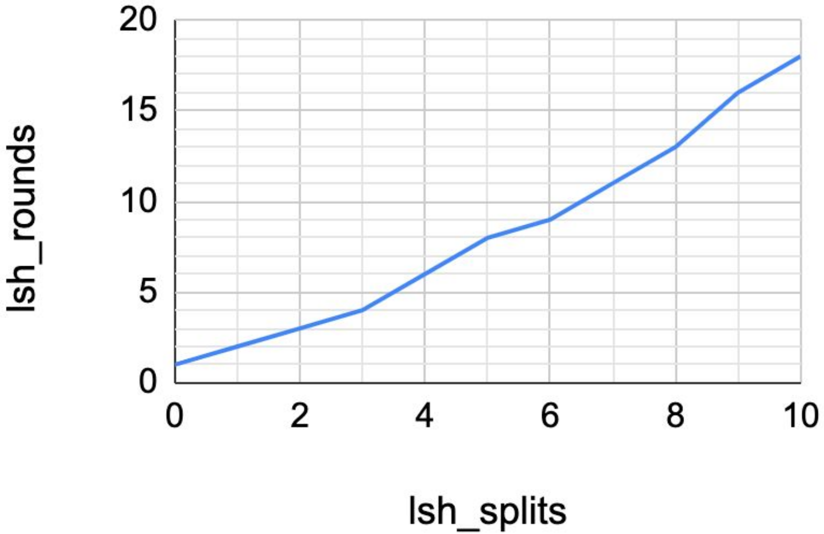
LSH Splits vs. Rounds

- 50K vertices
- 100-D embedding vectors
- 0.9 similarity threshold

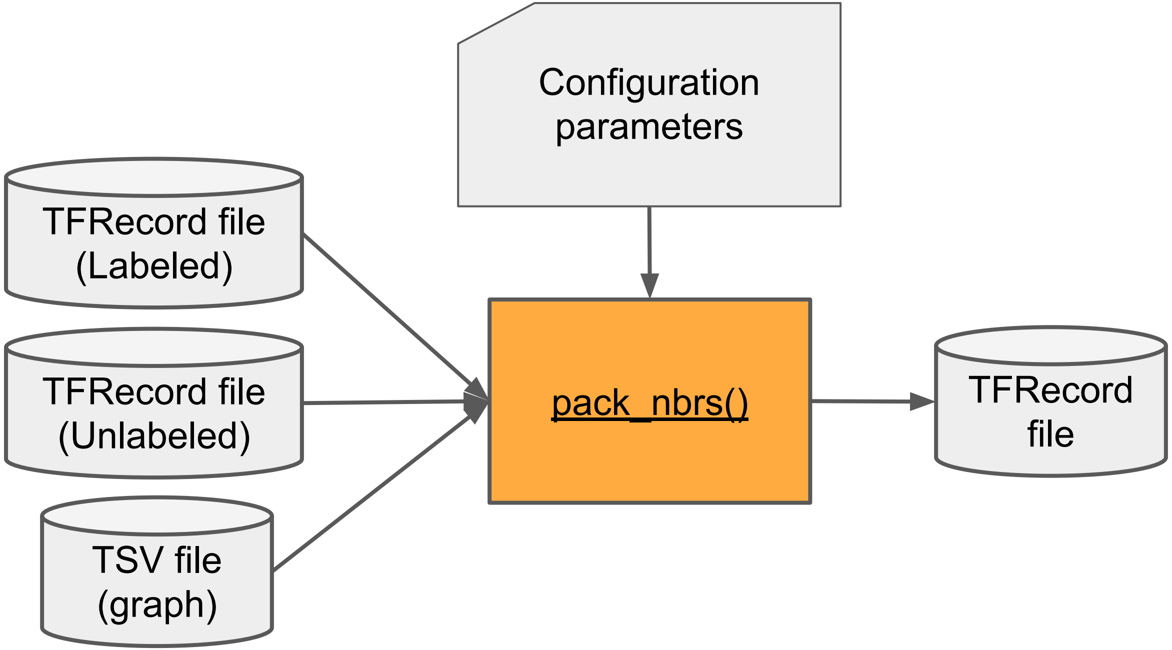
- Goal: Achieve 99.7+% recall of all edges resulting from lsh_splits = 0.

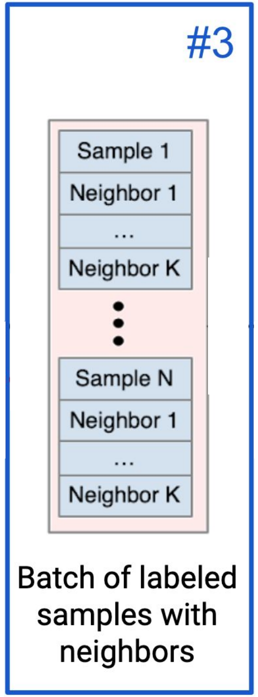






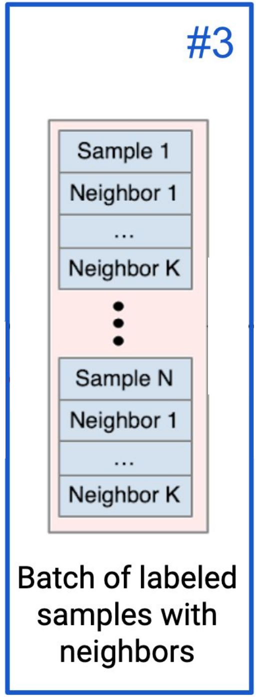
Packing Neighbors together





nsl.tools.pack_nbrs()

- labeled_examples_path: string
  - Pathname of TFRecord file containing labeled Examples
- unlabeled_example_path: string
  - Pathname of TFRecord file containing unlabeled Examples
- graph_path: string
  - Pathname of TSV file containing graph edges
- output_training_data_path: string
  - Pathname of TFRecord file where merged training Examples are written
- add_undirected_edges: boolean (default=False)
  - If True, all input graph edges are made symmetric
- max_nbrs: int (default=None)
  - Max # of neighbors to pack with each labeled Example
- id_feature_name: string (default="id")
  - Name of the feature containing the example ID



Running tools as binaries

Both data preprocessing tools can be run as binaries.

Graph Builder:

$ **`python -m neural_structured_learning.tools.build_graph`** \
[*flags*] *embedding_file.tfr... output_graph.tsv*

Pack Neighbors:

$ **`python -m neural_structured_learning.tools.pack_nbrs`** \
[*flags*] *labeled.tfr unlabeled.tfr graph.tsv output.tfr*