Formally Verified Implementation of the *K*-Nearest Neighbors Classification Algorithm

Bernny Velasquez, Jessica Herring, and **Nadeem Abdul Hamid**
Berry College, Georgia, USA

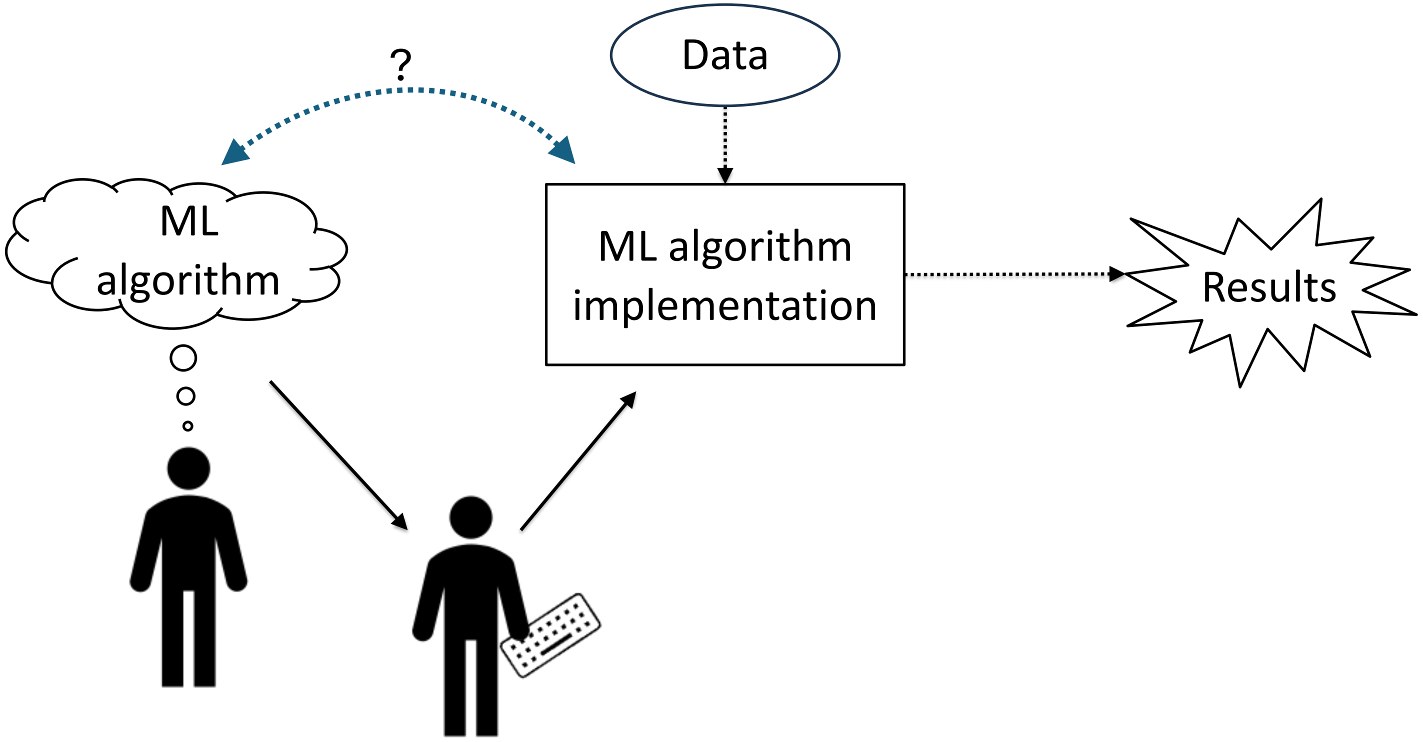
BERRY COLLEGE

TRUST LABORATORY
Transformative Reliability Underlying Software Technology

SBMF 2024, Vitória, Brazil

Implementing Machine Learning Algorithms

- Gap between the mathematical model and mechanics of implementation

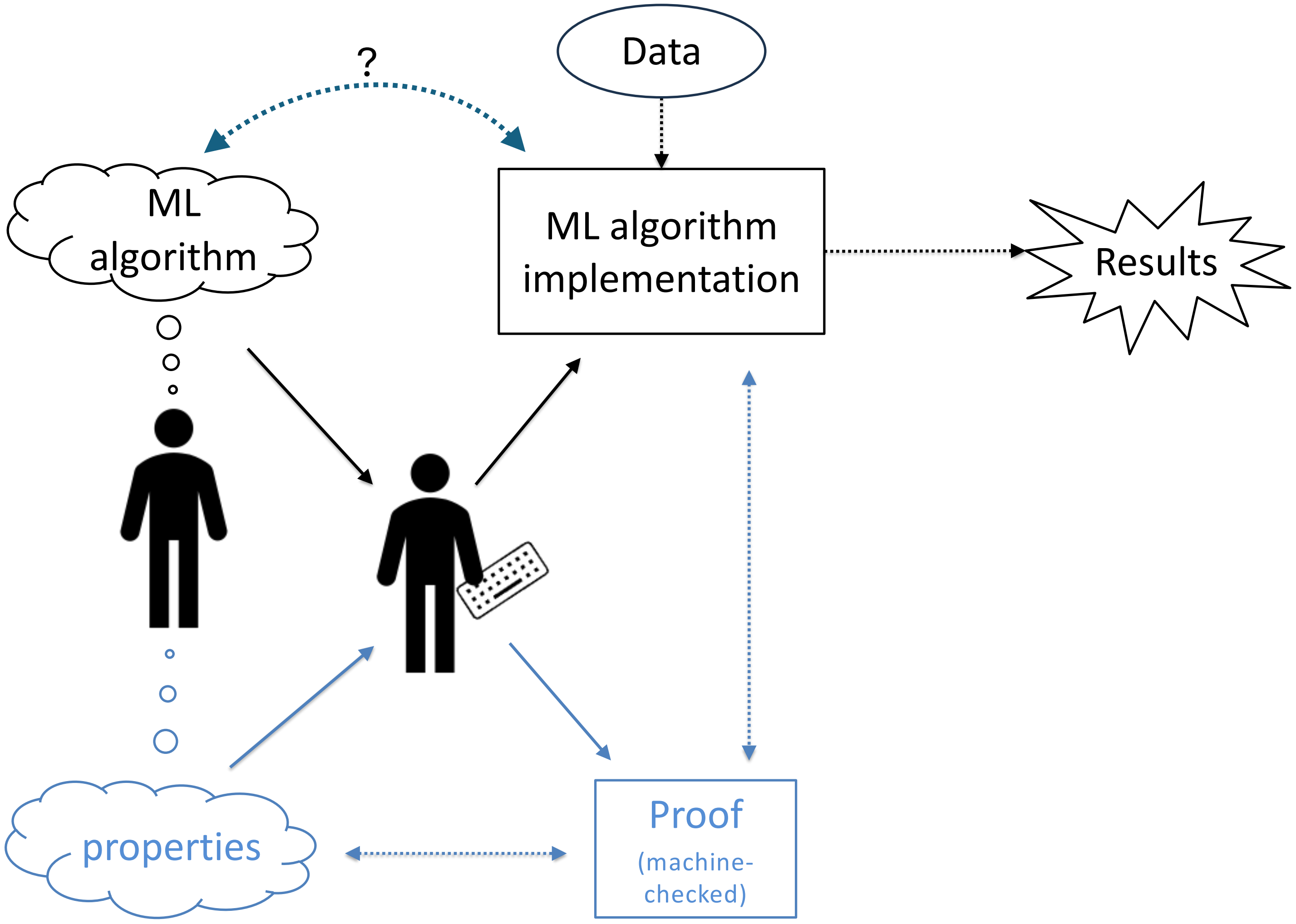


Implementing Machine Learning Algorithms

- Gap between the mathematical model and mechanics of implementation

- (Big Picture) Context for this work:

  Development of verified implementations of ML systems



Focus: Classification

- Does the program code for an ML algorithm faithfully implement the mathematical model/description?

- Focus on the mechanics of the algorithm, not meta-theoretical or application-specific properties

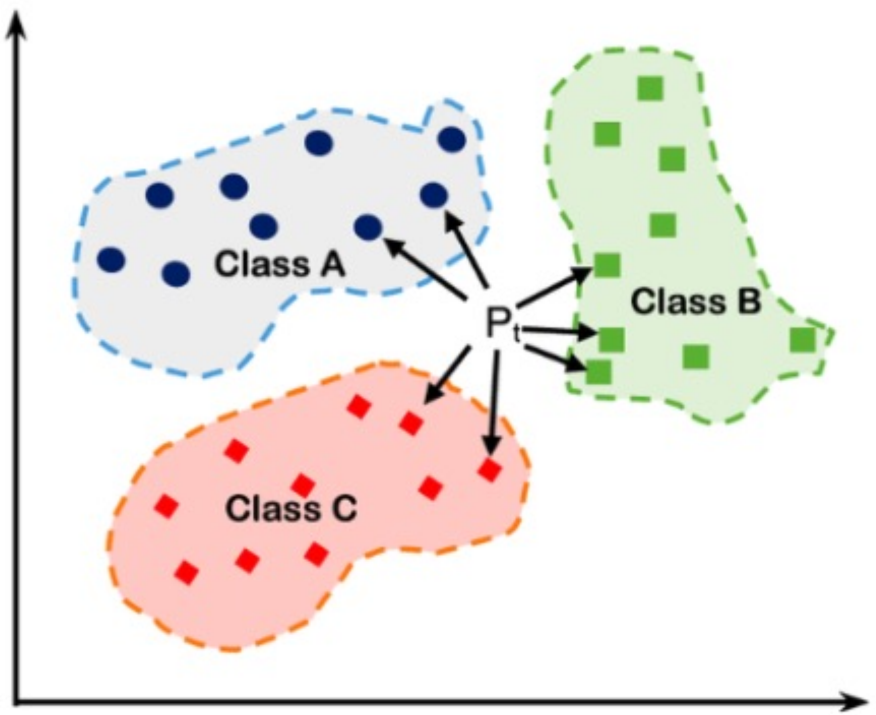


Image: medium.com

KNN (*K*-Nearest-Neighbors) Classification

- One of the oldest, well-known, widely used classification algorithms
  - Assigns class labels to observations based on previously seen data
  - Can also be used for regression
- Applied in a wide variety of domains (not just ML)
- Popularity can be attributed to its simplicity, ease of implementation, and high accuracy rates
- Although, there are known limitations of KNN search
  - (curse of dimensionality; scaling to large data sets)

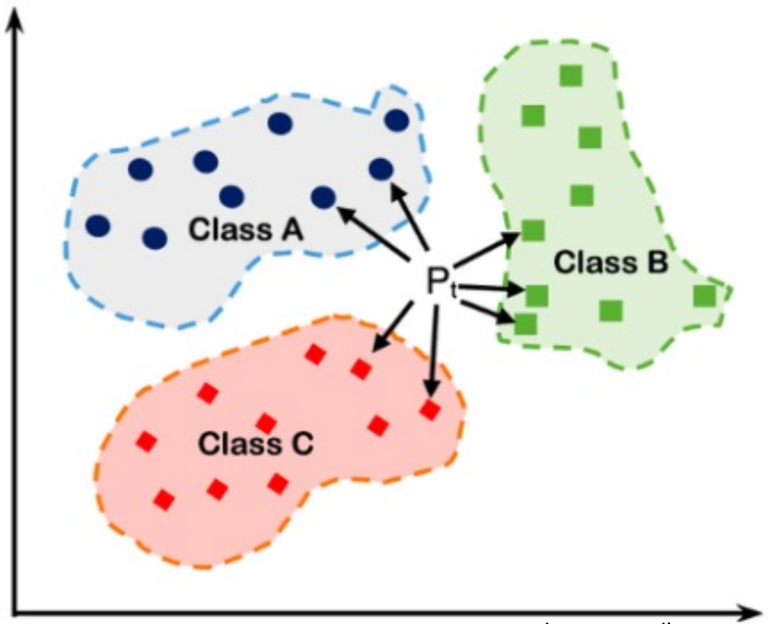


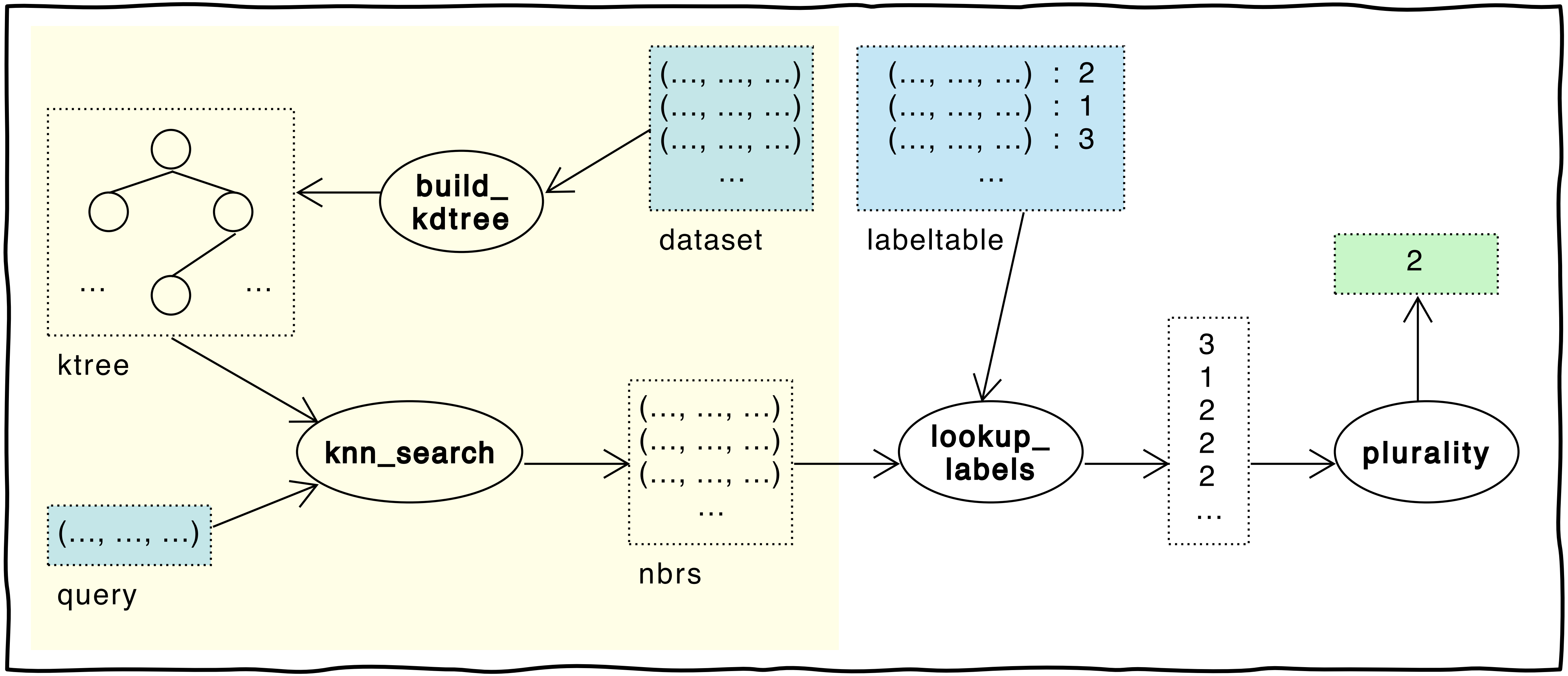
Image: medium.com

Contributions

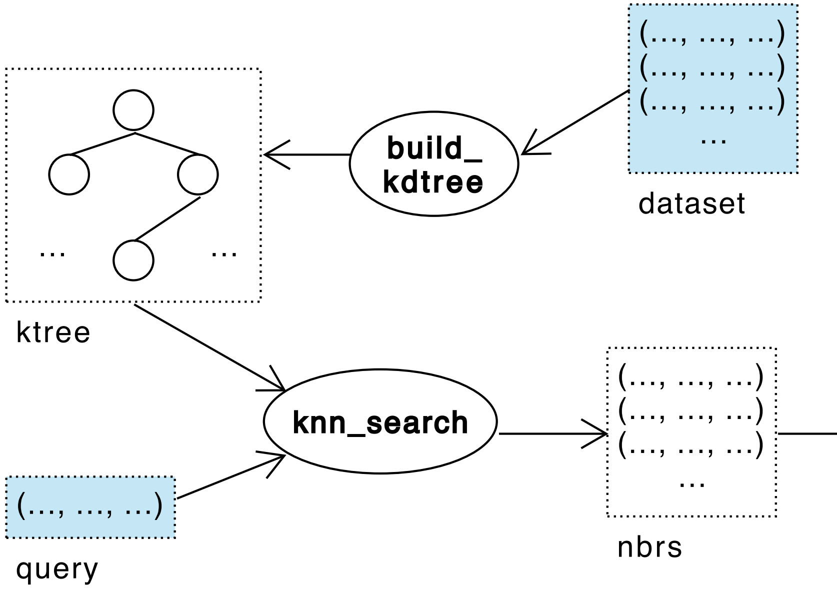
Mechanically verified implementation of a
KNN classification algorithm in the **Coq proof assistant**.

- ⭐ Integrating previously-verified data structures/algorithms
  - ***k*-d trees** and **AVL tree-based Map**
  - Generalized ***K*-nearest-neighbors search**
  - ⭐ **Plurality** algorithm
- ⭐ **Formal specification** and **verification**

Classifier Implementation



Prior Work: Verified KNN Search [Hamid, SAC 2024]



Construct a k-d tree structure from a list of **k**-dimensional **data points**

```
build_kdtree (k:nat) (data:list datapt) : kdtree
```

Produce a list of the **K** nearest data points (based on a given **distance metric**) to the **query** point among all the points in the **tree**.

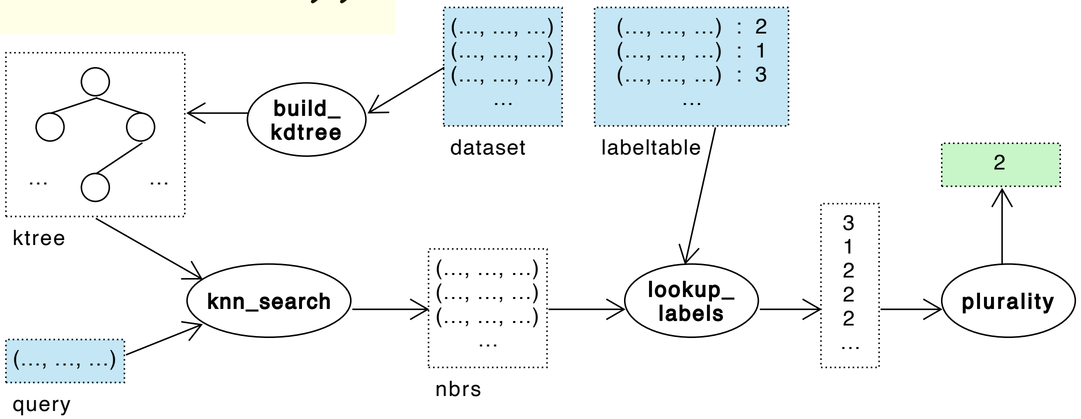
```
knn_search
(D:datapt -> datapt -> nat) (K:nat) (k:nat) (tree:kdtree) (query:datapt) : list datapt
```

Prior Work: Verified KNN Search [Hamid, SAC 2024]

```
Theorem knn_search_build_kdtree_correct :
 forall dist_metric, dist_metric_wf dist_metric ->
 forall (K:nat) (k : nat) (data : list datapt),    // Preconditions:
   0 < K ->                                       // at least one neighbor sought
   0 < length data ->                             // data is non-empty
   0 < k ->                                       // dimension space is non-empty
   (forall v' : datapt,                           // all data points well-formed (k-dim)
         In v' data -> length v' = k) ->
   forall tree query result,
      tree = (build_kdtree k data) ->             // If: the k-d tree built from data
      knn_search K k tree query = result ->       // produces result for a query point,
      exists leftover,                            // Then:
         length result = min K (length data)      // the result is length (at most) K,
         /\ Permutation data (result ++ leftover) // and is a sub-list of data,
         /\ all_in_leb (dist_metric query) result leftover. // and everything in
                 // result is closer in distance to the query than all the leftover part of data.
```

Classify algorithm

```
Definition classify (D : datapt -> datapt -> nat) (* dist metric *)
                    (K : nat)  (* number of neighbors *)
                    (k : nat)  (* dimensions of all points *)
                    (dataset : (list datapt))
                    (labeltable : LabelTable)
                    (query : datapt)
                    : option nat :=
    let ktree := (build_kdtree k dataset) in
    let nbrs := knn_search D K k ktree query in
        fst (plurality (lookup_labels labeltable nbrs)).
```



Computing Plurality

- Given a list of values, determine the most frequently occurring
- Produce a pair of a *potential* maximum frequency **value** and the maximum frequency **count** of any value in the given list
  - In case of a tie, produce **None** as the maximum frequency value
- To compute `plurality (v :: tail)` ,
  consider `plurality tail` and `cv = 1 + count v tail`
  - Case `(None, c)` and `c < cv` ➔ `v` is the new plurality value
  - Case `(None, c)` and `c >= cv` ➔ retain `(None, c)`
  - Case `(Some x, c)` and `c = cv` ➔ tie, so `(None, c)`
  - Case `(Some x, c)` and `c < cv` ➔ `v` is the new plurality value
  - Case `(Some x, c)` and `c > cv` ➔ retain `x`

Plurality Implementation

```
Function plurality (vals : list nat) : option nat * nat :=
  match vals with
  | nil    => (None, 0)
  | h :: t => match (plurality t) with
              | (None, c) => let c' := (1 + count t h) in
                  if c <? c' then (Some h, c') else (None, c)
              | (Some x, c) => let c' := (1 + count t h) in
                  if c =? c'      then (None, c)
                  else if c <? c' then (Some h, c')
                  else                 (Some x, c)
              end
  end.
```

Specification

```
Theorem classify_correct_some :
    forall dist_metric, dist_metric_wf dist_metric ->
    forall K k data labels query c,
    0 < K -> 0 < k ->
    length data >= K ->
    (forall d : datapt, List.In d data -> length d = k) ->  (* all data of dimension k *)
    (forall d : datapt, List.In d data -> FMap.In d labels) -> (* every pt has a label *)

    classify dist_metric K k data labels query = Some c ->

    exists near far classes,
        Permutation data (near ++ far) /\            (* the `near` portion of the data *)
        length near = K /\                           (* are the K                      *)
        all_in_leb (dist_metric query) near far /\   (* nearest neighbors              *)
        ClassesOf labels near classes /\     (* `classes` are the labels of the `near`s *)
        IsPlurality c classes.         (* c is the plurality of all the `near` labels *)
```

Specification (part 2) - completeness

```
.
.
.

classify dist_metric K k data labels query = None ->

~ exists c,
     IsPlurality c (lookup_labels labels
                      (knn_search dist_metric K k (build_kdtree k data) query)).
```

Supporting Predicates

```
Inductive ClassesOf (LT : LabelTable) : list datapt -> list nat -> Prop :=
    | classesOf_nil : ClassesOf LT nil nil
    | classesOf_cons : forall k v ks vs,
         MapsTo k v LT -> ClassesOf LT ks vs -> ClassesOf LT (k::ks) (v::vs).

Definition IsPlurality (m:nat) (ns:list nat) : Prop :=
    List.In m ns /\ (forall m’, m’ = m \/ count ns m’ < count ns m).
```

Reflections

- Coq standard library
  - `Function` (functional induction)
  - (In)Consistency – count_occ vs. count / eq_dec vs eqb
- User-defined tactics
  - Permutations
- Specification correctness
  - Alternate completeness ↔
- Development cost

```
eq_dec : forall x y : A, {x = y}+{x <> y}
   eqb : nat -> nat -> bool                .
```

Future Directions

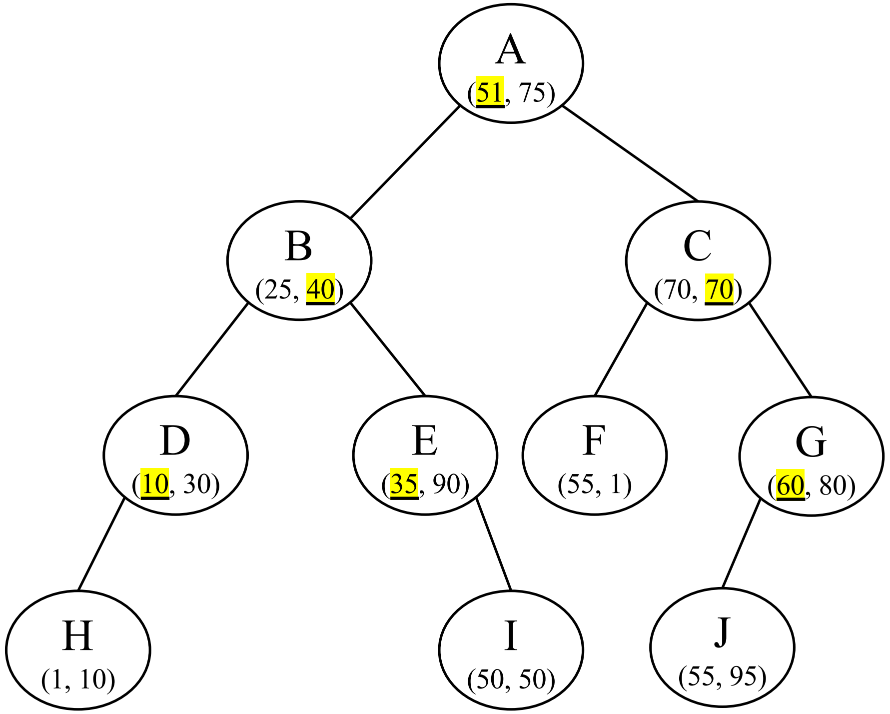
- KNN variations
  - Alternate tree data structures, dimension reduction, approximation, …
- Apply verification to “mainstream” language implementation
- Additional ML classification algorithms
  - Toolkit of specification approaches and verification techniques

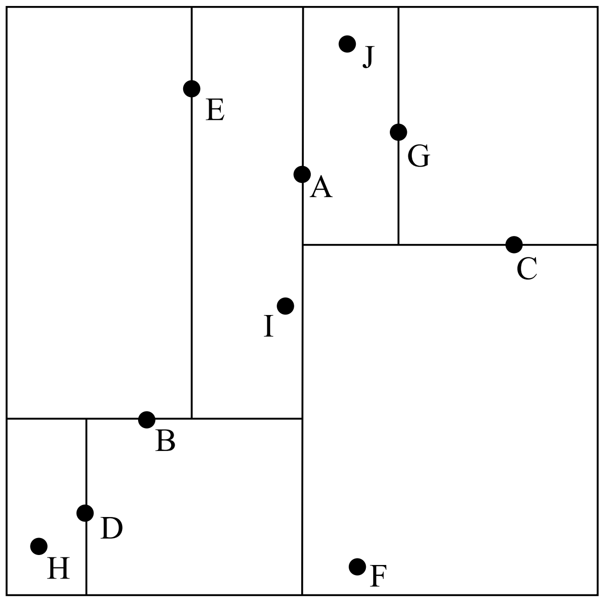
Thank you!

nadeem@acm.org

k-d trees

- Binary tree
- Nodes: *k*-dimensional data points
- Each level partitioned based on one of *k* dimensions
- Each subtree associated with an (implicit) bounding box
- Enables sub-linear NN search complexity through branch-and-bound





I: (50, 50)
A: (51, 75)
B: (25, 40)
C: (70, 70)
J: (55, 95)
H: (1, 10)
G: (60, 80)
F: (55, 1)
E: (35, 90)
D: (10, 30)

Lowercase ***k*** = dimension of data points;
Uppercase ***K*** = number of neighbors