# **ANN (Approximate Nearest Neighbor)**

An approximate nearest neighbor search algorithm is allowed to return points, whose distance from the query is at most *c* times the distance from the query to its nearest points.

The appeal of this approach is that, in many cases, an approximate nearest neighbor is almost as good as the exact one. In particular, if the distance measure accurately captures the notion of user quality, then small differences in the distance should not matter.

The ANN algorithm is able to solve multi-class classification tasks. The Apache Ignite implementation is a heuristic algorithm based upon searching of a small limited size N of candidate points (internally it uses a distributed KMeans clustering algorithm to find centroids) that can vote for class labels similar to a KNN algorithm.

The difference between KNN and ANN is that in the prediction phase all training points are involved in searching the k-nearest neighbors in the KNN algorithm, but in the ANN this search starts only on a small subset of candidate points.

NOTE: if N is set to the size of the training set, the ANN reduces to the KNN approach with enormous time spent in the training phase. Therefore, choose N comparable to K (10 x K, 100 x K for example).

## 

## **Model**

ANN classification output represents a class membership. An object is classified by the majority votes of its neighbors. The object is assigned to a particular class that is most common among its k nearest neighbors. k is typically a small positive integer. There is a special case when k is 1, then the object is simply assigned to the class of that single nearest neighbor.

Ignite currently supports these parameters for the ANN classification algorithm:

* k - a number of nearest neighbors
* distanceMeasure - one of the distance metrics provided by the ML framework such as Euclidean, Hamming or Manhattan
* KNNStrategy - could be SIMPLE or WEIGHTED (it enables a weighted k-NN algorithm)

NNClassificationModel knnMdl = trainer.fit(

...

).withK(5)

.withDistanceMeasure(**new** EuclideanDistance())

.withStrategy(NNStrategy.***WEIGHTED***);

## 

## **Trainer**

The trainer of the ANN model is based upon the KMeans algorithm to calculate the candidate subset and this is the reason that it has the same parameters as the KMeans algorithm to tune its hyperparameters. It builds not only the set of candidates but also their class-label distributions to vote for the class label during prediction phase.

Ignite currently supports the following parameters for ANNClassificationTrainer:

* k - a number of possible clusters
* maxIterations - one stop criteria (the other one is epsilon)
* epsilon - delta of convergence (delta between old and new centroid's values)
* distance - one of the distance metrics provided by the ML framework such as Euclidean, Hamming or Manhattan
* seed - one of initialization parameters which helps to reproduce models (trainer has a random initialization step to get the first centroids)

// Set up the trainer  
ANNClassificationTrainer trainer = new ANNClassificationTrainer()  
 .withDistance(new ManhattanDistance())  
 .withK(50)  
 .withMaxIterations(1000)  
 .withSeed(1234L)  
 .withEpsilon(1e-2);

// Build the model  
 NNClassificationModel knnMdl = trainer.fit(  
 ignite,  
 dataCache,  
 (k, v) -> VectorUtils.of(Arrays.copyOfRange(v, 1, v.length)),  
 (k, v) -> v[0]  
 ).withK(5)  
 .withDistanceMeasure(new EuclideanDistance())  
 .withStrategy(NNStrategy.WEIGHTED);

# **Example**

To see how ANNClassificationModel can be used in practice, try this [example](https://github.com/apache/ignite/blob/master/examples/src/main/java/org/apache/ignite/examples/ml/knn/ANNClassificationExample.java) that is available on GitHub and delivered with every Apache Ignite distribution.

The training dataset is the Iris dataset that can be loaded from the [UCI Machine Learning Repository](https://archive.ics.uci.edu/ml/datasets/iris).