Deconstructing the Temporal-Weighted K1K2NN Algorithm: A Comprehensive Pipeline Analysis for Dynamic Multi-Label Classification

An Introduction to Advanced Neighbor-Based Learning

The landscape of machine learning is perpetually evolving, driven by the increasing complexity and dimensionality of real-world data. Traditional classification paradigms, while foundational, often fall short when confronted with the nuanced, multi-faceted nature of modern datasets. Two significant challenges have emerged as central to the advancement of predictive modeling: the prevalence of multi-label instances, where a single data point can be associated with multiple categories simultaneously, and the inherent temporal dynamics of data, where the relevance and statistical properties of information change over time. The Temporal-Weighted K1K2NN (TW-K1K2NN) algorithm represents a sophisticated and targeted response to the confluence of these challenges, offering an integrated framework that addresses both label correlation and temporal decay within a unified, neighbor-based architecture. This report provides an exhaustive, step-by-step deconstruction of the TW-K1K2NN pipeline, elucidating its theoretical underpinnings, mathematical formalism, and practical applications.

The Evolution from Single-Label to Multi-Label Classification

Machine learning classification has traditionally been categorized into binary classification (two exclusive classes) and multi-class classification (more than two exclusive classes). A common thread in these paradigms is the assumption of mutual exclusivity; an instance belongs to one and only one category.[1] However, many contemporary problems defy this simplification. In domains such as bioinformatics, a single gene may be associated with multiple biological functions; in text analysis, a

document can cover several topics; and in clinical medicine, a drug can have a wide array of side effects.[3] This reality gives rise to the field of Multi-Label Classification (MLC).

Formally, in MLC, given an instance space X and a finite set of Q possible labels L={l1,l2,...,lQ}, the goal is to learn a function h:X→2L that maps an instance x∈X to a subset of labels Y⊆L.[5] The most intuitive approach to solving an MLC problem is through problem transformation, wherein the task is decomposed into multiple, independent binary classification problems—one for each label in

L.[5] This method, known as Binary Relevance, is simple to implement but carries a fundamental flaw: it completely ignores the potential correlations that exist between labels.[5] For example, in medical diagnostics, the presence of the label "hypertension" is highly correlated with the label "risk of stroke." An algorithm that treats these as independent phenomena discards a rich source of predictive information.

Recognizing this limitation, researchers developed more advanced algorithms designed to leverage label correlations. Methods like the Multi-Label k-Nearest Neighbors (ML-kNN) algorithm adapt the classic kNN framework by using the label sets of an instance's neighbors and applying the Maximum a Posteriori (MAP) principle to infer the most likely label set.[5] These "adapted algorithms" represent a crucial step forward, acknowledging that the label space has its own intrinsic structure. This problem is further magnified in the context of "extreme classification," a subfield focused on problems with an exceptionally large number of labels, where exploiting label structure is not just beneficial but computationally necessary.[8]

The Challenge of Temporal Dynamics in Machine Learning

Concurrent with the evolution of MLC, a separate challenge has been the increasing prevalence of data generated in dynamic, time-sensitive environments. Many systems, from biological networks to computer networks, are not static entities but evolve over time. This has led to the development of temporal graph theory, which models these systems as time-varying networks where nodes may be stable, but the interactions (edges) between them are timestamped and transient.[9] A temporal graph can be conceptualized as a sequence of static graph "snapshots," each representing the network's state at a specific point in time.[9]

Applying traditional, static analysis methods to such data is often inadequate and can be misleading. Aggregating all temporal interactions into a single, static graph—a common practice—obliterates the crucial information contained in the sequence and timing of events.[9] For instance, in a communication network, a path from node A to B to C is only valid if the A-to-B communication occurs

before the B-to-C communication. This concept of a "time-respecting path" is fundamental to understanding information flow and causality in temporal systems but is lost in a static representation.[10] This issue is particularly acute in fields like network intrusion detection, where attack patterns are defined by a specific sequence of actions over a short period [11], and in biomedical research, where the analysis of dynamic biological processes is essential for understanding disease progression and treatment effects.[9]

The Confluence of Challenges: Motivating TW-K1K2NN

The true frontier of modern machine learning lies at the intersection of these two challenges. Many of the most critical and complex real-world problems are simultaneously multi-label and temporal. The prediction of adverse drug reactions, for example, involves associating a drug (an instance) with a set of potential side effects (multiple labels), where the relevance of comparative data from other drugs may depend on when they were studied or approved (a temporal component).[4] Similarly, securing a computer network involves classifying a traffic flow with multiple potential attributes (e.g., {

DoS, Port Scan}), where the most relevant historical data for comparison is that which was generated moments ago, not days or weeks prior.[11]

An algorithm designed for this environment must therefore possess a dual capability: it must be able to model the complex, non-independent relationships within the label space while simultaneously accounting for the decaying relevance of information over time. This requires a synthesis of techniques from both MLC and time-series analysis. The development of an algorithm like TW-K1K2NN is not merely a technical combination of two separate methods. It signifies a more profound shift in modeling philosophy—a move away from solving isolated, simplified problems and toward creating integrated systems that holistically reflect the multi-faceted reality of data. It acknowledges that the "what" (the labels) and the "when" (the timestamp) are not

independent variables to be modeled in isolation but are intertwined aspects of a single observation. The TW-K1K2NN algorithm is architected to address this intertwined reality, providing a novel framework for intelligent prediction in dynamic, multi-label domains.

The Architectural Blueprint: The K1K2NN Framework

Before incorporating the complexities of time, it is essential to deconstruct the foundational architecture upon which the temporal weighting is built: the K1K2NN algorithm. Proposed by Das et al. for predicting COVID-19 drug side effects, K1K2NN is a novel multi-label classification approach that extends the principles of neighbor-based learning by operating in two distinct but complementary spaces: the instance feature space and the label co-occurrence space.[4] This two-stage process is designed to leverage not only the similarity between data points but also the statistical correlations between the labels themselves, making it more robust and powerful than its predecessors.

The architecture of K1K2NN represents a significant conceptual advance. Whereas traditional neighbor-based methods operate under the single assumption that similar instances should have similar labels, K1K2NN introduces a second, powerful assumption: that labels which frequently co-occur across the entire dataset are conceptually related, and this relationship can be used to refine and enrich predictions. This decoupling of two distinct forms of similarity—**instance similarity** in the feature space and **concept similarity** in the label space—is the cornerstone of the algorithm's design. It allows the model to correct for sparse or noisy local neighborhoods by drawing upon the global statistical structure of the label space, effectively creating a system that combines local evidence with global knowledge.

Foundational Principle: The k-Nearest Neighbors (kNN) Algorithm

At its core, K1K2NN is an evolution of the k-Nearest Neighbors (kNN) algorithm. kNN is a non-parametric, lazy learning method, meaning it does not build an explicit model during a training phase but instead memorizes the entire training dataset.[13] The

classification of a new, unseen instance occurs at prediction time. The algorithm identifies the

k instances in the training set that are "closest" to the new instance in the high-dimensional feature space. Proximity is typically measured using a distance metric, most commonly the Euclidean distance.[15] The new instance is then assigned the class label that is most frequent among its

k nearest neighbors—a process of majority voting.[15] Its simplicity, interpretability, and effectiveness in various domains have made it a foundational algorithm in machine learning.

The K1 Stage: Instance-Space Neighborhood Identification

The first phase of the K1K2NN algorithm is a direct application of the kNN principle to the multi-label context. For a given unlabeled test instance, this stage aims to identify a preliminary set of relevant labels based on its most similar counterparts in the training data.

The process is as follows: for a test instance represented by its feature vector, the algorithm computes a distance metric (e.g., Euclidean distance) between it and every instance in the training set.[4] These distances quantify the similarity in the feature space; for example, in the drug side effect domain, this would mean comparing the chemical properties of a new drug to all existing drugs in the database.[4]

The algorithm then sorts these distances and identifies the k1 training instances that have the smallest distance values. These instances are designated as the k1 nearest *instance neighbors*. The core idea is that these chemically or structurally similar drugs are likely to share similar biological effects, including side effects. The collection of all labels associated with these k1 neighbors is then aggregated. Through a majority voting mechanism, the labels that appear most frequently among these neighbors are selected to form an initial, candidate label set for the test instance.[4] This step provides the initial "evidence" for the prediction, grounded entirely in the local neighborhood of the feature space.

The K2 Stage: Label-Space Expansion via Co-occurrence

The second stage is the most innovative part of the K1K2NN architecture. It operates on the principle that "co-occurring [labels] may be assigned to chemically similar [instances]".[4] This stage takes the initial evidence from the K1 stage and refines it by leveraging the global correlation structure of the entire label space. It acts as a knowledge-based inference step that can augment the prediction with highly probable labels that might have been missed in the local K1 search due to data sparsity or noise.

The K2 stage involves a pre-computation step and a prediction-time search:

2.3.1. Building the Label Co-occurrence Matrix

During a one-time training phase, the algorithm analyzes the entire training dataset to map the relationships between all possible labels. It constructs a label co-occurrence matrix, a square matrix of size Q×Q (where Q is the total number of unique labels). Each entry (i,j) in this matrix stores a count of how many times label li and label lj appeared together in the label set of a single instance across the entire dataset. This matrix serves as the raw data for understanding label correlations.

2.3.2. Calculating Label-to-Label Similarity

Raw co-occurrence counts can be misleading, as very frequent labels will naturally co-occur with many other labels more often. To obtain a normalized measure of association, the algorithm computes a similarity score between every pair of labels. The K1K2NN paper specifies the use of the **Jaccard Similarity** for this purpose.[4] The Jaccard Similarity (or Jaccard Index) between two sets

A and B is defined as the size of their intersection divided by the size of their union [18]:

$$J(A,B) = |A \cup B| \, |A \cap B|$$

In this context, the "sets" for two labels, la and lb, are the sets of training instances that possess each label. The resulting Jaccard similarity score is a value between 0 and 1, where 1 indicates that the two labels always appear together and 0 indicates

they never do. This calculation is performed for all pairs of labels, yielding a dense, symmetric label similarity matrix. This matrix encapsulates the "concept similarity" for the entire domain. The Jaccard distance, defined as 1–J(A,B), can also be used to frame this as a distance metric learning problem, providing a fine-grained, real-valued measure of similarity between labels.[18]

2.3.3. Identifying k2 Nearest Labels

At prediction time, for each label identified in the initial set from the K1 stage, the algorithm performs a second neighbor search. This time, however, the search occurs in the *label space* using the pre-computed Jaccard similarity matrix. For a given label la from the initial set, the algorithm looks up its row (or column) in the similarity matrix and identifies the k2 other labels that have the highest Jaccard similarity score with la. These are the k2 nearest *label neighbors*.[4]

2.3.4. Final Label Set Augmentation

The final step of the K2 stage is to augment the prediction. The k2 nearest label neighbors found for each of the initial labels are collected and added to the candidate label set derived from the K1 stage. This step enriches the final prediction by incorporating labels that are conceptually and statistically linked to the initial findings.

This two-stage process demonstrates remarkable robustness. For example, consider a new drug whose k1 nearest neighbors in the feature space only exhibit the side effects {headache, nausea}. The K1 stage would only suggest these two labels. However, if the global label similarity matrix reveals a very strong Jaccard similarity between "nausea" and "dizziness" across thousands of drugs in the training set, the K2 stage will identify "dizziness" as a nearest label neighbor to "nausea" and add it to the final prediction. In this way, the K2 stage leverages the global statistical patterns of the data to correct for potential omissions or noise present in the local, instance-based neighborhood, making the final prediction more comprehensive and reliable.

Incorporating Time: Principles of Temporal Weighting

The K1K2NN framework provides a robust solution for handling multi-label classification by exploiting label correlations. However, in its standard form, it treats all training data as equally relevant, regardless of when it was generated. This assumption is untenable in dynamic environments where the underlying patterns and relationships in the data change over time. The "Temporal-Weighted" (TW) component of the TW-K1K2NN algorithm is designed specifically to address this limitation by introducing a mechanism that prioritizes more recent information, allowing the model to adapt to the evolving nature of the data.

The decision to incorporate temporal weighting is not merely a technical enhancement; it encodes a fundamental assumption about the nature of information in many real-world systems: **information has a shelf life**. The value of data often decays over time, a phenomenon that must be modeled explicitly for a predictive algorithm to remain accurate and relevant. The choice of a specific temporal decay function and its parameters is therefore not a trivial implementation detail but rather a hypothesis about the *rate of information obsolescence* within a given domain. This elevates the temporal weighting component from a simple parameter to a model of the system's inherent dynamics.

The Rationale for Temporal Weighting: Concept Drift and Data Recency

The primary motivation for temporal weighting is the phenomenon known as "concept drift." Concept drift occurs when the statistical properties of the target variable, which the model is trying to predict, change over time.[11] This means that a model trained on historical data may become progressively less accurate as the underlying data-generating process evolves.

This is a common issue in many application domains:

- **Network Intrusion Detection:** Malicious actors constantly develop new attack vectors and modify existing ones. An intrusion detection system trained on attack patterns from last year may be completely blind to a novel threat emerging today. Therefore, data from recent network traffic is far more indicative of the current threat landscape than older data.[11]

- **Drug Discovery and Pharmacovigilance:** The understanding of a drug's side effects can evolve. Rare side effects may only become apparent after a drug has been on the market for several years and used by a large population. Conversely, the relevance of data from older, less-used drugs may diminish. Therefore, temporal proximity can be a proxy for relevance.[4]
- **Location-Based Services:** The popularity or "social opinion" of a location (e.g., a restaurant or park) is highly dynamic. Recommendations based on check-ins or reviews from last year are likely to be less reliable than those from last week.[20]

In all these cases, the core principle is the same: data recency matters. A training instance that is temporally closer to the test instance should exert a greater influence on the final prediction.[20] Temporal weighting is the mechanism by which this principle is mathematically formalized and integrated into the learning algorithm.

Methodologies for Temporal Influence: The Exponential Decay Function

While several functions can model the decay of influence over time, such as linear decay or step decay (where the weight is reduced by a fixed factor after a certain number of time steps) [22], the most common and mathematically versatile method for modeling continuous-time influence is the

Exponential Decay function.[22]

Exponential decay describes a process where a quantity decreases at a rate proportional to its current value. This results in a rapid initial decrease, which then slows as the quantity gets smaller.[23] In the context of temporal weighting, this is an intuitive model: the "value" of a piece of information loses a percentage of its remaining relevance with each passing time unit. The general mathematical formula for exponential decay is:

$$y(t)=y0 \cdot e-\lambda t$$

where y0 is the initial quantity, t is the time elapsed, and λ is the decay constant. In a machine learning context, this is often expressed in a discrete form for implementation with optimizers or weighting schemes [24]:

$$weight=initial\_weight \cdot decay\_rate(decay\_stepsstep)$$

The key parameters that govern this function are:

- **Initial Weight (y0):** This represents the maximum influence, assigned to the most recent data point (i.e., when the time difference is zero). In weighting schemes, this is typically set to 1.0.[25]
- **Decay Rate (λ or decay_rate):** This is a crucial hyperparameter that controls the steepness of the decay. It is a floating-point number, typically between 0 and 1 in the discrete formulation. A decay rate close to 1.0 (e.g., 0.99) results in a slow, gradual decay, meaning older data retains its influence for longer. A smaller decay rate (e.g., 0.8) causes a rapid decay, heavily penalizing older data.[22] The selection of this parameter is domain-specific; a rapidly changing environment like network security would demand a steeper decay, while a more stable domain like long-term drug effects might use a slower one.
- **Time Delta (t or step):** This is the measure of time elapsed between the historical data point and the current query time. It can be measured in various units, such as seconds, days, or the number of data instances processed.[9]

Applying Temporal Weights in Neighbor-Based Algorithms

The exponential decay function can be seamlessly integrated into the kNN framework to make it time-aware. In a standard weighted kNN, the influence of a neighbor is often weighted by the inverse of its distance—closer neighbors get a bigger "vote".[26] A temporal weighted kNN extends this by making the influence of a neighbor a function of both its feature-space distance

and its temporal proximity.

One simple approach is to multiply the standard distance-based weight of a neighbor by its calculated temporal weight. This would scale down the influence of older neighbors during the final voting stage.[27] However, a more elegant and powerful approach is to incorporate the temporal weight directly into the distance metric used for the neighbor search itself. This is the strategy implied by the ranking function proposed in related research on spatio-temporal queries, which combines spatial distance and a temporal aggregate into a single score.[20] By modifying the distance calculation, the algorithm penalizes older samples at the most fundamental stage: the search for neighbors. This ensures that the set of

k neighbors itself is temporally relevant, rather than simply down-weighting a potentially irrelevant set of neighbors after the fact. This method of creating a

temporally-weighted distance is the one adopted by the TW-K1K2NN pipeline.

The Complete Pipeline: A Step-by-Step Analysis of Temporal-Weighted K1K2NN

By synthesizing the robust label-correlation mechanism of the K1K2NN framework with the adaptive principles of temporal weighting, the Temporal-Weighted K1K2NN (TW-K1K2NN) algorithm provides a comprehensive solution for dynamic multi-label classification. The complete pipeline elegantly integrates these components, creating a process that is sensitive to both the "what" (label structure) and the "when" (instance recency) of the data. This section provides a granular, step-by-step walkthrough of the algorithm's operation, from initial data formulation to the final synthesized prediction.

A crucial aspect of the TW-K1K2NN architecture is an implicit, yet powerful, assumption it makes about the nature of temporal decay. The temporal weighting is applied exclusively during the K1 stage—the search for similar instances in the feature space. The K2 stage—the expansion via label co-occurrence—operates on a static similarity matrix built from the entire, non-weighted training set. This architectural choice reveals a core belief embedded in the algorithm: **instance relevance decays with time, but conceptual relationships between labels are timeless**, or at least decay on a much longer timescale. The model assumes that while a specific network attack from yesterday is more relevant than one from last year (instance decay), the fundamental relationship that a "Port Scan" is conceptually linked to a "Firewall Probe" holds true across the entire dataset's history (conceptual stability). This design choice makes the algorithm particularly well-suited for domains where this assumption holds, but it also highlights a potential limitation in scenarios where the meanings and correlations of labels themselves drift over time.

Step 1: Input Data Formulation and Pre-processing

The first step in any machine learning pipeline is to define and prepare the data. The TW-K1K2NN algorithm requires a specific data structure that includes feature, label,

and temporal information.

- **Training Data:** The input is a training set S consisting of m instances, where each instance i is a triplet: $S=\{(x_i,Y_i,t_i)\}_{i=1}^{m}$.
  - x_i: This is a d-dimensional feature vector representing the observable attributes of the instance. For example, in drug discovery, x_i could be a vector of molecular descriptors and chemical properties.[4] In network security, it could be a vector of network flow statistics like duration, protocol type, and byte counts, as found in datasets like NSL-KDD.[5]
  - Y_i: This is the set of labels associated with instance i. It is a subset of the total label space L, i.e., $Y_i \subseteq L$.[6]
  - t_i: This is a numerical timestamp associated with instance i, indicating when the instance was recorded or observed.
- **Test Data:** The input for prediction is a single, unlabeled instance defined by its features and query time: (x_q,t_q). The goal is to predict its unknown label set, Y_q. The query time t_q is crucial, as it serves as the reference point for all temporal calculations.
- **Pre-processing:** Before any calculations are performed, the feature vectors x_i and x_q typically undergo normalization or standardization. This is a critical step for distance-based algorithms like kNN. It ensures that all features are on a comparable scale (e.g., for normalization or a standard normal distribution for standardization). Without this step, features with larger numerical ranges would disproportionately dominate the Euclidean distance calculation, leading to biased neighbor selection.[16]

Step 2: Temporal Weight Calculation for All Training Instances

With the query instance (x_q,t_q) defined, the algorithm's next step is to quantify the temporal relevance of every single instance in the training set S. This is achieved by applying an exponential decay function.

For each training instance i, the algorithm first calculates the time difference, or temporal delta, relative to the query time:

$$\Delta t_i = t_q - t_i$$

It is assumed that the query time t_q is greater than or equal to any training instance time t_i. This Δt_i represents the "age" of the training instance.

Next, an exponential decay function is used to compute a temporal weight wi for each training instance. This weight will be a value between 0 and 1.

$$w_i = e^{-\lambda \cdot \Delta t_i}$$

Here, λ is the **decay rate hyperparameter**, a small positive number that must be tuned for the specific domain. A larger λ value results in a steeper decay, meaning older data is penalized more heavily and loses its influence quickly. A smaller λ results in a slower decay, allowing older data to retain more relevance. The resulting weight wi will be close to 1.0 for very recent instances (where Δti is small) and will approach 0 for very old instances (where Δti is large).

Step 3: Temporally-Weighted K1 Neighbor Selection

This step is the heart of the temporal adaptation. It modifies the standard K1 neighbor search by integrating the temporal weights computed in Step 2. Instead of finding neighbors based solely on feature-space proximity, it finds neighbors based on a combined measure of feature similarity and temporal relevance.

First, for each training instance i, the algorithm calculates the standard Euclidean distance, $d_E(x_q, x_i)$, between the feature vector of the query instance x_q and the feature vector of the training instance x_i.[16]

Then, it computes the final **Temporally-Weighted Distance**, $d_{TW}(x_q, x_i)$, by modulating the Euclidean distance with the temporal weight wi:

$$d_{TW}(x_q, x_i) = \frac{d_E(x_q, x_i)}{w_i}$$

This formulation provides an elegant and intuitive way to combine the two criteria. Since wi is between 0 and 1, dividing by it has the effect of increasing the perceived distance for older samples. For a very recent sample, $w_i \approx 1$, and its distance is minimally affected ($d_{TW} \approx d_E$). For a very old sample, $w_i \to 0$, and its distance is inflated towards infinity, making it extremely unlikely to be selected as a neighbor. This directly implements the ranking principle from related spatio-temporal research within the kNN framework.[20]

Finally, the algorithm identifies the k1 training instances that have the smallest Temporally-Weighted Distance values. This set of k1 instances constitutes the temporally-aware nearest neighbors.

Step 4: Initial Label Set Generation

Once the k1 temporally-weighted nearest neighbors have been identified, the next step is to generate an initial candidate set of labels for the query instance xq. This is done by aggregating the label information from these neighbors.

The algorithm collects the known label sets, {Yi}, from each of the k1 neighbors. A voting or aggregation mechanism is then employed to determine the initial label set, which we can call Yq_initial. There are several possible strategies for this aggregation:

- **Majority Voting:** A label is included in Yq_initial if it is present in more than half of the k1 neighbors.
- **Thresholding:** A label is included if it appears in at least a certain number or percentage of the neighbors. This is analogous to the posterior probability calculation used in the standard ML-kNN algorithm.[5]
- **Union:** A simple approach is to take the union of all label sets from the k1 neighbors, although this can be noisy.

The choice of aggregation method is another design parameter of the model. The resulting set, Yq_initial, represents the prediction based on direct, temporally-weighted evidence from similar instances.

Step 5: K2 Label-Neighbor Identification (The Unchanged Core)

This step proceeds exactly as described in the non-temporal K1K2NN framework (Section 2.3). It is crucial to note that this stage is deliberately **not** time-dependent. It operates on the global, static label co-occurrence structure that was pre-computed from the entire training set.

Using the pre-calculated label-to-label Jaccard similarity matrix, the algorithm performs a second neighbor search. For each label that was included in the initial set Yq_initial, it looks up the k2 other labels that are most similar to it based on their Jaccard similarity score. This identifies labels that are conceptually and statistically related to the initial findings.

Let the set of all such new labels found via this label-space search be denoted as Yq_expansion.

Step 6: Final Prediction Synthesis

The final step of the pipeline is to combine the results from the temporally-aware instance-space search (K1) and the static concept-space search (K2) to produce the final, comprehensive prediction for the query instance.

The final predicted label set, Yq, is simply the union of the labels from the initial set and the expansion set:

$$Yq = Yq\_initial \cup Yq\_expansion$$

This final set contains labels supported by direct evidence from recent, similar instances, as well as labels supported by global, historical correlations. This synthesis allows the algorithm to produce predictions that are both adaptive to recent trends and robust against local data sparsity.

Mathematical Formalism and Algorithmic Complexity

A rigorous understanding of any algorithm requires a formal definition of its components and an analysis of its computational costs. This section provides the precise mathematical formalism for the Temporal-Weighted K1K2NN pipeline and analyzes its complexity during both the offline pre-computation phase and the online prediction phase.

Table of Notations

To ensure clarity and precision throughout the analysis, the following table defines all mathematical symbols used to describe the algorithm. Centralizing these definitions is standard practice in technical literature and is essential for reproducibility and

unambiguous communication of the model's mechanics.

Symbol	Description	Source/Context
S	The training set, consisting of m instances {(xi,Yi,ti)}i=1m	General ML
xi	The d-dimensional feature vector of the i-th training instance	5
Yi	The set of labels associated with instance xi, where Yi⊆L	5
ti	The timestamp associated with instance xi	9
(xq,tq)	The query instance, consisting of a feature vector and a query time	20
L	The complete set of Q possible labels in the domain, L={l1,...,lQ}	5
m	The total number of instances in the training set S	5
d	The dimensionality of the feature space for vectors xi	5
Q	The total number of unique labels in the label space L	5
k1,k2	The number of neighbors to consider in the K1 and K2 stages, respectively	4
dE(a,b)	The Euclidean distance between two vectors a and b	16

λ	The hyperparameter controlling the rate of exponential temporal decay	[25]
wi	The calculated temporal weight for the i-th training instance	Derived
dTW(a,b)	The Temporally-Weighted Distance between instance a and instance b	Derived from [20]
J(A,B)	The Jaccard similarity between two sets A and B	[18]
C	The Q×Q label co-occurrence matrix	Derived from [4]
SimL	The Q×Q label-to-label similarity matrix, based on Jaccard similarity	Derived

Formal Definitions

With the notation established, we can provide the formal mathematical definitions for the key components of the TW-K1K2NN algorithm.

- Euclidean Distance: The standard distance metric used in the feature space. For two d-dimensional vectors a=(a1,...,ad) and b=(b1,...,bd), the Euclidean distance is:

  $$dE(a,b)=\sum_{j=1}^{d}(a_j-b_j)^2$$
- Temporal Weight: The weight assigned to training instance i with respect to query time tq.

  $$w_i=e^{-\lambda(t_q-t_i)}$$
- Temporally-Weighted Distance: The modified distance metric used in the K1 neighbor search. It combines Euclidean distance and the temporal weight.

$$d_{TW}(x_q, x_i) = \text{widE}(x_q, x_i) = \sum_{j=1} d(x_{qj} - x_{ij})^2 \cdot e^{\lambda(t_q - t_i)}$$

- Label Co-occurrence Matrix (C): An integer matrix of size $Q \times Q$, where each element $C(l_a, l_b)$ is the count of training instances that are labeled with both label l_a and label l_b.

  $$C(l_a, l_b) = |\{i \in \{1, \ldots, m\} \mid \{l_a, l_b\} \subseteq Y_i\}|$$
- Label Jaccard Similarity Matrix (SimL): A real-valued matrix of size $Q \times Q$, where each element $\text{SimL}(l_a, l_b)$ is the Jaccard similarity between the sets of instances possessing labels l_a and l_b. Let $N(l) = \{i \mid l \in Y_i\}$ be the set of indices of instances with label l. Then:

  $$\text{SimL}(l_a, l_b) = J(N(l_a), N(l_b)) = \frac{|N(l_a) \cup N(l_b)|}{|N(l_a) \cap N(l_b)|} = \frac{|N(l_a)| + |N(l_b)| - C(l_a, l_b)}{C(l_a, l_b)}$$

Algorithmic Pseudocode

The entire prediction process can be formalized in the following pseudocode for a function Predict that takes the training data and a query instance as input and returns a predicted label set.

```
Algorithm: Temporal-Weighted K1K2NN Prediction

Function Predict(S, x_q, t_q, k1, k2, λ)
Input:
  S: Training set {(x_i, Y_i, t_i)} for i=1...m
  x_q: Feature vector of the query instance
  t_q: Timestamp of the query instance
  k1, k2: Number of neighbors for each stage
  λ: Exponential decay rate

Output:
  Y_q: Predicted label set for x_q
```

```
// Pre-computation (one-time cost, performed offline)
C ← ComputeLabelCooccurrenceMatrix(S)
Sim_L ← ComputeLabelJaccardSimilarity(C, S)

// Prediction Phase
Distances ← empty list
for i from 1 to m do:
    d_E ← EuclideanDistance(x_q, x_i)
    w_i ← exp(-λ * (t_q - t_i))
    d_TW ← d_E / w_i
    Add (d_TW, i) to Distances
end for

Sort Distances by d_TW in ascending order
K1_Neighbors_Indices ← Get first k1 indices from sorted Distances

Y_q_initial ← GenerateInitialLabels(S, K1_Neighbors_Indices) // e.g., via majority vote

Y_q_expansion ← empty set
for each label l_a in Y_q_initial do:
    Label_Similarities ← Get row for l_a from Sim_L
    Sort Label_Similarities in descending order
    K2_Neighbor_Labels ← Get top k2 labels from sorted Label_Similarities
    Add K2_Neighbor_Labels to Y_q_expansion
end for

Y_q ← Y_q_initial ∪ Y_q_expansion
return Y_q
```

Computational Complexity Analysis

The computational cost of TW-K1K2NN can be broken down into two distinct phases: an offline training/pre-computation phase and an online prediction phase.

- Training Phase (Offline Pre-computation):
  This phase involves building the label similarity matrix, which is done only once.

1. **Building the Label Co-occurrence Matrix (C):** This requires iterating through each of the m training instances. For each instance, we must consider all pairs of its labels. If the average number of labels per instance is $|Y_{avg}|$, this step has a complexity of approximately $O(m \cdot |Y_{avg}|^2)$.
2. Building the Jaccard Similarity Matrix (SimL): This involves iterating through the Q×Q co-occurrence matrix. For each pair of labels (la,lb), the calculation requires the marginal counts $|N(l_a)|$ and $|N(l_b)|$, which can be pre-calculated in $O(m \cdot |Y_{avg}|)$. The matrix construction itself is then $O(Q^2)$.
   The total pre-computation complexity is dominated by these two steps, resulting in $O(m \cdot |Y_{avg}|^2 + Q^2)$.

- Prediction Phase (Online, for a single query instance):
  This phase constitutes the "lazy" part of the algorithm, where the bulk of the computation occurs.
  1. **Calculating Temporally-Weighted Distances:** The algorithm must compute the distance from the query instance xq to all m training instances. Each distance calculation in a d-dimensional space takes O(d) time. Therefore, this step has a complexity of **$O(m \cdot d)$**.
  2. **Finding k1 Nearest Neighbors:** Sorting the m computed distances takes O(mlogm) time. Alternatively, using a selection algorithm like introselect can find the top k1 elements in average-case O(m) time. Thus, this step is between **O(m) and O(mlogm)**.
  3. **Generating Initial Label Set:** This depends on the voting scheme but is typically fast, around $O(k1 \cdot |Y_{avg}|)$.
  4. **Finding k2 Nearest Labels:** For each label in the initial set ($|Y_{q\_initial}|$), we must find its k2 nearest neighbors in the Q-dimensional label space. Sorting the similarities for one label takes O(QlogQ). This gives a complexity of **$O(|Y_{q\_initial}| \cdot Q\log Q)$**.

The overall prediction time complexity is the sum of these steps. In most practical scenarios where the number of training instances m is much larger than the feature dimension d and the number of labels Q, the bottleneck is the K1 neighbor search. Therefore, the prediction complexity is dominated by the distance calculations and sorting, making it approximately **$O(m \cdot d + m\log m)$**. This highlights the algorithm's primary computational challenge: its prediction time scales linearly with the size of the training set, which can be prohibitive for large-scale, low-latency applications.

Applications, Insights, and Future Directions

The true measure of an algorithm's value lies in its applicability to real-world problems, its inherent strengths and weaknesses, and its potential to inspire future research. The Temporal-Weighted K1K2NN algorithm, with its unique architecture, is particularly well-suited for a class of complex problems where both label interdependencies and temporal dynamics are critical. This section explores its primary use cases, provides a critical analysis of its performance characteristics, and outlines promising directions for future algorithmic development.

Domain-Specific Use Cases

The hybrid nature of TW-K1K2NN makes it a powerful tool in domains where data is both multi-faceted and time-stamped. Two primary applications stand out, directly reflecting the motivations behind its constituent components.

Predicting Drug Side Effects

This domain, which inspired the original K1K2NN paper, is a canonical example of a multi-label problem.[3]

- **Instances (xi):** Each instance is a drug, represented by a feature vector of its chemical and physical properties. These can include 1D/2D structural information, molecular descriptors, or SMILES strings.[4]
- **Labels (Yi):** The labels for each drug are the set of its known adverse side effects (e.g., {headache, nausea, fatigue}).
- **Timestamps (ti):** The timestamp could represent the drug's approval date, the date of a major clinical trial report, or the date a specific side effect was officially documented.
- **TW-K1K2NN Application:** When a new drug candidate (xq) is developed, the model can predict its likely side effect profile. The **K1 stage** would identify the k1 most chemically similar drugs from the existing database. The **temporal weighting** ensures that drugs analyzed or approved more recently, under modern protocols and with more comprehensive data, are given higher influence. This is

crucial as diagnostic criteria and reporting standards change over time. The **K2 stage** then provides a critical layer of inference. If the K1 neighbors suggest the side effect "drowsiness," the K2 stage, using its global knowledge of label co-occurrence, might augment this prediction with "impaired driving ability," a highly correlated but distinct side effect that may not have been present in the immediate neighbors.

Time-Sensitive Network Intrusion Detection

Network security is a domain defined by its temporal nature, where attacks are sequences of events and patterns evolve rapidly.[11]

- **Instances (xi):** Each instance is a network traffic flow, represented by a vector of features extracted from packet headers and flow statistics. Datasets like NSL-KDD provide such features, including connection duration, protocol type, service, and packet counts.[29]
- **Labels (Yi):** The labels classify the traffic flow's nature, such as {Normal} or a set of attack types like {DoS, Probe, R2L}. While often treated as multi-class, it can be framed as multi-label if a flow exhibits characteristics of multiple attack phases.
- **Timestamps (ti):** The precise time the network flow was captured.
- **TW-K1K2NN Application:** To classify a new, live traffic flow (xq), the model's **temporal weighting** is paramount. It will heavily prioritize finding neighbors from flows that occurred seconds or minutes ago, as these are most representative of the current network state and any ongoing attacks. The **K1 stage** finds the k1 most similar recent flows. The **K2 stage** helps in identifying sophisticated, multi-stage attacks. For example, if the K1 stage identifies a flow as a "Probe," the K2 stage might suggest the label "DoS," because probes are statistically very likely to be precursors to a Denial-of-Service attack, even if the query flow itself doesn't yet have DoS characteristics.

Critical Analysis and Recommendations

Like any algorithm, TW-K1K2NN has a distinct profile of strengths and weaknesses that practitioners must understand.

- **Strengths:**
  - **Hybrid Power:** Its greatest strength is its unique ability to synthesize instance-level feature similarity with global label-space correlation, providing a more holistic prediction than methods that focus on only one aspect.
  - **Adaptability to Concept Drift:** The explicit temporal weighting mechanism allows the model to gracefully adapt to changing data distributions, prioritizing recent data and discounting obsolete information.
  - **Interpretability:** Compared to complex deep learning models, TW-K1K2NN offers a degree of interpretability. A prediction can be explained by inspecting the k1 instance neighbors and the k2 label neighbors that contributed to the final result.
- **Weaknesses and Challenges:**
  - **Hyperparameter Sensitivity:** The model's performance is highly contingent on the optimal selection of three key hyperparameters: k1, k2, and the temporal decay rate λ. These values are domain-dependent and require extensive, computationally expensive tuning, typically via cross-validation.
  - **The "Curse of Dimensionality":** As a distance-based method, TW-K1K2NN is susceptible to the curse of dimensionality. In very high-dimensional feature spaces, the concept of Euclidean distance becomes less meaningful, as all points tend to become equidistant from each other. This can degrade the performance of the K1 neighbor search.[16]
  - **Computational Cost:** The primary drawback is its high prediction-time complexity of roughly O(m·d), where m is the size of the training set. This "lazy" learning approach can be prohibitively slow for applications requiring real-time predictions on very large datasets.
- **Recommendations for Practitioners:**
  1. **Hyperparameter Tuning:** Always perform rigorous grid search or randomized search with cross-validation to find the optimal k1, k2, and λ for your specific dataset.
  2. **Dimensionality Reduction:** For datasets with a very high number of features, apply feature selection or dimensionality reduction techniques (e.g., PCA, feature importance from tree-based models) *before* feeding the data to the TW-K1K2NN algorithm. This can mitigate the curse of dimensionality and improve both performance and speed.[29]
  3. **Approximate Nearest Neighbor (ANN) Search:** To address the computational bottleneck in large-scale applications, consider replacing the exact K1 neighbor search with an ANN technique like Locality Sensitive Hashing (LSH) [11] or specialized index structures like k-d trees. This trades a

small amount of accuracy for a significant improvement in prediction speed.

Table of Comparative Analysis

To contextualize the contributions of TW-K1K2NN, it is useful to compare it against its predecessors. The following table provides a clear conceptual map of its evolution and unique capabilities.

Algorithm	Handles Multi-Label?	Captures Label Correlation?	Is Time-Aware?	Key Mechanism
k-NN	No (Natively)	No	No	Majority vote of k nearest neighbors in feature space.[17]
ML-kNN	Yes	Partially (via Bayesian inference on neighbor labels)	No	MAP principle applied to the labels of k nearest neighbors.[5]
K1K2NN	Yes	Yes (Explicitly)	No	Two-stage search: k1 instance neighbors, then k2 label neighbors via Jaccard similarity.[4]
TW-K1K2NN	Yes	Yes (Explicitly)	Yes	K1K2NN framework with an exponential decay temporal weight applied to the instance-neighb or distance metric.[4]

Future Research Directions

The architecture of TW-K1K2NN, while powerful, also illuminates clear pathways for future research and enhancement.

- **Fully Temporal K1K2NN:** The current model assumes that while instance relevance decays, label correlations are static. This may not hold in all domains. A significant area for future work is to make the K2 stage time-aware as well. This could be achieved by constructing the label co-occurrence matrix using only data from a recent sliding window or by applying a temporal decay to the co-occurrence counts themselves. This would create a "Fully Temporal" model capable of adapting to drifts in both instance patterns and conceptual relationships.
- **Adaptive Decay Rate (λ):** The decay rate λ is currently a fixed, global hyperparameter. A more sophisticated model could learn an adaptive λ. This could be a function of the local data density or the observed rate of change in the data distribution, allowing the model to automatically adjust how quickly it "forgets" the past based on the stability of the environment.
- **Hybrid and Learned Distance Metrics:** The algorithm currently relies on Euclidean distance in the feature space. Performance could be improved by exploring other distance metrics (e.g., Cosine similarity for text, Manhattan distance) or by employing metric learning techniques. A learned distance metric could be trained to find a feature space transformation where distance more accurately reflects semantic similarity for a given task, potentially boosting the effectiveness of the K1 stage.

Works cited

1. An introduction to MultiLabel classification - GeeksforGeeks, accessed August 11, 2025, https://www.geeksforgeeks.org/machine-learning/an-introduction-to-multilabel-classification/
2. Multi-label classification for beginners with codes | by Mehul Gupta | Data Science in Your Pocket | Medium, accessed August 11, 2025, https://medium.com/data-science-in-your-pocket/multi-label-classification-for-beginners-with-codes-6b098cc76f99
3. K1K2NN: A Novel Multi-Label Classification Approach Based on Neighbors for Predicting COVID-19 Drug Side Effects - ResearchGate, accessed August 11, 2025,

https://www.researchgate.net/publication/379517983_K1K2NN_A_Novel_Multi-Label_Classification_Approach_Based_on_Neighbors_for_Predicting_COVID-19_Drug_Side_Effects
4. K1K2NN: A Novel Multi-Label Classification Approach Based on Neighbours to Predict Covid Drug Side Effects from Chemical Properties - ResearchGate, accessed August 11, 2025, https://www.researchgate.net/publication/369276680_K1K2NN_A_Novel_Multi-Label_Classification_Approach_Based_on_Neighbours_to_Predict_Covid_Drug_Side_Effects_from_Chemical_Properties?_share=1
5. A k-Nearest Neighbor Based Algorithm for Multi-label Classification - PALM, accessed August 11, 2025, https://palm.seu.edu.cn/zhangml/files/A%20k-nearest%20neighbor%20based%20algorithm%20for%20multi-label%20classification.pdf
6. Multi-Label Classification Algorithm for Adaptive Heterogeneous Classifier Group - MDPI, accessed August 11, 2025, https://www.mdpi.com/2227-7390/13/1/103
7. Solving Multi-Label Classification problems (Case studies included) - Analytics Vidhya, accessed August 11, 2025, https://www.analyticsvidhya.com/blog/2017/08/introduction-to-multi-label-classification/
8. Deep Learning Approach for Extreme Multi-label Text Classification - YouTube, accessed August 11, 2025, https://www.youtube.com/watch?v=yp01hpk1g4A
9. Temporal networks in biology and medicine: a survey on models, algorithms, and tools - PubMed Central, accessed August 11, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC9803903/
10. Using Time-Aware Graph Neural Networks to Predict Temporal Centralities in Dynamic Graphs - NIPS, accessed August 11, 2025, https://proceedings.neurips.cc/paper_files/paper/2024/file/3514dbacaebf0f38b25adfe59ed81a8a-Paper-Conference.pdf
11. A Fast kNN-based Approach for Time Sensitive Anomaly Detection ..., accessed August 11, 2025, https://www.iccs-meeting.org/archive/iccs2019/papers/115370056.pdf
12. Network based Intrusion Detection using Time aware LSTM Autoencoder - ResearchGate, accessed August 11, 2025, https://www.researchgate.net/publication/380993607_Network_based_Intrusion_Detection_using_Time_aware_LSTM_Autoencoder
13. Unification of K-Nearest Neighbor (KNN) with Distance Aware Algorithm for Intrusion Detection in Evolving Networks Like IoT - ResearchGate, accessed August 11, 2025, https://www.researchgate.net/publication/373837006_Unification_of_K-Nearest_Neighbor_KNN_with_Distance_Aware_Algorithm_for_Intrusion_Detection_in_Evolving_Networks_Like_IoT
14. Improvement of K-nearest Neighbors (KNN) Algorithm for Network Intrusion Detection Using Shannon-Entropy - Journal of Communications, accessed August 11, 2025, https://www.jocm.us/uploadfile/2021/0720/20210720030454657.pdf

15. A Combination Strategy of Feature Selection Based on an Integrated Optimization Algorithm and Weighted K-Nearest Neighbor to Improve the Performance of Network Intrusion Detection - MDPI, accessed August 11, 2025, https://www.mdpi.com/2079-9292/9/8/1206
16. Euclidean Distance Explained - Built In, accessed August 11, 2025, https://builtin.com/articles/euclidean-distance
17. skojaku/multilabel_knn: Multilabel classification algorithms based on k-nearest neighbor algorithms - GitHub, accessed August 11, 2025, https://github.com/skojaku/multilabel_knn
18. Learning Distance Metrics for Multi-Label Classification, accessed August 11, 2025, http://proceedings.mlr.press/v63/Gouk8.pdf
19. Multi-label classification performance metrics - Mastering Machine Learning with scikit-learn - Second Edition [Book] - O'Reilly Media, accessed August 11, 2025, https://www.oreilly.com/library/view/mastering-machine-learning/9781788299879/87b63eb8-f52c-496a-b73b-42f8aef549fb.xhtml
20. K-Nearest Neighbor Temporal Aggregate Queries - Rui Zhang, accessed August 11, 2025, https://www.ruizhang.info/publications/EDBT2015-KNNTA.pdf
21. TADILOF: Time Aware Density-Based Incremental Local Outlier Detection in Data Streams, accessed August 11, 2025, https://www.mdpi.com/1424-8220/20/20/5829
22. Learning Rate Decay - GeeksforGeeks, accessed August 11, 2025, https://www.geeksforgeeks.org/machine-learning/learning-rate-decay/
23. Real-World Examples of Exponential Decay - GeeksforGeeks, accessed August 11, 2025, https://www.geeksforgeeks.org/maths/real-world-examples-of-exponential-decay/
24. What is exponential decay in tf - ProjectPro, accessed August 11, 2025, https://www.projectpro.io/recipes/what-is-exponential-decay-tf
25. ExponentialDecay - Keras, accessed August 11, 2025, https://keras.io/api/optimizers/learning_rate_schedules/exponential_decay/
26. Application of the Weighted K-Nearest Neighbor Algorithm for Short-Term Load Forecasting, accessed August 11, 2025, https://www.mdpi.com/1996-1073/12/5/916
27. Correct implementation of weighted K-Nearest Neighbors - Stack Overflow, accessed August 11, 2025, https://stackoverflow.com/questions/49579819/correct-implementation-of-weighted-k-nearest-neighbors
28. Multivariate Correlation Analysis Technique Based on Euclidean Distance Map for Network Traffic Characterization - Edinburgh Napier University, accessed August 11, 2025, https://www.napier.ac.uk/research-and-innovation/research-search/outputs/multivariate-correlation-analysis-technique-based-on-euclidean-distance-map-for-network
29. Intrusion Detection System Using Euclidean Metrics and K ... - IJREAM, accessed August 11, 2025, http://ijream.org/papers/IJREAMV05I0246067.pdf

30. What is Euclidean distance and why is it important in machine learning? - EITCA Academy, accessed August 11, 2025, https://eitca.org/artificial-intelligence/eitc-ai-mlp-machine-learning-with-python/programming-machine-learning/euclidean-distance/examination-review-euclidean-distance/what-is-euclidean-distance-and-why-is-it-important-in-machine-learning/
31. Euclidean Distance Maximum Detection Rates versus Threshold Levels. - ResearchGate, accessed August 11, 2025, https://www.researchgate.net/figure/Euclidean-Distance-Maximum-Detection-Rates-versus-Threshold-Levels_fig4_248399900
32. Network Anomaly Detection for IoT Using Hyperdimensional Computing on NSL-KDD, accessed August 11, 2025, https://arxiv.org/html/2503.03031v1