readme.md for @push.rocks/smartfuzzy

> **Smart fuzzy matching for the modern developer** - Effortlessly match strings, sort objects, and search content with intelligent algorithms

A powerful TypeScript library that brings intelligent fuzzy matching to your applications. Whether you're building search features, autocomplete functionality, or data filtering systems, SmartFuzzy delivers the precision and flexibility you need.

□□ Features

- □□ **Precise String Matching** - Find closest matches in dictionaries with confidence scores
- □□ **Smart Object Sorting** - Sort objects by property similarity with customizable criteria
- □□ **Advanced Article Search** - Multi-field content search with intelligent weighting
- ⚡ **Lightning Fast** - Built on proven algorithms (Levenshtein distance + Fuse.js)
- □□ **TypeScript Native** - Full type safety and IntelliSense support
- □□ **Universal** - Works in Node.js and modern browsers

□□ Installation

Install using pnpm (recommended):

```
pnpm install @push.rocks/smartfuzzy
```

Or with your preferred package manager:

```
npm install @push.rocks/smartfuzzy
# or
yarn add @push.rocks/smartfuzzy
```

□□ Browser Compatibility

SmartFuzzy works in all modern environments:

Node.js

- **Node.js 16+** (ES2022 support required)
- Full TypeScript support with type definitions included

Browsers

- **Modern browsers** supporting ES2022 features
- Chrome 94+, Firefox 93+, Safari 15+, Edge 94+
- **No additional build setup required** - works with standard bundlers

TypeScript Setup

Ensure your `tsconfig.json` includes:

```
{
  "compilerOptions": {
    "target": "ES2022",
    "module": "NodeNext",
    "moduleResolution": "NodeNext",
    "esModuleInterop": true
  }
}
```

Bundle Size

- **Core library**: ~15KB minified + gzipped
- **Dependencies**: Fuse.js (~12KB), Leven (~2KB)
- **Total footprint**: ~29KB minified + gzipped

□□ Quick Start (30 seconds)

Get up and running with SmartFuzzy in under a minute:

```
import { Smartfuzzy } from '@push.rocks/smartfuzzy';

// 1. Create a fuzzy matcher
const fuzzy = new Smartfuzzy(['apple', 'banana', 'orange']);

// 2. Find the best match
const match = fuzzy.findClosestMatch('aple');  // Returns: 'apple'

// 3. That's it! □□
```

Need object searching? Use `ObjectSorter`:

```
import { ObjectSorter } from '@push.rocks/smartfuzzy';

const products = [{ name: 'iPhone' }, { name: 'Android' }];
const sorter = new ObjectSorter(products);
const results = sorter.sort('iphone', ['name']);
```

□□ Usage

SmartFuzzy is designed for developers who need intelligent matching without the complexity. Jump right in with these real-world examples!

□□ Quick Start

```
import { Smartfuzzy, ObjectSorter, ArticleSearch } from '@push.rocks/smartfuzzy';
```

□□ Smart String Matching

Perfect for autocomplete, spell-check, or finding the best match from a list:

```
const myDictionary = ['Sony', 'Deutsche Bahn', 'Apple Inc.', "Trader Joe's"];
const mySmartFuzzy = new Smartfuzzy(myDictionary);

// Adding additional dictionary entries
```

```
mySmartFuzzy.addToDictionary('Microsoft');
mySmartFuzzy.addToDictionary(['Google', 'Facebook']);

// Finding the closest match
const searchResult = mySmartFuzzy.findClosestMatch('Appl');
console.log(searchResult); // Output: "Apple Inc."

// Calculate similarity scores for all dictionary entries
const scores = mySmartFuzzy.calculateScores('Appl');
console.log(scores);
// Output: { 'Sony': 4, 'Deutsche Bahn': 11, 'Apple Inc.': 5, ... }
// Lower scores indicate better matches
```

This example demonstrates how to instantiate the `Smartfuzzy` class with a list of strings (dictionary) and add more entries to it. You can then use it to find the closest match or calculate similarity scores for a given search string.

□□ Intelligent Object Sorting

Transform any object array into a smart, searchable dataset:

```
interface ICar {
  brand: string;
  model: string;
}

const carList: ICar[] = [
  { brand: 'BMW', model: 'M3' },
  { brand: 'Mercedes Benz', model: 'E-Class' },
  { brand: 'Volvo', model: 'XC90' },
];

const carSorter = new ObjectSorter<ICar>(carList);

// Search and sort based on brand similarity
const searchResults = carSorter.sort('Benz', ['brand']);
console.log(searchResults); // Results will be sorted by relevance to 'Benz'
```

This scenario shows how to use `ObjectSorter` for sorting an array of objects based on how closely one of their string properties matches a search term. This is particularly useful for filtering or

autocomplete features where relevance is key.

□□ Powerful Content Search

Build sophisticated search experiences for articles, blog posts, or any content with multiple fields:

```
import { IArticle } from '@tsclass/tsclass/content';

const articles: IArticle[] = [
  {
    title: 'History of Berlin',
    content: 'Berlin has a rich history...',
    tags: ['history', 'Berlin'],
    timestamp: Date.now(),
    featuredImageUrl: null,
    url: null,
  },
  {
    title: 'Tourism in Berlin',
    content: 'Discover the vibrant city of Berlin...',
    tags: ['travel', 'Berlin'],
    timestamp: Date.now(),
    featuredImageUrl: null,
    url: null,
  },
];

const articleSearch = new ArticleSearch(articles);

// Perform a search across titles, content, and tags
const searchResult = await articleSearch.search('rich history');
console.log(searchResult); // Array of matches with relevance to 'rich history'
```

The `ArticleSearch` class showcases how to implement a search feature across a collection of articles with prioritization across different fields (e.g., title, content, tags). This ensures more relevant search results and creates a better experience for users navigating through large datasets or content libraries.

□□ Real-World Use Cases

Search-as-You-Type

Build responsive search experiences:

```
import { Smartfuzzy } from '@push.rocks/smartfuzzy';

const cities = ['New York', 'Los Angeles', 'Chicago', 'Houston', 'Phoenix'];
const citySearch = new Smartfuzzy(cities);

// User types "new yo"
const suggestions = citySearch.calculateScores('new yo');
// Returns: { 'New York': 2, 'Los Angeles': 8, ... }

// Show top 3 suggestions
const topSuggestions = Object.entries(suggestions)
  .sort(([,a], [,b]) => a - b)
  .slice(0, 3)
  .map(([city]) => city);
```

Data Deduplication

Clean up messy datasets:

```
import { ObjectSorter } from '@push.rocks/smartfuzzy';

const contacts = [
  { name: 'John Smith', email: 'john@example.com' },
  { name: 'Jon Smith', email: 'jon.smith@example.com' },  // Likely duplicate
  { name: 'Jane Doe', email: 'jane@example.com' }
];

const sorter = new ObjectSorter(contacts);

// Find potential duplicates for each contact
contacts.forEach(contact => {
```

```
  const matches = sorter.sort(contact.name, ['name']);
  if (matches.length > 1 && matches[0].score < 0.3) {
    console.log(`Potential duplicate: ${contact.name} ↔ ${matches[1].item.name}`);
  }
});
```

Smart Product Search

E-commerce search with typo tolerance:

```
import { ObjectSorter } from '@push.rocks/smartfuzzy';

const products = [
  { name: 'iPhone 15 Pro', category: 'Electronics', brand: 'Apple' },
  { name: 'MacBook Air', category: 'Computers', brand: 'Apple' },
  { name: 'AirPods Pro', category: 'Audio', brand: 'Apple' }
];

const productSearch = new ObjectSorter(products);

// User searches "macbok air" (with typos)
const results = productSearch.sort('macbok air', ['name', 'brand']);
// Correctly finds "MacBook Air" despite typos
```

Recommendation System

Content-based recommendations:

```
import { ArticleSearch } from '@push.rocks/smartfuzzy';

const articles = [
  { title: 'React Hooks Guide', tags: ['react', 'javascript'], content: '...' },
  { title: 'Vue.js Tutorial', tags: ['vue', 'javascript'], content: '...' },
  { title: 'Angular Components', tags: ['angular', 'typescript'], content: '...' }
];

const articleSearch = new ArticleSearch(articles);
```

```
// User reads about React, find similar content
const similar = await articleSearch.search('react javascript hooks');
// Returns articles ordered by relevance
```

□□ Error Handling

SmartFuzzy provides clear error messages and graceful degradation:

Input Validation

```
import { Smartfuzzy } from '@push.rocks/smartfuzzy';

const fuzzy = new Smartfuzzy(['apple', 'banana']);

try {
  // □ This will throw an error
  const result = fuzzy.findClosestMatch(123 as any);
} catch (error) {
  console.error('Error:', error.message); // "Input must be a string"
}
```

Graceful Degradation

```
// Empty dictionary returns null instead of throwing
const emptyFuzzy = new Smartfuzzy([]);
const result = emptyFuzzy.findClosestMatch('test'); // Returns: null

// Empty object array returns empty results
const emptyObjectSorter = new ObjectSorter([]);
const results = emptyObjectSorter.sort('test', ['name']); // Returns: []
```

Best Practices

```
import { Smartfuzzy, ObjectSorter } from '@push.rocks/smartfuzzy';

// □ Always validate your inputs
function safeSearch(query: unknown, dictionary: string[]) {
  if (typeof query !== 'string') {
    return null; // Or throw a custom error
  }

  if (!Array.isArray(dictionary) || dictionary.length === 0) {
    return null;
  }

  const fuzzy = new Smartfuzzy(dictionary);
  return fuzzy.findClosestMatch(query);
}

// □ Handle async operations properly
async function searchArticles(query: string, articles: IArticle[]) {
  try {
    const search = new ArticleSearch(articles);
    const results = await search.search(query);
    return results;
  } catch (error) {
    console.error('Search failed:', error);
    return []; // Return empty results on error
  }
}
```

□□ API Reference

Smartfuzzy Class

The core fuzzy matching class for string dictionaries.

Constructor

```
new Smartfuzzy(dictionary?: string[])
```

- **dictionary** (optional): Array of strings to search against

Methods

`findClosestMatch(searchString: string): string | null`

Find the best matching string from the dictionary.

- **searchString**: String to find a match for
- **Returns**: Best match or `null` if no match found
- **Throws**: Error if input is not a string

`calculateScores(searchString: string): TDictionaryMap`

Calculate similarity scores for all dictionary entries.

- **searchString**: String to score against
- **Returns**: Object mapping dictionary words to their scores (lower = better)

`addToDictionary(items: string | string[]): void`

Add new entries to the search dictionary.

- **items**: Single string or array of strings to add

ObjectSorter<T> Class

Generic object sorting with fuzzy matching on specified properties.

Constructor

```
new ObjectSorter<T>(objects?: T[])
```

- **objects** (optional): Array of objects to search within

Methods

`sort(searchString: string, keys: string[]): IFuzzySearchResult<T>[]`

Sort objects by property similarity to search string.

- **searchString**: String to match against object properties
- **keys**: Array of object property names to search within
- **Returns**: Array of matches sorted by relevance

- **Throws**: Error for invalid inputs

`IFuzzySearchResult<T>` Interface

```
interface IFuzzySearchResult<T> {
  item: T;              // The matched object
  refIndex: number;     // Original array index
  score?: number;       // Match score (lower = better)
}
```

ArticleSearch Class

Specialized search for article content with intelligent field weighting.

Constructor

```
new ArticleSearch(articles?: IArticle[])
```

- **articles** (optional): Array of articles to search

Methods

`search(searchString: string): Promise<IArticleSearchResult[]>`

Perform weighted search across article fields.

- **searchString**: Query to search for
- **Returns**: Promise resolving to array of matched articles
- **Field Weights**: Title (3x), Tags (2x), Content (1x)

`addArticle(article: IArticle): void`

Add a single article to the search collection.

- **article**: Article object to add

`IArticleSearchResult` Interface

```
interface IArticleSearchResult {
  item: IArticle;       // The matched article
  refIndex: number;     // Original array index
  score?: number;       // Match score
```

```
  matches?: Array<{     // Match details
    indices: Array<[number, number]>;
    key?: string;
    value?: string;
  }>;
}
```

⚡ Performance Guide

Time Complexity

- **Smartfuzzy.findClosestMatch**: O(n × m) where n = dictionary size, m = average string length
- **ObjectSorter.sort**: O(n × k × m) where k = number of keys to search
- **ArticleSearch.search**: O(n × f × m) where f = number of fields (title, content, tags)

Recommended Dataset Sizes

- **Small (< 1,000 items)**: Excellent performance, sub-millisecond responses
- **Medium (1,000 - 10,000 items)**: Good performance, 1-10ms responses
- **Large (10,000+ items)**: Consider chunking or server-side search for real-time UIs

Optimization Tips

1. Reuse Instances

```
// ✅ Good: Reuse the same instance
const fuzzy = new Smartfuzzy(largeDictionary);
const result1 = fuzzy.findClosestMatch('query1');
const result2 = fuzzy.findClosestMatch('query2');

// ❌ Avoid: Creating new instances repeatedly
const result1 = new Smartfuzzy(largeDictionary).findClosestMatch('query1');
const result2 = new Smartfuzzy(largeDictionary).findClosestMatch('query2');
```

2. Batch Operations

```
// ✅ Good: Calculate scores once, extract multiple matches
const scores = fuzzy.calculateScores('query');
const topMatches = Object.entries(scores)
  .sort(([,a], [,b]) => a - b)
  .slice(0, 5);

// ❌ Avoid: Multiple separate lookups
const match1 = fuzzy.findClosestMatch('query');
const match2 = fuzzy.findClosestMatch('query'); // Duplicate work
```

3. Optimize Search Keys

```
// ✅ Good: Search only necessary fields
const results = sorter.sort('query', ['name']); // Fast

// ❌ Avoid: Searching unnecessary fields
const results = sorter.sort('query', ['name', 'description', 'notes']); // Slower
```

4. Memory Management

```
// For very large datasets, consider chunking
function chunkedSearch(query: string, largeArray: any[], chunkSize = 1000) {
  const results = [];

  for (let i = 0; i < largeArray.length; i += chunkSize) {
    const chunk = largeArray.slice(i, i + chunkSize);
    const sorter = new ObjectSorter(chunk);
    results.push(...sorter.sort(query, ['name']));
  }

  return results.sort((a, b) => a.score - b.score);
}
```

Advanced Configuration

Custom Fuse.js Options

Current Implementation: The Fuse.js options are optimized for general use cases:

```
// Default configuration in SmartFuzzy
const fuseOptions = {
  shouldSort: true,
  threshold: 0.6,        // 0.0 = exact match, 1.0 = match anything
  location: 0,           // Start position for search
  distance: 100,         // Search distance from location
  maxPatternLength: 32,  // Maximum pattern length
  minMatchCharLength: 1  // Minimum match character length
};
```

Configuration Guidelines:

- **threshold: 0.0-1.0** - Lower values require closer matches
- **distance** - How far from `location` to search
- **location** - Where in the string to start searching (0 = beginning)

Custom Matching Behavior

While direct configuration isn't exposed yet, you can achieve custom behavior:

```
// For stricter matching, filter results by score
const fuzzy = new Smartfuzzy(['apple', 'application', 'apply']);
const scores = fuzzy.calculateScores('app');

// Only accept very close matches (score < 2)
const strictMatches = Object.entries(scores)
  .filter(([word, score]) => score < 2)
  .sort(([,a], [,b]) => a - b);

// For more lenient matching, use a higher threshold in your logic
const lenientMatches = Object.entries(scores)
  .filter(([word, score]) => score < 5)
  .sort(([,a], [,b]) => a - b);
```

Article Search Weighting

The ArticleSearch class uses intelligent field weighting:

```
// Built-in weighting (not directly configurable)
const searchWeights = {
  title: 3,    // Highest priority - titles are most important
```

```
  tags: 2,     // Medium priority - tags are descriptive
  content: 1   // Lower priority - content can be lengthy
};

// This means a match in the title has 3x more relevance than content
```

□□ Why Choose SmartFuzzy?

- □□ **Intelligent**: Uses proven algorithms for accurate matching
- ⚡ **Fast**: Optimized for performance in real-world applications
- □□ **Flexible**: Adapts to your specific use cases and data structures
- □□ **Reliable**: Comprehensive test coverage and TypeScript safety
- □□ **Well-Documented**: Clear examples and complete API documentation

□□ Troubleshooting & FAQ

Common Issues

"Cannot find module" errors

```
# Ensure you've installed the package
pnpm install @push.rocks/smartfuzzy

# For TypeScript projects, types are included automatically
```

Poor matching results

```
// If matches seem inaccurate, check your input data
const fuzzy = new Smartfuzzy(['apple', 'APPLE', 'Apple']);
// Consider normalizing case before adding to dictionary
const normalizedDict = ['apple', 'banana', 'orange'].map(s => s.toLowerCase());
const fuzzy2 = new Smartfuzzy(normalizedDict);
```

Performance issues with large datasets

```
// For > 10,000 items, consider limiting search scope
const scores = fuzzy.calculateScores('query');
const topResults = Object.entries(scores)
  .sort(([,a], [,b]) => a - b)
  .slice(0, 10); // Only get top 10 results
```

FAQ

Q: Can I search case-insensitively? A: SmartFuzzy is case-sensitive by default. Normalize your data:

```
const fuzzy = new Smartfuzzy(dict.map(s => s.toLowerCase()));
const result = fuzzy.findClosestMatch(query.toLowerCase());
```

Q: How do I handle special characters? A: Fuse.js handles Unicode well, but you may want to normalize:

```
const normalize = (str: string) => str.normalize('NFD').replace(/[\u0300-\u036f]/g, '');
```

Q: Can I weight object properties differently? A: Currently not directly configurable, but you can post-process results:

```
const results = sorter.sort(query, ['name', 'description']);
// Boost results that matched 'name' field
const boosted = results.map(r => ({
  ...r,
  score: r.matches?.some(m => m.key === 'name') ? r.score * 0.5 : r.score
}));
```

Q: What's the difference between `findClosestMatch` and `calculateScores`? A: `findClosestMatch` returns only the best match, while `calculateScores` returns scores for all dictionary entries, letting you implement custom ranking logic.

Q: How do I handle empty results? A: Always check for null/empty returns:

```
const match = fuzzy.findClosestMatch('query');
if (match === null) {
  console.log('No suitable match found');
}
```

Get Started Today

Ready to add intelligent search to your application? SmartFuzzy makes it easy:

1. Install the package
2. Import the classes you need
3. Start matching, sorting, and searching!

Perfect for building search bars, recommendation systems, data filters, and more.

License and Legal Information

This repository contains open-source code that is licensed under the MIT License. A copy of the MIT License can be found in the license file within this repository.

Please note: The MIT License does not grant permission to use the trade names, trademarks, service marks, or product names of the project, except as required for reasonable and customary use in describing the origin of the work and reproducing the content of the NOTICE file.

Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH and are not included within the scope of the MIT license granted herein. Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines, and any usage must be approved in writing by Task Venture Capital GmbH.

Company Information

Task Venture Capital GmbH
Registered at District court Bremen HRB 35230 HB, Germany

For any legal inquiries or if you require further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.

Revision #3
Created 2026-03-28 11:10:38 UTC by foss.global Team

Updated 2026-03-28 12:17:27 UTC by foss.global Team