Klasyfikator k-NN z szybkim indeksem sąsiadów k-d tree

Raport z projektu z przedmiotu Inteligentne Usługi Interaktywne

Jan Trusiłło (s180141@student.pg.edu.pl)

2024-01-26

Spis treści

Drzewo k-d

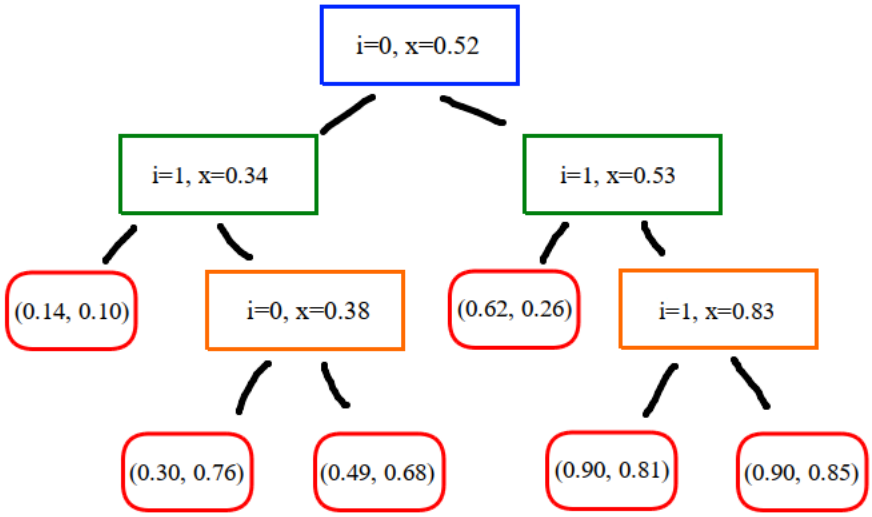
Drzewo k-d (k-d tree, z ang. drzewo k-wymiarowe) jest strukturą danych przeznaczoną do partycjonowania przestrzeni. Drzewo k-d ma postać drzewa binarnego, którego liśćmi są punkty k-wymiarowe.

Dla każdego węzła wewnętrznego drzewa k-d istnieje prostopadła do jednej z osi hiperpłaszczyzna, która dzieli punkty należące do lewego i prawego poddrzewa. Formalnie, każdy węzeł wewnętrzny k-d drzewa wyznacza hiperpłaszczyznę wyrażoną równaniem $v_i = x$, dla której dla dowolnego punktu l z lewego poddrzewa oraz dowolnego punktu r z prawego poddrzewa spełniona jest zależność

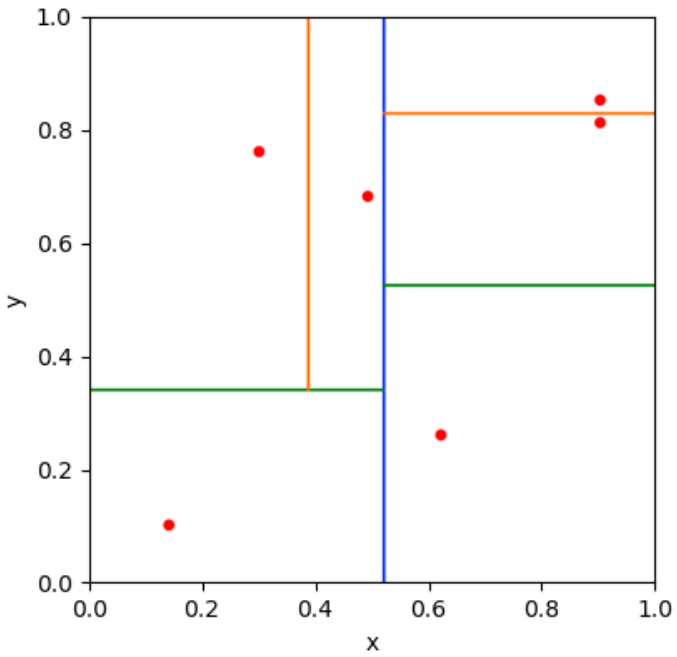
$l_i < x \leq r_i$.

Przykładowo, rozważmy zbiór 2-wymiarowych punktów:
$\{(0.90, 0.81), (0.14, 0.10), (0.49, 0.68), (0.30, 0.76), (0.90, 0.85), (0.62, 0.26)\}$

Jednym z możliwych drzew k-d dla tego zbioru jest:



Podział przestrzeni zdefiniowany przez to drzewo można zwizualizować następująco (kolory linii podziału odpowiadają kolorom węzłów w powyższym rysunku drzewa):



Ideą przyświecającą użyciu drzewa k-d dla zastosowań klasyfikatora k-NN jest możliwość szybkiego odrzucenia dużej liczby nieinteresujących nas punktów podczas szukania. Przykładowo, szukając w powyższym drzewie punktów oddalonych od punktu $(0.90, 0.81)$ o co najwyżej 0.15, możemy z pewnością stwierdzić, że nie interesują nas punkty z wartością pierwszej współrzędnej mniejszej niż 0.52, zatem możemy od razu odrzucić całe lewe poddrzewo korzenia.

Klasyfikator k-NN

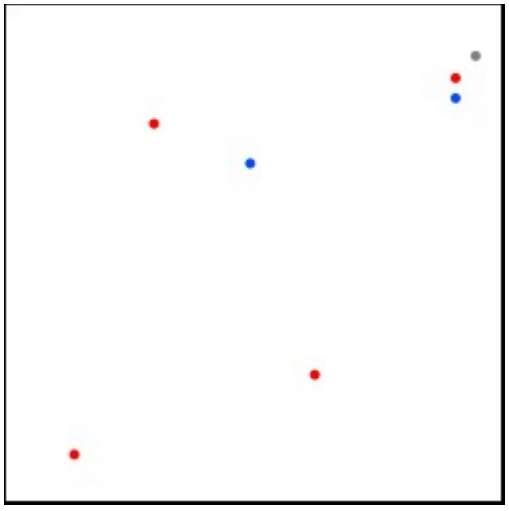
Jedną z najprostszych metod klasyfikacji jest metoda *k* najbliższych sąsiadów (k-NN). Zakłada ona istnienie zbioru uczącego, tj. zbioru punktów w n-wymiarowej przestrzeni o znanej klasie. Klasyfikacja nowego punktu polega na znalezieniu *k* najbliższych do niego punktów zbioru uczącego oraz wybraniu na ich podstawie najprawdopodobniejszej etykiety poprzez majority voting.

Rozważany dalej wariant klasyfikatora k-NN można zdefiniować formalnie jako piątkę (n, X, l, f, k), gdzie:

- n - liczba wymiarów przestrzeni,
- X - zbiór n-wymiarowych punktów uczących,
- $l : \mathbb{R}_n \to \mathbb{N}_+$ - funkcja przypisania klasy dla punktu uczącego
- $d : (\mathbb{R}_n, \mathbb{R}_n) \to \mathbb{R}$ - metryka odległości
- k - liczba najbliższych sąsiadów branych pod uwagę przy klasyfikacji, często jest to mała nieparzysta wartość (np. 3)

Rozważmy punkt *v*, który chcemy sklasyfikować. Przy założeniu, że $k \leq |X|$, istnieje zbiór $X_v = \{p \in X : d(p, v) < r\}$ dla pewnej najmniejszej wartości *r* takiej, że $|X_v| \geq k$. Prognozowaną klasą punktu *v* jest wówczas wartość *c* maksymalizująca moc zbioru $\{p \in X_v : l(p) = c\}$. Jeżeli takich wartości jest wiele, klasa wynikowa jest rozstrzygana poprzez np. minimalizację sumy odległości.

Jako przykładowy zbiór uczący przyjmijmy zbiór punktów z powyższego przykładu drzewa k-d:



Oprócz zbioru uczącego zauważmy punkt szary: naszym zadaniem jest przypisać do niego klasę. Dla $k = 1$ zostanie on przypisany do klasy czerwonej, gdyż najbliższy mu punkt należy do tejże klasy. Dla $k = 2$ przy użyciu strategii minimalizacji sumy odległości dla równolicznych grup punkt również zostanie przypisany do czerwonej klasy. Jednak dla $k = 3$ punkt zostanie przypisany do klasy niebieskiej, gdyż 2 z 3 najbliższych mu punktów zbioru uczącego należy do klasy niebieskiej.

Biblioteka

W ramach projektu zaimplementowano bibliotekę header-only w języku C++20, która dostarcza generyczną klasę k-d tree oraz korzystający z niej klasyfikator k-NN. Biblioteka dostępna jest na publicznym repozytorium GitHub: https://github.com/311Volt/iui_kdtree

Użytkowanie

Aby przygotować dane dla biblioteki, należy zdefiniować strukturę obserwacji:

```
struct Observation {
	std::array<float, 3> position;
	int label;
}
```

Mając dowolny zakres obserwacji (w postaci np. tablicy lub dowolnego kontenera STL), można stworzyć klasyfikator:

```
std::vector<Observations> observations;
// wypełnienie observations...

iui::KNNClassifier<iui::EuclideanDistanceMetric, int, float, 3>
	classifier(observations);
```

Przykładowy klasyfikator korzysta z euklidesowej metryki odległości, typ etykiety to int, typ elementu wektora obserwacji to float oraz wektor obserwacji jest 3-wymiarowy.

Mając obserwację testową, można użyć klasyfikatora:

```
std::array<float, 3> testPoint = {0.3f, 0.1f, 0.15f};
auto predictedLabel = classifier.predict(testPoint);
```

Metoda predict posiada opcjonalne parametry: k (domyślnie 3), początkowy promień poszukiwań (domyślnie 1e-6) oraz poprawną etykietę (w wypadku jej podania klasyfikator zbiera dane o swojej skuteczności)

Praktyczny przykład użycia biblioteki znajduje się w załączonym kodzie w pliku main.cpp (funkcja simpleUsageExample()).

Implementacja drzewa k-d

Szablony klasy KDTree (kdtree.hpp) implementuje zmodyfikowany wariant drzewa k-d, w którym liść może zawierać więcej niż jeden element.

Struktura drzewa

Drzewo posiada strukturę optymalizującą liczbę alokacji podczas konstrukcji oraz ograniczającą liczbę cache missów podczas jego odwiedzania. Na k-d drzewo składa się poniższe:

```
HyperboxType rootHyperbox_;
std::deque<Node> nodes_;
std::vector<EntryType> entries_;
```

Pola te można rozumieć następująco:

- rootHyperbox_ - hiperprostopadłościan ograniczający cały dataset uczący, zdefiniowany dwoma skrajnymi punktami
- nodes_ - węzły drzewa. Wybrano kontener std::deque ze względu na wydajność (na bibliotekach innych niż MSVC STL) oraz gwarantowaną stabilność referencji, która jest wymagana, gdyż elementy listy węzłów zawierają referencje do samych siebie.
- entries_ - wszystkie rekordy (pary punkt + etykieta). Dzięki przechowywaniu rekordów w sposób ciągły możliwe jest wydajne partycjonowanie drzewa oraz przechowywanie liści jako para wskaźnik+rozmiar (std::span) na zakres rekordów.

Węzeł drzewa jest zdefiniowany w następujący sposób:

```
struct Node {
	struct InnerNode {
		HyperboxSplitType split;
		Node* lchild;
		Node* rchild;
	};

	Node* parent;
	std::variant<std::span<const EntryType>, InnerNode> data;
};
```

Pomijając wskaźnik na rodzica (używany w praktyce tylko do debugu) węzeł jest de facto unią o dwóch alternatywach:

- liść - zakres rekordów (par punkt + etykieta) zbioru uczącego
- węzeł wewnętrzny, składający się z:
  - podziału hiperprostopadłościanu (para indeks osi + wartość)
  - wskaźników na lewe i prawe dziecko

Konstrukcja drzewa

Konstrukcję drzewa można w sposób przybliżony opisać następującym pseudokodem:

```
def createNode(int firstEntry, int lastEntry) -> Node:
	if lastEntry - firstEntry < leafThreshold
		return Leaf(firstEntry, lastEntry)

	split := findSplit(firstEntry, lastEntry)
	middleEntry := partition(firstEntry, lastEntry, split)

	return InnerNode(
		split = split,
		leftChild = createNode(firstEntry, middleEntry)
		rightChild = createNode(middleEntry, lastEntry)
	)

rootNode := createNode(0, entries.length())
```

Podsumowując, węzeł powstaje z zakresu rekordów, dla których znajdowany jest podział hiperprostopadłościanu, następnie rekordy są partycjonowane w miejscu według tego podziału i z powstałych dwóch podzakresów rekurencyjnie tworzone są poddrzewa.

Strategia równoważenia

Drzewo stosuje przybliżoną, zachłanną strategię równoważenia. Dla każdego węzła wewnętrznego losowane jest $2+2\log_2(n)$ indeksów wymiarów, a następnie dla każdego takiego indeksu i wykonywana jest następująca procedura:

1. Znajdź medianę wartości *i*-tej współrzędnej punktów w zadanym zakresie.
2. Dokonaj partycjonowania rekordów na 2 podzakresy według znalezionej mediany (tj. wg predykatu $v[i] < mediana$).
3. Dokonaj oceny partycjonowania w postaci liczby od 0 do 1, gdzie 0 oznacza najbardziej niezrównoważone podzakresy (możliwe tylko wtedy, gdy *i*-ta współrzędna wszystkich punktów jest identyczna), a 1 możliwie najlepiej zrównoważone zakresy (np. 39 → 19+20)
4. Jeżeli ocena przekracza ustalony próg (domyślnie 0.9), przerwij szukanie i zwróć podział równy $(i, mediana)$.
5. Zapisz otrzymany wynik (tj. parę podział + ocena) do listy prób.

Jeżeli pętla nie została wcześniej przerwana, wybierany jest podział z listy prób, dla którego wynik był najlepszy. Jeżeli jednak najlepszy wynik wyniósł 0, drzewo nie jest dzielone dalej i zostaje utworzony liść z zadanego zakresu elementów.

Wartość graniczna dla podziału

Jeżeli zakres elementów w poddrzewie nie obejmuje więcej niż 2 elementy bądź 128 bajtów (2 cache line'y na x86), nie odbywa się dalsze dzielenie drzewa. Optymalizacja ta jest podyktowana dwoma czynnikami:

- opłacalność pamięciowa - dla niższych wymiarowości zabieg ten znacznie zmniejsza rozmiar danych struktury drzewa w pamięci (dla testowanych danych zaobserwowano średnio 2.23-krotne zmniejszenie)
- opłacalność obliczeniowa - wczytanie 2-3 cache line'ów punktów z pamięci i obliczenie dla nich odległości jest szybsze niż odwiedzenie kilku dodatkowych dalekich od siebie w pamięci węzłów drzewa.

Test na datasecie z ostatniego rozdziału wykazał skuteczność tej optymalizacji: klasyfikator dla 3-wymiarowych problemów działa dzięki niej średnio o 60% szybciej, pomimo tego, że średnio odwiedza 3-krotnie więcej punktów.

Metoda walk()

Najważniejszą metodą klasy KDTree jest walk(fn, hboxPredicate), która przyjmuje 2 funktory jako argumenty: pierwszy to funkcja, która wykona się na każdym znalezionym punkcie, drugi zaś to predykat na hiperprostopadłościanie ograniczającym poddrzewo, dzięki któremu można odrzucić nieinteresujące nas poddrzewa. Metoda realizuje rekurencyjne odwiedzanie drzewa poprzez

wykorzystanie obiektu odwiedzającego (visitor), którego stanem jest aktualny hiperprostopadłościan.

Implementacja klasyfikatora k-NN

Klasyfikator k-NN przyjmuje z góry znany dataset oraz na etapie konstrukcji tworzy odpowiednie k-d tree.

Miary odległości

Miara odległości jest parametrem szablonu klasyfikatora. Aby zdefiniować klasę miary odległości, musi ona dostarczać 2 statyczne metody:

- `DistanceType distance(PointType a, PointType b)` → odległość dwóch punktów,
- `bool intersectsSearchSpace(HyperboxType hyperbox, PointType point, DistanceType maxDistance)` → predykat mówiący o tym, czy w hiperprostopadłościanie `hyperbox` może wystąpić punkt oddalony od `point` o nie więcej niż `maxDistance` - używane do szybkiego indeksowania

W bibliotecę zaimplementowano jedynie generyczną miarę Minkowskiego. `EuclideanDistanceMetric` oraz `ManhattanDistanceMetric` są aliasami na instancje szablonu `MinkowskiDistanceMetric` o parametrach odpowiednio 2 i 1. Należy wspomnieć, że pomimo faktu, że kompilator jest w stanie zoptymalizować np. pow(x, 0.5) na sqrt() oraz pow(x, 4) na (x*x)*(x*x), może on to zrobić jedynie z użyciem flagi -ffast-math, przez co traci zgodność z IEEE 754. Aby nie zmuszać użytkowników do użycia tej flagi, zdefiniowano szablony funkcji do pierwiastkowania i potęgowania, które zawierają specjalne przypadki dla małych N.

Redukcja wymiarowości

Opcjonalnym parametrem szablonu KNNClassifier jest reduktor wymiarowości oraz docelowa wymiarowość. Reduktor wymiarowości musi dostarczać konstruktor tworzący wewnętrzny stan z listy wektorów oraz metodę reduce przetwarzającą wektor o wymiarowości źródłowej na wektor o wymiarowości docelowej.

W bibliotece zaimplementowano jeden reduktor wymiarowości oparty na PCA. Wymaga on zainstalowanej biblioteki Eigen i jest on opcjonalny.

Metoda klasyfikacji

Metoda predict() klasyfikatora, służąca do klasyfikacji, ma za zadanie znaleźć *k* najbliższych do zadanego punktów w zbiorze uczącym oraz podjąć decyzję co do prognozowanej klasy.

Odbywa się to w następujących krokach:

1. Dobierany jest promień poszukiwań. Jeżeli użytkownik nie podał własnego, wybierane jest 10^{-6} lub dwukrotność odległości najdalszego punktu, który miał wpływ na wynik przy poprzedniej klasyfikacji.
2. Przy pomocy drzewa k-d szukane są wszystkie punkty, które są oddalone o nie więcej niż promień poszukiwań. W wypadku odwiedzenia punktów bardziej odległych odrzuca się je.

3. Jeżeli nie znaleziono co najmniej k takich punktów, promień poszukiwań jest podwajany i następuje powrót do punktu 2.
4. Zbiór k najbliższych znalezionych punktów jest zbiorem kandydatów.
5. Listę kandydatów przetwarza się na listę wyników - trójki (f, d, l), gdzie f to liczba wystąpień klasy, d to liczba odwrotna do sumy odległości, a l to powiązana etykieta.
6. Zwracana jest etykieta dla największego w porządku leksykograficznym elementu listy wyników.

Przykładowo, jeżeli klasyfikujemy punkt przy k=3 i lista kandydatów składa się z:

- punktu o klasie A odległego o 0.12,
- punktu o klasie A odległego o 0.25,
- punktu o klasie B odległego o 0.17

to lista wyników z niej utworzona ma następujące elementy:

- $(2, -0.37, A)$
- $(1, -0.17, B)$

Ponieważ 2 > 1, zwrócona etykieta wyniesie A.

Wyniki i benchmark

Wprowadzenie

Przetestowano otrzymane rozwiązanie na datasecie obrazów ręcznie pisanych cyfr MNIST.

Cechy datasetu:

- rozmiar zbioru uczącego: 5000
- rozmiar zbioru walidacyjnego: 500
- wymiarowość: 784

Kolumny poniższych tabel należy rozumieć następująco:

- n - liczba wymiarów, w których dokonuje się klasyfikacji (oznaczenie PCA oznacza użycie klasyfikatora z zastosowaniem reduktora wymiarowości iui::PrincipalComponentAnalysis)
- k - liczba najbliższych sąsiadów, na podstawie których dokonuje się predykcji (parametr k w k-NN)
- Accuracy - procent poprawnych klasyfikacji.
- Efficiency - procent punktów zbioru uczącego, które pominięto dzięki indeksowi k-d tree. Wartości zbliżone do 100% świadczą o opłacalności zastosowania indeksu k-d tree.
- Czas konstrukcji - czas spędzony na konstrukcji klasyfikatora, zanim będzie on gotowy do użytku - wlicza się w niego czas konstrukcji k-d drzewa i ew. reduktora wymiarowości.
- Czas testu - czas spędzony na klasyfikacji wszystkich elementów zbioru walidacyjnego.

Wszystkie testy zostały przeprowadzone w następującym środowisku:

- CPU: AMD Ryzen 7 3700X (taktowanie fabryczne 3.6 GHz)
- RAM: 32GB DDR4 @ 3200 MHz
- OS: Arch Linux (Linux 6.7.0, glibc 2.38)

Program testowy został skompilowany kompilatorem GCC 13.2 z odpowiadającą mu wersją libstdc++.

Wyniki

Cyfry MNIST, metryka Manhattan

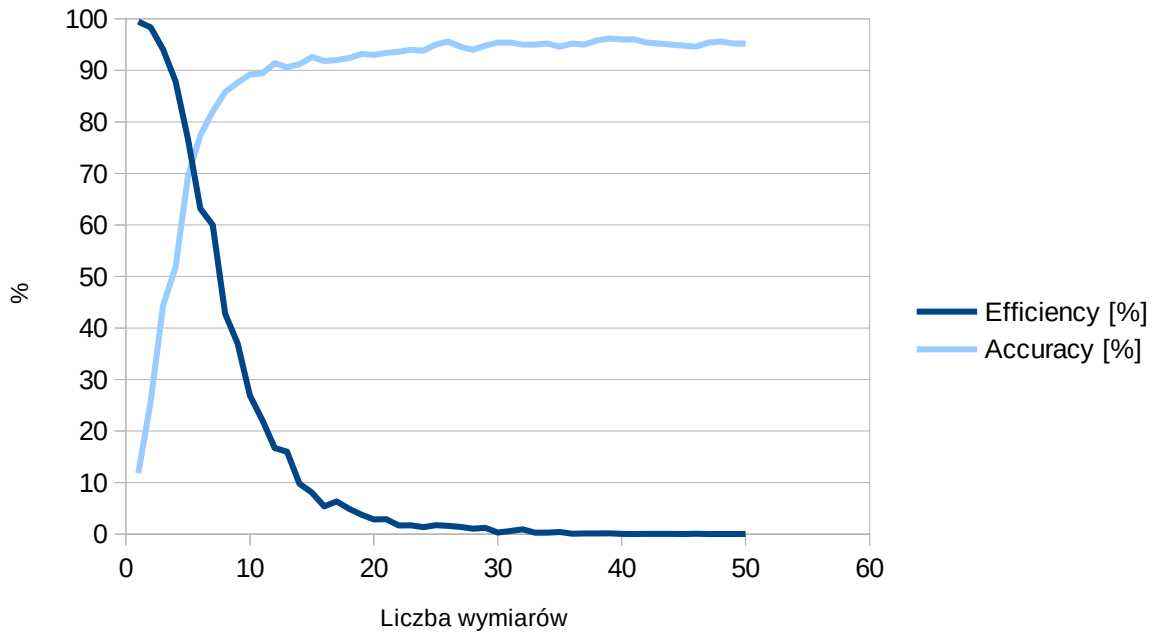
n	k	Accuracy [%]	Efficiency [%]	Czas konstrukcji [ms]	Czas testu [ms]
3 (PCA)	1	42.40	94.05	515.89	2.49
3 (PCA)	3	41.40	93.00	497.35	2.85
3 (PCA)	5	38.40	91.14	496.42	3.52
8 (PCA)	1	83.20	42.82	495.82	41.55
8 (PCA)	3	78.40	29.09	497.68	52.17
8 (PCA)	5	71.20	24.09	499.79	55.70
16 (PCA)	1	91.40	2.58	500.27	143.65
16 (PCA)	3	88.00	0.78	502.63	149.06
16 (PCA)	5	82.80	0.49	502.22	151.16
72 (PCA)	1	93.40	0.00	516.40	378.96
72 (PCA)	3	90.80	0.00	515.44	373.49
72 (PCA)	5	85.60	0.00	515.29	371.92
784	1	93.40	0.00	64.75	2363.20
784	3	89.40	0.00	82.80	2577.20
784	5	85.40	0.00	94.55	2549.61

Cyfry MNIST, metryka euklidesowa

n	k	Accuracy [%]	Efficiency [%]	Czas konstrukcji [ms]	Czas testu [ms]
3 (PCA)	1	43.80	95.84	501.42	2.01
3 (PCA)	3	44.40	94.61	504.37	2.47
3 (PCA)	5	38.60	93.40	505.68	2.97
8 (PCA)	1	82.20	63.17	506.29	28.15
8 (PCA)	3	78.40	50.51	503.43	38.19
8 (PCA)	5	72.60	43.79	502.57	43.44
16 (PCA)	1	92.00	18.49	505.13	100.84
16 (PCA)	3	88.80	10.23	503.54	112.23
16 (PCA)	5	84.20	9.25	504.84	114.84
72 (PCA)	1	94.20	0.14	517.42	382.92
72 (PCA)	3	92.00	0.04	518.15	377.82
72 (PCA)	5	87.60	0.04	516.88	378.40
784	1	94.40	0.00	72.44	2309.89
784	3	90.40	0.00	107.92	2224.98
784	5	86.00	0.00	83.61	2428.73

Efekt redukcji wymiarowości

Przeprowadzono test trafności i efektywności dla metryki euklidesowej dla każdej z liczby wymiarów w zakresie $[1;50]$. Wyniki przedstawiono na wykresie poniżej:



Wnioski

Indeksowanie przy użyciu drzewa k-d oraz zastosowane optymalizacje sprawiają, że po redukcji do 3 wymiarów udało się uzyskać wynik ok. 4 µs na pojedynczą klasyfikację. Osiągnięty wskaźnik efektywności (efficiency) świadczy o wysokiej opłacalności zastosowania drzewa k-d, jednak 3 wymiary okazują się niewystarczające dla problemu klasyfikacji cyfr, gdyż trafność takiego klasyfikatora jest niezadowalająca.

Dla wyższych wymiarowości zauważalny jest gwałtowny spadek opłacalności drzewa k-d. Jest to klasyczny przypadek tzw. klątwy wymiarowości. Dla badanego problemu drzewo k-d przestaje być opłacalne mniej więcej w momencie, w którym trafność klasyfikatora osiąga wartość porównywalną do tej osiąganej przez klasyfikator niewyposażony w reduktor wymiarowości.