جستجوی k نزدیک‌ترین همسایه تقریبی با روش ترکیب خطی

وحیده منعمی‌زاده [۱]، دانشجوی کارشناسی ارشد؛ جواد حمیدزاده[۲]، استادیار

۱- گروه مهندسی کامپیوتر - دانشگاه بین‌المللی امام‌رضا علیه‌السلام - مشهد - ایران - v.monemizadeh@imamreza.ac.ir
۲- دانشکده مهندسی کامپیوتر و فناوری اطلاعات - دانشگاه صنعتی سجاد - مشهد - ایران - j_hamidzadeh@sadjad.ac.ir

چکیده: مسئله جستجوی k نزدیک‌ترین همسایه تقریبی در ابعاد بالا یک مسئله کلاسیک در هندسه محاسباتی، شباهت تصویر و سایر زمینه‌های مشابه می‌باشد. در این مسئله، یک مجموعه داده متشکل از n نقطه در فضای d بعدی و یک پارامتر k داریم، هدف پیش‌پردازش مجموعه داده است به‌طوری‌که با داشتن یک نقطه پرس‌وجوی d بعدی Q داده‌شده بتوان k نقطه را یافت به‌طوری‌که k نزدیک‌ترین همسایه تقریبی به Q باشد. هدف این مقاله ارائه روشی جدید برای یافتن k نزدیک‌ترین همسایه تقریبی برای ابعاد بالا است. در روش پیشنهادی، ابتدا داده‌های با ابعاد بالای مجموعه داده مورد نظر درون فضای همینگ جاسازی‌شده، سپس با ترکیب خطی بردارهای تصادفی و داده‌های جاسازی‌شده در فضای همینگ، جدول‌های درهم‌سازی تشکیل می‌شود. آزمایش‌های زیادی بر روی پایگاه داده بزرگ تصاویر انجام گرفته است و نتایج گویای این نکته می‌باشد که این الگوریتم برای ماتریس‌های خلوت منجر به حاصل شدن جواب‌های مناسب‌تری خواهد شد. روش پیشنهادی با روش‌های جدید نیز مقایسه شده است که نتایج آزمایش‌ها و ارزیابی آن‌ها، نشان‌دهنده برتری روش پیشنهادی از نظر صحت نسبت به آن روش‌ها می‌باشد.

واژه‌های کلیدی: جستجوی k نزدیک‌ترین همسایه تقریبی، ابعاد بالا، ترکیب خطی، جاسازی، نفرین ابعاد، درهم‌سازی حساس به محل یا LSH.

Approximate k Nearest Neighbor Search with Linear Combination Method

V. Monemizadeh[1], MSc Student; J. Hamidzadeh[2], Assistant Professor

1- Computer Engineering Department, Imam Reza International University, Mashhad, Iran, Email: v.monemizadeh@imamreza.ac.ir
2- Faculty of Computer Engineering and Information Technology, Sadjad University of Technology, Mashhad, Iran, Email: j_hamidzadeh@sadjad.ac.ir

Abstract: Approximate k nearest neighbor search problem in high dimensional Euclidean spaces is a classical problem in computational geometry, image similarity search, video, and so on. In this problem, we are given a point set P of size n in the space $\mathbb{R}^d$ and a parameter k, the goal is to preprocess P. So that given a query point $q \epsilon \mathbb{R}^d$ we can return fairly fast k points in which the points are good approximations of the k nearest neighbors to Q in P. In this paper, an algorithm for searching k nearest neighbor is presented for high dimensional data. In this method, first, data with high-dimensional are embedded in hamming space, then with a linear combination of random vectors and embedded data in hamming space, hash tables are formed. We conduct extensive experiments for this algorithm on big dataset of handwriting English single-digit images. This algorithm led good results for sparse matrices. Experimental results show that the proposed algorithm has the better accuracy comparing to the new methods.

Keywords: Approximate k nearest neighbor search, high dimensional, linear combination, embedding, curse dimensional, locality sensitive hashing (LSH).

تاریخ ارسال مقاله: ۱۳۹۴/۱۲/۰۹
تاریخ اصلاح مقاله: ۱۳۹۵/۰۲/۲۳
تاریخ پذیرش مقاله: ۱۳۹۵/۰۶/۰۶
نام نویسنده مسئول: جواد حمیدزاده
نشانی نویسنده مسئول: ایران، مشهد، بلوار جلال آل احمد ۶۴، دانشگاه صنعتی سجاد، دانشکده مهندسی کامپیوتر و فناوری اطلاعات.

۱- مقدمه

جستجوی k نزدیک‌ترین همسایه برای یک نمونه پرس‌وجو[۱] در یک پایگاه داده خاص یکی از مسائل بسیار مهم در حوزه یادگیری و بینایی ماشین می‌باشد که به دلیل تنوع فراوان کاربردهای آن همواره مورد علاقه محققین بوده است [۳-۱]. در ادبیات طبقه‌بندی داده‌ها، روش‌هایی وجود دارد که داده‌ها را بر اساس فاصله نمونه جدید با نمونه‌های آموزشی طبقه‌بندی می‌کند [۴]. در مسئله جستجوی k نزدیک‌ترین همسایه، برای تعیین دسته یک نمونه جدید، فاصله اقلیدسی بین آن و همه نمونه‌های ذخیره شده در حافظه، محاسبه و k نمونه با کوچک‌ترین فاصله انتخاب می‌شوند. از این k نمونه، برچسب دسته اکثریت، به عنوان برچسب دسته نمونه ناشناخته در نظر گرفته می‌شود [۵]. در سال‌های اخیر، به دلیل گسترش انفجارگونه داده‌های روی وب به‌ویژه علاقه زیاد کاربران وب به داده‌های تصویری و ویدیویی [۱]، مسئله یافتن k نزدیک‌ترین همسایه برای یک نمونه پرس‌وجو در یک پایگاه داده با ابعاد بالا از اهمیت خاصی برخوردار شده است [۶]. با توجه به اینکه چنین پایگاه‌های داده‌ای حاوی میلیون‌ها نمونه می‌باشند روش‌های جستجوی جامع، غیرعملی بوده و در نتیجه یافتن روش‌های بازیابی و جستجوی کارآمد و بهینه امری ضروری و بسیار مهم می‌باشد.

در حالت کلی، برای پایگاه‌های داده با ابعاد بالا، تعداد ویژگی‌ها (یا تعداد ابعاد) ممکن است از هزارها تا میلیون‌ها تغییر کند. برای مثال، یک تصویر ۱۰۰۰×۱۰۰۰ می‌تواند به صورت یک بردار در یک فضای ۱۰۰۰۰۰۰-بعدی نمایش داده شود (یک پیکسل به ازای هر بعد) [۳]. مسئله یافتن نزدیک‌ترین همسایه برای داده‌های با ابعاد پایین (مثلاً بعد ۲ یا ۳) به‌خوبی حل شده است [۷]. اما متأسفانه وقتی تعداد بعد زیاد و زیادتر می‌شود تمام روش‌های شناخته‌شده جهت حل مسئله یافتن نزدیک‌ترین همسایه ناکارآمد و ناکارآمدتر می‌شوند و فضا و زمان مورد نیاز برای حل آن به‌طور نمایی با بعد زیاد می‌شود. در این حالت گفته می‌شود که این راه‌حل‌ها برای داده‌های با ابعاد بالا دچار مشکل نفرین ابعاد[۲] شده‌اند. علی‌رغم دهه‌ها تلاش شدید، حل کارآمدی برای این مسئله یافت نشده است [۷]. عدم موفقیت در حذف وابستگی نمایی به بعد، منجر به این شده است که محققان حدس بزنند که راه‌حل بهینه برای چنین مسائلی هنگامی‌که بعد به اندازه کافی بزرگ می‌شود وجود ندارد. از طرف دیگر در بسیاری از کاربردها و اهداف عملی مشاهده شده است که برای این مسئله نیازی به اصرار در یافتن حل دقیق وجود ندارد و یافتن یک جواب تقریبی نیز کافی است و تقریباً به اندازه یک جواب دقیق، خوب و قابل قبول می‌باشد. در نتیجه در حال حاضر سؤال اصلی این است که با توجه به خوب و قابل قبول بودن حل‌های تقریبی در بسیاری از اهداف عملی، آیا در این صورت امکان حذف وابستگی نمایی به بعد وجود دارد یا نه؟ علاوه‌براین، یک الگوریتم تقریبی کارا، با در نظر گرفتن تمام همسایه‌های نزدیک تقریبی و برگرداندن نزدیک‌ترین نقطه در بین آن‌ها، می‌تواند برای حل دقیق مسئله نزدیک‌ترین همسایه نیز استفاده شود. ازاین‌رو اکنون تلاش‌ها برای حل مسئله k نزدیک‌ترین همسایه تقریبی (ANN)[۳] متمرکز شده است [۳-۱، ۶، ۸، ۹].

یکی از روش‌های عمده جستجوی نزدیک‌ترین همسایه تقریبی (ANN) که توجه زیادی را به خود جذب کرده است، روش مبتنی بر درهم‌سازی[۴] می‌باشد [۲، ۳، ۱۰]. ایده اصلی درهم‌سازی برای ANN، استفاده از توابع درهم‌ساز مناسب جهت تبدیل ویژگی‌ها (یا نقاط) با ابعاد بالا به کدهای دودویی است که شباهت (فاصله) بین نقاط در فضای اصلی را حفظ کند. از این طریق (همان‌طور که آزمایش‌ها نشان داده)، درهم‌سازی مزایای زیادی در ذخیره داده و سرعت محاسبه برای مسئله جستجوی شباهت در فضاهای با ابعاد بالا نسبت به روش‌های مبتنی بر درخت‌ها یا روش‌های مبتنی بر وزن‌دهی (مانند روش وزن‌دهی hubness فازی) [۸]، فراهم می‌کند. از دیگر مزایای درهم‌ساز، می‌توان به پیاده‌سازی ساده آن نیز اشاره کرد [۱]. ازاین‌رو، درهم‌سازی تبدیل به یکی از روش‌های پرطرفدار برای جستجوی شباهت داده‌های با ابعاد بالا شده است و سرعت بالای آن برای ANN به‌ویژه برای داده‌های چندرسانه‌ای نشان داده شده است.

یکی از روش‌های بسیار شناخته‌شده، درهم‌سازی یک الگوریتم تصادفی درهم‌سازی به نام الگوریتم درهم‌سازی حساس به محل یا LSH[۵] می‌باشد [۱۱]. ایده اصلی الگوریتم LSH، درهم‌سازی نقاط با استفاده از برخی توابع درهم‌ساز است به‌طوری‌که برای هر تابع، احتمال برخورد[۶] برای اشیایی که به هم نزدیک‌ترند (شبیه‌ترند) از اشیایی که از هم دورتر هستند، بسیار بالاتر است [۱]. در این صورت می‌توان همسایه‌های نزدیک را به‌وسیله درهم‌سازی نقطه پرس‌وجو و بازیابی عناصر ذخیره‌شده در جعبه (یا باکت[۷]) حاوی آن نقطه پرس‌وجو به دست آورد [۱].

در این مقاله الگوریتمی به نام ترکیب خطی ارائه می‌شود. در این روش، به دلیل اینکه روش درهم‌ساز برای فضای همینگ سریع اجرا می‌شود بنابراین، ابتدا نمونه‌های آموزشی، به فضای همینگ جاسازی می‌شوند سپس با ترکیب خطی بردارهای تصادفی و داده‌های جاسازی‌شده در فضای همینگ، جدول‌های درهم‌سازی تشکیل می‌شود و نتایج آن طی آزمایش‌هایی بر روی فاکتورهای تأثیرگذار مانند تعداد همسایه‌ها، ظرفیت جعبه، تعداد مراحل تکرار و تعداد بردارهای تصادفی انتخاب‌شده، بررسی شده است. روش پیشنهادی با روش‌های جدید نیز مقایسه شده است که نتایج نشان‌دهنده برتری روش پیشنهادی از نظر صحت نسبت به آن روش‌ها می‌باشد.

بنابراین، نوآوری این روش در ترکیب روش‌های جستجوی ANN مبتنی بر درهم‌ساز و نزدیک‌ترین همسایگان تقریبی در فوق‌مکعب با هم ترکیب می‌باشد. لازم به ذکر است که بردارهای مورد بحث در روش نزدیک‌ترین همسایگان تقریبی در فوق‌مکعب را نیز به صورت تصادفی و مطابق توزیع برنولی انتخاب می‌کنیم.

ساختار ادامه مقاله به شرح زیر است:

در بخش دوم، کارهای پیشین مورد بررسی قرار گرفته است. جزییات روش پیشنهادی در بخش سوم بیان شده است. در بخش چهارم نتایج آزمایش‌ها و تفسیر آن‌ها بیان شده است. نتیجه‌گیری و بیان کارهای آینده در بخش آخر ذکرشده است.

۲- کارهای پیشین

راه‌حل‌های مختلفی برای نزدیک‌ترین همسایه تقریبی ارائه شده است اما به‌طورکلی این راه‌حل‌ها را می‌توان در چهار دسته عمده تقسیم‌بندی نمود که عبارت‌اند: روش‌های جستجوی ANN مبتنی بر درخت، روش‌های جستجوی ANN مبتنی بر وزن‌دهی hubness و فازی، روش‌های جستجوی ANN مبتنی بر درهم‌ساز و نزدیک‌ترین همسایگان تقریبی در فوق‌مکعب می‌باشد. در ادامه به معرفی آن‌ها می‌پردازیم.

۲-۱- روش‌های جستجوی ANN مبتنی بر درخت

این دسته از روش‌ها در واقع عمل جستجو را از طریق پارتیشن‌بندی کردن ناحیه تحت جستجو انجام می‌دهند. روش درخت-kd یکی از مشهورترین رویکردهای مبتنی بر درخت برای مسئله جستجوی ANN می‌باشد [۷]. گونه‌های اولیه این روش در فضاهای با ابعاد پایین بسیار کارآمد می‌باشد [۸] اما عملکرد آن‌ها برای داده‌های با ابعاد بالا سریعاً افت می‌کنند [۹]. جهت افزایش سرعت این روش، گونه‌های اصلاح‌شده‌ای از درخت-kd ارائه شده است که از آن جمله می‌توان به درخت‌های-kd تصادفی [۱۲] اشاره کرد که جهت افزایش سرعت جستجوی ANN ارائه شده‌اند. از دیگر روش‌های مبتنی بر درخت می‌توان درخت مستطیلی[۸] (درخت-R) [۱۰] و بهبودهای آن مانند درخت-SS [۱۱] و درخت-SR [۱۳]، درخت-M [۱۴] و درخت-RP [۱۵] را نام برد.

۲-۲- روش‌های جستجوی ANN مبتنی بر وزن‌دهی hubness و فازی

این دسته شامل دو روش عمده می‌باشد [۶]: الگوریتم نزدیک‌ترین همسایگی فازی[۹] و روش kNN مبتنی بر وزن‌دهی hubness[۱۰].

در این دسته از روش‌ها که جدید هم می‌باشند سعی می‌شود به‌جای تلاش جهت اجتناب از مشکل نفرین ابعاد از طریق مشاهده یک زیرفضای ویژگی با ابعاد پایین‌تر، از برخی ویژگی‌های ذاتی مربوط به داده‌های ابعاد بالا استفاده شود. یکی از این ویژگی‌ها، hubness است. hubness (یا مرکزیت) به معنای تمایل برخی نقاط داده در مجموعه داده‌های با ابعاد بالا برای قرار گرفتن در اغلب لیست‌های kNN سایر نقاط (در مقایسه با نقاط دیگر) می‌باشد [۱۶]. از این خاصیت می‌توان برای اهداف خوشه‌بندی داده استفاده کرد و روش‌های خوشه‌بندی مبتنی بر hubness بر اساس ارزش بالقوه استفاده از نقاط hub در خوشه‌بندی طراحی می‌شوند.

همان‌طور که می‌دانیم در روش‌های مبتنی بر وزن‌دهی معمولاً عمل وزن‌دهی مبتنی بر موقعیت یک عنصر در لیست kNN یا فاصله آن عنصر تا نقطه داده مشاهده شده انجام می‌گیرد. در روش وزن‌دهی hubness عمل وزن‌دهی با توجه به تعداد دفعاتی که یک عنصر در لیست‌های kNN سایر نقاط مجموعه داده قرار می‌گیرد انجام می‌شود. برای این منظور، فرض کنید $N_k(x_i)$ برابر نمره hubness یک عنصر x_i باشد و به صورت تعداد دفعاتی که x_i در لیست‌های kNN سایر نقاط مجموعه داده قرار می‌گیرد تعریف شود [۶]. این عدد می‌تواند به دو بخش مطابق $N_k(x_i)=GN_k(x_i)+BN_k(x_i)$ تقسیم شود. $GN_k(x_i)$ تعداد رخدادهای خوب می‌باشد یعنی تعداد دفعاتی که x_i در لیست‌های kNN نقاط دیگر قرار می‌گیرد و برچسب آن با برچسب آن نقاط یکسان است. $BN_k(x_i)$ تعداد رخدادهای بد می‌باشد یعنی تعداد دفعاتی که x_i در لیست‌های kNN نقاط دیگر قرار می‌گیرد اما برچسب آن با برچسب آن نقاط متفاوت است. از آنجا که نقاط با یک نمره hubness بد بالا، اطلاعات نادرست و گمراه‌کننده را به دست می‌دهد در نتیجه تأثیر تعیین‌کننده‌ای بر نتیجه طبقه‌بند kNN دارد. اگر وزن یک عنصر x_i در روش‌های kNN مبتنی بر وزن‌دهی با $w(x_i,k)$ نمایش دهیم آن‌گاه در روش مبتنی بر نمره hubness این عدد به صورت $w(x_i,k)=e^{-h_b(x_i,k)}$ تعریف می‌شود که در آن $h_b(x_i,k)=\frac{BN_k(x_i)-\mu_{BN_k}}{\sigma_{BN_k}}$ برابر hubness بد استاندارد شده است و μ_{BN_k} متوسط hubness بد و σ_{BN_k} انحراف معیار آن می‌باشد. به دلیل اینکه در این روش ممکن است نویز دردسرساز شود، اخیراً روش‌هایی از جمله NWkNN، AkNN، HIkNN و NHBkNN برای حل این مشکل ارائه شده است [۱۶].

در روش وزن‌دهی فازی، بردار وزنی برای ارزیابی میزان تشابه نمونه‌ها لازم است [۱۷]. در روش‌های فازی kNN به هر عنصر x_i عددی مانند $u_{ci}=u_c(x_i)$ نسبت می‌دهند که عبارت است از درجه عضویت x_i به کلاس c که این عدد u_{ci} بایستی یک سری قیود را برآورده کند. عدد u_{ci} معمولاً برحسب موقعیت یک عنصر در لیست kNN یا فاصله آن عنصر تا نقطه داده مشاهده شده تعیین می‌شود. روش وزن‌دهی hubness فازی با ترکیب دو روش فازی و hubness از مزایای هر دو روش بهره می‌برد. در روش وزن‌دهی hubness فازی، درجه عضویت x_i به کلاس c (یعنی عدد u_{ci}) نه برحسب فاصله که برحسب تعداد رخدادهای x_i در لیست‌های kNN سایر نقاط تعیین می‌شود [۱۶].

۲-۳- روش‌های جستجوی ANN مبتنی بر درهم‌ساز

با توجه به اینکه روش پیشنهادی در این مقاله جزء روش‌های مبتنی بر درهم‌ساز است این دسته را مفصل‌تر معرفی می‌کنیم. همان‌طور که پیش‌تر بیان شد ایده اصلی الگوریتم‌های مبتنی بر درهم‌ساز تولید کدهای دودویی برای نقاط داده است که شباهت بین هر دو تا از آن‌ها را حفظ کند. فرض کنید یک مجموعه داده $\boldsymbol{X}\in\mathbb{R}^{d\times n}$ شامل n تا نقطه d-بعدی داده شده است. یک الگوریتم درهم‌ساز، از c تابع درهم‌ساز برای تولید یک جاساز همینگ c بیتی $\boldsymbol{Y}\in\mathbb{B}^{c\times n}$ استفاده می‌کند [۷]. یکی از روش‌های جستجوی ANN مبتنی بر درهم‌ساز، استفاده از رتبه‌بندی

همینگ است [۷] بدین ترتیب که ابتدا فاصله همینگ بین یک نقطه پرس‌وجو و تمام نقاط پایگاه داده محاسبه می‌شود. سپس یک رتبه‌بندی همینگ بر اساس فواصل همینگ محاسبه‌شده، انجام می‌گیرد [۷]. می‌توان جهت سرعت‌بخشی به محاسبه فاصله همینگ بین هر دو کد دودویی از عملیات سخت‌افزاری سطح پایینی همچون XOR استفاده کرد. با این حال، کماکان برای یافتن نزدیک‌ترین همسایه، $o(n)$ نیاز است [۷]. به دلیل اینکه روش درهم‌ساز از یک اندیس جهت عملیات جستجو استفاده می‌کند که همین عامل باعث افزایش سرعت می‌شود، بنابراین برای تسریع عمل جستجو می‌توان از این روش استفاده نمود. با اختصاص هر نقطه داده درون یک جعبه درهم‌ساز c بیتی متناظر با کد دودویی c بیتی آن، یک اندیس درهم‌ساز می‌تواند ساخته شود. در نتیجه برای یک نقطه پرس‌وجو داده شده، سه مرحله زیر جهت انجام عملیات جستجو می‌تواند استفاده شود [۷]:

۱) **مرحله کدگذاری:** در این مرحله می‌توان نقطه پرس‌وجو را با استفاده از c تابع درهم‌ساز، به یک کد دودویی c بیتی تبدیل نمود که این بیت‌ها، آدرس جدول درهم‌ساز را تشکیل می‌دهند؛

۲) **مرحله مکان‌یابی:** تمام نقاط داده در جعبه‌هایی که درون یک شعاع همینگ r_h از کد دودویی نقطه پرس‌وجو می‌افتند مشخص و برگردانده می‌شوند؛ و

۳) **مرحله اسکن خطی:** یک اسکن خطی روی این نقاط انجام می‌گیرد تا همسایه‌های مورد نیاز را برگرداند. به هر حال، برای یافتن نزدیک‌ترین همسایه واقعی، روش‌های درهم‌ساز موجود مجبورند تا شعاع r_h بزرگی را استفاده کنند که این کار، زمان مکان‌یابی و زمان اسکن خطی را به‌طور نمایی افزایش می‌دهد.

یک مسئله درهم‌ساز نوعی می‌تواند به این صورت خلاصه شود:

با داشتن n نقطه داده $\boldsymbol{X}=[\boldsymbol{x}_1,\cdots,\boldsymbol{x}_n]\in\mathbb{R}^{d\times n}$ که هرکدام از این نقاط دارای d-بعد می‌باشند نیاز است تا c تابع درهم‌ساز جهت نگاشت یک نقطه داده $\boldsymbol{x}$ به یک کد c بیتی پیدا کنیم [۷]:

$$H(\boldsymbol{x})=\left[h_1(\boldsymbol{x}),h_2(\boldsymbol{x}),\cdots,h_c(\boldsymbol{x})\right] \qquad (۱)$$

که $h_k(\boldsymbol{x})\in\{0,1\}$ تابع درهم‌ساز k-ام می‌باشد. به عنوان مثال، برای درهم‌سازی مبتنی بر تصویر کردن، داریم [۱۸]:

$$h_k(\boldsymbol{x})=sgn\left(\boldsymbol{F}\left(W_k^T\boldsymbol{x}+\boldsymbol{t}_k\right)\right) \qquad (۲)$$

که W_k بردار تصویر و $\boldsymbol{t}_k$ عرض از مبدأ می‌باشد. هدف الگوریتم‌های درهم‌ساز مختلف در یافتن $\boldsymbol{F}$، W_k و $\boldsymbol{t}_k$ مختلف نسبت به توابع هدف مختلف می‌باشد.

یکی از شناخته‌شده‌ترین و محبوب‌ترین الگوریتم‌های درهم‌ساز، الگوریتم درهم‌سازی حساس به محل یا LSH می‌باشد [۱۹] که این درهم‌ساز مبتنی بر تصویرسازی تصادفی است (یعنی LSH از W_k به‌طور تصادفی تولیدشده استفاده می‌کند) [۷]. تابع $\boldsymbol{F}$ در LSH یک تابع همانی است. در نتیجه، برای LSH داریم:

$$h_k(\boldsymbol{x})=\begin{cases}1, & if \quad W_k^T\boldsymbol{x}\geq 0\\ 0, & otherwise\end{cases} \qquad (۳)$$

که W_k برداری تولیدشده از یک توزیع گوسی با متوسط صفر $\mathcal{N}(0,1)$ و بعدی برابر با ورودی $\boldsymbol{x}$ می‌باشد. تعمیم‌های زیادی برای LSH پیشنهاد شده است [۲۲-۲۰] که از آن جمله می‌توان LSH مبتنی بر آنتروپی [۲۰] و LSH کرنلی [۲۱] را نام برد.

در بخش بعد، به یکی از اساسی‌ترین مشکلات الگوریتم LSH اشاره می‌کنیم و راه‌حل آن را نیز شرح خواهیم داد.

۲-۳-۱- الگوریتم جاسازی

الگوریتم LSH تاکنون در موارد کاربردی زیادی استفاده‌شده است. با این حال، LSH دارای یک اشکال اساسی است و آن این است که این روش فقط هنگامی‌که نقاط ورودی در فضای همینگ باشند سریع و آسان است. همان‌طور که در [۱۹] ذکرشده، با جاسازی فضای l_2 درون فضای l_1 و سپس فضای l_1 درون فضای همینگ می‌توان الگوریتم LSH را به نرم l_2 تعمیم داد. از آنجا که برای مسئله جستجوی شباهت تفاوتی میان استفاده از نرم l_2 یا l_1 نمی‌باشد، می‌توان فاصله را با نرم l_1 تعریف کرد. در نتیجه برای مسئله جستجوی ANN لازم است تا ابتدا به صورت مناسبی فضای l_1 را درون فضای همینگ جاسازی نماییم. بدین منظور الگوریتم ۱ جهت جاسازی فضای l_1 درون فضای همینگ در [۲۳] پیشنهاد شده است. برای این منظور باید در بین تمام نمونه‌های آموزشی، بزرگ‌ترین عدد را پیدا کرده و آن را m بنامیم. سپس، با داشتن m، به ازای هر داده آموزشی d-بعدی V_i، $i\in\{1,...,num\_sample\}$، یک بردار دودویی به طول $m\times d$ بیت مانند V_i' در نظر می‌گیریم. هر m بیت در این بردار دودویی مربوط به یک ویژگی یا بعد است. عملیات مقداردهی m بیت متناظر با هر ویژگی (بعد) بدین صورت است که اگر ویژگی j-ام مربوط به داده آموزشی i-ام دارای مقدار $v_{i,j}$ باشد $(v_{i,j}\leq m)$ آنگاه $v_{i,j}$ بیت سمت چپ را با عدد ۱ پر می‌کنیم و مابقی بیت‌ها را $(m-v_{i,j})$ صفر قرار می‌دهیم [۲۳].

در شبه‌کد الگوریتم ۱، بردار V، بردار d عنصری هر نمونه آموزشی که d همان تعداد ابعاد (یا تعداد ویژگی‌های) هر نمونه آموزشی است و بردار V'، نسخه جاسازی‌شده مربوط به نمونه آموزشی متناظر آن می‌باشد. توجه داشته باشید که هر نمونه آموزشی که d بعد یا ویژگی داشت به عناصر صفر و یک تبدیل شده است. d در اینجا تعداد ابعاد مجموعه داده می‌باشد. متغیر num_sample بیانگر تعداد نمونه‌های آموزشی در این شبه‌کد می‌باشد.

Algorithm 1: Embedding
Input: $(V_1,\ldots,V_{\text{num\_sample}})$
For $i=1$ to num_sample
  For $j=0$ to $d-1$
    $V'_{i,j}[mj,\ldots,m(j+1)-1]$ = Put from left to right $V_{i,j}$ ones followed by $(m-V_{i,j})$ zeroes, where $V_{i,j}$ is j-th element of i-th vector.
Output: $(V'_1,\ldots,V'_{\text{num\_sample}})$

الگوریتم ۱: الگوریتم جاسازی [۲۳]

۲-۴- نزدیک‌ترین همسایگان تقریبی در فوق‌مکعب[11]

برای مکعب d-بعدی یک الگوریتم نزدیک‌ترین همسایه تقریبی وجود دارد که، تمام نقاط پایگاه داده و نقاط پرس‌وجو در $\{0,1\}^d$ می‌باشند و فواصل با فاصله همینگ اندازه‌گیری شده‌اند [۲۴]. به عبارت دیگر، یکی از روش‌های کاهش بعد، استفاده از فوق‌مکعب می‌باشد که از طریق تصویرسازی در فضاهای کوچک متعامد، فوق‌مکعب مذکور ساخته می‌شود. اولین ایده پشت الگوریتم مکعبی، طراحی یک آزمون جداگانه برای هر فاصله ℓ می‌باشد [۲۴]. یک پرس‌وجوی Q داده شده است، یک چنین آزمونی یا یک بردار از پایگاه داده‌ای با فاصله حداکثر $(1+\epsilon)\ell$ از Q را برمی‌گرداند، یا اطلاع می‌دهد که هیچ بردار پایگاه داده‌ای در فاصله ℓ یا کم‌تر از Q وجود ندارد. با توجه به چنین آزمونی، جستجوی نزدیک‌ترین همسایه تقریبی با استفاده از جستجوی دودویی بر روی $\ell \epsilon \{1,2,\ldots,d\}$ (و همچنین بررسی فاصله صفر که می‌تواند با استفاده از هر ساختار داده فرهنگ لغتی مناسب انجام شود) انجام می‌پذیرد [۲۴].

یک آزمون- β ی τ به صورت زیر تعریف می‌شود: یک زیرمجموعه C از مختصات مکعب با انتخاب هر عنصر به‌طور مستقل در $\{1,2,\ldots,d\}$ و به‌طور تصادفی با احتمال β انتخاب کنید. برای هر یک از مختصات انتخاب‌شده i، به‌طور مستقل و یکنواخت و تصادفی $r_i \in \{0,1\}$ را انتخاب می‌کنیم. برای $v \in Q_d$، مقدار τ در v را تعریف می‌کنیم، $\tau(v)$ به صورت زیر نشان داده می‌شود:

$$\tau(v) = \sum_{i \in C} r_i . v_i \,(mod\ 2) \qquad (۴)$$

به‌طور مشابه، آزمون می‌تواند به عنوان انتخاب یک بردار $\vec{R} \in \{0,1\}^d$ مشاهده شود به‌طوری‌که هر وارده با مقدار صفر با احتمال "بالا" (یعنی، $1-\beta/2$) و مقدار یک با احتمال "پایین" انتخاب شود (یعنی، $\beta/2$). با این دید، مقدار آزمون روی $v \in Q_d$ توسط ضرب داخلی[12] با $\vec{R}$ و سپس محاسبه نتیجه به پیمانه ۲ صورت می‌گیرد [۲۴].

در بخش بعد روش پیشنهادی خود جهت حل مسئله جستجوی k نزدیک‌ترین همسایه تقریبی را که مبتنی بر ترکیب خطی از داده جاسازی‌شده است به‌طور کامل توضیح می‌دهیم.

۳- روش پیشنهادی

الگوریتم LCAkNN[13] در این مقاله جهت حل مسئله جستجوی k نزدیک‌ترین همسایه تقریبی در ابعاد بالا می‌پردازیم که مبتنی بر الگوریتم درهم‌سازی حساس به محل (LSH) می‌باشد. این روش را می‌توان در دو گام کلی بیان کرد: گام پیش‌پردازش و گام جستجوی k نزدیک‌ترین همسایه تقریبی برای یک داده پرس‌وجو. گام اول خود شامل دو الگوریتم کوچک‌تر است. الگوریتم اول به دلیل (الف) پیچیدگی و بار محاسباتی بالای فرآیند جستجوی داده‌های با ابعاد بالا در فضای اقلیدسی، (ب) عدم‌وجود تفاوت میان استفاده از نرم l_2 یا l_1 برای مسئله جستجوی شباهت و (ج) سریع و آسان بودن LSH فقط برای فضای همینگ، داده‌های با ابعاد بالا را درون فضای همینگ جاسازی می‌کند [۲۳]. سپس الگوریتم دوم، با ترکیب خطی بردارهای تصادفی و داده‌های جاسازی‌شده در فضای همینگ، جدول‌های درهم‌سازی تشکیل می‌شود که فرآیند جستجو در این جدول‌ها بسیار کارآمدتر از جستجو در پایگاه داده می‌باشد. گام دوم نیز شامل الگوریتمی است که با استفاده از جدول‌های درهم‌سازی تشکیل شده، برای یک داده پرس‌وجو عملیات یافتن k نزدیک‌ترین همسایه تقریبی را انجام می‌دهد.

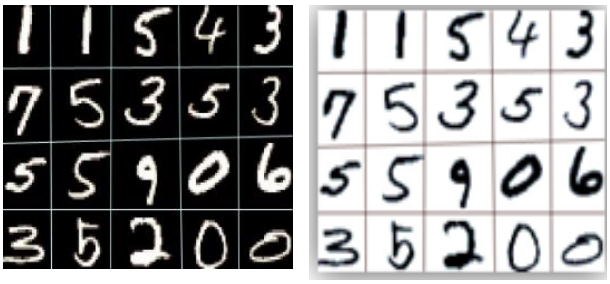
بنابراین به‌طور خلاصه می‌توان گفت، در این مقاله دو روش جستجوی ANN مبتنی بر درهم‌ساز و نزدیک‌ترین همسایگان تقریبی در فوق‌مکعب را با هم ترکیب و بردارها را نیز تصادفی و مطابق توزیع برنولی تعیین می‌کنیم. بدین ترتیب، جدول‌های درهم‌ساز تشکیل می‌شوند و در نهایت بر روی چندین پارامتر آزمایش‌ها صورت گرفته خواهد شد و برای هرکدام، مقادیر مطلوب پارامتر مذکور به دست خواهد آمد.

۳-۱- مجموعه داده

ما در این مقاله، از مجموعه داده اعداد رقمی دست‌نویس MNIST استفاده کرده‌ایم [۲۵]. این مجموعه داده شامل ۶۰۰۰۰ نمونه آموزشی و یک مجموعه آزمایشی ۱۰۰۰۰ عنصری می‌باشد. این مجموعه داده، یک زیرمجموعه از مجموعه بزرگ‌تر NIST می‌باشد. هر نمونه از این مجموعه داده، شامل تصویر یک عدد رقمی است که به صورت یک بردار نمایش داده شده است. این بردار شامل ۷۸۴ عنصر می‌باشد و هرکدام از این ۷۸۴ عنصر، یک پیکسل از تصویرمان را تشکیل می‌دهد که به صورت یک عدد بین ۰ تا ۲۵۵ که همان شدت نور پیکسل مورد نظر است. هر پیکسل را یک بعد یا ویژگی در نظر می‌گیریم بنابراین مجموعه داده‌های ما شامل ۶۰۰۰۰ نمونه آموزشی هرکدام با ۷۸۴ بعد می‌باشد. هر بردار ۷۸۴ عنصری به صورت یک ماتریس ۲۸× ۲۸ تایی می‌توان نمایش داد که تصویر رقم در واقع در ماتریس۲۰× ۲۰ میانی و در مرکز ماتریس ۲۸× ۲۸ قرار گرفته است (شکل ۱ را ببینید). لازم به ذکر است که در مجموعه داده‌ای که آزمایش‌ها بر روی آن انجام شده است (منظور MNIST) m برابر با ۲۵۵ خواهد بود (زیرا مجموعه داده MNIST از تصاویر اعداد دست‌نویس تشکیل شده است که بیش‌ترین مقدار، شدت نور ۲۵۵ می‌باشد).

در این مقاله برای مقایسه روش پیشنهادی با روش‌های موجود در مقاله‌های [۶، ۱۶] از چند مجموعه داده استفاده‌شده است که در جدول ۱ مشخصات این مجموعه داده‌ها ذکرشده است.

همان‌طور که در شکل ۱-الف مشاهده می‌شود ماتریس مذکور را می‌توان با تبدیل پیکسل‌های سیاه و سفید به یکدیگر به‌منظور استفاده در الگوریتم جاسازی (به دلیل اینکه شدت پیکسل رنگ سیاه که همان اعداد دست‌نویس هستند، صفر می‌باشد و بنابراین اعمال الگوریتم جاسازی بر روی عدد صفر بی‌معنی می‌باشد بنابراین به راحتی با تبدیل پیکسل‌های سیاه به سفید و بالعکس می‌توان الگوریتم جاسازی را استفاده نمود)، برعکس شده‌اند و به شکل ۱-ب تبدیل نمودیم. آزمایش‌ها بر روی کامپیوتر با پردازنده i۵ Intel و ۶Gb RAM انجام شده است.



(ب) (الف)

شکل ۱: نمونه‌ای از خروجی مجموعه داده MNIST؛ (الف) ۲۰ نمونه از این اعداد را نمایش می‌دهد، (ب) پیکسل‌های سیاه و سفید به‌منظور استفاده در الگوریتم جاسازی، برعکس شده‌اند

جدول ۱: مشخصات مجموعه داده‌ها

مجموعه داده	اندازه تعداد نمونه‌ها	تعداد ویژگی (d)	مقدار حداکثر ویژگی (m)	تعداد بیت‌های جاسازی‌شده $(m \times d)$
Ozone	۲۵۳۴	۷۲	۱۰۴۸۵	۷۵۴۹۲۰
Vehicle	۸۴۶	۱۸	۱۰۱۸	۱۸۳۲۴
Mfeat-fourier	۲۰۰۰	۷۶	۸۰	۶۰۸۰
Ionosphere	۳۵۱	۳۴	۲	۶۸
MNIST	۶۰۰۰۰	۷۸۴	۲۵۵	۱۹۹۹۲۰
haberman	۳۰۶	۳	۸۳	۲۴۹

۳-۲- الگوریتم ترکیب خطی

در الگوریتم ترکیب خطی (LCAkNN)، تعداد s بردار به صورت زیر تولید می‌کنیم: s بردار به اندازه $m \times d$ بیت که هر درایه این بردارها از ۰ و ۱ تشکیل شده‌اند و احتمال تولید هرکدام طبق توزیع برنولی تعیین می‌شود (قابل توجه است که این احتمال می‌تواند همان‌طور که در جدول ۶ آمده است تغییر پیدا کند و نتایج جالبی حاصل می‌شود). حال هر نمونه همینگ جاسازی‌شده حاصل از الگوریتم ۱ را در تک‌تک این s بردار ضرب اسکالر کرده و باقی‌مانده به پیمانه ۲ محاسبه شده و حاصل را به عنوان آدرس جعبه آن نمونه در نظر می‌گیرد. (برای مثال، فرض کنید $m=5, s=2, d=4$ باشد و نمونه آموزشی V با ۴ بعد به صورت زیر داشته باشیم:

۳	۰	۵	۱

که جاسازی‌شده آن خواهد شد:

۱۱۱۰۰	۰۰۰۰۰	۱۱۱۱۱	۱۰۰۰۰

حال فرض می‌کنیم ۲ بردار تصادفی به صورت زیر داشته باشیم:

s_1:

۱۰۱۱۱	۰۰۰۱۰	۰۰۰۰۰	۱۱۱۰۰

s_2:

۰۰۰۱۰	۱۱۰۰۰	۰۰۱۰۰	۰۱۰۰۰

آن‌گاه ضرب اسکالر نمونه آموزشی V در s_1 و s_2 به صورت زیر خواهد شد:

$V.s_1 = 3$ و $V.s_2 = 1$ که باقی‌مانده آن‌ها بر ۲ برابر با ۱ و ۱ خواهد شد. بنابراین آدرس نمونه مورد نظر ۱۱ خواهد شد.

همان‌طور که به صورت شهودی نیز قابل مشاهده می‌باشد، هرچه تعداد بردارهای تصادفی کمتر باشد (s)، به دلیل اینکه طول آدرس جدول درهم‌ساز کوچک می‌شود بنابراین ممکن است آدرس مشابه زیادی تولید شود که در این صورت خطا نیز افزایش می‌یابد و هرچه تعداد بردارها بیشتر باشد، خطا کاهش می‌یابد. نتایج جدول ۵ و شکل ۵ نیز گویای همین مطلب می‌باشد.) حال برای تک‌تک نمونه‌های آموزشی، این آدرس را محاسبه و در صورت برابر بودن هرکدام از آن‌ها با آدرس یکی از جعبه‌های قبلی و در صورتی که اندازه آن جعبه از $3k$ تجاوز نکرده باشد، به آن جعبه الحاق می‌نماید. در غیر این صورت، یک جعبه جدید برای نمونه مورد نظر اختصاص می‌دهد. جهت کاهش خطا می‌توان متغیری به نام r را تعریف کرد و آزمایش‌ها را r بار انجام داد. آزمایش‌ها نیز نشان داد که چنین متغیری تأثیر به سزایی در افزایش صحت الگوریتم‌ها دارد.

Algorithm 2: Linear Combination Approximate k Nearest Neighbor (LCAkNN)

Input: $\left(V_1^{'},\ldots,V_{\text{num\_sample}}^{'}\right)$

Let s = ? be the dimension of the target space
Let r = ? be the number of repetitions
For l = ۱ to r
  Let $R_1,\ldots,R_s$ be random vectors such that

$$R_i[j]=\begin{cases}0 & :1-p_1\\ 1 & :p_1\end{cases}$$

  For i = 1 to num_sample
    Let $V_i^{''}=\left(R_1V_i^{'},R_2V_i^{'},\ldots,R_sV_i^{'}\right)$
    If $V_i^{''}$ is a bucket and contains at most $3k$ vectors then
      Add $V_i^{'}$ to $V_i^{''}$
    If $V_i^{''}$ does not exists then
      Create $V_i^{''}$ and add V_i to $V_i^{''}$

Output: Put each sample in its bucket.

الگوریتم ۲: ترکیب خطی

۳-۳- الگوریتم پرس‌وجو

الگوریتم پرس‌وجو، الگوریتمی جهت پرس‌وجو برای الگوریتم ترکیب خطی جاسازی‌شده می‌باشد. هنگامی‌که الگوریتم ترکیب خطی جاسازی‌شده بر روی تمام نمونه‌های آموزشی اعمال شود، حال نوبت به پرس‌وجو می‌رسد. بدین منظور، ابتدا آدرس جعبه پرس‌وجوی مذکور را طبق روشی که در الگوریتم ۲ شرح داده شده است، به دست آورده سپس نمونه‌هایی که آدرس جعبه آن‌ها با آدرس جعبه پرس‌وجوی مورد نظر برابر باشد را به مجموعه z اضافه می‌کند. در انتها، نرم l_1 هریک از نمونه‌های موجود در z را از پرس‌وجوی مورد نظر محاسبه کرده و k تا از کوچک‌ترین آن‌ها را به عنوان k نزدیک‌ترین همسایه به آن پرس‌وجو، برمی‌گرداند. بردار Q، بردار d عنصری هر نمونه آموزشی هر نمونه آموزشی، بردار $Q^{'}$، نسخه جاسازی‌شده مربوط به نمونه آموزشی متناظر

آن و ضرب داخلی S بردار تصادفی و بردار Q' را در بردار Q'' متناظر قرار می‌دهد. ابتدا تمامی محتویات جعبه پرس‌وجو تحت عنوان متغیر w به درون متغیر z انتقال داده می‌شود که متغیر w زیرمجموعه‌ای از مجموعه بزرگ v می‌باشد. سپس به اندازه تعداد عناصر متغیر z ($|Z|$)، فاصله نرم ۱ هر یک از عناصر w تا پرس‌وجو محاسبه شده و از کوچک به بزرگ مرتب می‌شوند و k تای کوچک آن به عنوان k نزدیک‌ترین همسایه تقریبی برگردانده می‌شود.

Algorithm 3: Query for LCAkNN ($Q \in \mathbb{R}^d, k$)
Input: Query (Q)
Let Z be initialized to an empty set
Let $I_1,\ldots,I_r$ be the random vector sets that each set has s random vectors
For $l=1$ to r
  Let Q' be the embeded of Q
  Let Q'' be the inner product of Q' and I_l
  If bucket Q'' exists then
    Add vectors of bucket Q'' to set Z
    $Z=\{W_1,\ldots,W_{|Z|}\}\subseteq\{V_1,\ldots,V_{\text{num\_sample}}\}$
For $i=1$ to $|Z|$
  Let $a_i=|Q-W_i|_1=L_1(Q,W_i)$
Sort $W_1,\ldots,W_{|Z|}$ based on $a_1,\ldots,a_{|Z|}$
Output the first k vectors from the sorted list.
Output: the first k vectors from the sorted list.

الگوریتم ۳: پرس‌وجو برای ترکیب خطی

۴- آزمایش‌ها و نتایج

الگوریتم پیشنهادی طی آزمایش‌هایی مورد ررسی قرار گرفته است و نتایج در بخش ۴-۱ آورده شده است.

۴-۱- نتایج الگوریتم نمونه‌برداری

الگوریتم پیشنهادی بر روی مجموعه داده MNIST با تغییر پارامترهای تعداد همسایگان، تعداد تکرار (r)، ظرفیت جعبه، احتمال انتخاب عدد یک و تعداد بردارهای تصادفی آزمایش شد و نتایج به شرح زیر است:

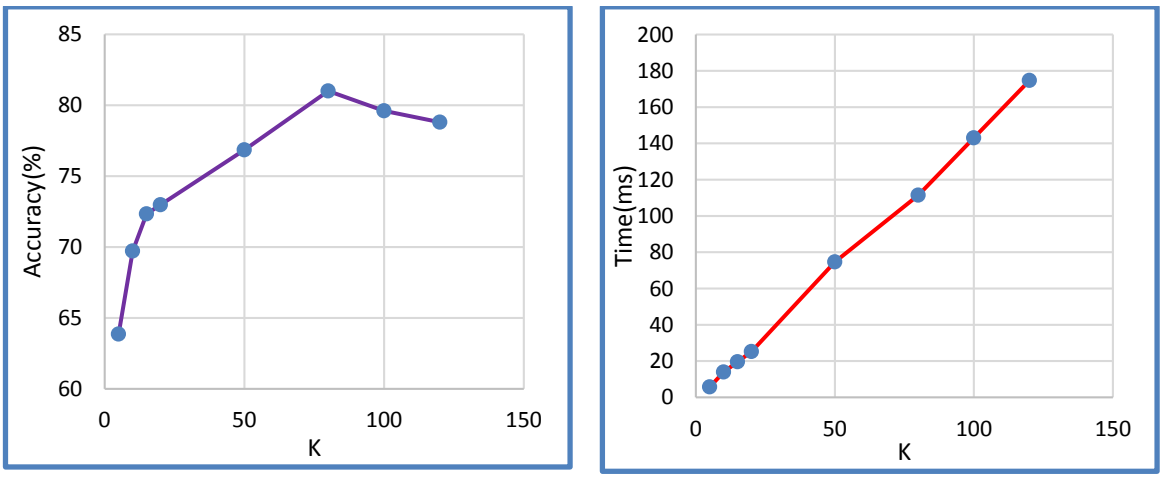
۴-۱-۱- تعداد همسایه‌های متفاوت

جدول ۲، الگوریتم LCAkNN با تعداد ۴ بردار تصادفی، با احتمال انتخاب عدد یک برابر با ۰/۰۰۰۰۱ و با تکرار ۵ مرحله ($r=5$)، برای kهای مختلف و بر روی تمامی مجموعه داده MNIST انجام گرفت. سطر اول میانگین درصد صحت را نشان می‌دهد و سطر دوم میانگین زمان اجرای هر پرس‌وجو به میلی‌ثانیه در ۱۰ بار اجرا نشان می‌دهد. همان‌طور که در جدول ۲ و شکل ۲ می‌بینیم، با افزایش k، صحت در حال بهبود می‌باشد اما در عمل، برای مقادیر k بیش‌تر از ۱۵، این افزایش چشم‌گیر نمی‌باشد. با توجه به اینکه زمان اجرا نیز افزایش می‌یابد بنابراین مقرون‌به‌صرفه نمی‌باشد که k را بیش‌تر از مقدار ۱۵ افزایش دهیم در نتیجه به نظر می‌آید $k=15$ یک نقطه خوب می‌باشد بدان معنی که در این نقطه یک مصالحه بین زمان و صحت می‌باشد بنابراین در تمام آزمایش‌های بعدی، $k=15$ انتخاب خواهد شد.

همان‌طور که می‌دانیم با افزایش تعداد همسایه‌ها، صحت باید افزایش یابد اما این افزایش در یک نقطه به دلیل افزایش خطا، افت خواهد کرد (در مقدار ۸۰ این افت رخ خواهد داد).

جدول ۲: میزان نرخ صحت بر اساس تعداد همسایه‌های انتخابی

تعداد همسایه (k)	۵	۱۰	۱۵	۲۰	۵۰	۸۰	۱۰۰	۱۲۰
میانگین صحت (٪)	۶۳/۸۶	۶۹/۷۱	۷۲/۳۳	۷۲/۹۸	۷۶/۸۵	۸۱	۷۹/۶	۷۸/۸
میانگین زمان (ms)	۵/۷۸	۱۳/۹۸	۱۹/۶۱	۲۵/۲۰	۷۴/۶۱	۱۱۱/۴	۱۴۳/۰۳	۱۷۴/۷۲



(الف) (ب)

شکل ۲: روند افزایش k؛ (الف) صحت و (ب) زمان مربوط به آزمایش‌ها را نشان می‌دهد.

۴-۱-۲- تعداد تکرارها متفاوت

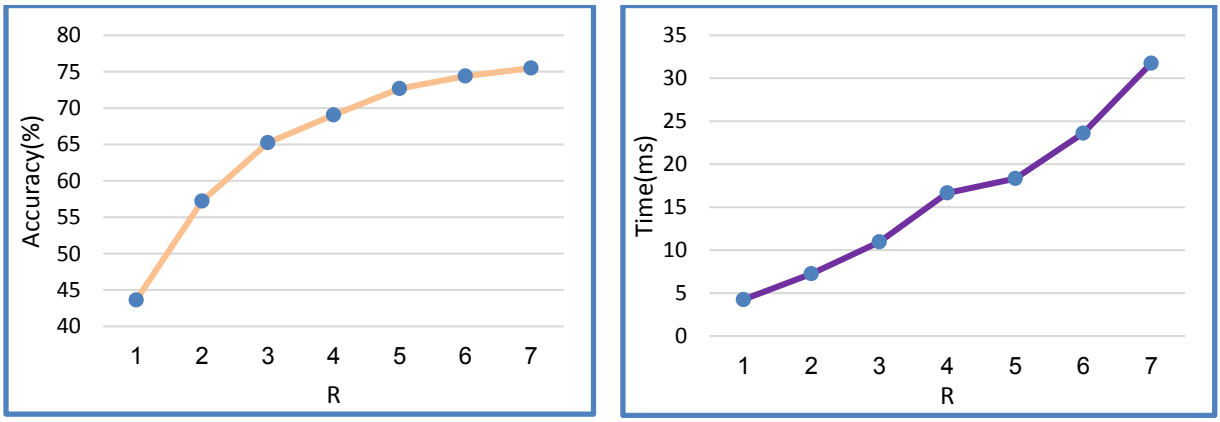
جدول ۳ الگوریتم LCAkNN را با تعداد ۴ بردار تصادفی، با احتمال انتخاب عدد یک برابر با ۰/۰۰۰۰۱ و برای k مساوی با ۱۵ و برای تکرارهای مختلف (r های مختلف) و بر روی تمامی پایگاه داده انجام گرفته است. سطر اول، میانگین صحت این ۱۰ بار اجرا و سطر آخر هر آزمایش، میانگین زمانی ۱۰ بار اجرا را نشان می‌دهد (لازم به ذکر است زمان‌ها به میلی‌ثانیه می‌باشد و زمان هر پرس‌وجو در سطر آخر جدول ۳ نشان داده شده است).

همان‌طور که در جدول ۳ و شکل ۳ مشاهده می‌شود، با افزایش r صحت در حال بهبود می‌باشد اما در عمل برای مقادیر r بیش‌تر از ۵، این افزایش چشم‌گیر نمی‌باشد و با توجه به اینکه زمان اجرا نیز افزایش می‌یابد بنابراین مقرون‌به‌صرفه نمی‌باشد که r را از ۵ افزایش دهیم در نتیجه به نظر می‌آید که $r=5$ یک نقطه خوب می‌باشد این بدان معنی می‌باشد که در این نقطه یک مصالحه بین زمان و صحت می‌باشد بنابراین در تمام آزمایش‌ها، $r=5$ انتخاب خواهد شد. نتایج به شرح زیر می‌باشد:

به دلیل تأثیر معکوس تعداد مراحل یا همان تکرار الگوریتم بر روی خطا (هرچه متغیر r یا همان تعداد تکرار الگوریتم، افزایش یابد، خطا نیز کاهش می‌یابد) بنابراین صحت افزایش می‌یابد اما به تبع آن زمان نیز افزایش می‌یابد.

جدول ۳: روند تأثیر تعداد مراحل بر روی صحت

تعداد مراحل (r)	۱	۲	۳	۴	۵	۶	۷
میانگین صحت (٪)	۴۳/۶۲	۵۷/۲۲	۶۵/۲۴	۶۹/۰۵	**۷۲/۶۷**	۷۴/۳۹	۷۵/۵۰
میانگین زمان (ms)	۴/۲۴	۷/۲۴	۱۰/۹۵	۱۶/۶۵	**۱۸/۳۲**	۲۳/۶۰	۳۱/۷۴



(الف) (ب)

شکل ۳: روند افزایش r؛ (الف) صحت و (ب) زمان مربوط به آزمایش‌ها را نشان می‌دهد

۴-۱-۳- ظرفیت جعبه‌های متفاوت

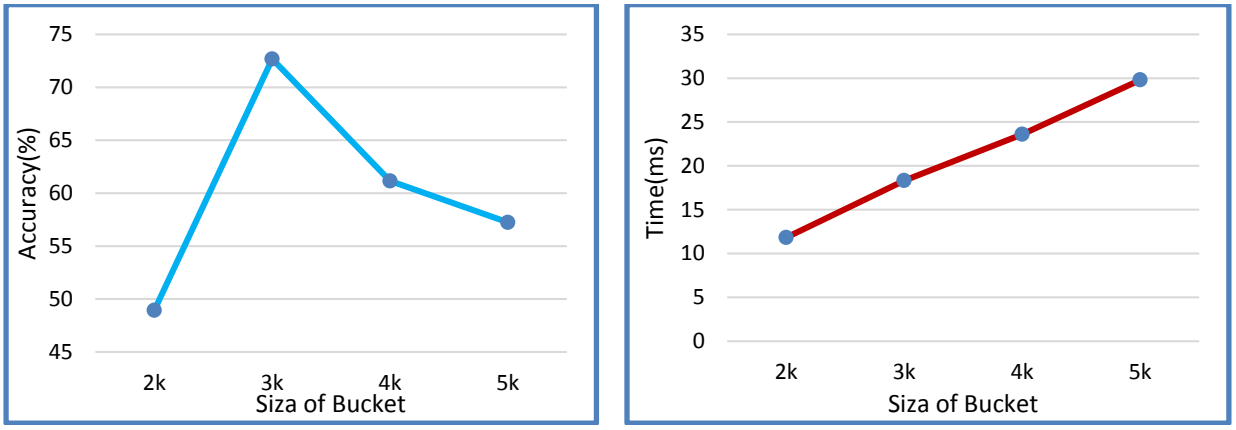
در الگوریتم LCAkNN حداکثر اندازه جعبه‌ها در آزمایش‌ها بررسی شد. جدول ۴ الگوریتم LCAkNN را برای k مساوی با ۱۵ و برای ظرفیت جعبه‌های مختلف و با تکرار ۵ مرحله‌ای ($r=5$) و با ۴ بردار تصادفی با احتمال انتخاب عدد یک برابر با ۰/۰۰۰۰۱ و بر روی تمامی پایگاه داده انجام گرفته است. سطر اول، میانگین صحت و سطر دوم، میانگین زمانی ۱۰ بار اجرا را نشان می‌دهد (لازم به ذکر است زمان‌ها به میلی‌ثانیه می‌باشد و زمان هر پرس‌وجو در سطر آخر جدول ۴ نشان داده شده است).

همان‌طور که در جدول ۴ و شکل ۴ دیده می‌شود بهترین ظرفیت $3k$ می‌باشد زیرا در این حالت هم صحت و هم‌زمان اجرای پرس‌وجو به میزان مطلوبی قابل قبول می‌باشند. بنابراین در تمام آزمایش‌ها این ظرفیت به‌منظور حداکثر اندازه جعبه‌ها در نظر گرفته شده است.

در خصوص ظرفیت جعبه‌ها، این نکته قابل ذکر است که مقدار بسیار زیاد و کم این پارامتر، باعث افزایش خطا می‌شود بنابراین مقداری میانی در این خصوص، مطلوب به نظر می‌رسد و آزمایش‌ها نیز به مقدار مطلوب $3k$ انجام گرفته است.

جدول ۴: روند تأثیر ظرفیت جعبه بر روی صحت

اندازه جعبه	2k	3k	4k	5k
میانگین صحت (٪)	۴۸/۹۴	**۷۲/۶۷**	۶۱/۱۵	۵۷/۲۴
میانگین زمان (ms)	۱۱/۸۲	**۱۸/۳۲**	۲۳/۶۰	۲۹/۸۱



(الف) (ب)

شکل ۴: روند افزایش اندازه جعبه؛ (الف) صحت و (ب) زمان مربوط به آزمایش‌ها را نشان می‌دهد

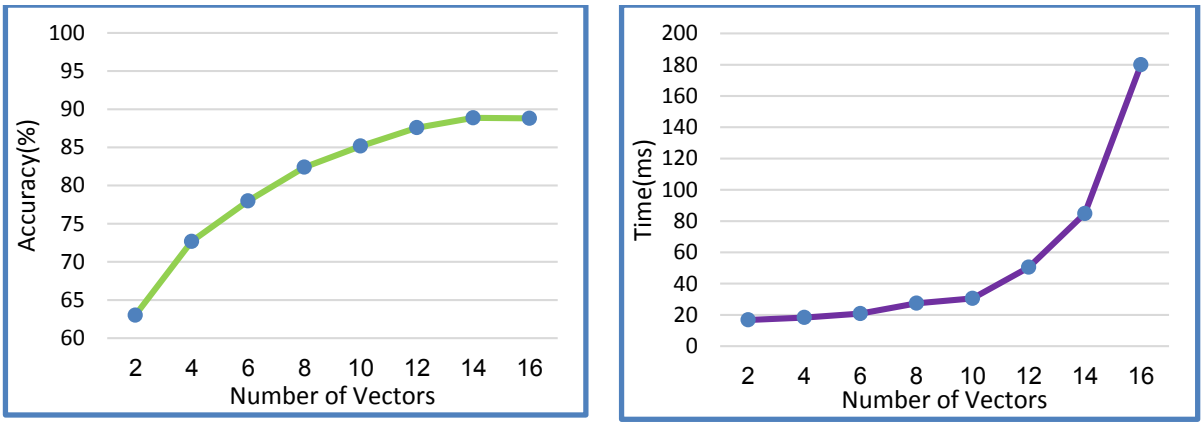
۴-۱-۴- تعداد بردارهای تصادفی متفاوت

جدول ۵ الگوریتم LCAkNN را با احتمال انتخاب عدد یک برابر با ۰/۰۰۰۰۱ و برای k مساوی با ۱۵ و برای بردارهای تصادفی مختلف و با تکرار ۵ مرحله‌ای ($r=5$)، بر روی تمامی پایگاه داده انجام گرفته است. سطر اول، میانگین صحت و سطر آخر هر آزمایش، میانگین زمانی ۱۰ بار اجرا را نشان می‌دهد (لازم به ذکر است زمان‌ها به میلی‌ثانیه می‌باشد و زمان هر پرس‌وجو در سطر آخر جدول ۵ نشان داده شده است). هنگامی‌که تعداد بردارهای تصادفی افزایش می‌یابد، زمان انجام الگوریتم بسیار کند خواهد شد. همان‌طور که در جدول ۵ و شکل ۵ مشاهده می‌شود، با افزایش تعداد بردارهای تصادفی، صحت در حال بهبود می‌باشد اما در عمل با تعداد بیش از ۸ بردار، این افزایش چشم‌گیر نمی‌باشد و با توجه به اینکه زمان اجرا نیز افزایش می‌یابد بنابراین مقرون‌به‌صرفه نمی‌باشد که تعداد بردارها را بیش‌تر از تعداد ۸ عدد افزایش دهیم در نتیجه به نظر می‌آید هنگامی‌که ۸ بردار انتخاب شود یک حالت خوب می‌باشد به بیان دیگر در این حالت یک مصالحه بین زمان و صحت می‌باشد.

لازم به ذکر است که متغیر s به صورت مضارب عدد ۲ مقداردهی شده است. در واقع به دلیل اینکه با افزایش این متغیر، رشد زمان اجرای الگوریتم محسوس می‌باشد بنابراین سعی شده است این افزایش به آرامی صورت گیرد. بنابراین، هرچه این مقدار افزایش یابد، مقدار ضرب اسکالر به واقعیت نزدیک‌تر می‌شود و بنابراین صحت افزایش می‌یابد اما به تبع آن زمان نیز افزایش می‌یابد. به عبارت دیگر، هرچه تعداد بردارهای تصادفی کمتر باشد، به دلیل اینکه طول آدرس جدول درهم‌ساز کوچک می‌شود بنابراین ممکن است آدرس مشابه زیادی تولید شود که در این صورت خطا نیز افزایش می‌یابد و هرچه تعداد بردارها بیشتر باشد، خطا کاهش می‌یابد.

جدول ۵: روند تأثیر تعداد بردارهای تصادفی انتخاب‌شده بر روی صحت

تعداد بردارهای تصادفی	۲	۴	۶	۸	۱۰	۱۲	۱۴	۱۶
میانگین صحت (٪)	۶۳/۰۰	۷۲/۶۷	۷۷/۹۷	**۸۲/۳۹**	۸۵/۱۶	۸۷/۵۸	۸۸/۷۸	۸۸/۸۰
میانگین زمان (ms)	۱۶/۷۸	۱۸/۳۲	۲۰/۸۵	**۲۷/۴۲**	۳۰/۵۶	۵۰/۴۷	۸۴/۸۲	۱۷۹/۹۷



(الف) (ب)

شکل ۵: روند افزایش تعداد بردارهای انتخابی؛ (الف) صحت و (ب) زمان مربوط به آزمایش‌ها را نشان می‌دهد

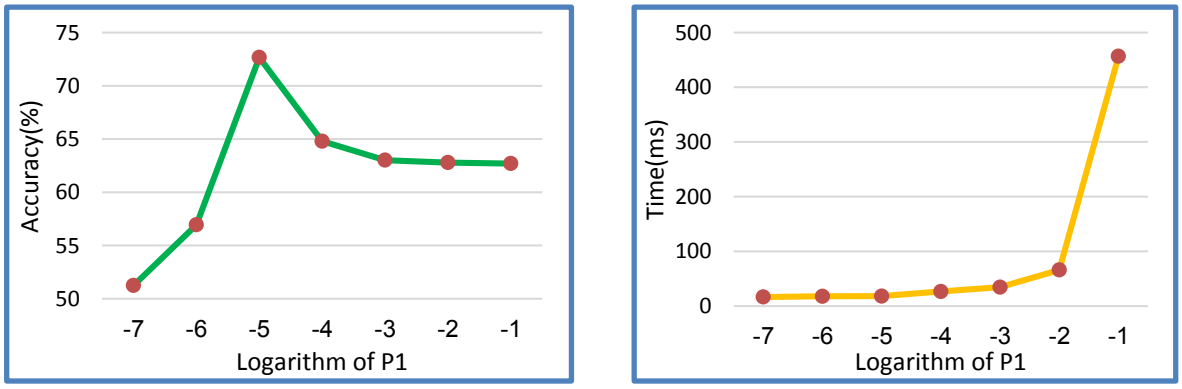
۴-۱-۵- احتمالات متفاوت

جدول ۶ الگوریتم LCAkNN برای k مساوی با ۱۵ و با ۴ بردار تصادفی با احتمالات انتخاب متفاوت عدد یک و با تکرار ۵ مرحله‌ای ($r=5$) و بر روی تمامی پایگاه داده انجام گرفته است. سطر اول، میانگین صحت و سطر آخر هر آزمایش، میانگین زمانی برای ۱۰ بار اجرا را نشان می‌دهد (لازم به ذکر است زمان‌ها به میلی‌ثانیه می‌باشد و زمان هر پرس‌وجو در سطر آخر جدول ۶ نشان داده شده است). زمانی که احتمال انتخاب عدد یک ۰/۱ باشد، به‌طور متوسط ۲۰۰۰۰ عدد یک تولید می‌شود و در نتیجه زمان انجام الگوریتم و انجام ضرب اسکالر به دلیل وجود تعداد یک زیاد، بسیار کند خواهد شد. این موضوع در جدول ۶ نشان داده شده است. همان‌طور که در جدول ۶ و شکل ۶ مشاهده می‌شود، افزایش احتمال وقوع ۱، صحت جواب‌ها ابتدا زیاد می‌شود تا اینکه برای $p_1=0.00001$ به حداکثر خود می‌رسد سپس افزایش احتمال رخداد ۱، باعث کاهش صحت جواب‌ها می‌شود. احتمال $p_1=0.00001$ به معنای این است که برای هر بردار جاسازی‌شده، هر بردار نمونه به‌طور متوسط دو مقدار ۱ دارد، در نتیجه دو بیت نمونه می‌گیرد. بنابراین نتایج به شرح زیر می‌باشد:

در مورد روند تأثیر احتمالات متفاوت انتخاب عدد یک بر روی صحت، به نظر می‌رسد هرچه این احتمال به سمت صفر میل نماید، به دلیل اینکه تعداد بیت‌های بردارهای تصادفی ایجاد شده تماماً صفر خواهند شد و ضرب اسکالر در عمل بی‌اثر خواهد شد. همچنین هرچه این احتمال به سمت یک میل نماید، به دلیل اینکه تعداد بیت‌های بردارهای تصادفی ایجاد شده تماماً یک خواهند شد و ضرب نقطه‌ای در عمل بی‌اثر خواهد شد بنابراین مقدار میانی بین ۰ و ۱ برای این متغیر منطقی‌تر می‌باشد.

جدول ۶: روند تأثیر احتمالات متفاوت انتخاب عدد یک بر روی صحت

احتمال انتخاب عدد یک (p_1)	۰/۱	۰/۰۱	۰/۰۰۱	۰/۰۰۰۱	۰/۰۰۰۰۱	۰/۰۰۰۰۰۱	۰/۰۰۰۰۰۰۱
میانگین صحت (٪)	۶۲/۷۰	۶۲/۷۹	۶۳/۰۲	۶۴/۸۰	**۷۲/۶۷**	۵۶/۹۴	۵۱/۲۴
میانگین زمان (ms)	۴۵۶/۴۴	۶۶/۱۳	۳۴/۶۶	۲۶/۷۸	**۱۸/۳۲**	۱۷/۹۰	۱۶/۵۹



(الف) (ب)

شکل ۶: روند افزایش احتمال وقوع ۱؛ (الف) صحت و (ب) زمان مربوط به آزمایش‌ها را نشان می‌دهد

۴-۱-۶- مقایسه روش‌های پیشنهادی با چندین روش پیشین

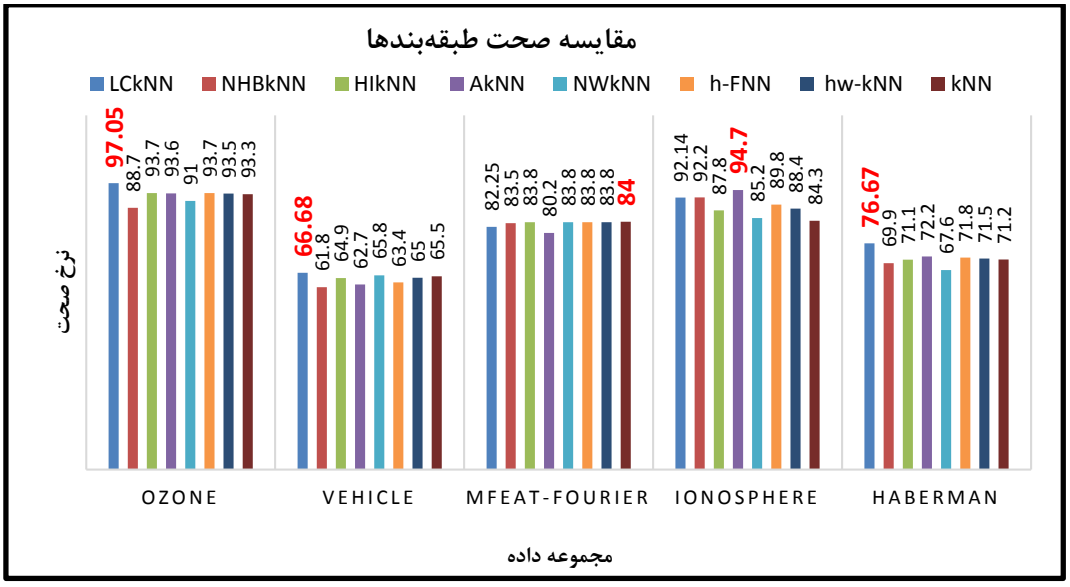
روش پیشنهادی با چند روش و بر روی ۵ مجموعه داده مقایسه شده است و برتری روش پیشنهادی قابل مشاهده می‌باشد. لازم به ذکر است که برای این مقایسه، به دلیل استفاده مقاله [۶، ۱۶] از روش اعتبارسنجی متقاطع ۱۰ تایی[۱۴]، ما نیز همین روش را به کار برده‌ایم. همان‌طور که در جدول ۶ مشاهده می‌شود روش پیشنهادی نسبت به سایر روش‌ها بهبود داشته است. جدول ۷ تعداد بردارهای استفاده‌شده برای انجام مقایسه صحت طبقه‌بندها را نشان می‌دهد. شکل ۷ این مقایسه را به‌خوبی نشان می‌دهد و برتری روش پیشنهادی نسبت به روش‌های پیشنهادی ذکرشده در مقاله [۶] و روش‌های پیشنهادی در مقاله [۱۶] را نشان می‌دهد. از آنجایی که داده‌ها در فضای با ابعاد بالا، در بسیاری از مواقع، اغلب ماتریس خلوت می‌باشند، بنابراین این الگوریتم برای ماتریس‌های خلوت منجر به حاصل شدن جواب‌های مناسب‌تری خواهد شد بنابراین علت بد بودن بعضی از جواب‌های روش پیشنهادی این می‌باشد که مجموعه داده مورد نظر در آن حالت، تشکیل ماتریس چگال می‌دهند نه ماتریس خلوت. نتایج به شرح زیر می‌باشد:

جدول ۷: مقایسه صحت طبقه‌بندها

روش		kNN	hw-kNN [۶]	h-FNN [۶]	NWkNN [۶]	AkNN [۶]	HIkNN [۶]	NHBkNN [۶]	LCAkNN (روش پیشنهادی)
مجموعه داده	Ozone	۹۳/۳	۹۳/۵	۹۳/۷	۹۱/۰	۹۳/۶	۹۳/۷	۸۸/۷	۹۷/۰۵
	Vehicle	۶۵/۵	۶۵/۰	۶۳/۴	۶۵/۸	۶۲/۷	۶۴/۹	۶۱/۸	٦٦/٦٨
	Mfeat-fourier	۸٤/۰	۸۳/۸	۸۳/۸	۸۳/۸	۸۰/۲	۸۳/۸	۸۳/۵	۸۲/۲۵
	Ionosphere	۸۴/۳	۸۸/۴	۸۹/۸	۸۵/۲	۹٤/۷	۸۷/۸	۹۲/۲	۹۲/۱۴
	haberman	۷۱/۲	۷۱/۵	۷۱/۸	۶۷/۶	۷۲/۲	۷۱/۱	۶۹/۹	۷٦/٦۷

جدول ۸: تعداد بردارهای استفاده‌شده در مقایسه صحت طبقه‌بندها

تعداد بردار استفاده‌شده		LCAkNN
مجموعه داده	Ozone	۴
	Vehicle	۲۵۶
	Mfeat-fourier	۴۰۰
	Ionosphere	۱۵۰
	haberman	۶۴



شکل ۷: نمودار مقایسه صحت طبقه‌بندها

۵- نتیجه‌گیری و کارهای آینده

در این مقاله، روش LCAkNN بر اساس ضرب داخلی بردارهای تصادفی ارائه شده است. روش مزبور با روش‌های مطرح مرز دانش مقایسه شده است و نتایج آزمایش‌ها نشان‌دهنده برتری نسبی روش پیشنهادی از نظر صحت نسبت به روش‌های مرز دانش می‌باشد. از آنجایی که داده‌ها در فضای با ابعاد بالا، در مواقع زیادی، اغلب ماتریس خلوت می‌باشند، بنابراین روش مزبور زمانی مناسب می‌باشد که تراکم بردارها، خلوت باشد. در این صورت باید با احتمال بیش‌تری هر عنصر بردار تصادفی که در هر بردار ضرب می‌شود را یک بگذاریم (به عبارت دیگر، احتمال اینکه هر عنصری در بردار تصادفی یک باشد باید بیش‌تر باشد). هرچه بردار چگال‌تر باشد، برای اینکه صحت LCAkNN بهتر شود باید عناصر بردارهای تصادفی با احتمال بیش‌تری صفر باشند یا به عبارت دیگر عناصر کم‌تری از هر بردار با هم ترکیب شده و باقی‌مانده آن‌ها بر ۲ محاسبه گردد. بهترین دقت برای بردارهای خلوت زمانی بوده است که از LCAkNN استفاده کرده‌ایم و با احتمال ۰/۰۰۰۰۱ هر عنصر تصادفی را یک نسبت داده‌ایم یا به عبارت دیگر، LCAkNN دو عنصر از هر بردار را با هم به دست آورده‌ایم. در این حالت صحت ۸۸/۸۰ به دست آمده است. در روش LCAkNN، برای بردارهای چگال، اگر عناصر زیادی را با هم ترکیب کنیم (با احتمال زیادی هر عنصر بردار تصادفی یک شود) سرعت بسیار بالا و صحت بسیار پایین خواهد آمد ولی اگر احتمال یک شدن بسیار پایین باشد، سرعت پایین و صحت بالا می‌رود. نتایج آزمایش‌ها نشان‌دهنده این نکته می‌باشد که هرچه r و تعداد بردارهای تصادفی انتخاب‌شده را افزایش دهیم صحت روبه‌بهبود می‌رود اما زمان انجام الگوریتم پرس‌وجو نیز به شدت افزایش می‌یابد. نشان دادیم که k و r و تعداد بیت‌های تصادفی انتخاب‌شده به ترتیب ۱۵ و ۵ باشد به مصالحه خوبی بین زمان و صحت دست می‌یابیم.

به عنوان کارهای آینده، می‌توان پیشنهاد کرد که روش درهم‌ساز را با وزن‌دهی hubness فازی ترکیب نمود بدین صورت که پس از اینکه تقلیل بعد صورت گرفت و تعداد ابعاد کاهش یافت حال می‌توان وزن‌دهی hubness فازی را اعمال نمود و نتایج را به دست آورد.

مراجع

[1] J. Wang, X. S. Xu, S. Guo, L. Cui and X. L. Wang, "Linear unsupervised hashing for ANN search in Euclidean space," *Neurocomputing*, 2015.

[2] S. Har-Peled, P. Indyk and R. Motwani, "Approximate nearest neighbor: towards removing the curse of dimensionality," *Theory of Computing*, vol. 8, no. 1, pp. 321-350, 2012.

[3] M. Muja and D. G. Lowe, "Scalable nearest neighbor algorithms for high dimensional data," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 36, no. 11, pp. 2227-2240, 2014.

[4] J. Hamidzadeh, "IRDDS: Instance reduction based on Distance-based decision surface," *Journal of AI and Data Mining*, vol. 3, no. 2, pp. 121-130, 2015.

[۵] مهرداد حیدری ارجلو، سید قدرت‌اله سیف‌السادات، مرتضی رزاز، «یک روش هوشمند تشخیص جزیره در شبکه توزیع دارای تولیدات پراکنده مبتنی بر تبدیل موجک و نزدیک‌ترین k-همسایگی (kNN)»، *مجله مهندسی برق دانشگاه تبریز*، جلد ۴۳، شماره ۱، صفحات ۱۵-۲۶، ۱۳۹۲.

[6] N. Tomasev, M. Radovanovic, D. Mladenic and M. Ivanovic, "The role of hubness in clustering high-dimensional data," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 26, no. 3, pp. 739-751, 2014.

[7] Z. Jin, D. Zhang, Y. Hu, S. Lin, D. Cai and X. He, "Fast and accurate hashing via iterative nearest neighbors expansion," *Cybernetics, IEEE Transactions on*, vol. 44, no. 11, pp. 2167-2177, 2014.

[8] J. H. Friedman, J. L. Bentley and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Transactions on Mathematical Software (TOMS)*, vol. 3, no. 3, pp. 209-226, 1977.

[9] K. Beyer, J. Goldstein, R. Ramakrishnan and U. Shaft, "When is "nearest neighbor" meaningful?," *In Database Theory—ICDT'99*, pp. 217-235, 1999.

[10] A. Guttman, "R-trees: a dynamic index structure for spatial searching," *ACM*, vol. 14, no. 2, pp. 47-57, 1984.

[11] N. Beckmann, H. P. Kriegel, R. Schneider and B. Seeger, "The R*-tree: an efficient and robust access method for points and rectangles," *ACM*, vol. 19, no. 2, pp. 322-331, 1990.

[12] C. Silpa-Anan and R. Hartley, "Optimised KD-trees for fast image descriptor matching," *In Computer Vision and Pattern Recognition*, pp. 1-8, 2008.

[13] N. Katayama and S. I. Satoh, "The SR-tree: An index structure for high-dimensional nearest neighbor queries," *In ACM SIGMOD Record*, vol. 26, no. 2, pp. 369-380, 1997.

[14] P. Ciaccia, M. Patella and P. Zezula, "DEIS-CSITE-CNR," *In Proceedings of the International Conference on Very Large Data Bases*, vol. 23, 1997.

[15] S. Dasgupta and Y. Freund, "Random projection trees and low dimensional manifolds," *In Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, pp. 537-546, 2008.

[16] N. Tomašev and K. Buza, "Hubness-aware kNN classification of high-dimensional data in presence of label noise," *Neurocomputing*, vol. 160, pp. 157-172, 2015.

[۱۷] سیدمهدی مظفری، حسن منصف، حمید فلقی، «رویکردی جدید بر توسعه پست‌های فوق‌توزیع مبتنی بر وزن‌دهی فازی ویژگی‌های الکتریکی و جغرافیایی شبکه توزیع مورد مطالعه»، *مجله مهندسی برق دانشگاه تبریز*، جلد ۴۲، شماره ۱، صفحات ۸۵-۹۱، ۱۳۹۱.

[18] J. Wang, S. Kumar and S. F. Chang, "Sequential projection learning for hashing with compact codes," *In Proceedings of the 27th International Conference on Machine Learning*, pp. 1127-1134, 2010.

[19] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," *In Foundations of Computer Science*, pp. 459-468, 2006.

[20] R. Panigrahy, "Entropy based nearest neighbor search in high dimensions," *In Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, pp. 1186-1195, 2006.

[21] B. Kulis and K. Grauman, "Kernelized locality-sensitive hashing," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 34, no. 6, pp. 1092-1104, 2012.

[22] T. Trzcinski, V. Lepetit and P. Fua, "Thick boundaries in binary space and their influence on nearest-neighbor search," *Pattern Recognition Letters*, vol. 33, no. 16, pp. 2173-2180, 2012.
[23] A. Gionis, P. Indyk and R. Motwani, "Similarity search in high dimensions via hashing," *In VLDB*, vol. 99, pp. 518-529, 1999.
[24] E. Kushilevitz, R. Ostrovsky and Y. Rabani, "Efficient search for approximate nearest neighbor in high dimensional spaces," *SIAM Journal on Computing*, vol. 30, no. 2, pp. 457-474, 2000.
[25] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, 1998.

زیرنویس‌ها

[1] Query
[2] Curse Dimensional
[3] Approximate Nearest Neighbor
[4] Hashing
[5] Locality Sensitive Hashing
[6] Collision
[7] Bucket
[8] Rectangle-tree
[9] Fuzzy nearest neighbor algorithm
[10] Hubness-Weighted kNN
[11] Hypercube
[12] Inner Product
[13] Linear Combination Approximate k Nearest Neighbor
[14] 10-fold cross-validation