

Azure Cognitive Search でのフルテキスト検索

[アーティクル] • 2023/09/27

この記事は、Azure Cognitive Search におけるフルテキスト検索のしくみについて理解を深める必要がある開発者を対象としています。テキストクエリに関して、Azure Cognitive Search はほとんどの状況で速やかに適切な結果を返します。しかし一見、間違っているのではないか、と思うような結果が返されることも皆無ではありません。このような状況では、Lucene による 4 段階から成るクエリ実行（クエリ解析、字句解析、文書のマッチング、スコア付け）についての背景知識があると、具体的にどのような変更をクエリ パラメーターやインデックス構成に加えれば目的の結果が生成されるかが特定しやすくなります。

① 注意

Azure Cognitive Search では、フルテキスト検索に Apache Lucene[↗] が使われていますが、Lucene の機能がそのままの形で統合されているわけではありません。Microsoft は、Azure Cognitive Search にとって重要なシナリオを実現する Lucene の機能を選んで公開、拡張しています。

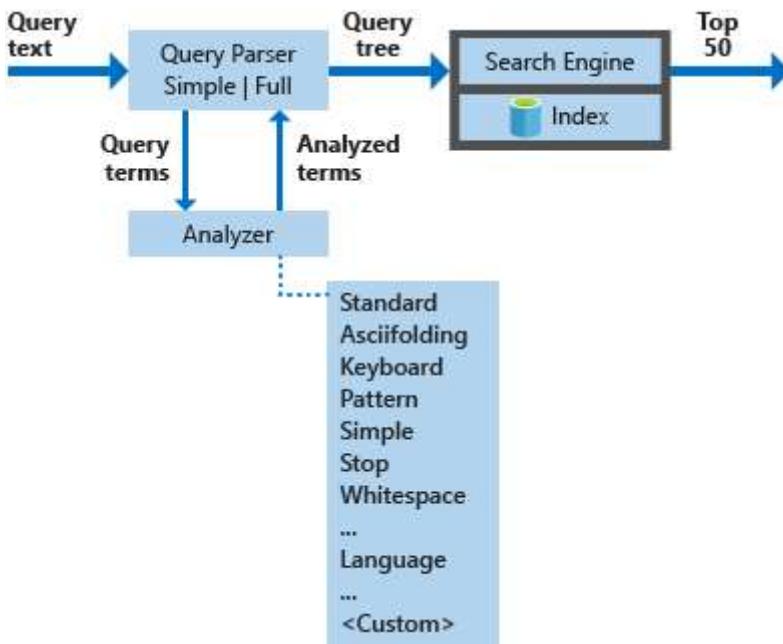
アーキテクチャの概要と図

クエリの実行には次の 4 つの段階があります。

1. クエリ解析
2. 字句解析
3. 文書検索
4. ポイントの計算

フルテキスト検索クエリは、クエリ テキストを解析して検索語と演算子を抽出することから始まります。2 つのパーサーがあり、速度と複雑さを選択できます。次に分析段階では、個々の検索語がしばしば分解され、新しい形に再構築されます。この手順では、できるだけ広い範囲から "一致と見なす候補" が得られます。検索エンジンは、インデックスをスキャンして、一致する語句を持つドキュメントを検索し、各一致をスコア付けします。その後、一致する個々の文書に割り当てられた関連度スコアに基づいて結果セットが並べ替えられます。このようにランク順に並んだリストの先頭の結果が、呼び出し元のアプリケーションに返されます。

以下の図は、検索要求の処理に使用されるコンポーネントを示しています。



主要コンポーネント 機能の説明

クエリ パーサー	クエリ演算子から検索語を切り離し、検索エンジンに送るクエリ構造(クエリツリー)を作成します。
アナライザ	検索語に対する字句解析を実行します。このプロセスには、検索語の変換、削除、拡大が伴う場合があります。
インデックス	索引付けの対象文書から抽出した検索可能な語句を効率よく体系的に格納するデータ構造です。
検索エンジン	転置インデックスの内容に基づいて、一致する文書を検索してスコア付けします。

検索要求の構造

検索要求は、結果セットで返すべき内容を詳細に規定した仕様です。最も単純な形式は、どのような種類の条件も含まれていない空のクエリです。しかしそれより現実的な例では、パラメーターや複数の検索語を伴うのが一般的です。場合によっては、検索範囲を特定のフィールドに限定したり、フィルター式や並べ替え規則が使われたりすることもあります。

次の例は、REST API を使用して Azure Cognitive Search に送信できる検索要求です。

```

POST /indexes/hotels/docs/search?api-version=2020-06-30
{
    "search": "Spacious, air-condition* +\"Ocean view\"",
    "searchFields": "description, title",
}
  
```

```
    "searchMode": "any",
    "filter": "price ge 60 and price lt 300",
    "orderby": "geo.distance(location, geography'POINT(-159.476235
22.227659)'')",
    "queryType": "full"
}
```

この要求に対して、検索エンジンは次の操作を実行します。

1. 価格が \$60 以上 \$300 未満の文書を検索します。
2. クエリを実行します。この例では、検索クエリが `"Spacious, air-condition*`
`+\"Ocean view\""` という語句で構成されています (通常はユーザーが句読点を入力することはありませんが、この例では、アナライザーによる処理を説明するために入えて含めています)。

このクエリの場合、検索エンジンは、"searchFields" に指定された説明フィールドとタイトルフィールドをスキャンして、`"Ocean view"` を含む文書、さらに `"spacious"` という語句、または `"air-condition"` というプレフィックスで始まる語句を探します。明示的に必須指定 (+) されていない語句に関して、マッチング対象を任意 (既定) とするか、すべてとするかが、"searchModel" パラメーターで指定されています。

3. 結果として得られた一連のホテルを特定の地理的位置に近い順に並べ替え、結果を呼び出し元のアプリケーションに返します。

この記事では主に、検索クエリ: `"Spacious, air-condition* +\"Ocean view\""` の処理について取り上げています。フィルター処理と並べ替えについては取り上げません。 詳細については、[Search API のリファレンス ドキュメント](#)を参照してください。

第 1 段階: クエリ解析

前出のとおり、検索要求の最初の行がクエリ文字列です。

```
"search": "Spacious, air-condition* +\"Ocean view\"",
```

クエリ パーサーは、検索語から演算子 (この例の `*` や `+`) を切り離し、検索クエリを "サブクエリ" に分解します。次の種類のサブクエリがサポートされています。

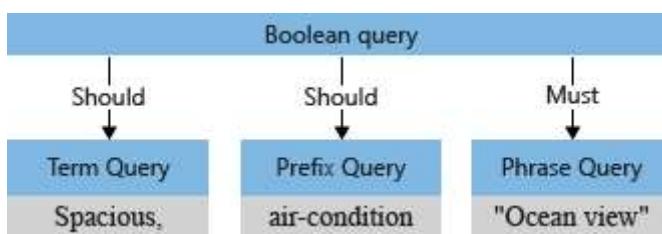
- "単語検索": 独立した語を検索 (`spacious` など)
- "フレーズ検索": 引用符で囲まれた語句を検索 (`ocean view` など)

- ・ "プレフィックス検索": 語句 + プレフィックス演算子 * を検索 (air-condition など)

サポートされているクエリの種類すべての一覧については、[Lucene のクエリ構文](#)に関するページを参照してください

文書が一致していると見なされるために検索条件との一致が "必須 (must)" であるのか "勧告 (should be)" であるのかは、サブクエリに関連付けられている演算子によって決まります。たとえば、`+ "Ocean view"` は + 演算子が指定されているので検索条件との一致が "必須" となります。

クエリパーサーは、サブクエリを "クエリツリー" (クエリを表す内部的な構造) に再構築して検索エンジンに渡します。クエリ解析の第 1 段階で、クエリツリーは次のようにになります。



サポートされるパーサー: Simple と Full Lucene

Azure Cognitive Search では、`simple` (既定) と `full` の 2 種類のクエリ言語が使用されます。どちらのクエリ言語を使うかは、検索要求で `queryType` パラメーターの設定で指定します。その指定に基づいて、クエリパーサーが演算子と構文を解釈します。

- ・ [Simple クエリ言語](#)は直感的で安定しており、多くの場合、クライアント側の処理を行わなくてもユーザー入力をそのまま解釈するのに適しています。Web 検索エンジンで多く使われているクエリ演算子がサポートされます。
- ・ [Full Lucene クエリ言語](#)は、`queryType=full` を設定することによって利用できます。より多くの演算子やクエリの種類 (ワイルドカード、あいまい一致、正規表現、フィールド指定検索など) がサポートされることで、既定の Simple クエリ言語が拡張されます。たとえば、Simple クエリ構文で送信された正規表現は、式としてではなくクエリ文字列と解釈されます。この記事で紹介している要求の例では、Full Lucene クエリ言語を使用しています。

searchMode がパーサーに及ぼす影響

解析方法に作用する検索要求パラメーターとしては、他にも "searchMode" パラメーターがあります。これは、ブールクエリの既定の演算子、つまり `any` (既定) または `all` を制御します。

"searchMode=any"とした場合(既定)、spaciousとair-conditionとの間に区切り記号として置かれた空白はOR(||)の働きをするため、サンプルクエリテキストは次のクエリテキストに相当します。

```
Spacious,||air-condition*+"Ocean view"
```

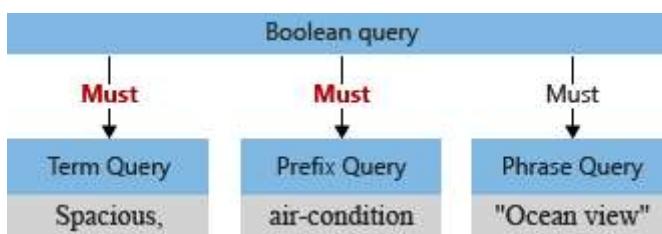
ブルクエリの構造において、明示的な演算子(+ "Ocean view" の+など)の意味ははつきりしています。つまり検索条件との一致は"必須(must)"です。それに比べて、残りの語句(spaciousとair-condition)の解釈はあいまいです。検索エンジンが探すべきなのは、ocean viewとspaciousとair-conditionの"すべて"との一致でしょうか。それとも、ocean viewに加えて、残りの2つの語句のうち、"どちらか一方"のみが含まれていればよいのでしょうか。

既定("searchMode=any")では、検索エンジンはより広い解釈を想定します。どちらか一方のフィールドが一致していればよい(should)の意味、つまり"or"のセマンティクスで解釈されます。先ほど例に挙げた、2つの"should"演算を含んだクエリツリーは既定の動作を示しています。

では、"searchMode=all"と設定したらどうなるでしょうか。この場合は、空白文字が"and"演算と解釈されます。残りの2つの語句が両方とも文書に存在したときに初めて一致と見なされます。最終的にサンプルクエリは次のように解釈されます。

```
+Spacious,+air-condition*+"Ocean view"
```

変更後のクエリツリーは次のようになり、一致する文書は3つすべてのサブクエリの積集合となります。



① 注意

"searchMode=all"より"searchMode=any"を選ぶ場合は、代表的なクエリを実行したうえで判断することをお勧めします。普段から演算子を指定するユーザー(ドキュメントストアを検索するときなど)は、"searchMode=all"で得られるブルクエリの構造の方が直感的にわかりやすいかもしれません。 "searchMode"と演算

子の相互作用について詳しくは、「Simple クエリ構文」に関するページをご覧ください。

第 2 段階: 字句解析

クエリツリーが構築された後、"単語検索" と "フレーズ検索" のクエリがアナライザーによって加工されます。アナライザーは、パーサーから渡されたテキスト入力を受け取ってそのテキストを加工してから、トークン化した語句をクエリツリーに組み入れます。

最も一般的な字句解析は、特定の言語に固有の規則に従って検索語を変換する *言語分析です。

- 検索語を単語の原形にします。
- **ストップワード**、つまり検索上の重要性がさほど高くない単語 (英語であれば "the" や "and") を削除します
- 複合語をその構成要素に分解します。
- 単語の大文字を小文字に変換します。

通常はこれらの操作をひととおり適用することで、ユーザーによって入力されたテキストと、インデックスに格納されている語句との相違点が取り除かれます。こうした操作はテキスト処理の範囲を超えており、言語そのものに対する深い知識が必要となります。この言語知識のレイヤーを追加するために、Azure Cognitive Search は、Lucene と Microsoft から提供されているさまざまな言語アナライザーに対応しています。

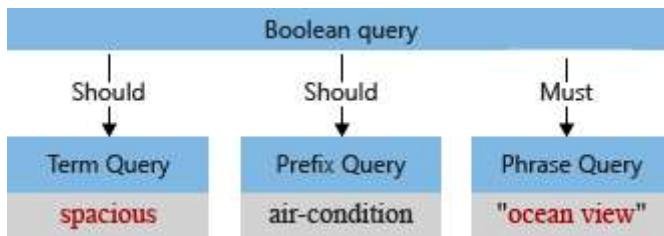
① 注意

解析要件は、実際のシナリオによって大きく異なります。ごく最低限で済む場合もあれば、膨大な作業が必要となる場合もあります。字句解析の難易度は、あらかじめ定義されているいづれかのアナライザーを選択するか、**カスタム アナライザー**を独自に作成するかによって決まります。アナライザーは、検索可能なフィールドにその適用対象が限定されており、フィールド定義の一環として指定されます。これにより、フィールドごとに多様な字句解析を行うことができます。指定されなかった場合は、"標準" の Lucene アナライザーが使用されます。

この例では、解析前の最初のクエリツリーに "Spacious," という語句があり、大文字の "S" とコンマはクエリパーサーによって検索語の一部として解釈されています (コンマはクエリ言語の演算子とは見なされません)。

既定のアナライザーは、この語句を加工する際、"ocean view" と "spacious" に含まれている大文字を小文字に変換し、さらにコンマを削除します。変更後のクエリツリー

は次のようにになります。



アナライザーの動作テスト

アナライザーの動作は、[Analyze API](#) を使ってテストすることができます。解析したいテキストを入力すると、指定したアナライザーからどのような語句が生成されるかを確認できます。たとえば、標準アナライザーで "air-condition" というテキストがどのように加工されるかを確認するには、次のように要求します。

```
JSON

{
  "text": "air-condition",
  "analyzer": "standard"
}
```

標準アナライザーは、入力テキストを次の 2 つのトークンに分解します。その際、開始オフセットと終了オフセット (一致部分の強調表示に使用される) やその位置 (フレーズの照合に使用される) など、各種の属性を使ってそれらに注釈を付けます。

```
JSON

{
  "tokens": [
    {
      "token": "air",
      "startOffset": 0,
      "endOffset": 3,
      "position": 0
    },
    {
      "token": "condition",
      "startOffset": 4,
      "endOffset": 13,
      "position": 1
    }
  ]
}
```

字句解析の例外

字句解析が適用されるのは、語句全体を必要とする種類の検索(単語検索とフレーズ検索)だけです。語句全体を必要としない種類の検索(プレフィックス検索、ワイルドカード検索、正規表現検索など)やあいまい検索には適用されません。前出の例の `air-condition*` という語を使ったプレフィックス検索も含め、こうした種類の検索は、解析段階を経ずに直接クエリツリーに追加されます。こうした種類の検索語に対して適用される変換は、大文字から小文字への変換だけです。

第3段階: 文書検索

ここでいう文書検索とは、一致する語句がインデックスに存在する文書を見つけることです。この段階は、例を使用するとよくわかります。まず、次のような単純なスキマを使用した `hotels` というインデックスを考えてみましょう。

```
JSON

{
    "name": "hotels",
    "fields": [
        { "name": "id", "type": "Edm.String", "key": true, "searchable": false },
        { "name": "title", "type": "Edm.String", "searchable": true },
        { "name": "description", "type": "Edm.String", "searchable": true }
    ]
}
```

さらに、このインデックスには、以下の4つの文書が追加されているとします。

```
JSON

{
    "value": [
        {
            "id": "1",
            "title": "Hotel Atman",
            "description": "Spacious rooms, ocean view, walking distance to
the beach."
        },
        {
            "id": "2",
            "title": "Beach Resort",
            "description": "Located on the north shore of the island of
Kaua'i. Ocean view."
        },
        {
            "id": "3",
            "title": "Playa Hotel",
            "description": "Comfortable, air-conditioned rooms with ocean
view."
        }
    ]
}
```

```
    },
    {
        "id": "4",
        "title": "Ocean Retreat",
        "description": "Quiet and secluded"
    }
]
```

語句のインデックス作成方法

検索を理解するには、インデックス作成の基本をいくつかを把握しておくと役立ちます。保存の単位は転置インデックスで、検索可能なフィールドごとに 1 つ存在します。転置インデックス内には、全文書から抽出されたすべての語句を並べ替えたリストが存在します。それぞれの語句は、それが出現する一連の文書に対応付けられています（以下の例を参照）。

転置インデックスに含める語句を得るために、検索エンジンは、クエリの加工時と同様の字句解析を文書の内容に対して実行します。

1. "テキスト入力" はアナライザーに渡され、小文字への変換や句読点の削除など、アナライザーの構成に応じた処理が行われます。
2. "トークン" は字句解析の出力です。
3. "用語" はインデックスに追加されます。

検索語の体裁をインデックスに登録されている語句の体裁と合わせるために、通常は検索操作とインデックス作成操作に同じアナライザーが使用されますが、必ずしも同じである必要はありません。

① 注意

Azure Cognitive Search では、追加の `indexAnalyzer` および `searchAnalyzer` フィールド パラメーターを使用して、インデックス作成と検索に別々のアナライザーを指定することができます。指定しなかった場合、`analyzer` プロパティで設定されたアナライザーが、インデックス作成と検索の両方に使用されます。

文書サンプルの転置インデックス

もう一度先ほどの例を見てみましょう。`title` フィールドの転置インデックスは、次のようにになります。

任期	ドキュメント リスト
atman	1

任期	ドキュメント リスト
beach	2
hotel	1、3
ocean	4
playa	3
resort	3
retreat	4

title フィールドの場合、*hotel*だけが 2 つの文書 (1 と 3) に出現します。

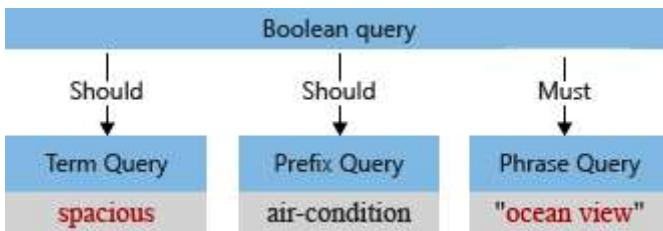
description フィールドのインデックスは次のようにになっています。

任期	ドキュメント リスト
air	3
and	4
beach	1
conditioned	3
comfortable	3
distance	1
island	2
kaua'i	2
located	2
north	2
ocean	1, 2, 3
of	2
on	2
通知の停止	4
会議室	1、3
secluded	4

任期	ドキュメント リスト
shore	2
spacious	1
the	1, 2
to	1
表示	1, 2, 3
walking	1
with	3

インデックスが作成された語句に対する検索語の照合

上記の転置インデックスを踏まえてサンプル クエリに戻り、一致する文書がどのように検索されるかを見ていきましょう。最終的なクエリツリーは、次のようになっていましたことを思い出してください。



クエリの実行中、検索可能なフィールドに対して個々の検索が別々に実行されます。

- 単語検索 "spacious" と一致する文書は 1 件です (Hotel Atman)。
- プレフィックス検索 "air-condition*" と一致する文書はありません。

これは、場合によっては開発者の混乱を招く動作です。 "air-conditioned" という語句はこの文書内に存在しますが、既定のアナライザーによって 2 つの単語に分割されています。部分的な語句を含むプレフィックス クエリは、解析されないことに注意してください。そのため、転置インデックスで、"air-condition" というプレフィックスを含む語句を検索しても見つかりません。

- フレーズ検索 "ocean view" では、"ocean" と "view" という 2 つの語句が検索され、元の文書内で両者が近いかどうかがチェックされます。文書 1、2、3 の description フィールドに、この検索との一致が見つかります。文書 4 の title には ocean という語が存在しますが、一致とは見なされないことに注意してください。検索対象は "ocean view" というフレーズであって、個々の単語ではありません。

① 注意

特定のフィールドを `searchFields` パラメーター (検索要求の例を参照) で指定しない限り、検索クエリは、Azure Cognitive Search インデックス内の検索可能なすべてのフィールドに対して個別に実行されます。選択したフィールドのいずれかで一致する文書が返されます。

全体として、この例のクエリの場合、一致する文書は 1、2、3 です。

第 4 段階: スコア付け

検索結果セット内のすべての文書には、関連度スコアが割り当てられます。関連度スコアの機能は、ユーザーからの問い合わせ (検索クエリ) に対して最適解となる文書に対し、相対的に高いランクを与えることです。このスコアは、一致した語句の統計学的な特性に基づいて計算されます。スコア付けの式の核となるのは [TF/IDF \(Term Frequency-Inverse Document Frequency\)](#) です。TF/IDF は、出現頻度の低い語句と高い語句を含んだ検索において、出現頻度の低い語句を含んだ結果に、より高いランクを与えます。たとえば、Wikipedia の記事をすべて含んだ架空のインデックスでは、*the president* というクエリに一致した文書のうち、*president* で一致した文書の方が *the* で一致した文書よりも関連性が高いと見なされます。

スコア付けの例

冒頭のサンプル クエリで見つかった 3 つの文書を思い出してください。

```
search=Spacious, air-condition* +"Ocean view"
```

JSON

```
{
  "value": [
    {
      "@search.score": 0.25610128,
      "id": "1",
      "title": "Hotel Atman",
      "description": "Spacious rooms, ocean view, walking distance to the beach."
    },
    {
      "@search.score": 0.08951007,
      "id": "3",
      "title": "Budget Hotel",
      "description": "Affordable accommodation near the beach, free Wi-Fi, breakfast included." }
  ]
}
```

```
        "title": "Playa Hotel",
        "description": "Comfortable, air-conditioned rooms with ocean view."
    },
    {
        "@search.score": 0.05967338,
        "id": "2",
        "title": "Ocean Resort",
        "description": "Located on a cliff on the north shore of the island of Kauai. Ocean view."
    }
]
```

文書 1 は、クエリに対して最も高い関連度で一致しています。なぜなら、*spacious* という単語と *ocean view* という必須のフレーズの両方が *description* フィールドに出現するためです。その他の 2 つの文書は、*ocean view* しか一致していません。しかし文書 2 と文書 3 は、クエリに対して同じように一致しているにもかかわらず、関連度スコアが異なるのはなぜでしょうか。これは、スコア付けの式の構成要素が TF/IDF だけではないためです。この場合、文書 3 の方が、*description* が短いために、少しだけ高いスコアが割り当てられています。フィールドの長さやその他の要因が関連度スコアに与える影響については、[Lucene の実際に役立つスコア付けの式](#)に関するページを参照してください。

一部の検索の種類 (ワイルドカード、プレフィックス、正規表現) は、文書全体のスコアに対して常に一定のスコアをもたらします。これによって、ランクには影響を与えずに、クエリ拡張によって見つかった一致を結果に反映することができます。

このことが重要である理由を例で説明します。ワイルドカード検索 (プレフィックス検索を含む) は、本質的に解釈があいまいです。入力されるのは文字列の一部であり、まったく異なる、膨大な数の語句と一致する可能性があるからです ("tour*" と入力した場合、"tours"、"tourettes"、"tourmaline" などとの一致が検出されます)。こうした結果の性質上、用語の相対的な重みを適切に推測することができません。そのため、ワイルドカード検索、プレフィックス検索、正規表現検索では、結果のスコア付けを行う際に、語句の出現頻度が無視されます。部分的な語句と完全な語句とを含んだ複数の構成要素から成る検索要求では、予期しない一致が偏重されないよう、部分的な入力から得られた結果については、定数スコアを割り当てたうえで反映されます。

関連性のチューニング

Azure Cognitive Search の関連度スコアは、次の 2 つおりの方法でチューニングできます。

1. **スコアリングプロファイル**: ランク付けされた結果リストにおいて、一連のルールに基づく重みを文書に与えます。このページの例では、*title* フィールドに一致

の見つかった文書の方が、`description` フィールドに一致の見つかった文書よりも関連性が高いと見なすことができます。加えて、仮にホテルごとの料金フィールドをインデックスに含めた場合、料金の低い方の文書に高い重みを与えることもできます。[スコアリング プロファイルを検索インデックスに追加する方法](#)の詳細について学習します。

2. **項目ブースト** (Full Lucene クエリ構文のみで使用可能): クエリツリーの任意の構成要素に適用できるブースト演算子 `^` が用意されています。このページの例では、`air-condition^*` というプレフィックスで検索する代わりに、`air-condition^2||air-condition^*` のように単語検索にブーストを適用することもできます。そうすれば、`air-condition` で完全一致する語句と、プレフィックスで一致する語句との両方を検索したうえで、完全一致の語句で一致した文書の方に、より高いランクを与えることができます。[クエリでの語句ブースト](#)の詳細について学習します。

分散されたインデックスにおけるスコア付け

Azure Cognitive Search のすべてのインデックスは自動的に複数のシャードに分割されます。これにより、Microsoft は、サービスのスケールアップまたはスケールダウンの間に、複数のノードにインデックスをすばやく分散することができます。検索要求は送信されると、各シャードに対して別々に送られます。その後、各シャードから得られた結果がマージされ、スコア順に並べ替えられます (他に並べ替えが定義されていない場合)。スコアリング関数は、シャード内のすべてのドキュメントで逆ドキュメント頻度に対して検索語頻度を重み付けするものであり、すべてのシャードで重み付けするものではないことを知ることが重要です。

つまり、まったく同じ文書でも、それらが異なるシャードに存在していれば、関連度スコアが異なる "可能性がある" ということです。さいわい、インデックス内の文書数が増えるにつれて語句の分布が平準化され、そのような差異は総じて消失します。文書がどのシャードに配置されるかを推測することは不可能です。しかし文書は、そのキーが変化しなければ、常に同じシャードに割り当てられます。

一般に、順序の不变性が重要な場合、文書を並べ替えるための特性として、文書のスコアはあまり適していません。たとえば、まったく同じスコアの文書が 2 つあるとして、同一クエリを繰り返し実行したときに、どちらの文書が上位にランク付けされるかを毎回確実に予測することができません。文書スコアは、検索結果群における特定の文書の相対的な関連度の高さを測る、おおよその目安としてのみ使用してください。

まとめ

商用検索エンジンが多くの人々の支持を得たことで、プライベートデータに対するフルテキスト検索への期待が高まってきました。ほぼすべての検索について言えることですが、語句の綴りが間違っていたり不完全であったりしても自分の意図がきちんと反映されるほどの利便性を、私たちは検索エンジンに求めるようになっています。指定してもいない同義語やほぼ同等の語句に基づいた検索結果が得られることさえ、当然のように感じてしまうほどです。

技術的な観点でいえばフルテキスト検索はきわめて複雑で、洗練された言語分析と検索語の加工(関連性の高い結果を得るために検索語を抽出、展開、変換する処理)への秩序立ったアプローチとが要求されます。こうした本質的な複雑さもあって、検索結果はさまざまな要因によって左右されます。フルテキスト検索のメカニズムをしっかりと理解しておけば、予期しない結果に対処しようとする際に、はっきりとその効果を実感できるでしょう。

この記事では、Azure Cognitive Search の観点からフルテキスト検索について詳しく見てきました。ここで身に付けた知識が、検索時に遭遇しやすい問題の原因や解決策を判断するうえでの一助となればさいわいです。

次のステップ

- サンプルインデックスを構築し、さまざまな検索を試してその結果を確認します。詳しい手順については、[ポータルでのインデックスの構築と照会](#)に関するページを参照してください。
- [Search Documents](#) の例に関するセクションや[単純なクエリ構文](#)で紹介されている他のクエリ構文をポータルの検索エクスプローラーで試します。
- 検索アプリケーションにおけるランク付けをチューニングする方法については、[スコアリング プロファイル](#)に関するページを参照してください。
- 言語に固有の字句解析器を適用する方法について書かれた記事を参照します。
- 特定のフィールドに対して最小限の処理または特殊な処理を適用するための[カスタム アナライザー](#)を構成します。

関連項目

[Search Documents REST API](#)

[単純なクエリ構文](#)

[Full Lucene クエリ構文](#)

検索結果の処理方法

Azure AI Search のベクター

[アーティクル] • 2024/04/09

ベクトル検索とは、コンテンツの数値表現に対するインデックス付けとクエリの実行をサポートする情報取得のアプローチです。コンテンツはプレーンテキストではなく数値であるため、照合はクエリ ベクトルに最も類似したベクトルに基づいて行われます。これにより、次のシナリオでの照合が可能になります。

- 意味的または概念的な類似性 ("dog" と "canine" は概念的には似ているが言語的には異なる)
- 多言語コンテンツ (英語では "dog"、ドイツ語では "hund")
- 複数のコンテンツ タイプ (プレーンテキストの "dog" と画像ファイル内の犬の写真)

この記事では、Azure AI 検索でのベクトルの概要について説明します。また、他の Azure サービスとの統合についても説明し、ベクトル検索の開発に関する用語と概念についても説明します。

最初にこの記事を読んで基礎知識を得ることをお勧めしますが、それはかまわないのですぐに使いたいという方は、これらの手順を実行してください。

- ✓ インデックス用の埋め込みを提供するか、インデクサー パイプラインで埋め込み (プレビュー) を生成する
- ✓ ベクトルインデックスを作成する
- ✓ ベクトルクエリの実行

ベクトルクイックスタートまたは GitHub のコード サンプル[を](#)を使用して開始することもできます。

ベクトル検索をサポートできるシナリオ

ベクトル検索には次のようなシナリオがあります。

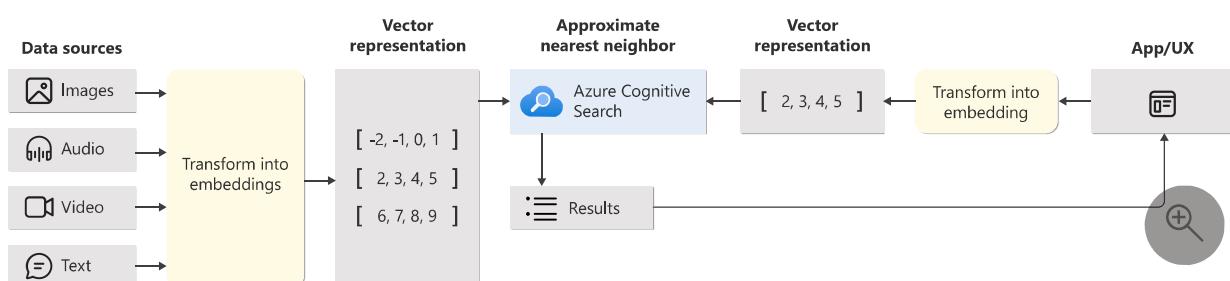
- 類似性検索。OpenAI 埋め込みなどの埋め込みモデルや SBERT などのオープンソース モデルを使用してテキストをエンコードし、やはりベクトルとしてエンコードされたクエリを使用してドキュメントを取得します。
- さまざまなコンテンツ タイプ (マルチモーダル) で検索します。画像とテキストをマルチモーダル埋め込み (たとえば、Azure OpenAI の OpenAI CLIP[や GPT-4 Turbo with Vision](#) を使用) を使用してエンコードし、両方のコンテンツ タイプのベクトルで構成される埋め込みスペースをクエリします。

- ハイブリッド検索。** Azure AI 検索のハイブリッド検索では、同じ要求でベクトルおよびキーワード クエリの実行を参照します。ベクトルサポートはフィールドレベルで実装されており、ベクトルフィールドと検索可能なテキストフィールドの両方を含むインデックスがあります。クエリは並列で実行され、結果は1つの応答にマージされます。必要に応じて、[セマンティックランク付け](#)を追加して、Bing を動作させているのと同じ言語モデルを使用して、L2 の再ランク付けによって精度をさらに高めます。
- 多言語検索。** 複数の言語でトレーニングされたモデルとチャット モデルを埋め込んで、ユーザーの母国語での検索エクスペリエンスを提供できます。翻訳をより詳細に制御する必要がある場合は、ハイブリッド検索シナリオで Azure AI Search が非ベクトルコンテンツに対して提供する[多言語機能](#)を追加できます。
- フィルター選択されたベクトル検索。** クエリ要求にはベクトルクエリと[フィルタ一式](#)を含めることができます。フィルターはテキストフィールドと数値フィールドに適用され、メタデータフィルターに役立ち、フィルター条件に基づいて検索結果を含めたり除外したりするのに役立ちます。ベクトルフィールド自体はフィルター処理できませんが、フィルター可能なテキストフィールドまたは数値フィールドを設定できます。検索エンジンは、ベクトルクエリの実行前または実行後にフィルターを処理できます。
- ベクトルデータベース。** Azure AI Search には、クエリを実行するデータが格納されます。長期メモリやナレッジベース、あるいは取得拡張生成 (RAG) アーキテクチャやベクトルを使用するあらゆるアプリケーションの基礎データが必要な場合は、[純粋なベクトルストア](#)として使用します。

Azure AI Searchでのベクトル検索のしくみ

ベクトルのサポートには、検索インデックスからのベクトル埋め込みのインデックス作成、格納、クエリが含まれます。

次の図は、ベクトル検索のインデックス作成とクエリのワークフローを示しています。



インデックス作成側では、Azure AI Search はベクトル埋め込みを受け取り、[ニアレスターネイバーアルゴリズム](#)を使用して、同様のベクトルをインデックス内の近い場所に

配置します。 内部では、各ベクトルフィールドのベクトルインデックスが作成されます。

ソースコンテンツから埋め込みを Azure AI Search に取り込む方法は、アプローチとプレビュー機能が使用できるかどうかによって異なります。 OpenAI、Azure OpenAI、および任意の数のプロバイダーのモデルを使用して、テキスト、画像、モデルでサポートされているその他のコンテンツ タイプなど、さまざまなソースコンテンツに対して埋め込みをベクトル化または生成できます。その後、事前にベクトル化したコンテンツをベクトルストアのベクトルフィールドにプッシュすることができます。これが一般公開されているアプローチです。 プレビュー機能を使用できる場合、Azure AI Search はインデクサー パイプラインで統合されたデータ チャンクとベクトル化を提供します。 リソース (エンドポイントと Azure OpenAI への接続情報) は引き続き提供されますが、Azure AI Search はすべての呼び出しを行い、移行を処理します。

クエリ側では、クライアントアプリケーションで、通常はプロンプトワークフローを使用して、ユーザーからクエリ入力を収集します。その後、入力をベクトルに変換するエンコード手順を追加し、Azure AI Search 上のインデックスにベクトル クエリを送信して類似性検索を行うことができます。 インデックス作成と同様に、統合ベクトル化 (プレビュー) をデプロイして、質問をベクトルに変換できます。 どちらの方法でも、Azure AI Search では、要求された `k` ニアレストネイバー (kNN) を含むドキュメントが結果に返されます。

Azure AI Search は、ベクトル検索とキーワード検索を並行して実行するハイブリッドシナリオをサポートしており、統合された結果セットを返しますが、これはしばしば、ベクトル検索やキーワード検索のみよりも優れた結果を提供します。ハイブリッドの場合、ベクトルコンテンツと非ベクトルコンテンツは、並列して実行されるクエリに対して、同じインデックスに取り込まれます。

可用性と料金

ベクトル検索は、すべてのリージョンのすべての Azure AI Search レベルの一部として追加料金なしで利用できます。

2024 年 4 月 3 日以降に作成された新しいサービスでは、ベクトルインデックスのためにより高いクォータをサポートしています。

ベクトル検索は次で利用できます。

- データのインポートとベクトル化ウィザードを使用した Azure portal
- Azure REST API、バージョン 2023-11-01
- .NET ↗、Python ↗、JavaScript ↗ 用の Azure SDK
- Azure AI Studio や Azure OpenAI Studio などその他の Azure オファリング。

① 注意

2019年1月1日より前に作成された一部の古い検索サービスは、ベクトルワーカークロードをサポートしないインフラストラクチャにデプロイされています。ベクトルフィールドをスキーマに追加しようとしてエラーが表示された場合、それはサービスが古いためです。このような場合は、ベクトル機能を試すために新しい検索サービスを作成する必要があります。

Azure の統合と関連サービス

Azure AI Search は、Azure AI プラットフォーム全体で深く統合されています。次の表に、ベクトルワーカークロードで役立ついくつかの要素を示します。

[+] テーブルを展開する

Product	統合
Azure OpenAI Studio	データプレイグラウンドとのチャットで、[独自のデータを追加する] は、データの基盤と会話型検索のために Azure AI Search を使用します。これは、データとチャットするための最も簡単かつ高速なアプローチです。
Azure OpenAI	Azure OpenAI には埋め込みモデルとチャットモデルが用意されています。デモとサンプルでは、 text-embedding-ada-002 を対象とします。テキスト用の埋め込みを生成するには、Azure OpenAI をお勧めします。
Azure AI Services	Image Retrieval Vectorize Image API (プレビュー) では、画像コンテンツのベクトル化がサポートされます。画像用の埋め込みを生成するには、この API をお勧めします。
Azure データ プラットフォーム: Azure Blob Storage、Azure Cosmos DB	インデクサーを使用してデータインジェストを自動化し、 統合ベクトル化 (プレビュー) を使用して埋め込みを生成できます。Azure AI Search では、Azure Blob インデクサーと Azure Cosmos DB for NoSQL インデクサー の 2 つのデータソースからベクトルデータのインデックスを自動的に作成できます。詳細については、「 検索インデックスにベクトルフィールドを追加する 」を参照してください。

これは、[LangChain](#) などのオープンソースフレームワークでも一般的に使用されています。

ベクトル検索の概念

ベクトルを初めて使用する場合、このセクションではいくつかの主要な概念について説明します。

ベクトル検索について

ベクトル検索は、ドキュメントとクエリがplain textではなくベクトルとして表現される場合の情報取得の方法です。ベクトル検索では、機械学習モデルがソース入力(テキスト、画像、その他のコンテンツ)のベクトル表現を生成します。コンテンツの数学表現を使用することによって、検索シナリオの共通基盤が提供されます。すべてがベクトルであれば、関連する元のコンテンツがクエリとは異なるメディアや言語であっても、クエリはベクトル空間で一致するものを見つけることができます。

ベクトル検索を使用する理由

検索可能なコンテンツがベクトルとして表されると、クエリは類似するコンテンツ内の近い一致を見つけることができます。ベクトル生成に使用される埋め込みモデルは、どの単語と概念が類似しているかを認識し、結果のベクトルを埋め込み空間内で近くに配置します。たとえば、"クラウド"と"霧"に関するベクトル化されたソースドキュメントは、意味的に類似しているため、構文上的一致ではない場合も"霧"に関するクエリで表示される可能性が高くなります。

埋め込みベクトル化

"埋め込み"は、テキストのセマンティックな意味や画像などの他のコンテンツの表現を読み取る機械学習モデルによって作成された、コンテンツまたはクエリの特定の種類のベクトル表現です。自然言語機械学習モデルは、単語間のパターンや関係を識別するために、大量のデータでトレーニングされます。トレーニング中に、"エンコーダー"と呼ばれる中間ステップで、入力を実数のベクトルとして表現する方法を学習します。トレーニングが完了すると、中間ベクトル表現がモデルの出力になるように、これらの言語モデルを変更できます。結果として得られる埋め込みは高次元ベクトルであり、[埋め込みの概要 \(Azure OpenAI\)](#)に関する記事で説明されているように、同じような意味を持つ単語がベクトル空間で互いに近くになります。

関連する情報の取得におけるベクトル検索の有効性は、ドキュメントとクエリの意味を結果のベクトルに抽出する埋め込みモデルの有効性に依存します。最適なモデルは、それらが代表するデータの種類によって適切にトレーニングされています。Azure OpenAI text-embedding-ada-002などの既存のモデルを評価したり、問題領域で直接トレーニングされた独自のモデルを使用したり、汎用モデルを微調整したりできます。Azure AI Searchでは、選ぶモデルに制約が課されないため、データに最適なものを選んでください。

ベクトル検索に対して効果的な埋め込みを作成するには、入力サイズの制限を考慮することが重要です。埋め込みを生成する前に、[データをチャンクするためのガイドライン](#)

ンに従うことをお勧めします。このベストプラクティスのおかげで、埋め込みによって関連情報が正確に読み取られ、より効率的なベクトル検索が可能になります。

埋め込み空間とは

"埋め込み空間"は、ベクトルクエリのコーパスです。検索インデックス内では、埋め込み空間は、同じ埋め込みモデルからの埋め込み値が設定されているすべてのベクトルフィールドです。機械学習モデルでは、個々の単語、語句、またはドキュメント(自然言語処理の場合)、画像、またはその他の形式のデータを、高次元空間の座標を表す実数のベクトルで構成される表現にマッピングすることで、埋め込み空間を作成します。この埋め込みスペースでは、類似項目は近くに配置され、異なる項目は離れた場所に配置されます。

たとえば、さまざまな種類の犬について説明するドキュメントは、埋め込み空間で互いに近くに集められます。猫に関するドキュメントは互いに近くに集まりますが、犬のクラスターから遠く離れており、それでも動物としては近くになります。クラウドコンピューティングなどの異なる概念は、はるかに遠く離れています。実際には、これらの埋め込み空間は抽象的で、人間が解釈できる明確に定義された意味はありませんが、中核となる概念は同じです。

ニアレストネイバー検索

ベクトル検索では、検索エンジンは埋め込みスペース内のベクトルをスキャンして、クエリベクトルに最も近いベクトルを識別します。この手法は "ニアレストネイバー検索"と呼ばれます。[ニアレストネイバー](#)は、項目間の類似性を定量化するのに役立ちます。ベクトルの類似性が高い場合は、元のデータも同様であることを示します。高速なニアレストネイバー検索を容易にするために、検索エンジンでは最適化を実行するか、データ構造およびデータパーティション分割を使用して検索領域を削減します。各ベクトル検索アルゴリズムは、最小待機時間、最大スループット、再現率、メモリを最適化する際に、ニアレストネイバーの問題をさまざまな方法で解決します。類似性を計算するために、類似性メトリックでは距離を計算するためのメカニズムを提供します。

Azure AI Search では現在、次のアルゴリズムがサポートされています。

- Hierarchical Navigable Small World (HNSW): HNSW は、データ分散が不明であるか、頻繁に変更される可能性がある、高いリコールと待機時間の短い用途に最適化された主要な ANN アルゴリズムです。高次元のデータ ポイントを階層グラフ構造に整理することで、高速でスケーラブルな類似性検索を可能にしながら、検索精度と計算コストのトレードオフを調整できます。このアルゴリズムでは、高速ランダムアクセスのためにすべてのデータ ポイントがメモリ内に存在する必要

があるため、このアルゴリズムではベクトルインデックス サイズのクオータが使用されます。

- 完全な K ニアレストネイバー (KNN): クエリベクトルとすべてのデータ ポイントの間の距離を計算します。計算負荷が高いので、小規模なデータセットに最適です。このアルゴリズムではデータ ポイントの高速ランダム アクセスが不要なため、このアルゴリズムではベクトルインデックス サイズのクオータが使用されません。ただし、このアルゴリズムではニアレストネイバーのグローバルセットが提供されます。

インデックス定義内で 1 つ以上のアルゴリズムを指定し、ベクトルフィールドごとに使用するアルゴリズムを指定できます。

- インデックスとフィールドにアルゴリズムを指定するベクトルストアを作成します。
- 完全な KNN の場合、[2023-11-01](#)、[2023-10-01-Preview](#)、または REST API バージョンを対象とする Azure SDK ベータ ライブラリを使用します。

インデックスの作成時にインデックスの初期化に使用されるアルゴリズム パラメーターは不变であり、インデックスの作成後に変更することはできません。ただし、クエリ時間の特性 (`efSearch`) に影響を与えるパラメーターは変更することができます。

さらに、HNSW アルゴリズムを指定するフィールドは、クエリ要求パラメーター `"exhaustive": true` を使用した完全な KNN 検索もサポートします。ただし、その逆は当てはまりません。`exhaustiveKnn` に対してフィールドがインデックス付けされている場合、効率的な検索'を可能にする追加のデータ構造が存在しないため、クエリで HNSW を使用することはできません。

近似ニアレストネイバー

近似ニアレストネイバー検索 (ANN) は、ベクトル空間で一致を検索するためのアルゴリズムの種類です。この種類のアルゴリズムでは、検索空間を大幅に削減してクエリ処理を高速化するため、さまざまなデータ構造またはデータ パーティション分割方法が採用されます。

ANN アルゴリズムでは、精度がいくらか犠牲になりますが、近似ニアレストネイバーをスケーラブルかつ迅速に取得できるため、最新の情報取得の用途で効率と精度のバランスを取るのに最適です。アルゴリズムのパラメーターを調整して、検索用途のリコール、待機時間、メモリ、ディスク フットプリントの要件を微調整できます。

Azure AI Search では、ANN アルゴリズムに HNSW が使用されます。

次のステップ

- クイックスタートを試す
- ベクトルインデックス作成の詳細を確認する
- ベクトルクエリの詳細を確認する
- Azure Cognitive Search と LangChain: 拡張ベクトル検索機能のシームレスな統合 ↗

Azure AI Search でベクトルやフルテキストを使用したハイブリッド検索

[アーティクル] • 2024/01/31

ハイブリッド検索は、フルテキストとベクトル クエリを組み合わせたものです。検索可能なプレーンテキスト コンテンツと生成された埋め込み両方を含む検索インデックスに対してクエリを実行します。 クエリの目的として、ハイブリッド検索とは次のようなものです。

- `search` および `vectors` クエリ パラメータ両方を含む単一のクエリ要求です。
- 並行して実行されます。
- クエリ応答のマージされた結果の場合、[Reciprocal Rank Fusion \(RRF\)](#) を使用してスコアリングされます。

この記事では、ハイブリッド検索のコンセプト、利点、制限について説明します。 この[埋め込みビデオ](#)で、ハイブリッド検索が高品質のチャットスタイルアプリやコパイロットアプリにどのように貢献するかの説明と短いデモをご覧ください。

ハイブリッド検索が機能するしくみ

Azure AI Search では、埋め込みを含むベクター フィールドをテキスト フィールドと数値フィールドと共に使用できるため、並列で実行されるハイブリッド クエリを作成できます。ハイブリッド クエリでは、単一の検索要求で、フィルター処理やファセット処理、並べ替え、スコアリング プロファイル、[セマンティック ランク付け](#)といった既存の機能を利用できます。

ハイブリッド検索では、BM25 や HNSW などの異なるランク付け機能を利用し、フルテキストおよびベクトル クエリの両方の結果を組み合わせます。[逆ランク 融合 \(RRF\)](#) アルゴリズムによって結果がマージされます。 クエリ応答は、RRF を使用して各クエリから最も関連性の高い一致を選択する結果セットを 1 つだけ提供します。

ハイブリッド クエリの構造

ハイブリッド検索は、プレーンテキストと数値、地理空間検索の地理座標、テキストのチャンクの数学的表現のためのベクトルなど、さまざまな[データ型](#)のフィールドを含む検索インデックスを持つことを前提とします。ベクトル クエリにより、オートコンプリートや検索候補などのクライアント側のインタラクションを除き、Azure AI Search のほぼすべてのクエリ機能を使用できます。

典型的なハイブリッド クエリは、次のようにになります(簡潔にするためベクトルを省略しています)。

```
HTTP

POST https://{{searchServiceName}}.search.windows.net/indexes/hotels-vector-quickstart/docs/search?api-version=2023-11-01
    content-type: application/JSON
{
    "count": true,
    "search": "historic hotel walk to restaurants and shopping",
    "select": "HotelId, HotelName, Category, Description, Address/City, Address/StateProvince",
    "filter": "geo.distance(Location, geography'POINT(-77.03241 38.90166)') le 300",
    "facets": [ "Address/StateProvince"],
    "vectors": [
        {
            "value": [ <array of embeddings> ]
            "k": 7,
            "fields": "DescriptionVector"
        },
        {
            "value": [ <array of embeddings> ]
            "k": 7,
            "fields": "Description_frVector"
        }
    ],
    "queryType": "semantic",
    "queryLanguage": "en-us",
    "semanticConfiguration": "my-semantic-config"
}
```

重要なポイントは次のとおりです。

- `search` はフルテキスト検索クエリを指定します。
- `vectors` はベクトルクエリです。複数設定して複数のベクトルフィールドを対象とすることができます。埋め込みスペースに多言語コンテンツが含まれる場合、言語アナライザーや翻訳を介さずに、ベクトルクエリで一致を検出できます。
- `select` は結果で返すフィールドを指定します。人間が判読できるテキストフィールドを指定することもできます。
- `filters` は地理空間の検索、またはその他の包含条件や除外条件(駐車場を含めるかどうかなど)を指定できます。この例における地理空間のクエリでは、ワシントン D.C. から半径 300 キロ以内にあるホテルを検出します。
- `facets` はハイブリッド クエリで返された結果のファセットバケットの計算に使用できます。

- queryType=semantic はセマンティック ランク付けを呼び出します。機械読解を適用し、より関連性の高い検索結果を表示させます。

フィルターとファセットは、フルテキスト検索に使用した逆インデックスおよびベクトル検索に使用したベクトルインデックスとは異なるインデックス内のデータ構造を対象とします。そのため、フィルターとファセット処理が実行されると、検索エンジンの応答では、ハイブリッド検索結果に操作上の結果が適用されます。

このクエリには orderby がないことがわかります。明示的な並べ替え順序は、関連性でランク付けされた結果をオーバーライドするため、類似性と BM25 の関連性が必要な場合は、クエリで並べ替えを省略します。

上記のクエリに対する応答は、この例のようになります。

HTTP

```
{
  "@odata.count": 3,
  "@search.facets": {
    "Address/StateProvince": [
      {
        "count": 1,
        "value": "NY"
      },
      {
        "count": 1,
        "value": "VA"
      }
    ]
  },
  "value": [
    {
      "@search.score": 0.03333333507180214,
      "@search.rerankerScore": 2.5229012966156006,
      "HotelId": "49",
      "HotelName": "Old Carrabelle Hotel",
      "Description": "Spacious rooms, glamorous suites and residences, rooftop pool, walking access to shopping, dining, entertainment and the city center.",
      "Category": "Luxury",
      "Address": {
        "City": "Arlington",
        "StateProvince": "VA"
      }
    },
    {
      "@search.score": 0.032522473484277725,
      "@search.rerankerScore": 2.111117362976074,
      "HotelId": "48",
      "HotelName": "Nordick's Motel",
      "Description": "Only 90 miles (about 2 hours) from the nation's"
    }
  ]
}
```

```
capital and nearby most everything the historic valley has to offer.  
Hiking? Wine Tasting? Exploring the caverns? It's all nearby and we have  
specially priced packages to help make our B&B your home base for fun while  
visiting the valley.",  
        "Category": "Boutique",  
        "Address": {  
            "City": "Washington D.C.",  
            "StateProvince": null  
        }  
    }  
}  
]
```

ハイブリッド検索を選択する理由

ハイブリッド検索は、ベクトル検索とキーワード検索の長所を組み合わせたものです。ベクトル検索のメリットは、逆インデックスにキーワードの一致がない場合でも、検索クエリと概念的に似た情報が検索されることです。キーワード検索やフルテキスト検索のメリットは、精度の高さと、最初の結果の品質を向上させるセマンティックランク付けを適用できる機能です。製品コードや、極めて特殊な専門用語、日付、人の名前に対するクエリなど、一部のシナリオでは、完全一致を識別できるため、キーワード検索を使用すると、より良いパフォーマンスを発揮します。

実際のデータセットとベンチマークデータセットに対するベンチマークテストでは、セマンティックランク付けを使用したハイブリッド検索の場合、検索の関連性に大きな利点があることが示されています。

次のビデオでは、有用な AI 応答を生成するための最適なグラウンディングデータが、ハイブリッド検索でどのように提供されるかについて説明します。

<https://www.youtube-nocookie.com/embed/Xwx1DJ0OqCk>

関連項目

[ハイブリッド検索とランク付けでベクトル検索の性能を上回る \(技術ブログ\)](#)

Azure AI Search での取得拡張生成 (RAG)

[アーティクル] • 2024/04/22

取得拡張生成 (RAG) は、グラウンディング データを提供する情報取得システムを追加することで、ChatGPT などの大規模言語モデル (LLM) の機能を拡張するアーキテクチャです。情報取得システムを追加すると、応答を作成するときに LLM によって使用されるグラウンディング データを制御できます。エンタープライズ ソリューションの場合、RAG アーキテクチャは、ベクトル化されたドキュメントや画像、およびそのコンテンツの埋め込みモデルがある場合は、その他のデータ形式から取得された "エンタープライズ コンテンツ" に生成 AI を制限できることを意味します。

どの情報取得システムを使用するかによって LLM への入力が決定されるため、この決定は重要です。情報取得システムは、次の情報を提供する必要があります。

- 必要な頻度で、すべてのコンテンツに対して、大規模に読み込んで更新するインデックス作成戦略。
- クエリ機能と関連性のチューニング。システムは、関連する結果を、LLM 入力のトークンの長さの要件を満たすのに必要な短い形式で返す必要があります。
- データと操作の両方のセキュリティ、グローバル展開、信頼性。
- インデックス作成用の埋め込みモデル、および取得のためのチャット モデルまたは言語理解モデルとの統合。

Azure AI Search は、RAG アーキテクチャにおける [情報取得のための実証済みのソリューション](#) です。Azure クラウドのインフラストラクチャとセキュリティを備えたインデックス作成とクエリ機能を提供します。コードやその他のコンポーネントを使用して、財産的価値のあるコンテンツに対する生成 AI のすべての要素を含む包括的な RAG ソリューションを設計できます。

① 注意

Copilot と RAG の概念は初めてですか? 「[ベクトル検索と、生成 AI アプリの最新の取得](#)」をご覧ください。

Azure AI Search を使用した RAG へのアプローチ

Microsoft は、RAG ソリューションで Azure AI Search を使用するためのいくつかの組み込み実装を用意しています。

- Azure AI Studio。ベクトルインデックスと取得拡張を使用します。
- Azure OpenAI Studio。ベクトルの有無にかかわらず、検索インデックスを使用します。
- Azure Machine Learning。プロンプト フローでベクトルストアとして検索インデックスを使用します。

キュレーションされたアプローチを使用すると、簡単に作業を開始できますが、アーキテクチャをより詳細に制御するには、カスタム ソリューションが必要です。これらのテンプレートでは、以下でエンドツー エンドのソリューションが作成されます。

- [Python](#) ↗
- [.NET](#) ↗
- [JavaScript](#) ↗
- [Java](#) ↗

この記事の残りの部分では、Azure AI Search がカスタム RAG ソリューションにどのように適合するかについて説明します。

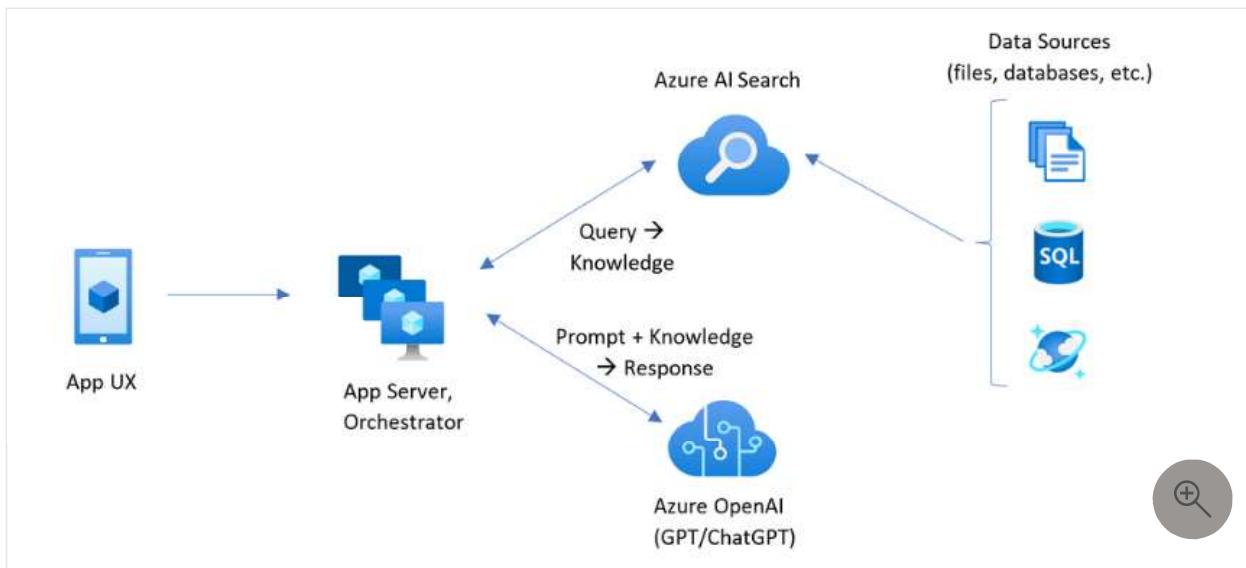
Azure AI Search のカスタム RAG パターン

パターンの大まかな概要は次のとおりです。

- ユーザーの質問または要求 (プロンプト) から始めます。
- Azure AI Search に送信して、関連情報を見つけます。
- 上位の検索結果を LLM に送信します。
- LLM の自然言語理解と推論機能を使用して、最初のプロンプトに対する応答を生成します。

Azure AI Search が LLM プロンプトに入力を提供しますが、モデルのトレーニングはしません。RAG アーキテクチャでは、追加のトレーニングはありません。LLM はパブリック データを使用して事前トレーニングされますが、取得コンポーネントからの情報によって拡張された応答を生成します。

Azure AI Search を含む RAG パターンには、次の図に示す要素があります。



- ユーザー エクスペリエンスのためのアプリ UX (Web アプリ)
- アプリ サーバーまたはオーケストレーター (統合と調整レイヤー)
- Azure AI Search (情報取得システム)
- Azure OpenAI (生成 AI 用の LLM)

Web アプリはユーザー エクスペリエンスを提供し、プレゼンテーション、コンテキスト、ユーザー操作を提供します。ユーザーからの質問またはプロンプトは、ここから始まります。入力は統合レイヤーを通過します。最初に情報を取得して検索結果を取得しますが、さらに LLM に移動してコンテキストと意図を設定します。

アプリ サーバーまたはオーケストレーターは、情報の取得と LLM の間のハンドオフを調整する統合コードです。1つのオプションは、[LangChain](#) を使用してワークフローを調整することです。LangChain は [Azure AI Search](#) と統合されるため、Azure AI Search を [取得コンポーネント](#) としてワークフローに簡単に含めることができます。セマンティック カーネルも別のオプションです。

情報取得システムは、検索可能なインデックス、クエリ ロジック、ペイロード (クエリ 応答) を提供します。検索インデックスには、ベクトルまたはベクトル以外のコンテンツを含めることができます。ほとんどのサンプルとデモにはベクトル フィールドが含まれていますが、必須ではありません。クエリは、キーワード (または用語) とベクトルクエリを処理できる Azure AI Search の既存の検索エンジンを使用して実行されます。インデックスは、定義したスキーマに基づいて事前に作成され、ファイル、データベース、またはストレージからソース化されたコンテンツと共に読み込まれます。

LLM は、元のプロンプトに加えて、Azure AI Search からの結果を受け取ります。LLM は結果を分析し、応答を作成します。LLM が ChatGPT の場合、ユーザーの対話は会話のやり取りである可能性があります。Davinci を使用している場合、プロンプトは完全に構成された回答である可能性があります。Azure ソリューションでは Azure OpenAI が使用される可能性が最も高いですが、この特定のサービスに対するハードな依存関係はありません。

Azure AI 検索では、プロンプト フローやチャットの保持のためのネイティブ LLM 統合は提供されないため、オーケストレーションと状態を処理するコードを記述する必要があります。完全なソリューションに必要なブループリントについては、デモ ソース ([Azure-Samples/azure-search-openai-demo](#)) を確認できます。また、LLM と統合する RAG ベースの Azure AI 検索ソリューションを作成するには、Azure AI Studio または Azure OpenAI Studio を使用することをお勧めします。

Azure AI Search の検索可能なコンテンツ

Azure AI Search では、検索可能なすべてのコンテンツは、検索サービスでホストされている検索インデックスに格納されます。検索インデックスは、応答時間がミリ秒レベルの高速なクエリを実現するために設計されているため、内部データ構造はその目標をサポートするために存在します。そのため、検索インデックスにはインデックス付きコンテンツが保存されます。コンテンツ ファイル全体 (PDF 全体や画像など) は保存されません。内部では、データ構造には [トークン化されたテキスト](#) の逆インデックス、埋め込み用のベクトルインデックス、逐語的一致が必要な場合 (フィルター、あいまい検索、正規表現クエリなど) の変更されていないテキストが含まれます。

RAG ソリューションのデータを設定するときは、Azure AI Search でインデックスを作成して読み込む機能を使用します。インデックスには、ソース コンテンツを複製または表すフィールドが含まれます。インデックス フィールドは単純な転送 (ソース ドキュメントのタイトルまたは説明が検索インデックスのタイトルまたは説明になる) であるか、画像の表現またはテキストの説明を生成するベクトル化やスキル処理などの外部プロセスの出力を含む場合があります。

検索するコンテンツの種類をご存知のことと思われる所以、各コンテンツ タイプに適用できるインデックス作成機能を検討します。

 [テーブルを開く](#)

コンテンツ タイプ	付けられたインデックス	機能
text	トークン、変更されていないテキスト	インデクサーは、Azure Storage や Cosmos DB などの他の Azure リソースからプレーンテキストをフルできます。インデックスに 任意の JSON コンテンツをプッシュ することもできます。処理中のテキストを変更するには、 アナライザー と ノーマライザー を使用して、インデックス作成中に字句処理を追加します。 同意語マップ は、クエリで使用される可能性のある用語がソース ドキュメントにない場合に便利です。
text	ベクトル ¹	テキストをチャンクして外部でベクトル化し、インデックスに ベクトルフィールド としてインデックスを付けることができます。

コンテンツ	付けられたインデックス	機能
image	トークン、変更されていないテキスト ²	OCR と画像解析の スキル では、テキスト認識やイメージ特性のために画像を処理できます。 画像情報は検索可能なテキストに変換され、インデックスに追加されます。 エンデックスにはインデクサーの要件があります。
image	ベクトル ¹	画像は、画像コンテンツを数学的に表現するために外部でベクトル化し、インデックスに ベクトルフィールドとしてインデックスを付ける ことができます。 OpenAI CLIP などのオープンソースモデルを使用して、同じ埋め込み空間内のテキストと画像をベクトル化できます。

¹[ベクトルサポート](#)の一般提供機能では、データ チャンクとベクトル化のために他のライブラリまたはモデルを呼び出す必要があります。しかし、[垂直統合 \(プレビュー\)](#)にはこれらの手順が埋め込まれています。両方のアプローチを示すコード サンプルについては、[azure-search-vector リポジトリ](#) を参照してください。

²[スキル](#)は AI エンリッチメントの組み込みサポートです。 OCR と画像分析の場合、インデックス作成パイプラインは Azure AI Vision API の内部呼び出しを行います。これらのスキルは、抽出された画像を処理のために Azure AI に渡し、Azure AI Search によってインデックス付けされたテキストとして出力を受け取ります。

ベクトルは、異なるコンテンツ (複数のファイル形式と言語) に最適な設備を提供します。これは、コンテンツが数学表現で汎用的に表現されるためです。また、ベクトルでは類似性検索もサポートされています。つまり、ベクトルクエリに最も似た座標で照合します。トークン化された用語で照合するキーワード検索 (または用語検索) と比較すると、類似性検索の方が微妙です。コンテンツまたはクエリにあいまいな点や解釈の要件がある場合は、より適切な選択肢です。

Azure AI Search でのコンテンツの取得

データが検索インデックスに格納されたら、Azure AI Search のクエリ機能を使用してコンテンツを取得します。

RAG 以外のパターンでは、クエリは検索クライアントからラウンド トリップを行います。クエリが送信され、検索エンジンで実行され、応答がクライアント アプリケーションに返されます。応答 (検索結果) は、インデックス内で見つかった逐語的なコンテンツのみで構成されます。

RAG パターンでは、検索エンジンと LLM の間でクエリと応答が調整されます。ユーザーの質問またはクエリは、検索エンジンと LLM の両方にプロンプトとして転送されま

す。検索結果は検索エンジンから戻り、LLM にリダイレクトされます。ユーザーに返される応答は生成 AI で、LLM からの合計または回答のどちらかです。

Azure AI Search には、新しい回答を構成するクエリの種類はありません (セマンティック検索やベクトル検索でさえも)。LLM だけが生成 AI を提供します。クエリの作成に使用される Azure AI Search の機能を次に示します。

テーブルを展開する

クエリ機能	目的	使用する理由
単純または完全な Lucene 構文	テキストと非ベクトル数値コンテンツに対するクエリ実行	フルテキスト検索は、類似一致ではなく、完全一致に最適です。フルテキスト検索クエリは、 BM25 アルゴリズム を使用してランク付けされ、スコアリングプロファイルによる関連性チューニングをサポートします。また、フィルターとファセットもサポートされています。
フィルターとファセット	テキストまたは数値(非ベクトル)フィールドにのみ適用されます。包含条件または除外条件に基づいて検索対象領域を減らします。	クエリに精度を追加します。
セマンティックランク付け	セマンティックモデルを使用して BM25 結果セットを再ランク付けします。LLM 入力として役立つ短い形式のキャプションと回答を生成します。	スコアリングプロファイルよりも簡単で、コンテンツによっては、関連性チューニングのためのより信頼性の高い手法です。
ベクトル検索	クエリ文字列が 1 つ以上のベクトルである類似性検索のベクトルフィールドに対するクエリ実行。	ベクトルは、あらゆる種類のコンテンツを任意の言語で表すことができます。
ハイブリッド検索	上記のクエリ手法の一部またはすべてを組み合わせます。ベクトルおよび非ベクトルクエリは並列で実行され、統合された結果セットで返されます。	ハイブリッド クエリを使用した場合、精度とリコールにおけるメリットが最も多くなります。

クエリ応答を構造化する

クエリの応答は LLM に入力を提供するため、検索結果の品質は成功に不可欠です。結果は表形式の行セットです。結果の構成や構造は次に依存します。

- 応答に含まれるインデックスの部分を決定するフィールド。
- インデックスにおける一致を示す行。

フィールドは、属性が "取得可能" である場合に検索結果に表示されます。インデックススキーマ内のフィールド定義には属性があり、これがフィールドが応答で使用されるかどうかを決定します。"取得可能" フィールドだけがフルテキストクエリまたはベクトルクエリ結果で返されます。既定では、すべての "取得可能" フィールドが返されますが、"選択" を使用してサブセットを指定できます。"取得可能" 以外に、フィールドに制限はありません。フィールドには、任意の長さまたは型を指定できます。長さについて、Azure AI Search にはフィールド長の上限はありませんが、[API 要求のサイズ](#)には制限があります。

行ではクエリとの一致が、関連性、類似性、またはその両方でランク付けされます。既定では、結果はフルテキスト検索の場合は上位 50 件、ベクトル検索の場合は K ニアレストネイバーに制限されます。既定値を変更して制限を (最大 1000 ドキュメントまで) 増減できます。top および skip ページングパラメーターを使用して、結果を一連のページングされた結果として取得することもできます。

関連性でランク付けする

複雑なプロセスや大量のデータを処理する場合や、ミリ秒レベルの応答が期待されている場合、各ステップで価値を高め、最終的な結果の品質を向上させることが重要です。情報取得側において、関連性のチューニングは、LLM に送信される結果の品質を向上させるアクティビティです。結果には、最も関連性の高い、または最も類似した一致するドキュメントだけを含める必要があります。

関連性は、キーワード (非ベクトル) 検索とハイブリッドクエリ (非ベクトルフィールドに対する) に適用されます。Azure AI Search では、類似性検索とベクトルクエリの関連性のチューニングはありません。[BM25 ランク付け](#)は、フルテキスト検索のランク付けアルゴリズムです。

関連性のチューニングは、BM25 ランク付けを強化する機能によってサポートされます。これらのアプローチには、次が含まれます。

- [スコアリングプロファイル](#)。これは、特定の検索フィールドまたはその他の条件で一致が見つかった場合に検索スコアを向上させます。
- [セマンティックランク付け](#)。これは、Bing のセマンティックモデルを使用して結果を並べ替え、元のクエリに合わせてセマンティックに適合するように BM25 結果セットを再ランク付けします。

比較テストおよびベンチマークテストでは、テキストフィールドとベクトルフィールドを含むハイブリッドクエリに、BM25 ランクの結果に対するセマンティックランク付けを補足することで、最も関連性の高い結果が生成されます。

RAG シナリオの Azure AI Search クエリのコード例

次のコードは、デモ サイトからの [retrievethenread.py](#) ファイルからコピーされます。ハイブリッド クエリの検索結果から LLM の `content` が生成されます。より簡単なクエリを記述できますが、この例では、セマンティック再ランク付けとスペルチェックを使用したベクトル検索とキーワード検索が含まれています。デモでは、このクエリを使用して初期コンテンツを取得します。

Python

```
# Use semantic ranker if requested and if retrieval mode is text or hybrid
(vectors + text)
if overrides.get("semantic_ranker") and has_text:
    r = await self.search_client.search(query_text,
                                         filter=filter,
                                         query_type=QueryType.SEMANTIC,
                                         query_language="en-us",
                                         query_speller="lexicon",
                                         semantic_configuration_name="default",
                                         top=top,
                                         query_caption="extractive|highlight-false")
if use_semantic_captions else None,
                                         vector=query_vector,
                                         top_k=50 if query_vector else None,
                                         vector_fields="embedding" if query_vector
else None)
else:
    r = await self.search_client.search(query_text,
                                         filter=filter,
                                         top=top,
                                         vector=query_vector,
                                         top_k=50 if query_vector else None,
                                         vector_fields="embedding" if query_vector
else None)
if use_semantic_captions:
    results = [doc[self.sourcepage_field] + ": " + nonewlines(" .
".join([c.text for c in doc['@search.captions']])) async for doc in r]
else:
    results = [doc[self.sourcepage_field] + ": " +
nonewlines(doc[self.content_field]) async for doc in r]
content = "\n".join(results)
```

統合コードと LLM

Azure AI Search を含む RAG ソリューションでは、完全なソリューションを作成するために、他のコンポーネントとコードが必要です。前のセクションでは、Azure AI Search を使用した情報の取得と、検索可能なコンテンツの作成とクエリに使用される

機能について説明しましたが、このセクションでは LLM の統合と相互作用について説明します。

デモ リポジトリ内のノートブックは、LLM に検索結果を渡すためのパターンを示しているため、出発点として最適です。RAG ソリューションのほとんどのコードは LLM の呼び出しで構成されているため、この記事の範囲外であるこれらの API のしくみを理解する必要があります。

[chat-read-retrieve-read.ipynb](#) ノートブックの次のセルブロックは、チャットセッションのコンテキストでの検索呼び出しを示しています。

Python

```
# Execute this cell multiple times updating user_input to accumulate chat
history
user_input = "Does my plan cover annual eye exams?"

# Exclude category, to simulate scenarios where there's a set of docs you
can't see
exclude_category = None

if len(history) > 0:
    completion = openai.Completion.create(
        engine=AZURE_OPENAI_GPT_DEPLOYMENT,
        prompt=summary_prompt_template.format(summary="\n".join(history),
question=user_input),
        temperature=0.7,
        max_tokens=32,
        stop=["\n"])
    search = completion.choices[0].text
else:
    search = user_input

# Alternatively simply use search_client.search(q, top=3) if not using
semantic ranking
print("Searching:", search)
print("-----")
filter = "category ne '{}'".format(exclude_category.replace("'", ""))
if exclude_category else None
r = search_client.search(search,
                        filter=filter,
                        query_type=QueryType.SEMANTIC,
                        query_language="en-us",
                        query_speller="lexicon",
                        semantic_configuration_name="default",
                        top=3)
results = [doc[KB_FIELDS_SOURCEPAGE] + ": " +
          doc[KB_FIELDS_CONTENT].replace("\n", "").replace("\r", "") for doc in r]
content = "\n".join(results)

prompt = prompt_prefix.format(sources=content) + prompt_history + user_input
+ turn_suffix
```

```
completion = openai.Completion.create(
    engine=AZURE_OPENAI_CHATGPT_DEPLOYMENT,
    prompt=prompt,
    temperature=0.7,
    max_tokens=1024,
    stop=["<|im_end|>", "<|im_start|>"])

prompt_history += user_input + turn_suffix + completion.choices[0].text +
"\n<|im_end|>" + turn_prefix
history.append("user: " + user_input)
history.append("assistant: " + completion.choices[0].text)

print("\n-----\n".join(history))
print("\n-----\nPrompt:\n" + prompt)
```

ファースト ステップ¹

- Azure AI Studio を使用して検索インデックスを作成します。
- Azure OpenAI Studio と "データ持ち込み"を使用して、プレイグラウンドで既存の検索インデックスに対するプロンプトを試します。この手順は、使用するモデルを決定するのに役立ち、RAG シナリオで既存のインデックスがどの程度適切に動作するかを示します。
- Azure AI 検索チームによって構築された "データとのチャット" ソリューション アクセラレータ²は、独自のカスタム RAG ソリューションを作成するのに役立ちます。
- エンタープライズ チャット アプリテンプレート³では、Contoso と Northwind の架空の医療保険ドキュメントを使用して、Azure リソース、コード、サンプルのグラウンドティング データをデプロイします。このエンドツー エンド ソリューションを使用すると、運用チャット アプリをわずか 15 分ほどで利用できます。これらのテンプレートのコードは、いくつかのプレゼンテーションで取り上げられる azure-search-openai-demo です。次のリンクでは言語固有のバージョンが提供されます。
 - .NET⁴
 - Python⁵
 - JavaScript⁶
 - Java⁷
- インデックス作成の概念と戦略を確認して、データを取り込む方法と更新方法を決定します。ベクトル検索、キーワード検索、ハイブリッド検索のどれを使用するかを決定します。検索する必要があるコンテンツの種類と実行するクエリの種類によって、インデックスの設計が決まります。

- クエリの作成について確認して、検索要求の構文と要件の詳細について確認します。

① 注意

一部の Azure AI Search 機能は人による操作を目的としており、RAG パターンでは役に立ちません。具体的には、オートコンプリートと候補をスキップできます。ファセットや orderby などの他の機能は役立つ可能性がありますが、RAG シナリオでは一般的ではありません。

関連項目

- 取得拡張生成: インテリジェント自然言語処理モデルの作成の合理化 ↗
- Azure Machine Learning プロンプト フローを使用した取得拡張生成
- Azure Cognitive Search と LangChain: 拡張ベクトル検索機能のシームレスな統合 ↗

Azure AI Search でクエリを実行する

[アーティクル] • 2023/11/15

Azure AI Search では、フリーフォーム テキスト検索から高度に指定されたクエリ パターンやベクトル検索まで、さまざまなシナリオのクエリ コンストラクトがサポートされます。すべてのクエリは、検索可能なコンテンツを格納する検索インデックスに対して実行されます。

クエリの種類

クエリ	検索可能なコンテンツ	説明
フルテキスト検索	トーカン化された用語の逆インデックス。	フルテキストクエリでは、任意の数の検索ドキュメント内で高速スキャンできるように構造化された転置インデックスを反復処理します。この場合、すべてのフィールドで一致が見つかる可能性があります。テキストは分析されてフルテキスト検索用にトーカン化されます。
ベクトル検索	生成された埋め込みのベクトルインデックス。	ベクトル クエリでは、検索インデックスのベクトル フィールドが反復処理されます。
ハイブリッド検索	1 つの検索インデックス上で記述したすべて。	1 つのクエリ要求でテキスト検索とベクトル検索を組み合わせます。テキスト検索は、"検索可能" および "フィルター処理可能" なフィールドのプレーンテキストコンテンツに対して機能します。ベクトル検索は、ベクトル フィールド内のコンテンツに対して機能します。
その他	プレーンテキストと英数字のコンテンツ。	ソース ドキュメントから逐語的に抽出された未加工のコンテンツは、地理空間検索、あいまい検索、フィールド検索などのフィルターおよびパターン マッチング クエリをサポートします。

この記事では最後のカテゴリ、つまり、フィルターやその他の特殊なクエリ フォームに使用される、元のソースからそのまま抽出された、プレーン テキストと英数字のコンテンツに対して機能するクエリに焦点を当てます。

オートコンプリートとクエリ候補

[オートコンプリート](#)や[結果候補](#)は、`search` の代替手段です。これは、部分文字列の入力 (各文字の後) に基づいて一連のクエリ要求を入力につれて検索する方式で行いま

す。 このチュートリアルで説明しているように、`autocomplete` と `suggestions` パラメーターは一緒に使用することも、個別に使用することができますが、`search` と一緒に使用することはできません。 完成した用語とクエリ候補はどちらもインデックスの内容から派生されます。 このエンジンは、インデックスに存在しない文字列や候補を返しません。 詳細については、「[オートコンプリート \(REST API\)](#)」と「[検索候補 \(REST API\)](#)」をご覧ください。

フィルター検索

フィルターは、Azure AI Search を元に構築したアプリで広く使用されています。 アプリケーションページでは、フィルターは多くの場合、ユーザー向けのフィルター処理のためにリンクナビゲーション構造のファセットとして視覚化されます。 フィルターは、インデックス付きコンテンツのスライスを公開するために、内部的にも使用されます。 たとえば、製品カテゴリに対してフィルターを使用して検索ページを初期化したり、インデックスに英語とフランス語の両方のフィールドが含まれている場合は、言語を初期化したりすることができます。

次の表に示すように、特殊なクエリ フォームを呼び出すフィルターが必要な場合もあります。 指定されていない検索 (`search=*`) を含むフィルターを使用することも、用語、語句、演算子、およびパターンを含むクエリ文字列を含むフィルターを使用することもできます。

ユーザ	説明
ーシナ	
リオ	
範囲フィルター	Azure AI Search では、範囲クエリは filter パラメーターを使用して作成されます。 詳細と例については、「 範囲フィルターの例 」をご覧ください。
—	
ファセットナビゲーション	ファセットナビゲーションツリーでは、ユーザーがファセットを選択できます。 フィルターでサポートされている場合、クリックするたびに検索結果が絞り込まれます。 ファセットによって指定された条件に一致しなくなったドキュメントを除外するフィルターによって、各ファセットがサポートされます。

① 注意

フィルター式で使用されるテキストは、クエリ処理中には分析されません。 テキスト入力は、照合に成功するか失敗するかのどちらかになる逐語的な文字パターンで、大文字と小文字が区別されます。 フィルター式は OData 構文を使用して構築され、インデックス内のすべての "フィルター可能な" フィールドの `filter` パ

ラメーターで渡されます。 詳細については、「Azure AI Search のフィルター」をご覧ください。

地理空間検索

地理空間検索では、"近くを検索" またはマップベースの検索エクスペリエンスのために、場所の緯度と経度の座標との一致を検索します。 Azure AI Search では、次の手順に従って地理空間検索を実装できます。

- [Edm.GeographyPoint](#)、[Collection\(Edm.GeographyPoint\)](#)、[Edm.GeographyPolygon](#) のいずれかの種類のフィルター可能フィールドを定義します。
- 受信ドキュメントに適切な座標が含まれていることを確認します。
- インデックス作成が完了したら、フィルターと[地理空間関数](#)を使用するクエリを作成します。

地理空間検索では、距離にキロメートルを使用します。 座標は `(longitude, latitude)` の形式で指定されます。

地理空間検索のフィルターの例を次に示します。 このフィルターでは、地理的なポイント (この例ではワシントン D.C.) から半径 300 キロ以内に座標を持つ他の `Location` フィールドを検索インデックスから見つけます。 結果にアドレス情報を返し、場所による自己ナビゲーションのためにオプションの `facets` 句を含めます。

HTTP

```
POST https://{{searchServiceName}}.search.windows.net/indexes/hotels-vector-quickstart/docs/search?api-version=2023-07-01-Public
{
    "count": true,
    "search": "*",
    "filter": "geo.distance(Location, geography'POINT(-77.03241 38.90166)') le 300",
    "facets": [ "Address/StateProvince" ],
    "select": "HotelId, HotelName, Address/StreetAddress, Address/City, Address/StateProvince",
    "top": 7
}
```

詳細および例については、[地理空間検索の例](#)に関するページをご覧ください。

ドキュメントの検索

前述のクエリ フォームとは対照的に、このフォームでは、対応するインデックス検索またはスキヤンを行わずに、[IDによる検索ドキュメント](#)を1つ取得します。1つのドキュメントのみが要求され、返されます。ユーザーが検索結果で項目を選択する場合、ドキュメントの取得と詳細ページでのフィールドの設定が典型的な応答になり、ドキュメントの検索はこれをサポートする操作になります。

高度な検索: あいまい、ワイルドカード、近接、正規表現

高度なクエリ フォームは、特定のクエリ動作をトリガーする完全な Lucene パーサーと演算子に依存します。

クエリの種類	使用法	例/詳細情報
フィールド検索	<code>search</code> パラメーター、 <code>queryType=full</code>	1つのフィールドを対象とする複合クエリ式を作成します。 フィールド検索の例
あいまい検索	<code>search</code> パラメーター、 <code>queryType=full</code>	構造やスペリングが似ている語句を照合します。 あいまい検索の例
近接検索	<code>search</code> パラメーター、 <code>queryType=full</code>	ドキュメント内で近くにある語句を検索します。 近接検索の例
用語ブースト	<code>search</code> パラメーター、 <code>queryType=full</code>	ブーストされた語を含むドキュメントの順位を、含まないドキュメントよりも引き上げます。 用語ブーストの例
正規表現検索	<code>search</code> パラメーター、 <code>queryType=full</code>	正規表現の内容に基づいて照合します。 正規表現の例
ワイルドカードまたはプレフィックス検索	<code>*</code> <code>~</code> または <code>?</code> を使用した <code>search</code> パラメーター、 <code>queryType=full</code>	プレフィックスとチルダ (<code>~</code>) または1つの文字 (<code>?</code>) に基づいて照合します。 ワイルドカード検索の例

次のステップ[°]

クエリの実装について詳しく見るには、構文ごとに例を確認します。フルテキスト検索を初めて使用する場合は、クエリ エンジンの機能をよく理解しておくことをお勧めします。

- 簡易クエリの例
- 高度なクエリを作成するための Lucene 構文のクエリの例
- Azure AI Search でのフルテキスト検索のしくみgit