Vector Databases & Hybrid Search

Semantic + Keyword Search for Business Intelligence

FAISS, Embeddings, and Real-World Applications

Module Overview

What You'll Learn

- **Hybrid search concepts** - Combining semantic and keyword search
- **Embeddings** - Converting text to numbers for AI understanding
- **FAISS** - Vector database for similarity search
- **FTS5** - SQLite's full-text search capabilities
- **Weight modification** - Tuning search results for your use case

What is Hybrid Search?

The Best of Both Worlds

Hybrid search combines **two different search approaches**:

1. **Keyword Search (FTS5 + BM25)** - Finds exact matches and phrases
2. **Semantic Search (FAISS + Embeddings)** - Understands meaning and context

Why Hybrid? Sometimes you want exact matches ("social media strategy"), sometimes you want related concepts ("marketing approach" → finds "brand strategy").

Understanding the Components

1. Embeddings: Converting Text to Numbers

What are embeddings?

- Embeddings convert text into a list of numbers (vectors)
- Similar text gets similar numbers
- Our app uses OpenAI's `text-embedding-3-small` model (1536 dimensions)

Embeddings Example

How It Works

```
# Text: "marketing strategy"
# Embedding: [0.1, -0.3, 0.8, 0.2, ...] (1536 numbers)

# Text: "brand approach"
# Embedding: [0.2, -0.2, 0.7, 0.3, ...] (similar numbers = similar meaning)
```

Similar numbers = similar meaning

Embeddings Implementation

OpenAI Integration

```
# services/embedding_service.py
async def generate_embedding(self, text: str) -> List[float]:
    response = await self.client.embeddings.create(
        model="text-embedding-3-small",
        input=text
    )
    return response.data[0].embedding  # Returns 1536 numbers
```

1536-dimensional vectors for semantic understanding

Understanding the Components

2. FAISS: Vector Database

What is FAISS?

- Facebook AI Similarity Search
- Stores and searches through vectors efficiently
- Finds similar vectors quickly (cosine similarity)

Vector Database Comparison

Choosing the Right Solution

Feature	FAISS	ChromaDB	Pinecone	PostgreSQL + pgvector
Setup	Simple, local	Easy, local/cloud	Cloud service	Moderate, local/cloud
Cost	Free	Free/paid	Paid	Free/paid
Performance	Very fast	Good	Good	Good
Scalability	Self-managed	Self-managed	Managed	Self-managed

FAISS Implementation

Current Setup

```python
# services/hybrid_search_service.py
class HybridSearchService:
    def __init__(self):
        self.faiss_index = faiss.IndexFlatIP(1536)  # Inner product index
        self.embedding_dimension = 1536
```

Inner product index for cosine similarity

Switching Vector Databases

Easy Migration Strategy

The vector database implementation is abstracted in our `HybridSearchService` class:

1. **Create a new vector database adapter** (e.g., `PostgreSQLVectorAdapter`)
2. **Implement the same interface** as the current FAISS implementation
3. **Update the service initialization** to use the new adapter
4. **Migrate existing vector data** to the new database

PostgreSQL + pgvector Example

Alternative Implementation

```python
# New adapter class
class PostgreSQLVectorAdapter:
    def __init__(self, connection_string: str):
        self.conn = psycopg2.connect(connection_string)

    def add_vectors(self, vectors: List[List[float]], ids: List[int]):
        # Insert vectors into PostgreSQL with pgvector
        pass

    def search(self, query_vector: List[float], k: int = 10):
        # Use pgvector's similarity search
        pass

# Update HybridSearchService
class HybridSearchService:
    def __init__(self, vector_db_type: str = "faiss"):
```

Understanding the Components

3. FTS5: Full-Text Search

What is FTS5?

- SQLite's Full-Text Search extension
- Searches through text content efficiently
- Uses BM25 ranking algorithm

FTS5 Implementation

Virtual Table Setup

```sql
-- Virtual table for searchable content
CREATE VIRTUAL TABLE chunks_fts USING fts5(
    content,
    chunk_type,
    content='chunks',
    content_rowid='rowid'
);
```

BM25 ranking for keyword relevance

Understanding Weight Modification

The Hybrid Score Formula

```
hybrid_score = (bm25_weight × BM25_score) + (cosine_weight × cosine_similarity_score)
```

Default Weights:

- `bm25_weight = 0.35` (35% keyword search)
- `cosine_weight = 0.65` (65% semantic search)

Weight Modification Scenarios

Scenario 1: More Keyword Focus

```
# For exact term matching (legal documents, code)
bm25_weight = 0.7     # 70% keyword
cosine_weight = 0.3   # 30% semantic
```

Use case: Legal documents, code repositories

Weight Modification Scenarios

Scenario 2: More Semantic Focus

```
# For creative content (marketing, research)
bm25_weight = 0.2     # 20% keyword
cosine_weight = 0.8   # 80% semantic
```

Use case: Marketing content, research papers

Weight Modification Scenarios

Scenario 3: Balanced Search

```
# For general content
bm25_weight = 0.5    # 50% keyword
cosine_weight = 0.5  # 50% semantic
```

Use case: General business documents

Testing Different Weights

API Integration

```
# API call with custom weights
GET /search/hybrid?q=marketing&bm25_weight=0.2&cosine_weight=0.8
```

Easy experimentation with different weight combinations

Learning Path

Phase 1: Understanding the Basics

1. **Study the embedding service** (`services/embedding_service.py`)
2. **Examine chunk creation** (`services/chunking_service.py`)
3. **Test different search methods** via the UI at `/search`

Learning Path

Phase 2: Experimenting with Weights

1. **Modify weights** in the search UI
2. **Compare results** between keyword, semantic, and hybrid
3. **Test with different content types**

Learning Path

Phase 3: Extending the System

1. **Add new content types** (PDFs, images)
2. **Implement custom scoring** algorithms
3. **Add search filters** and faceted search

Learning Path

Phase 4: Advanced Features

1. **Add search suggestions** and autocomplete
2. **Implement search analytics** and user behavior tracking
3. **Add real-time search** updates

Architecture Deep Dive

Data Flow

1. User types search query
2. Query gets embedded (text → numbers)
3. FAISS finds similar vectors
4. FTS5 finds keyword matches
5. Results get combined with weights
6. Final ranked results returned

Key Files to Study

Implementation Details

- `services/hybrid_search_service.py` - Core hybrid search logic
- `services/embedding_service.py` - OpenAI integration
- `services/chunking_service.py` - Content processing
- `routes/marketing.py` - Search API endpoints
- `templates/search.html` - Frontend interface

Extending for Real-World Use

Adding New Content Types

```python
# Example: Add PDF support
class PDFChunkingService:
    def chunk_pdf(self, pdf_path: str) -> List[Chunk]:
        # Extract text from PDF
        # Split into chunks
        # Generate embeddings
        pass
```

Extensible architecture for any content type

Custom Scoring Algorithms

Example: Boost Recent Content

```python
# Example: Boost recent content
def custom_score(self, chunk, base_score):
    recency_boost = self.calculate_recency_boost(chunk.created_at)
    return base_score * recency_boost
```

Customize ranking for your business needs

Performance Optimization

Caching Frequent Searches

```python
# Example: Caching frequent searches
@lru_cache(maxsize=1000)
def cached_embedding(self, text: str):
    return self.generate_embedding(text)
```

Improve performance with intelligent caching

Monitoring and Analytics

Search Metrics to Track

- **Query performance** (response time)
- **Result relevance** (click-through rates)
- **Popular queries** and search patterns
- **Search method effectiveness** (hybrid vs keyword vs semantic)

Analytics Implementation

Tracking Search Metrics

```python
# Add to search endpoints
async def track_search(query: str, method: str, results_count: int):
    # Log search metrics
    # Update analytics database
    pass
```

Data-driven search optimization

Troubleshooting Common Issues

Issue 1: No Search Results

- **Check**: Are chunks being created?
- **Check**: Are embeddings being generated?
- **Check**: Is FAISS index built?

Troubleshooting Common Issues

Issue 2: Poor Search Quality

- **Try**: Adjusting weights
- **Try**: Different chunk sizes
- **Try**: Different embedding models

Troubleshooting Common Issues

Issue 3: Slow Performance

- **Check**: FAISS index size
- **Check**: Database query performance
- **Consider**: Adding caching

Corporate Database Use Case

10,000 Customer Analysis

This architecture is perfect for analyzing a corporate database of 10,000 customers/stakeholders for cross-selling opportunities.

Why This Architecture Works for Customer Analysis

Key Benefits

1. **Semantic Understanding** - Find customers with similar interests even if they used different words
2. **Hybrid Search** - Combine exact matches (company names, industries) with semantic matches (similar business needs)
3. **Scalable** - FAISS handles 10,000+ records efficiently
4. **Real-time** - Update customer profiles and immediately searchable

Customer Data Structure

Implementation for Customer Analysis

```python
class CustomerProfile:
    company_name: str
    industry: str
    business_needs: str  # Rich text description
    current_products: List[str]
    pain_points: str
    company_size: str
    location: str
    engagement_level: str  # webinar, whitepaper, trial, etc.
```

Search Use Cases

Customer Analysis Queries

- **"Find companies like Acme Corp"** → Semantic search finds similar business profiles
- **"Manufacturing companies needing automation"** → Hybrid search finds exact + related matches
- **"SaaS companies with 50-200 employees"** → Filtered semantic search

Cross-Selling Opportunities

Sales Team Queries

```python
# Example search queries for sales teams:
queries = [
    "companies using our CRM who might need marketing automation",
    "enterprise clients who haven't tried our analytics product",
    "SMB customers ready for enterprise features",
    "companies in healthcare needing compliance tools"
]
```

Enhanced Features for Sales

Advanced Customer Analysis

- **Customer Similarity** - "Show me 10 companies most similar to [current customer]"
- **Gap Analysis** - "What products do similar companies use that this customer doesn't?"
- **Engagement Scoring** - Boost customers with high engagement in search results
- **Timing Analysis** - Find customers who engaged 6+ months ago but haven't converted

Performance at Scale

Handling Large Datasets

- **10,000 customers** - FAISS handles this easily (can scale to millions)
- **Real-time updates** - New customer data immediately searchable
- **Fast queries** - Sub-second response times even with complex searches

Business Value

ROI for Customer Analysis

- **Increase Revenue** - Find cross-selling opportunities automatically
- **Improve Targeting** - Better customer segmentation and personalization
- **Save Time** - Sales teams find relevant prospects instantly
- **Data-Driven** - Use actual customer behavior and needs, not just demographics

Further Learning Resources

Additional Study Materials

- **FAISS Documentation**: https://faiss.ai/
- **OpenAI Embeddings Guide**: https://platform.openai.com/docs/guides/embeddings
- **SQLite FTS5**: https://www.sqlite.org/fts5.html
- **Vector Search Best Practices**: https://weaviate.io/blog/vector-search-best-practices

Next Steps

Immediate Actions

1. **Experiment** with the search interface
2. **Modify weights** and see how results change
3. **Add your own content** and test search quality
4. **Extend the system** with new features
5. **Deploy** to a real environment

Key Takeaways

What Makes Hybrid Search Powerful

1. **Combines best of both worlds** - Exact matches + semantic understanding
2. **Highly tunable** - Adjust weights for your specific use case
3. **Scalable** - Handles large datasets efficiently
4. **Real-world ready** - Perfect for business applications
5. **Extensible** - Easy to add new content types and features

Key Takeaways

Business Applications

- **Customer analysis** - Find similar customers and cross-selling opportunities
- **Document search** - Intelligent content discovery
- **Knowledge management** - Organize and find information efficiently
- **Research** - Discover related concepts and ideas
- **Content recommendation** - Suggest relevant content to users

Ready to Build?

Start Learning

The best way to learn is by doing!

Start with small modifications and gradually build more complex features.

Let's create something amazing! 🚀