

- Read
 - Write
 - Glob
 - Grep
-

DSPy + Haystack Integration

Goal

Use DSPy's optimization capabilities to automatically improve prompts in Haystack pipelines.

When to Use

- You have existing Haystack pipelines
- Manual prompt tuning is tedious
- Need data-driven prompt optimization
- Want to combine Haystack components with DSPy optimization

Inputs

Input	Type	Description
<code>haystack_pipeline</code>	<code>Pipeline</code>	Existing Haystack pipeline
<code>trainset</code>	<code>list[dspy.Example]</code>	Training examples
<code>metric</code>	<code>callable</code>	Evaluation function

Outputs

Output	Type	Description
<code>optimized_prompt</code>	<code>str</code>	DSPy-optimized prompt
<code>optimized_pipeline</code>	<code>Pipeline</code>	Updated Haystack pipeline

Workflow

Phase 1: Build Initial Haystack Pipeline

```

from haystack import Pipeline
from haystack.components.generators import OpenAIGenerator
from haystack.components.builders import PromptBuilder
from haystack.components.retrievers.in_memory import InMemoryBM25Retriever
from haystack.document_stores.in_memory import InMemoryDocumentStore

# Setup document store
doc_store = InMemoryDocumentStore()
doc_store.write_documents(documents)

# Initial generic prompt
initial_prompt = """
Context: {{context}}
Question: {{question}}
Answer:
"""

# Build pipeline
pipeline = Pipeline()
pipeline.add_component("retriever", InMemoryBM25Retriever(document_store=doc_store))
pipeline.add_component("prompt_builder", PromptBuilder(template=initial_prompt))
pipeline.add_component("generator", OpenAIGenerator(model="gpt-3.5-turbo"))

pipeline.connect("retriever", "prompt_builder.context")
pipeline.connect("prompt_builder", "generator")

```

Phase 2: Create DSPy RAG Module

```

import dspy

class HaystackRAG(dspy.Module):
    """DSPy module wrapping Haystack retriever."""

    def __init__(self, retriever, k=3):
        super().__init__()
        self.retriever = retriever
        self.k = k
        self.generate = dspy.ChainOfThought("context, question -> answer")

    def forward(self, question):
        # Use Haystack retriever
        results = self.retriever.run(query=question, top_k=self.k)
        context = [doc.content for doc in results['documents']]

        # Use DSPy for generation
        pred = self.generate(context=context, question=question)
        return dspy.Prediction(context=context, answer=pred.answer)

```

Phase 3: Define Custom Metric

```
from haystack.components.evaluators import SASEvaluator

# Haystack semantic evaluator
sas_evaluator = SASEvaluator(model="sentence-transformers/all-MiniLM-L6-v2")

def mixed_metric(example, pred, trace=None):
    """Combine semantic accuracy with conciseness."""

    # Semantic similarity (Haystack SAS)
    sas_result = sas_evaluator.run(
        ground_truth_answers=[example.answer],
        predicted_answers=[pred.answer]
    )
    semantic_score = sas_result['score']

    # Conciseness penalty
    word_count = len(pred.answer.split())
    conciseness = 1.0 if word_count <= 20 else max(0, 1 - (word_count - 20) / 50)

    return 0.7 * semantic_score + 0.3 * conciseness
```

Phase 4: Optimize with DSPy

```
from dspy.teleprompt import BootstrapFewShot

dspy.configure(lm=dspy.LM("openai/gpt-3.5-turbo"))

# Create DSPy module with Haystack retriever
rag_module = HaystackRAG(retriever=pipeline.get_component("retriever"))

# Optimize
optimizer = BootstrapFewShot(
    metric=mixed_metric,
    max_bootstrapped_demos=4,
    max_labeled_demos=4
)

compiled = optimizer.compile(rag_module, trainset=trainset)
```

Phase 5: Extract and Apply Optimized Prompt

```

def extract_dspy_prompt(compiled_module):
    """Extract the optimized prompt from compiled DSPy module."""
    # Get the predictor's demos and instructions
    predictor = compiled_module.generate

    demos = getattr(predictor, 'demos', [])

    # Build prompt with few-shot examples
    prompt_parts = ["Answer questions using the provided context.\n"]

    for demo in demos:
        prompt_parts.append(f"Context: {demo.context}")
        prompt_parts.append(f"Question: {demo.question}")
        prompt_parts.append(f"Answer: {demo.answer}\n")

    prompt_parts.append("Context: {{context}}")
    prompt_parts.append("Question: {{question}}")
    prompt_parts.append("Answer:")

    return "\n".join(prompt_parts)

optimized_prompt = extract_dspy_prompt(compiled)

```

Phase 6: Build Optimized Haystack Pipeline

```

# Create new pipeline with optimized prompt
optimized_pipeline = Pipeline()
optimized_pipeline.add_component("retriever", InMemoryBM25Retriever(document_store=doc_
_store))
optimized_pipeline.add_component("prompt_builder", PromptBuilder(template=optim-
ized_prompt))
optimized_pipeline.add_component("generator", OpenAIGenerator(model="gpt-3.5-turbo"))

optimized_pipeline.connect("retriever", "prompt_builder.context")
optimized_pipeline.connect("prompt_builder", "generator")

```

Production Example

```

import dspy
from dspy.teleprompt import BootstrapFewShot
from haystack import Pipeline, Document
from haystack.components.generators import OpenAIGenerator
from haystack.components.builders import PromptBuilder
from haystack.components.retrievers.in_memory import InMemoryBM25Retriever
from haystack.document_stores.in_memory import InMemoryDocumentStore
import logging

logger = logging.getLogger(__name__)

class HaystackDSPyOptimizer:
    """Optimize Haystack pipelines using DSPy."""

    def __init__(self, document_store, lm_model="openai/gpt-3.5-turbo"):
        self.doc_store = document_store
        self.retriever = InMemoryBM25Retriever(document_store=document_store)
        dspy.configure(lm=dspy.LM(lm_model))

    def create_dspy_module(self, k=3):
        """Create DSPy module wrapping Haystack retriever."""

        class RAGModule(dspy.Module):
            def __init__(inner_self):
                super().__init__()
                inner_self.generate = dspy.ChainOfThought("context, question -> answer")

            def forward(inner_self, question):
                results = self.retriever.run(query=question, top_k=k)
                context = [doc.content for doc in results.get('documents', [])]

                if not context:
                    return dspy.Prediction(context=[], answer="No relevant information found.")

                pred = inner_self.generate(context=context, question=question)
                return dspy.Prediction(context=context, answer=pred.answer)

        return RAGModule()

    def optimize(self, trainset, metric=None):
        """Run DSPy optimization."""

        metric = metric or (lambda ex, pred, trace=None:
                            ex.answer.lower() in pred.answer.lower())

        module = self.create_dspy_module()

        optimizer = BootstrapFewShot(
            metric=metric,
            max_bootstrapped_demos=4,
            max_labeled_demos=4
        )

        compiled = optimizer.compile(module, trainset=trainset)
        logger.info("DSPy optimization complete")

        return compiled

    def build_optimized_pipeline(self, compiled_module):
        """Create Haystack pipeline with DSPy-optimized prompt."""

```

```

# Extract demos from compiled module
demos = getattr(compiled_module.generate, 'demos', [])

# Build optimized prompt template
prompt_lines = ["Answer the question using the context provided.\n"]

for i, demo in enumerate(demos[:4]): # Limit to 4 examples
    prompt_lines.append(f"Example {i+1}:")
    prompt_lines.append(f"Context: {demo.context[:500]}...") # Truncate
    prompt_lines.append(f"Question: {demo.question}")
    prompt_lines.append(f"Answer: {demo.answer}\n")

prompt_lines.extend([
    "Now answer this question:",
    "Context: {{context}}",
    "Question: {{question}}",
    "Answer:"
])

optimized_prompt = "\n".join(prompt_lines)

# Build pipeline
pipeline = Pipeline()
pipeline.add_component("retriever", InMemoryBM25Retriever(document_store=self.doc_store))
pipeline.add_component("prompt_builder", PromptBuilder(template=optimized_prompt))
pipeline.add_component("generator", OpenAIGenerator(model="gpt-3.5-turbo"))

pipeline.connect("retriever", "prompt_builder.context")
pipeline.connect("prompt_builder", "generator")

return pipeline

# Usage
optimizer = HaystackDSPyOptimizer(doc_store)
compiled = optimizer.optimize(trainset)
pipeline = optimizer.build_optimized_pipeline(compiled)

# Run optimized pipeline
result = pipeline.run({"retriever": {"query": "What is photosynthesis?"}})

```

Best Practices

1. **Match retrievers** - Use same retriever in DSPy module as Haystack pipeline
2. **Custom metrics** - Combine Haystack evaluators with DSPy optimization
3. **Prompt extraction** - Carefully map DSPy demos to Haystack template format
4. **Test both** - Validate DSPy module AND final Haystack pipeline

Limitations

- Prompt template conversion can be tricky
- Some Haystack features don't map directly to DSPy
- Requires maintaining two codebases initially
- Complex pipelines may need custom integration