VisionThing

February 2, 2026

[1]:
```python
#psychometrics.ai

import os
from PIL import Image
```

[2]:
```python
#collect pngs and check they open

import glob

# Collect only PNGs from the current folder
image_paths = sorted(glob.glob("./*.png"))[:30]

print(f"Using {len(image_paths)} PNGs:")
for p in image_paths:
    print(" -", p)

# Quick open-check to catch any corrupted files early
bad = []
for p in image_paths:
    try:
        with Image.open(p) as img:
            img.verify()  # verifies file integrity
    except Exception as e:
        bad.append((p, str(e)))

if bad:
    print("  Some images failed integrity check:")
    for p, err in bad:
        print(f" - {p}: {err}")
    raise SystemExit("Please fix or remove the bad files before proceeding.")
```

```
Using 30 PNGs:
 - ./A-1.png
 - ./A-10.png
 - ./A-11.png
 - ./A-12.png
 - ./A-13.png
 - ./A-14.png
 - ./A-15.png
 - ./A-16.png
 - ./A-17.png
 - ./A-18.png
 - ./A-19.png
 - ./A-2.png
 - ./A-20.png
 - ./A-21.png
 - ./A-22.png
 - ./A-23.png
 - ./A-24.png
 - ./A-25.png
 - ./A-26.png
 - ./A-27.png
 - ./A-28.png
 - ./A-29.png
 - ./A-3.png
 - ./A-30.png
 - ./A-4.png
 - ./A-5.png
 - ./A-6.png
 - ./A-7.png
 - ./A-8.png
 - ./A-9.png
```

[3]:
```python
from sentence_transformers import SentenceTransformer
import torch

device = "cuda" if torch.cuda.is_available() else "cpu"

# IMPORTANT: use the correct model ID
model = SentenceTransformer("sentence-transformers/clip-ViT-B-32",␣
 ↪device=device)

print("Loaded:", type(model))
print("Device:", device)
```

```
/opt/anaconda3/lib/python3.11/site-
packages/sentence_transformers/cross_encoder/CrossEncoder.py:11:
TqdmExperimentalWarning: Using `tqdm.autonotebook.tqdm` in notebook mode. Use
`tqdm.tqdm` instead to force console mode (e.g. in jupyter console)
  from tqdm.autonotebook import tqdm, trange
```

```
Loaded: <class 'sentence_transformers.SentenceTransformer.SentenceTransformer'>
Device: cpu
```

[4]:
```python
import numpy as np
```

```
# Create embeddings for the 30 images
embeddings = model.encode(
    image_paths,               # the list of your PNGs
    batch_size=8,              # fine for small sets
    convert_to_numpy=True,
    normalize_embeddings=True, # good for cosine similarity
    show_progress_bar=True
)

print("Embeddings created.")
print("Shape:", embeddings.shape)  # should be (30, 512)
```

```
Batches:   0%|          | 0/4 [00:00<?, ?it/s]

Embeddings created.
Shape: (30, 512)
```

[5]:
```
import pandas as pd

# image_paths = [...]      # already created earlier
# embeddings = [...]       # already computed earlier

df = pd.DataFrame(embeddings)
df.insert(0, "filename", image_paths)

# Show a scrollable full-width table inside Jupyter
from IPython.display import display, HTML
display(HTML(df.to_html(max_rows=200, max_cols=200, notebook=True)))
```

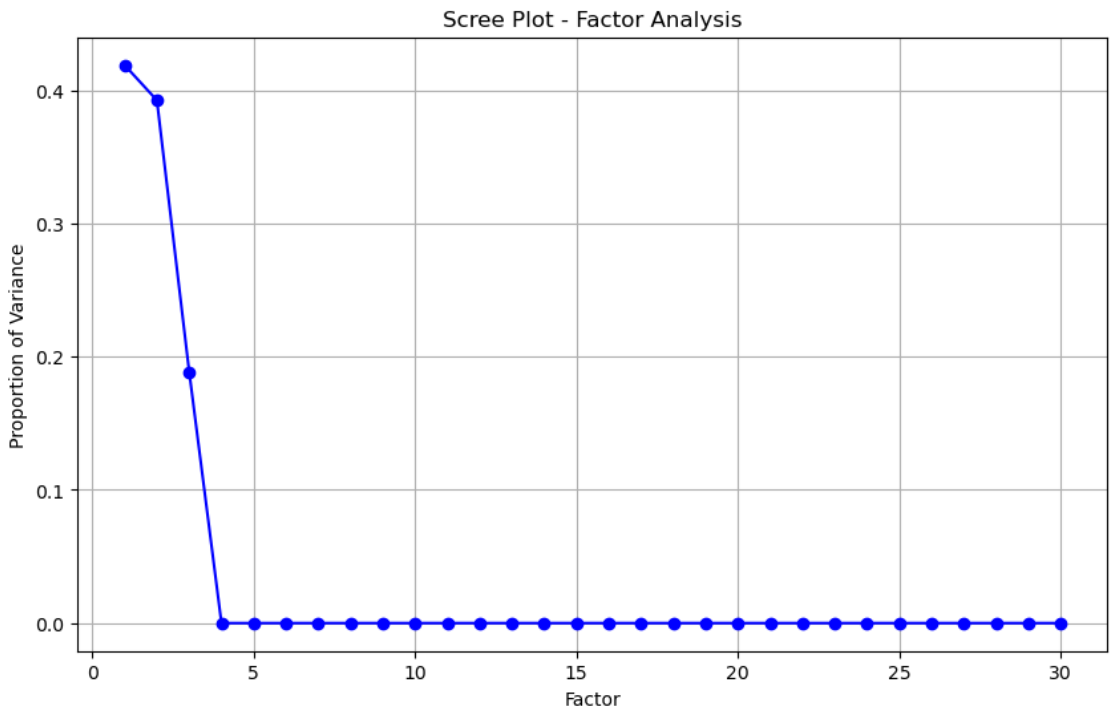
```
<IPython.core.display.HTML object>
```

[6]:
```
from sklearn.metrics.pairwise import cosine_similarity

# Compute cosine similarity matrix
similarity_matrix = cosine_similarity(embeddings)

# Create labeled DataFrame
filenames = [os.path.basename(p) for p in image_paths]
sim_df = pd.DataFrame(similarity_matrix, columns=filenames)
sim_df.insert(0, "filename", filenames)

# Display
display(HTML(sim_df.to_html(max_rows=200, max_cols=200, notebook=True)))
```

```
<IPython.core.display.HTML object>
```

[7]:
```
from sklearn.decomposition import FactorAnalysis
import matplotlib.pyplot as plt
```

```python
# Extract just the similarity values (exclude filename column)
sim_matrix = sim_df.iloc[:, 1:].values

# Factor Analysis with Maximum Likelihood
fa = FactorAnalysis(n_components=None, rotation=None)
fa.fit(sim_matrix)

# Scree plot (using noise variance as proxy for explained variance)
# Note: FA doesn't have explained_variance_ratio_ like PCA
# We can plot the components' variances
component_vars = np.var(fa.transform(sim_matrix), axis=0)
total_var = np.sum(component_vars)
explained_var_ratio = component_vars / total_var

plt.figure(figsize=(10, 6))
plt.plot(range(1, len(explained_var_ratio) + 1),
         explained_var_ratio, 'bo-')
plt.xlabel('Factor')
plt.ylabel('Proportion of Variance')
plt.title('Scree Plot - Factor Analysis')
plt.grid(True)
plt.show()

# Get loadings with labels
loadings = fa.components_.T
loadings_df = pd.DataFrame(
    loadings,
    index=filenames,
    columns=[f'Factor{i+1}' for i in range(loadings.shape[1])]
)
display(HTML(loadings_df.to_html(max_rows=200, max_cols=200, notebook=True)))
```



```
<IPython.core.display.HTML object>
```

[8]:
```
import pandas as pd

# Read the CSV file
form1 = pd.read_csv('form1.csv')

# Check what the columns are actually named
print("Column names:", form1.columns.tolist())
print("First few rows:")
print(form1.head())

# Assign using .values to avoid index mismatch
loadings_df['dis'] = form1.iloc[:, 0].values  # First column (dis)
loadings_df['dif'] = form1.iloc[:, 1].values  # Second column (dif)

# Display the updated DataFrame
display(HTML(loadings_df.to_html(max_rows=200, max_cols=200, notebook=True)))
```

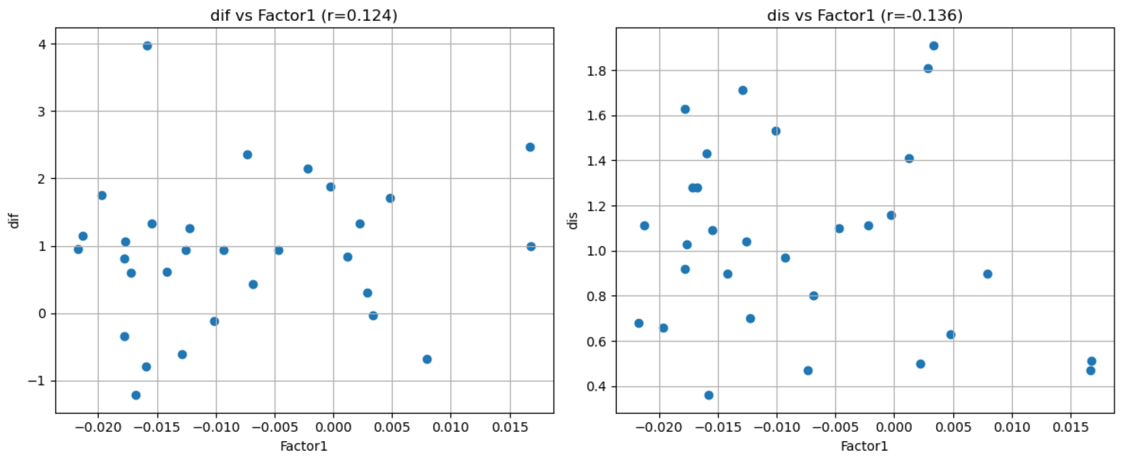
```
Column names: ['dis', 'dif']
First few rows:
    dis   dif
0  1.11  1.15
1  1.10  0.94
2  1.28 -1.22
```

```
3  1.09  1.33
4  0.68  0.95
```

```
<IPython.core.display.HTML object>
```

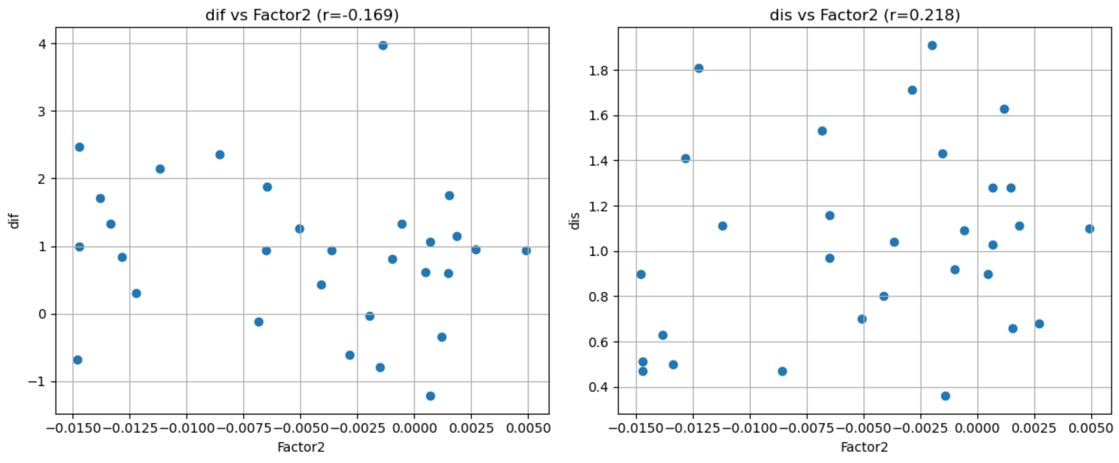
[9]:
```
# Correlations between factor 1 and dis and dif

corr_dif_factor1 = loadings_df['dif'].corr(loadings_df['Factor1'])
corr_dis_factor1 = loadings_df['dis'].corr(loadings_df['Factor1'])
print(f"dif vs Factor1: r = {corr_dif_factor1:.3f}")
print(f"dis vs Factor1: r = {corr_dis_factor1:.3f}")

# Scatter plots
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

axes[0].scatter(loadings_df['Factor1'], loadings_df['dif'])
axes[0].set_xlabel('Factor1')
axes[0].set_ylabel('dif')
axes[0].set_title(f'dif vs Factor1 (r={corr_dif_factor1:.3f})')
axes[0].grid(True)

axes[1].scatter(loadings_df['Factor1'], loadings_df['dis'])
axes[1].set_xlabel('Factor1')
axes[1].set_ylabel('dis')
axes[1].set_title(f'dis vs Factor1 (r={corr_dis_factor1:.3f})')
axes[1].grid(True)

plt.tight_layout()
plt.show()
```

```
dif vs Factor1: r = 0.124
dis vs Factor1: r = -0.136
```



[10]:
```
# Correlations between factor 2 and dis and dif

corr_dif_factor2 = loadings_df['dif'].corr(loadings_df['Factor2'])
corr_dis_factor2 = loadings_df['dis'].corr(loadings_df['Factor2'])
print(f"dif vs Factor2: r = {corr_dif_factor2:.3f}")
print(f"dis vs Factor2: r = {corr_dis_factor2:.3f}")

# Scatter plots
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

axes[0].scatter(loadings_df['Factor2'], loadings_df['dif'])
axes[0].set_xlabel('Factor2')
axes[0].set_ylabel('dif')
axes[0].set_title(f'dif vs Factor2 (r={corr_dif_factor2:.3f})')
axes[0].grid(True)

axes[1].scatter(loadings_df['Factor2'], loadings_df['dis'])
axes[1].set_xlabel('Factor2')
axes[1].set_ylabel('dis')
axes[1].set_title(f'dis vs Factor2 (r={corr_dis_factor2:.3f})')
axes[1].grid(True)

plt.tight_layout()
plt.show()
```

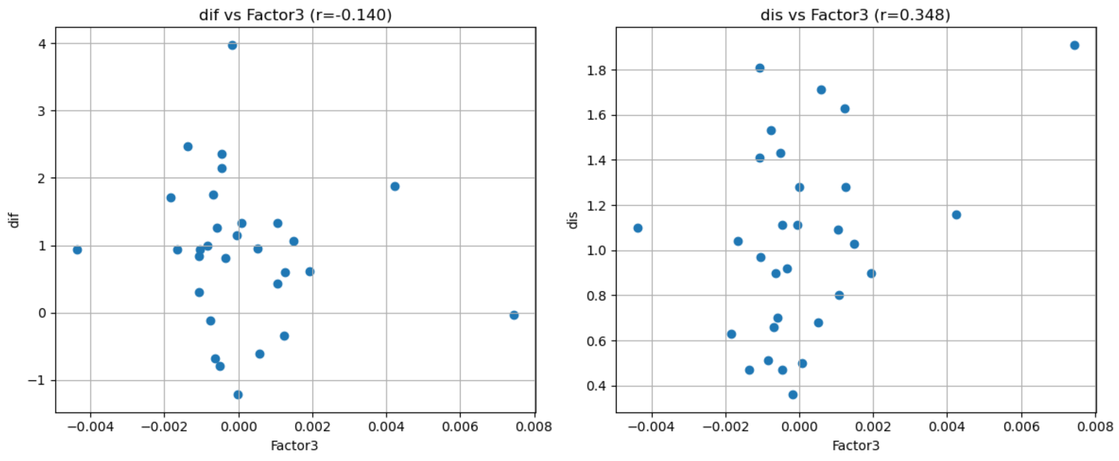
```
dif vs Factor2: r = -0.169
dis vs Factor2: r = 0.218
```



[11]:
```
# Correlations between factor 3 and dis and dif

corr_dif_factor3 = loadings_df['dif'].corr(loadings_df['Factor3'])
```

```
corr_dis_factor3 = loadings_df['dis'].corr(loadings_df['Factor3'])
print(f"dif vs Factor3: r = {corr_dif_factor3:.3f}")
print(f"dis vs Factor3: r = {corr_dis_factor3:.3f}")

# Scatter plots
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

axes[0].scatter(loadings_df['Factor3'], loadings_df['dif'])
axes[0].set_xlabel('Factor3')
axes[0].set_ylabel('dif')
axes[0].set_title(f'dif vs Factor3 (r={corr_dif_factor3:.3f})')
axes[0].grid(True)

axes[1].scatter(loadings_df['Factor3'], loadings_df['dis'])
axes[1].set_xlabel('Factor3')
axes[1].set_ylabel('dis')
axes[1].set_title(f'dis vs Factor3 (r={corr_dis_factor3:.3f})')
axes[1].grid(True)

plt.tight_layout()
plt.show()
```

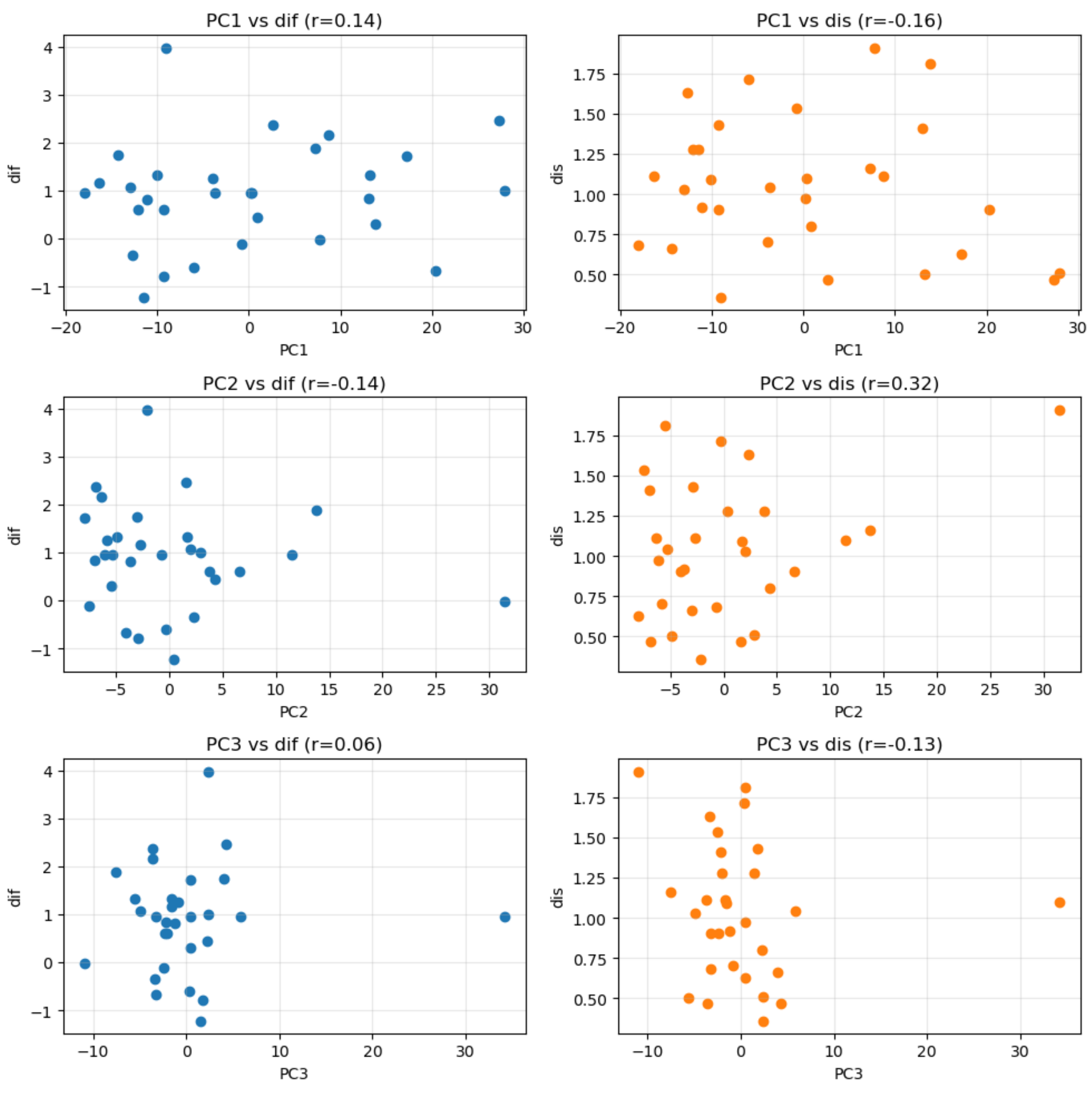
```
dif vs Factor3: r = -0.140
dis vs Factor3: r = 0.348
```



[12]:
```
# --- Try PCA on raw embeddings + correlations with IRT params (dif, dis)

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from scipy.stats import pearsonr

# ---- Gather inputs ----
# embeddings: np.ndarray shape (n_items, 512)
# filenames: list[str] length n_items
# dif/dis: arrays-like length n_items (fallback: read from loadings_df if␣
↪present)
def _arr(x):
    return np.asarray(x, dtype=float).reshape(-1)

n_items = embeddings.shape[0]
if 'filenames' not in globals():
    # Try to reconstruct from earlier sim_df or file paths if needed
    try:
        filenames = list(loadings_df.index)
    except Exception as _:
        filenames = [f"item_{i+1}" for i in range(n_items)]

# Pull dif/dis from globals or loadings_df if available
if 'dif' in globals() and 'dis' in globals():
    dif_vec, dis_vec = _arr(dif), _arr(dis)
else:
    try:
        dif_vec = _arr(loadings_df['dif'].values)
        dis_vec = _arr(loadings_df['dis'].values)
    except Exception as e:
        raise RuntimeError("Need vectors 'dif' and 'dis' (or loadings_df['dif'/
↪'dis']).") from e

assert len(dif_vec) == n_items and len(dis_vec) == n_items, "Length mismatch␣
↪for dif/dis vs embeddings."

# ---- PCA on raw embeddings ----
X = embeddings  # (n_items, 512)
Xz = StandardScaler(with_mean=True, with_std=True).fit_transform(X)
pca = PCA(n_components=min(10, Xz.shape[0]))  # take up to 10 PCs or n_items
scores = pca.fit_transform(Xz)                # (n_items, n_components)
expl_var = pca.explained_variance_ratio_

# Build tidy results frame
pc_cols = [f"PC{i+1}" for i in range(scores.shape[1])]
res_df = pd.DataFrame(scores, columns=pc_cols)
res_df.insert(0, "filename", filenames)
res_df["dif"] = dif_vec
res_df["dis"] = dis_vec
```

```
# ---- Correlate PCs with dif & dis ----
corr_rows = []
for j, pc in enumerate(pc_cols, start=1):
    r_dif, p_dif = pearsonr(res_df[pc], res_df["dif"])
    r_dis, p_dis = pearsonr(res_df[pc], res_df["dis"])
    corr_rows.append({
        "PC": f"PC{j}",
        "ExplainedVar(%)": round(100*expl_var[j-1], 2) if j-1 < len(expl_var)␣
 ↪else np.nan,
        "r(PC, dif)": round(r_dif, 3), "p_dif": p_dif,
        "r(PC, dis)": round(r_dis, 3), "p_dis": p_dis
    })
corr_df = pd.DataFrame(corr_rows)

# ---- Print quick summary ----
print("== PCA on raw embeddings ==")
print(f"Items: {n_items}, Features: {X.shape[1]}")
print("\nTop PCs variance (%):", [round(100*v, 2) for v in expl_var[:5]])
print("\nCorrelations of PCs with dif and dis:")
display(corr_df)

# ---- Quick scatter diagnostics for top 3 PCs ----
k = min(3, scores.shape[1])
fig, axes = plt.subplots(k, 2, figsize=(10, 3.3*k))
axes = np.atleast_2d(axes)
for i in range(k):
    pc = pc_cols[i]
    r_dif, _ = pearsonr(res_df[pc], res_df["dif"])
    r_dis, _ = pearsonr(res_df[pc], res_df["dis"])
    # PC vs dif
    ax = axes[i, 0]
    ax.scatter(res_df[pc], res_df["dif"], c="tab:blue")
    ax.set_xlabel(pc); ax.set_ylabel("dif")
    ax.set_title(f"{pc} vs dif (r={r_dif:.2f})")
    ax.grid(True, alpha=.3)
    # PC vs dis
    ax = axes[i, 1]
    ax.scatter(res_df[pc], res_df["dis"], c="tab:orange")
    ax.set_xlabel(pc); ax.set_ylabel("dis")
    ax.set_title(f"{pc} vs dis (r={r_dis:.2f})")
    ax.grid(True, alpha=.3)

plt.tight_layout()
plt.show()

# ---- Optional: export the per-item PC scores & IRT params ----
# res_df.to_csv("pca_embedding_vs_irt.csv", index=False)
```

```
== PCA on raw embeddings ==
Items: 30, Features: 512

Top PCs variance (%): [32.05, 12.22, 10.3, 8.75, 5.56]

Correlations of PCs with dif and dis:
     PC  ExplainedVar(%)  r(PC, dif)     p_dif  r(PC, dis)     p_dis
0   PC1            32.05       0.141  0.458439      -0.164  0.387055
1   PC2            12.22      -0.139  0.462894       0.318  0.086680
2   PC3            10.30       0.057  0.765581      -0.131  0.489806
3   PC4             8.75       0.060  0.753443      -0.053  0.779854
4   PC5             5.56       0.010  0.959202       0.081  0.670957
5   PC6             4.50       0.065  0.733924       0.222  0.237748
6   PC7             3.18       0.198  0.295263      -0.107  0.573190
7   PC8             2.84       0.096  0.614234      -0.134  0.481732
8   PC9             2.33      -0.083  0.661331      -0.156  0.409502
9  PC10             2.10      -0.010  0.958012      -0.003  0.989374
```



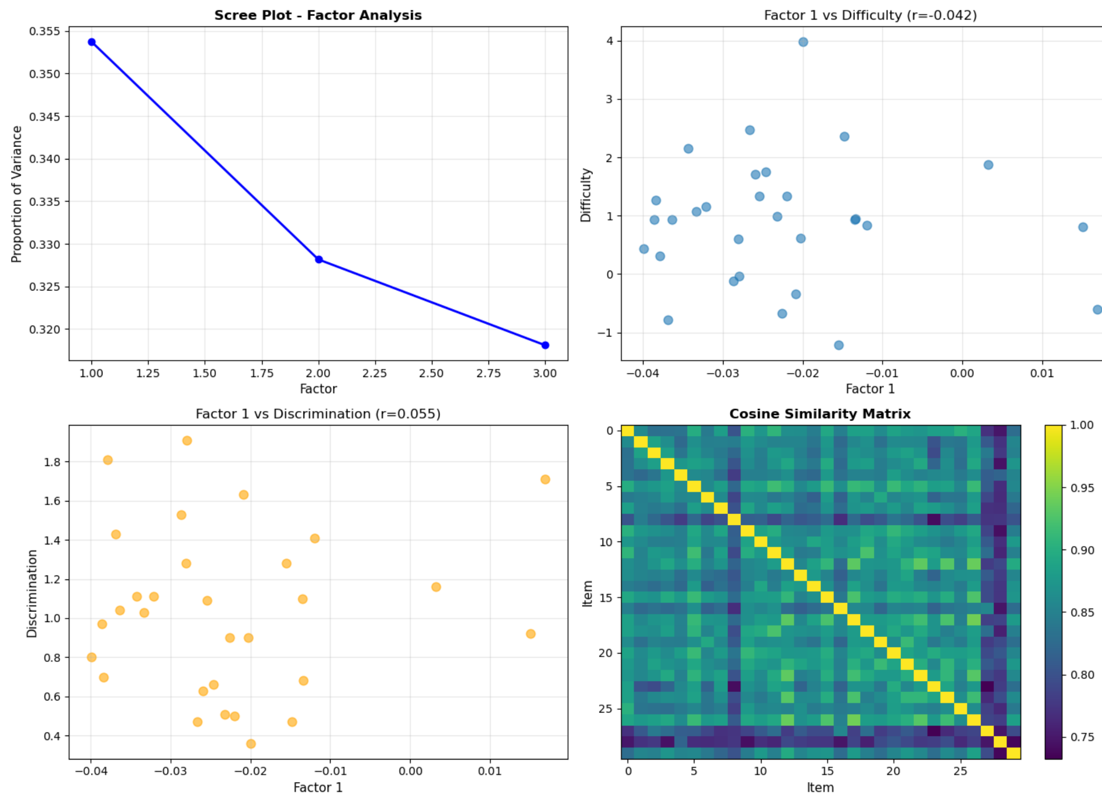
[13]:
```python
"""
Full Pseudo Factor Analysis for Matrix Reasoning Items
Uses multimodal LLM embeddings instead of CLIP
"""

import os
import glob
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from PIL import Image
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.decomposition import FactorAnalysis
from scipy.stats import pearsonr
import base64
from io import BytesIO

# ============================================================================
# CONFIGURATION - ENTER YOUR DETAILS HERE
# ============================================================================

# Choose your API: 'openai', 'anthropic', or 'google'
API_PROVIDER = 'openai'

# API Key - reads from environment variable
# Set in terminal before running:
#   Mac/Linux:   export OPENAI_API_KEY='your-key-here'
#   Windows:     set OPENAI_API_KEY=your-key-here
API_KEY = os.getenv('OPENAI_API_KEY') or os.getenv('ANTHROPIC_API_KEY') or os.
 ↪getenv('GOOGLE_API_KEY')

# Image paths
IMAGE_FOLDER = './'  # Folder containing your PNG files
IMAGE_PATTERN = '*.png'  # Pattern to match images
MAX_IMAGES = 30  # Limit number of images

# IRT parameters CSV
IRT_CSV = 'form1.csv'  # Should have columns: 'dis', 'dif'

# Number of factors for FA
N_FACTORS = 3

# ============================================================================
# HELPER FUNCTIONS
# ============================================================================

def encode_image_base64(image_path):
    """Convert image to base64 string"""
    with open(image_path, "rb") as f:
        return base64.b64encode(f.read()).decode('utf-8')

def get_embeddings_openai(image_paths, api_key):
    """Get embeddings from OpenAI GPT-4V"""
    from openai import OpenAI
    client = OpenAI(api_key=api_key)

    embeddings = []
    for img_path in image_paths:
        print(f"Processing {os.path.basename(img_path)}...")
        # Encode image
        base64_image = encode_image_base64(img_path)

        # Get embedding via completion with special prompt
        response = client.chat.completions.create(
            model="gpt-4o",
            messages=[{
                "role": "user",
                "content": [
                    {"type": "text", "text": "Analyze this matrix reasoning␣
 ↪item. Describe its structure, complexity, and reasoning demands in detail."},
                    {"type": "image_url", "image_url": {"url": f"data:image/png;
 ↪base64,{base64_image}"}}
                ]
            }],
            max_tokens=300
        )

        # Use text embedding of the response as proxy
        text_response = response.choices[0].message.content
        embed_response = client.embeddings.create(
            input=text_response,
            model="text-embedding-3-large"
        )
        embeddings.append(embed_response.data[0].embedding)

    return np.array(embeddings)

def get_embeddings_anthropic(image_paths, api_key):
    """Get embeddings from Claude via description method"""
    import anthropic
    client = anthropic.Anthropic(api_key=api_key)

    embeddings = []
    for img_path in image_paths:
        print(f"Processing {os.path.basename(img_path)}...")

        base64_image = encode_image_base64(img_path)

        # Get detailed description
        message = client.messages.create(
            model="claude-3-5-sonnet-20241022",
            max_tokens=300,
            messages=[{
                "role": "user",
                "content": [
                    {
                        "type": "image",
                        "source": {
                            "type": "base64",
                            "media_type": "image/png",
                            "data": base64_image,
                        },
                    },
                    {
                        "type": "text",
                        "text": "Analyze this matrix reasoning item. Describe:␣
↪(1) number and types of objects, (2) spatial arrangement, (3) transformation␣
↪rules, (4) cognitive complexity, (5) reasoning demands. Be specific and␣
↪systematic."
                    }
                ],
            }]
        )

        # Convert description to embedding using a sentence transformer
        from sentence_transformers import SentenceTransformer
        model = SentenceTransformer('all-mpnet-base-v2')
        text_response = message.content[0].text
        embedding = model.encode(text_response, convert_to_numpy=True)
        embeddings.append(embedding)

    return np.array(embeddings)

def get_embeddings_google(image_paths, api_key):
    """Get embeddings from Google Gemini"""
    import google.generativeai as genai
    genai.configure(api_key=api_key)

    model = genai.GenerativeModel('gemini-1.5-flash')

    embeddings = []
    for img_path in image_paths:
        print(f"Processing {os.path.basename(img_path)}...")

        img = Image.open(img_path)

        response = model.generate_content([
            "Analyze this matrix reasoning item systematically. Describe:␣
↪structural elements, transformation patterns, cognitive demands, and␣
↪problem-solving complexity.",
            img
        ])
        # Convert description to embedding
        from sentence_transformers import SentenceTransformer
        embed_model = SentenceTransformer('all-mpnet-base-v2')
        embedding = embed_model.encode(response.text, convert_to_numpy=True)
        embeddings.append(embedding)

    return np.array(embeddings)

# ============================================================================
# MAIN ANALYSIS
# ============================================================================

def run_full_analysis():
    print("="*70)
    print("PSEUDO FACTOR ANALYSIS FOR MATRIX REASONING ITEMS")
    print("="*70)

    # 1. Load images
    print("\n[1] Loading images...")
    image_paths = sorted(glob.glob(os.path.join(IMAGE_FOLDER, IMAGE_PATTERN)))[:
 ↪MAX_IMAGES]
    print(f"Found {len(image_paths)} images")

    filenames = [os.path.basename(p) for p in image_paths]

    # 2. Load IRT parameters
    print("\n[2] Loading IRT parameters...")
    irt_df = pd.read_csv(IRT_CSV)
    dif = irt_df['dif'].values[:len(image_paths)]
    dis = irt_df['dis'].values[:len(image_paths)]
    print(f"Loaded {len(dif)} difficulty and {len(dis)} discrimination values")

    # 3. Get embeddings
    print(f"\n[3] Getting embeddings from {API_PROVIDER.upper()}...")

    if API_PROVIDER == 'openai':
        embeddings = get_embeddings_openai(image_paths, API_KEY)
    elif API_PROVIDER == 'anthropic':
        embeddings = get_embeddings_anthropic(image_paths, API_KEY)
    elif API_PROVIDER == 'google':
        embeddings = get_embeddings_google(image_paths, API_KEY)
    else:
        raise ValueError(f"Unknown API provider: {API_PROVIDER}")

    print(f"Embeddings shape: {embeddings.shape}")

    # 4. Compute cosine similarity matrix
```

```python
print("\n[4] Computing similarity matrix...")
sim_matrix = cosine_similarity(embeddings)

# 5. Factor Analysis
print(f"\n[5] Running Factor Analysis (n_factors={N_FACTORS})...")
fa = FactorAnalysis(n_components=N_FACTORS, rotation=None)
fa.fit(sim_matrix)

# Get loadings
loadings = fa.components_.T
factor_scores = fa.transform(sim_matrix)

# 6. Create results dataframe
print("\n[6] Organizing results...")
results_df = pd.DataFrame(
    loadings,
    columns=[f'Factor{i+1}' for i in range(N_FACTORS)]
)
results_df.insert(0, 'filename', filenames)
results_df['dif'] = dif
results_df['dis'] = dis

# 7. Compute correlations
print("\n[7] Computing correlations between factors and IRT parameters...")
print("\n" + "="*70)
print("CORRELATIONS: FACTORS vs IRT PARAMETERS")
print("="*70)

for i in range(N_FACTORS):
    factor_col = f'Factor{i+1}'
    r_dif, p_dif = pearsonr(results_df[factor_col], results_df['dif'])
    r_dis, p_dis = pearsonr(results_df[factor_col], results_df['dis'])

    print(f"\n{factor_col}:")
    print(f"  vs Difficulty:     r = {r_dif:+.3f}  (p = {p_dif:.4f})")
    print(f"  vs Discrimination: r = {r_dis:+.3f}  (p = {p_dis:.4f})")

# 8. Visualizations
print("\n[8] Creating visualizations...")

# Scree plot
component_vars = np.var(factor_scores, axis=0)
total_var = np.sum(component_vars)
explained_var_ratio = component_vars / total_var

fig, axes = plt.subplots(2, 2, figsize=(14, 10))
```

```
    # Scree plot
    axes[0, 0].plot(range(1, N_FACTORS + 1), explained_var_ratio, 'bo-',␣
↪linewidth=2)
    axes[0, 0].set_xlabel('Factor', fontsize=11)
    axes[0, 0].set_ylabel('Proportion of Variance', fontsize=11)
    axes[0, 0].set_title('Scree Plot - Factor Analysis', fontsize=12,␣
↪fontweight='bold')
    axes[0, 0].grid(True, alpha=0.3)

    # Factor1 vs parameters
    axes[0, 1].scatter(results_df['Factor1'], results_df['dif'], alpha=0.6,␣
↪s=60)
    r_dif, _ = pearsonr(results_df['Factor1'], results_df['dif'])
    axes[0, 1].set_xlabel('Factor 1', fontsize=11)
    axes[0, 1].set_ylabel('Difficulty', fontsize=11)
    axes[0, 1].set_title(f'Factor 1 vs Difficulty (r={r_dif:.3f})', fontsize=12)
    axes[0, 1].grid(True, alpha=0.3)

    axes[1, 0].scatter(results_df['Factor1'], results_df['dis'], alpha=0.6,␣
↪s=60, color='orange')
    r_dis, _ = pearsonr(results_df['Factor1'], results_df['dis'])
    axes[1, 0].set_xlabel('Factor 1', fontsize=11)
    axes[1, 0].set_ylabel('Discrimination', fontsize=11)
    axes[1, 0].set_title(f'Factor 1 vs Discrimination (r={r_dis:.3f})',␣
↪fontsize=12)
    axes[1, 0].grid(True, alpha=0.3)

    # Similarity matrix heatmap
    im = axes[1, 1].imshow(sim_matrix, cmap='viridis', aspect='auto')
    axes[1, 1].set_title('Cosine Similarity Matrix', fontsize=12,␣
↪fontweight='bold')
    axes[1, 1].set_xlabel('Item', fontsize=11)
    axes[1, 1].set_ylabel('Item', fontsize=11)
    plt.colorbar(im, ax=axes[1, 1])

    plt.tight_layout()
    plt.savefig('pfa_results.png', dpi=150, bbox_inches='tight')
    print("Saved: pfa_results.png")
    plt.show()

    # 9. Save results
    print("\n[9] Saving results...")
    results_df.to_csv('pfa_loadings_with_irt.csv', index=False)
    print("Saved: pfa_loadings_with_irt.csv")

    sim_df = pd.DataFrame(sim_matrix, columns=filenames, index=filenames)
```

```python
    sim_df.to_csv('similarity_matrix.csv')
    print("Saved: similarity_matrix.csv")

    print("\n" + "="*70)
    print("ANALYSIS COMPLETE!")
    print("="*70)

    return results_df

# =============================================================================
# RUN
# =============================================================================

if __name__ == "__main__":
    # Check API key
    if API_KEY == 'your-api-key-here':
        print("ERROR: Please enter your API key in the API_KEY variable")
    else:
        results = run_full_analysis()
        print("\nResults preview:")
        print(results.head())
```

```
======================================================================
PSEUDO FACTOR ANALYSIS FOR MATRIX REASONING ITEMS
======================================================================

[1] Loading images…
Found 30 images

[2] Loading IRT parameters…
Loaded 30 difficulty and 30 discrimination values

[3] Getting embeddings from OPENAI…
Processing A-1.png…
Processing A-10.png…
Processing A-11.png…
Processing A-12.png…
Processing A-13.png…
Processing A-14.png…
Processing A-15.png…
Processing A-16.png…
Processing A-17.png…
Processing A-18.png…
Processing A-19.png…
Processing A-2.png…
Processing A-20.png…
Processing A-21.png…
Processing A-22.png…
Processing A-23.png…
Processing A-24.png…
Processing A-25.png…
Processing A-26.png…
Processing A-27.png…
Processing A-28.png…
Processing A-29.png…
Processing A-3.png…
Processing A-30.png…
Processing A-4.png…
Processing A-5.png…
Processing A-6.png…
Processing A-7.png…
Processing A-8.png…
Processing A-9.png…
Embeddings shape: (30, 3072)

[4] Computing similarity matrix…

[5] Running Factor Analysis (n_factors=3)…

[6] Organizing results…

[7] Computing correlations between factors and IRT parameters…

======================================================================
CORRELATIONS: FACTORS vs IRT PARAMETERS
======================================================================

Factor1:
  vs Difficulty:      r = -0.042  (p = 0.8255)
  vs Discrimination:  r = +0.055  (p = 0.7740)

Factor2:
  vs Difficulty:      r = -0.102  (p = 0.5933)
  vs Discrimination:  r = -0.029  (p = 0.8810)

Factor3:
  vs Difficulty:      r = -0.036  (p = 0.8522)
  vs Discrimination:  r = +0.073  (p = 0.7025)

[8] Creating visualizations…
Saved: pfa_results.png
```



```
[9] Saving results…
Saved: pfa_loadings_with_irt.csv
Saved: similarity_matrix.csv

======================================================================
ANALYSIS COMPLETE!
======================================================================

Results preview:
   filename   Factor1   Factor2   Factor3   dif   dis
0   A-1.png -0.032143 -0.019830  0.009578  1.15  1.11
1  A-10.png -0.013488 -0.005540 -0.018043  0.94  1.10
2  A-11.png -0.015542 -0.000804 -0.023565 -1.22  1.28
3  A-12.png -0.025428  0.001947 -0.021314  1.33  1.09
4  A-13.png -0.013348  0.001396  0.003446  0.95  0.68
```

[21]:
```
# let's try predicting from CLIP for dif and dis

"""
Comprehensive prediction of IRT parameters from CLIP embeddings
Includes: Ridge, Lasso, ElasticNet, Random Forest, Gradient Boosting, XGBoost
```

```
All with proper cross-validation and feature resampling
"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score, KFold, GridSearchCV
from sklearn.linear_model import Ridge, Lasso, ElasticNet
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from scipy.stats import pearsonr
import warnings
warnings.filterwarnings('ignore')

# Optional: XGBoost (install with: pip install xgboost)
try:
    from xgboost import XGBRegressor
    HAS_XGBOOST = True
except ImportError:
    HAS_XGBOOST = False
    print("Note: XGBoost not installed. Install with: pip install xgboost")

# ============================================================================
# ASSUMES YOU ALREADY HAVE THESE FROM YOUR NOTEBOOK:
# - embeddings: np.array shape (30, 512) - CLIP embeddings
# - dif: np.array shape (30,) - difficulty parameters
# - dis: np.array shape (30,) - discrimination parameters
# ============================================================================

# If running standalone, uncomment and load:
# embeddings = np.load('clip_embeddings.npy')
# irt_df = pd.read_csv('form1.csv')
# dif = irt_df['dif'].values
# dis = irt_df['dis'].values

def run_comprehensive_regression(X, y_dif, y_dis, target_name='dif'):
    """
    Run multiple regression models with cross-validation

    Parameters:
    -----------
    X : array-like, shape (n_items, n_features)
        CLIP embeddings (typically 30 x 512)
    y_dif : array-like, shape (n_items,)
        Difficulty parameters
    y_dis : array-like, shape (n_items,)
```

```
        Discrimination parameters
    target_name : str
        Which target to predict ('dif' or 'dis')
    """

    y = y_dif if target_name == 'dif' else y_dis
    n_items, n_features = X.shape

    print("="*80)
    print(f"PREDICTING {target_name.upper()} FROM CLIP EMBEDDINGS")
    print("="*80)
    print(f"Items: {n_items}, Features: {n_features}")
    print(f"Target range: [{y.min():.2f}, {y.max():.2f}]")
    print()

    # Cross-validation setup
    # With 30 items, use 5-fold CV (6 items per fold)
    cv = KFold(n_splits=5, shuffle=True, random_state=42)

    # Store results
    results = []

    # ========================================================================
    # 1. RIDGE REGRESSION (L2 regularization)
    # ========================================================================
    print("[1/6] Ridge Regression (L2 penalty)...")

    ridge_pipeline = Pipeline([
        ('scaler', StandardScaler()),
        ('ridge', Ridge())
    ])

    ridge_params = {
        'ridge__alpha': [0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]
    }

    ridge_grid = GridSearchCV(
        ridge_pipeline, ridge_params,
        cv=cv, scoring='r2', n_jobs=-1
    )
    ridge_grid.fit(X, y)

    ridge_cv_scores = cross_val_score(
        ridge_grid.best_estimator_, X, y, cv=cv, scoring='r2'
    )

    results.append({
        'Model': 'Ridge',
        'Best_Params': f"α={ridge_grid.best_params_['ridge__alpha']}",
        'CV_R²_mean': ridge_cv_scores.mean(),
        'CV_R²_std': ridge_cv_scores.std(),
        'Best_Model': ridge_grid.best_estimator_
    })

    print(f"  Best α: {ridge_grid.best_params_['ridge__alpha']}")
    print(f"  CV R²: {ridge_cv_scores.mean():.3f} ± {ridge_cv_scores.std():.
↪3f}")
    print()

    # ============================================================================
    # 2. LASSO REGRESSION (L1 regularization, feature selection)
    # ============================================================================
    print("[2/6] Lasso Regression (L1 penalty, sparse)...")

    lasso_pipeline = Pipeline([
        ('scaler', StandardScaler()),
        ('lasso', Lasso(max_iter=5000))
    ])

    lasso_params = {
        'lasso__alpha': [0.001, 0.01, 0.1, 1.0, 10.0]
    }

    lasso_grid = GridSearchCV(
        lasso_pipeline, lasso_params,
        cv=cv, scoring='r2', n_jobs=-1
    )
    lasso_grid.fit(X, y)

    lasso_cv_scores = cross_val_score(
        lasso_grid.best_estimator_, X, y, cv=cv, scoring='r2'
    )

    # Count non-zero coefficients (feature selection)
    lasso_grid.best_estimator_.fit(X, y)
    n_selected = np.sum(lasso_grid.best_estimator_.named_steps['lasso'].coef_ !
↪= 0)

    results.append({
        'Model': 'Lasso',
        'Best_Params': f"α={lasso_grid.best_params_['lasso__alpha']},␣
↪{n_selected} features",
        'CV_R²_mean': lasso_cv_scores.mean(),
        'CV_R²_std': lasso_cv_scores.std(),
        'Best_Model': lasso_grid.best_estimator_
    })

    print(f"  Best  : {lasso_grid.best_params_['lasso__alpha']}")
    print(f"  Selected features: {n_selected}/{n_features}")
    print(f"  CV R²: {lasso_cv_scores.mean():.3f} ± {lasso_cv_scores.std():.
↪3f}")
    print()

    # ========================================================================
    # 3. ELASTIC NET (L1 + L2 regularization)
    # ========================================================================
    print("[3/6] ElasticNet (L1 + L2 penalty)...")

    enet_pipeline = Pipeline([
        ('scaler', StandardScaler()),
        ('enet', ElasticNet(max_iter=5000))
    ])

    enet_params = {
        'enet__alpha': [0.01, 0.1, 1.0, 10.0],
        'enet__l1_ratio': [0.1, 0.5, 0.9]  # 0=Ridge, 1=Lasso
    }

    enet_grid = GridSearchCV(
        enet_pipeline, enet_params,
        cv=cv, scoring='r2', n_jobs=-1
    )
    enet_grid.fit(X, y)

    enet_cv_scores = cross_val_score(
        enet_grid.best_estimator_, X, y, cv=cv, scoring='r2'
    )

    results.append({
        'Model': 'ElasticNet',
        'Best_Params': f" ={enet_grid.best_params_['enet__alpha']},␣
↪l1={enet_grid.best_params_['enet__l1_ratio']}",
        'CV_R²_mean': enet_cv_scores.mean(),
        'CV_R²_std': enet_cv_scores.std(),
        'Best_Model': enet_grid.best_estimator_
    })

    print(f"  Best  : {enet_grid.best_params_['enet__alpha']}")
    print(f"  Best l1_ratio: {enet_grid.best_params_['enet__l1_ratio']}")
    print(f"  CV R²: {enet_cv_scores.mean():.3f} ± {enet_cv_scores.std():.3f}")
    print()
```

```
# ==========================================================================
# 4. RANDOM FOREST (Bootstrap aggregating with feature resampling)
# ==========================================================================
print("[4/6] Random Forest (Bootstrap + Feature Resampling)...")

# Random Forest naturally does bootstrap resampling (bootstrap=True by␣
↪default)
# and feature resampling (max_features < n_features)

rf_params = {
    'n_estimators': [100, 200, 500],
    'max_depth': [3, 5, 7, None],
    'min_samples_split': [2, 5, 10],
    'max_features': ['sqrt', 'log2', 0.3]  # Feature resampling
}

rf = RandomForestRegressor(random_state=42, n_jobs=-1)
rf_grid = GridSearchCV(
    rf, rf_params,
    cv=cv, scoring='r2', n_jobs=-1
)
rf_grid.fit(X, y)

rf_cv_scores = cross_val_score(
    rf_grid.best_estimator_, X, y, cv=cv, scoring='r2'
)

results.append({
    'Model': 'RandomForest',
    'Best_Params': f"trees={rf_grid.best_params_['n_estimators']},␣
↪depth={rf_grid.best_params_['max_depth']}",
    'CV_R²_mean': rf_cv_scores.mean(),
    'CV_R²_std': rf_cv_scores.std(),
    'Best_Model': rf_grid.best_estimator_
})

print(f"  Best n_estimators: {rf_grid.best_params_['n_estimators']}")
print(f"  Best max_depth: {rf_grid.best_params_['max_depth']}")
print(f"  Best max_features: {rf_grid.best_params_['max_features']}")
print(f"  CV R²: {rf_cv_scores.mean():.3f} ± {rf_cv_scores.std():.3f}")
print()

# ==========================================================================
# 5. GRADIENT BOOSTING (Sequential boosting)
# ==========================================================================
print("[5/6] Gradient Boosting (Sequential Boosting)...")
```

```
gb_params = {
    'n_estimators': [100, 200, 500],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.8, 1.0]  # Stochastic boosting (resampling)
}

gb = GradientBoostingRegressor(random_state=42)
gb_grid = GridSearchCV(
    gb, gb_params,
    cv=cv, scoring='r2', n_jobs=-1
)
gb_grid.fit(X, y)

gb_cv_scores = cross_val_score(
    gb_grid.best_estimator_, X, y, cv=cv, scoring='r2'
)

results.append({
    'Model': 'GradientBoosting',
    'Best_Params': f"trees={gb_grid.best_params_['n_estimators']},␣
↪lr={gb_grid.best_params_['learning_rate']}",
    'CV_R²_mean': gb_cv_scores.mean(),
    'CV_R²_std': gb_cv_scores.std(),
    'Best_Model': gb_grid.best_estimator_
})

print(f"  Best n_estimators: {gb_grid.best_params_['n_estimators']}")
print(f"  Best learning_rate: {gb_grid.best_params_['learning_rate']}")
print(f"  Best subsample: {gb_grid.best_params_['subsample']}")
print(f"  CV R²: {gb_cv_scores.mean():.3f} ± {gb_cv_scores.std():.3f}")
print()

# ==========================================================================
# 6. XGBOOST (Advanced boosting with regularization)
# ==========================================================================
if HAS_XGBOOST:
    print("[6/6] XGBoost (Regularized Boosting)...")

    xgb_params = {
        'n_estimators': [100, 200, 500],
        'max_depth': [3, 5, 7],
        'learning_rate': [0.01, 0.1, 0.2],
        'subsample': [0.8, 1.0],
        'colsample_bytree': [0.8, 1.0],  # Feature resampling per tree
        'reg_alpha': [0, 0.1, 1.0],  # L1 regularization
```

```
            'reg_lambda': [1.0, 10.0]  # L2 regularization
        }

        xgb = XGBRegressor(random_state=42, n_jobs=-1)
        xgb_grid = GridSearchCV(
            xgb, xgb_params,
            cv=cv, scoring='r2', n_jobs=-1
        )
        xgb_grid.fit(X, y)

        xgb_cv_scores = cross_val_score(
            xgb_grid.best_estimator_, X, y, cv=cv, scoring='r2'
        )

        results.append({
            'Model': 'XGBoost',
            'Best_Params': f"trees={xgb_grid.best_params_['n_estimators']},␣
↪lr={xgb_grid.best_params_['learning_rate']}",
            'CV_R²_mean': xgb_cv_scores.mean(),
            'CV_R²_std': xgb_cv_scores.std(),
            'Best_Model': xgb_grid.best_estimator_
        })

        print(f"  Best n_estimators: {xgb_grid.best_params_['n_estimators']}")
        print(f"  Best learning_rate: {xgb_grid.best_params_['learning_rate']}")
        print(f"  CV R²: {xgb_cv_scores.mean():.3f} ± {xgb_cv_scores.std():.
↪3f}")
        print()
    else:
        print("[6/6] XGBoost skipped (not installed)")
        print()

    # ==========================================================================
    # SUMMARY
    # ==========================================================================
    results_df = pd.DataFrame(results)
    results_df = results_df.sort_values('CV_R²_mean', ascending=False)

    print("="*80)
    print(f"RESULTS SUMMARY - Predicting {target_name.upper()}")
    print("="*80)
    print(results_df[['Model', 'CV_R²_mean', 'CV_R²_std', 'Best_Params']].
↪to_string(index=False))
    print()

    best_model_name = results_df.iloc[0]['Model']
    best_r2 = results_df.iloc[0]['CV_R²_mean']
    best_model = results_df.iloc[0]['Best_Model']

    print(f"  BEST MODEL: {best_model_name} (CV R² = {best_r2:.3f})")
    print()

    # ==========================================================================
    # VISUALIZATION
    # ==========================================================================
    fig, axes = plt.subplots(1, 2, figsize=(14, 5))

    # Bar plot of CV R²
    ax = axes[0]
    models = results_df['Model'].values
    r2_means = results_df['CV_R²_mean'].values
    r2_stds = results_df['CV_R²_std'].values

    colors = ['#1f77b4' if m != best_model_name else '#ff7f0e' for m in models]
    bars = ax.barh(models, r2_means, xerr=r2_stds, color=colors, alpha=0.8)
    ax.set_xlabel('Cross-Validated R²', fontsize=12)
    ax.set_title(f'Model Comparison - Predicting {target_name.upper()}',␣
↪fontsize=13, fontweight='bold')
    ax.axvline(0, color='black', linewidth=0.8)
    ax.grid(axis='x', alpha=0.3)

    # Predicted vs Actual for best model
    ax = axes[1]
    best_model.fit(X, y)
    y_pred = best_model.predict(X)

    ax.scatter(y, y_pred, alpha=0.6, s=80, edgecolors='black', linewidth=0.5)

    # Perfect prediction line
    min_val, max_val = y.min(), y.max()
    ax.plot([min_val, max_val], [min_val, max_val], 'r--', linewidth=2,␣
↪label='Perfect prediction')

    # Correlation
    r, p = pearsonr(y, y_pred)
    ax.text(0.05, 0.95, f'r = {r:.3f}\nR² = {best_r2:.3f}',
            transform=ax.transAxes, fontsize=11,
            verticalalignment='top', bbox=dict(boxstyle='round',␣
↪facecolor='wheat', alpha=0.5))

    ax.set_xlabel(f'Actual {target_name.upper()}', fontsize=12)
    ax.set_ylabel(f'Predicted {target_name.upper()}', fontsize=12)
    ax.set_title(f'{best_model_name} - Predicted vs Actual', fontsize=13,␣
↪fontweight='bold')
```

```
    ax.legend()
    ax.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.savefig(f'regression_results_{target_name}.png', dpi=150,␣
↪bbox_inches='tight')
    print(f"Saved: regression_results_{target_name}.png")
    plt.show()

    return results_df, best_model

# ============================================================================
# RUN ANALYSIS
# ============================================================================

if __name__ == "__main__":
    # Load/check required data
    print("="*80)
    print("LOADING DATA")
    print("="*80)

    # Check if embeddings exist
    try:
        print(f"  Embeddings found: {embeddings.shape}")
    except NameError:
        print("  ERROR: 'embeddings' not defined")
        print("  Re-run your CLIP encoding cell first!")
        import sys
        sys.exit(1)

    # Load IRT parameters from CSV
    try:
        import pandas as pd
        form1 = pd.read_csv('form1.csv')
        dif = form1['dif'].values
        dis = form1['dis'].values
        print(f"  Loaded IRT parameters: {len(dif)} items")
        print(f"  Difficulty range: [{dif.min():.2f}, {dif.max():.2f}]")
        print(f"  Discrimination range: [{dis.min():.2f}, {dis.max():.2f}]")
    except FileNotFoundError:
        print("  ERROR: 'form1.csv' not found")
        print("  Make sure form1.csv is in the current directory")
        import sys
        sys.exit(1)
    except Exception as e:
        print(f"  ERROR loading IRT parameters: {e}")
        import sys
        sys.exit(1)

    print()

    # Predict difficulty
    print("\n" + "="*80)
    print("PART 1: PREDICTING DIFFICULTY")
    print("="*80 + "\n")
    results_dif, best_model_dif = run_comprehensive_regression(
        embeddings, dif, dis, target_name='dif'
    )

    # Predict discrimination
    print("\n" + "="*80)
    print("PART 2: PREDICTING DISCRIMINATION")
    print("="*80 + "\n")
    results_dis, best_model_dis = run_comprehensive_regression(
        embeddings, dif, dis, target_name='dis'
    )

    # Final summary
    print("\n" + "="*80)
    print("FINAL SUMMARY")
    print("="*80)
    print(f"\nBest for DIFFICULTY: {results_dif.iloc[0]['Model']} "
          f"(CV R² = {results_dif.iloc[0]['CV_R²_mean']:.3f})")
    print(f"Best for DISCRIMINATION: {results_dis.iloc[0]['Model']} "
          f"(CV R² = {results_dis.iloc[0]['CV_R²_mean']:.3f})")
    print()
```

```
Note: XGBoost not installed. Install with: pip install xgboost
================================================================================
LOADING DATA
================================================================================
 Embeddings found: (30, 512)
 Loaded IRT parameters: 30 items
  Difficulty range: [-1.22, 3.98]
  Discrimination range: [0.36, 1.91]


================================================================================
PART 1: PREDICTING DIFFICULTY
================================================================================

================================================================================
PREDICTING DIF FROM CLIP EMBEDDINGS
================================================================================
Items: 30, Features: 512
```

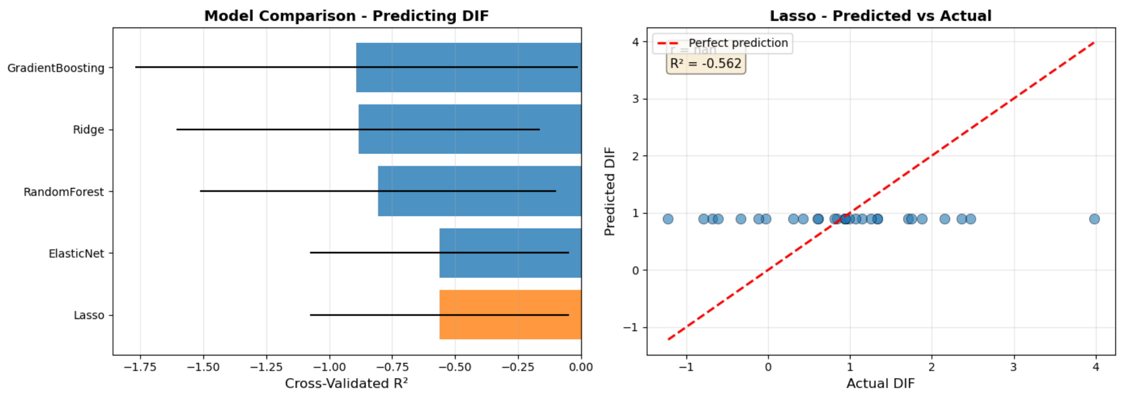
```
Target range: [-1.22, 3.98]

[1/6] Ridge Regression (L2 penalty)…

huggingface/tokenizers: The current process just got forked, after parallelism
has already been used. Disabling parallelism to avoid deadlocks…
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true |
false)
huggingface/tokenizers: The current process just got forked, after parallelism
has already been used. Disabling parallelism to avoid deadlocks…
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true |
false)
huggingface/tokenizers: The current process just got forked, after parallelism
has already been used. Disabling parallelism to avoid deadlocks…
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true |
false)
huggingface/tokenizers: The current process just got forked, after parallelism
has already been used. Disabling parallelism to avoid deadlocks…
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true |
false)
huggingface/tokenizers: The current process just got forked, after parallelism
has already been used. Disabling parallelism to avoid deadlocks…
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true |
false)
huggingface/tokenizers: The current process just got forked, after parallelism
has already been used. Disabling parallelism to avoid deadlocks…
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true |
false)
huggingface/tokenizers: The current process just got forked, after parallelism
has already been used. Disabling parallelism to avoid deadlocks…
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true |
false)
huggingface/tokenizers: The current process just got forked, after parallelism
has already been used. Disabling parallelism to avoid deadlocks…
```

```
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true |
false)
huggingface/tokenizers: The current process just got forked, after parallelism
has already been used. Disabling parallelism to avoid deadlocks…
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true |
false)
huggingface/tokenizers: The current process just got forked, after parallelism
has already been used. Disabling parallelism to avoid deadlocks…
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true |
false)
huggingface/tokenizers: The current process just got forked, after parallelism
has already been used. Disabling parallelism to avoid deadlocks…
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true |
false)
```

```
  Best  : 1000.0
  CV R²: -0.884 ± 0.721

[2/6] Lasso Regression (L1 penalty, sparse)…
  Best  : 1.0
  Selected features: 0/512
  CV R²: -0.562 ± 0.514

[3/6] ElasticNet (L1 + L2 penalty)…
```

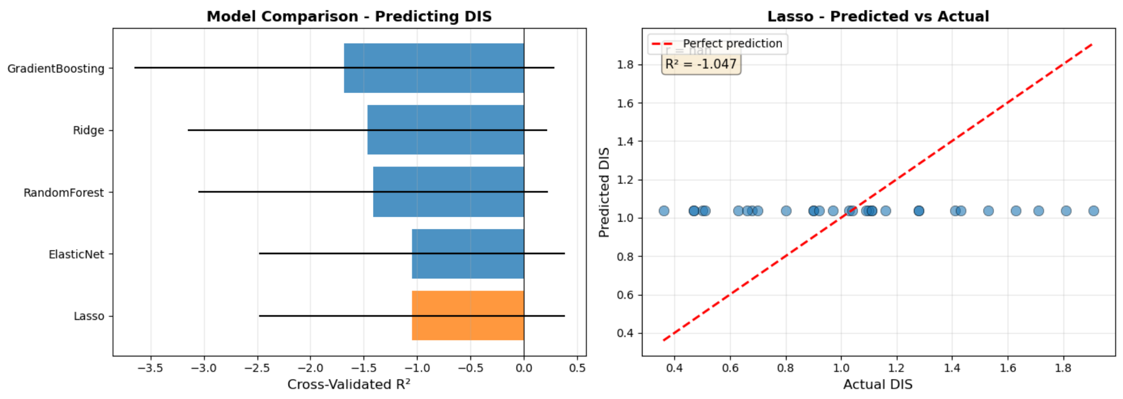
```
/opt/anaconda3/lib/python3.11/site-
packages/sklearn/linear_model/_coordinate_descent.py:631: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 4.330e-03, tolerance: 3.115e-03
  model = cd_fast.enet_coordinate_descent(
/opt/anaconda3/lib/python3.11/site-
packages/sklearn/linear_model/_coordinate_descent.py:631: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 2.534e-03, tolerance: 1.955e-03
  model = cd_fast.enet_coordinate_descent(
/opt/anaconda3/lib/python3.11/site-
packages/sklearn/linear_model/_coordinate_descent.py:631: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
```

```
gap: 3.294e-03, tolerance: 3.115e-03
  model = cd_fast.enet_coordinate_descent(
/opt/anaconda3/lib/python3.11/site-
packages/sklearn/linear_model/_coordinate_descent.py:631: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 1.011e-02, tolerance: 3.115e-03
  model = cd_fast.enet_coordinate_descent(
/opt/anaconda3/lib/python3.11/site-
packages/sklearn/linear_model/_coordinate_descent.py:631: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 4.754e-03, tolerance: 2.704e-03
  model = cd_fast.enet_coordinate_descent(
/opt/anaconda3/lib/python3.11/site-
packages/sklearn/linear_model/_coordinate_descent.py:631: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 3.401e-03, tolerance: 1.955e-03
  model = cd_fast.enet_coordinate_descent(
/opt/anaconda3/lib/python3.11/site-
packages/sklearn/linear_model/_coordinate_descent.py:631: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 1.565e-02, tolerance: 3.016e-03
  model = cd_fast.enet_coordinate_descent(

  Best  : 1.0
  Best l1_ratio: 0.9
  CV R²: -0.562 ± 0.514

[4/6] Random Forest (Bootstrap + Feature Resampling)…
  Best n_estimators: 500
  Best max_depth: 5
  Best max_features: log2
  CV R²: -0.806 ± 0.706

[5/6] Gradient Boosting (Sequential Boosting)…
  Best n_estimators: 100
  Best learning_rate: 0.01
  Best subsample: 0.8
  CV R²: -0.892 ± 0.878

[6/6] XGBoost skipped (not installed)

================================================================================
RESULTS SUMMARY - Predicting DIF
================================================================================
           Model  CV_R²_mean  CV_R²_std        Best_Params
           Lasso   -0.561683   0.514341  =1.0, 0 features
      ElasticNet   -0.561683   0.514341       =1.0, l1=0.9
    RandomForest   -0.806240   0.705928 trees=500, depth=5
           Ridge   -0.884478   0.720942            =1000.0
GradientBoosting   -0.892150   0.878359 trees=100, lr=0.01

 BEST MODEL: Lasso (CV R² = -0.562)

Saved: regression_results_dif.png
```



```
================================================================================
PART 2: PREDICTING DISCRIMINATION
================================================================================

================================================================================
PREDICTING DIS FROM CLIP EMBEDDINGS
================================================================================
Items: 30, Features: 512
Target range: [0.36, 1.91]

[1/6] Ridge Regression (L2 penalty)…
  Best  : 1000.0
  CV R²: -1.466 ± 1.687

[2/6] Lasso Regression (L1 penalty, sparse)…
  Best  : 1.0
  Selected features: 0/512
  CV R²: -1.047 ± 1.431

[3/6] ElasticNet (L1 + L2 penalty)…

/opt/anaconda3/lib/python3.11/site-
```

```
packages/sklearn/linear_model/_coordinate_descent.py:631: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 5.250e-04, tolerance: 3.660e-04
  model = cd_fast.enet_coordinate_descent(

  Best  : 1.0
  Best l1_ratio: 0.5
  CV R²: -1.047 ± 1.431

[4/6] Random Forest (Bootstrap + Feature Resampling)…
  Best n_estimators: 100
  Best max_depth: 3
  Best max_features: log2
  CV R²: -1.414 ± 1.640

[5/6] Gradient Boosting (Sequential Boosting)…
  Best n_estimators: 100
  Best learning_rate: 0.01
  Best subsample: 0.8
  CV R²: -1.682 ± 1.971

[6/6] XGBoost skipped (not installed)

================================================================================
RESULTS SUMMARY - Predicting DIS
================================================================================
           Model  CV_R²_mean  CV_R²_std        Best_Params
           Lasso   -1.046656   1.431416  =1.0, 0 features
      ElasticNet   -1.046656   1.431416       =1.0, l1=0.5
    RandomForest   -1.413974   1.640421 trees=100, depth=3
           Ridge   -1.466376   1.687025            =1000.0
GradientBoosting   -1.682395   1.970836 trees=100, lr=0.01

 BEST MODEL: Lasso (CV R² = -1.047)

Saved: regression_results_dis.png
```



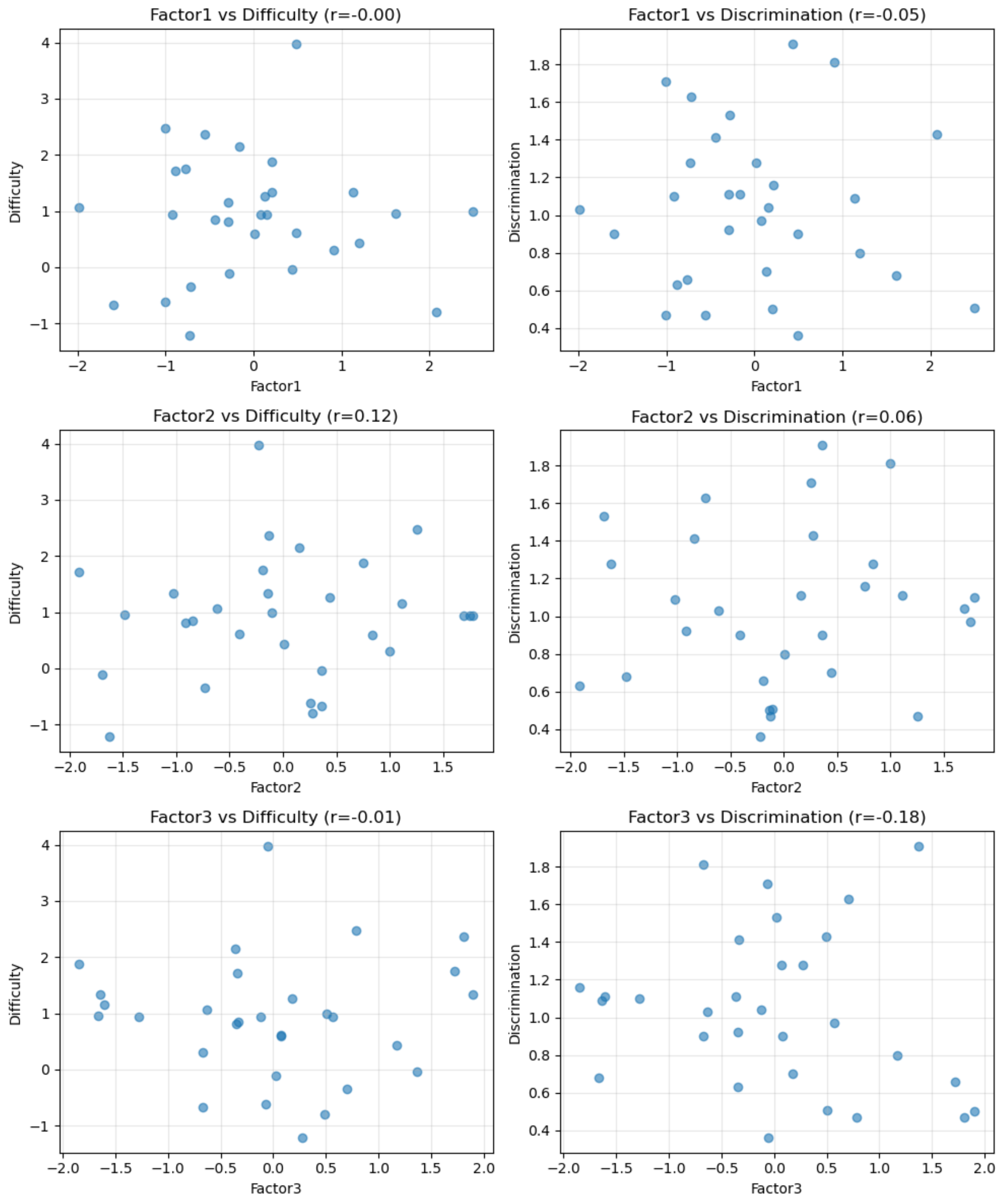
```
================================================================================
FINAL SUMMARY
================================================================================

Best for DIFFICULTY: Lasso (CV R² = -0.562)
Best for DISCRIMINATION: Lasso (CV R² = -1.047)
```

[27]:
```python
# ===== TRY 1: DINOv2 Embeddings + Factor Analysis (FIXED) =====
import torch
import numpy as np
import pandas as pd
from PIL import Image
from transformers import AutoImageProcessor, AutoModel
from factor_analyzer import FactorAnalyzer
from factor_analyzer.factor_analyzer import calculate_bartlett_sphericity,␣
 ↪calculate_kmo
import matplotlib.pyplot as plt
from scipy.stats import pearsonr

print("Loading DINOv2 model...")
processor = AutoImageProcessor.from_pretrained('facebook/dinov2-base')
model = AutoModel.from_pretrained('facebook/dinov2-base')
model.eval()

# Extract embeddings
print("Extracting DINOv2 embeddings...")
dino_embeddings = []
for img_path in image_paths:  # Assumes you have image_paths list
    img = Image.open(img_path).convert('RGB')
    inputs = processor(images=img, return_tensors="pt")
    with torch.no_grad():
        outputs = model(**inputs)
        # Use [CLS] token (first token)
        embedding = outputs.last_hidden_state[:, 0, :].squeeze().numpy()
    dino_embeddings.append(embedding)

dino_embeddings = np.array(dino_embeddings)
print(f"DINOv2 embeddings shape: {dino_embeddings.shape}")

# Standardize
from sklearn.preprocessing import StandardScaler
dino_scaled = StandardScaler().fit_transform(dino_embeddings)

# Check factorability
print("\n--- Factorability Tests ---")
chi_square, p_value = calculate_bartlett_sphericity(dino_scaled)
print(f"Bartlett's test:  ²={chi_square:.2f}, p={p_value:.4f}")
kmo_all, kmo_model = calculate_kmo(dino_scaled)
print(f"KMO: {kmo_model:.3f}")

# Factor Analysis with 3 factors
print("\n--- Factor Analysis (3 factors, ML) ---")
fa = FactorAnalyzer(n_factors=3, rotation='varimax', method='ml')
fa.fit(dino_scaled)

# Get loadings - THIS IS THE FIX
loadings = fa.loadings_  # This is (768, 3) - features x factors
# We need (30, 3) - items x factors, so we get factor scores instead
factor_scores = fa.transform(dino_scaled)  # This gives (30, 3)

loadings_df = pd.DataFrame(
    factor_scores,  # CHANGED: Use factor scores, not loadings
    columns=[f'Factor{i+1}' for i in range(3)],
    index=[f"item_{i+1}" for i in range(len(dino_embeddings))]
)

# Add IRT parameters
loadings_df['dif'] = form1['dif'].values  # CHANGED: Use form1
loadings_df['dis'] = form1['dis'].values

print("\nFactor scores (first 10 items):")
print(loadings_df.head(10))

# Correlate factors with IRT parameters
print("\n--- Correlations with IRT parameters ---")
for factor in ['Factor1', 'Factor2', 'Factor3']:
    r_dif, p_dif = pearsonr(loadings_df[factor], loadings_df['dif'])
    r_dis, p_dis = pearsonr(loadings_df[factor], loadings_df['dis'])
```

```python
    print(f"{factor}:")
    print(f"  r(factor, dif) = {r_dif:.3f}, p = {p_dif:.4f}")
    print(f"  r(factor, dis) = {r_dis:.3f}, p = {p_dis:.4f}")

# Visualize
fig, axes = plt.subplots(3, 2, figsize=(10, 12))
for i, factor in enumerate(['Factor1', 'Factor2', 'Factor3']):
    # Factor vs difficulty
    axes[i, 0].scatter(loadings_df[factor], loadings_df['dif'], alpha=0.6)
    axes[i, 0].set_xlabel(factor)
    axes[i, 0].set_ylabel('Difficulty')
    r_dif, _ = pearsonr(loadings_df[factor], loadings_df['dif'])
    axes[i, 0].set_title(f'{factor} vs Difficulty (r={r_dif:.2f})')
    axes[i, 0].grid(True, alpha=0.3)

    # Factor vs discrimination
    axes[i, 1].scatter(loadings_df[factor], loadings_df['dis'], alpha=0.6)
    axes[i, 1].set_xlabel(factor)
    axes[i, 1].set_ylabel('Discrimination')
    r_dis, _ = pearsonr(loadings_df[factor], loadings_df['dis'])
    axes[i, 1].set_title(f'{factor} vs Discrimination (r={r_dis:.2f})')
    axes[i, 1].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

print("\n  DINOv2 + FA complete!")
```

```
Loading DINOv2 model…
Extracting DINOv2 embeddings…
DINOv2 embeddings shape: (30, 768)

--- Factorability Tests ---
Bartlett's test:  ²=-inf, p=1.0000
KMO: nan

--- Factor Analysis (3 factors, ML) ---

Factor scores (first 10 items):
          Factor1   Factor2   Factor3   dif   dis
item_1  -0.288610  1.114952 -1.603965  1.15  1.11
item_2  -0.918159  1.782717 -1.276086  0.94  1.10
item_3  -0.731592 -1.623617  0.274825 -1.22  1.28
item_4   1.132702 -1.026272 -1.641842  1.33  1.09
item_5   1.613172 -1.482189 -1.665563  0.95  0.68
item_6  -1.982107 -0.617042 -0.632650  1.07  1.03
item_7   0.486858 -0.408625  0.075851  0.61  0.90
item_8   0.015957  0.838686  0.071655  0.60  1.28
```

```
item_9   0.212099  0.754642 -1.844838  1.88  1.16
item_10  0.436989  0.363259  1.370189 -0.03  1.91

--- Correlations with IRT parameters ---
Factor1:
  r(factor, dif) = -0.003, p = 0.9857
  r(factor, dis) = -0.049, p = 0.7968
Factor2:
  r(factor, dif) = 0.123, p = 0.5169
  r(factor, dis) = 0.058, p = 0.7627
Factor3:
  r(factor, dif) = -0.011, p = 0.9557
  r(factor, dis) = -0.177, p = 0.3488
```



```
DINOv2 + FA complete!
```

[ ]: