# Modern Dimensionality Reduction in Orange and Python

Greg Baker

Week 10 — 14th October 2025

## Setting the stage

- Thanks for sticking with the text mining whirlwind — today we pivot to pictures.
- Goal: how to squish big, messy feature spaces into 2D maps we can actually eyeball.
- We'll lean on Orange so you can recreate the demo in the lab and mirror it in Python.
- Three tools on the menu: first encounter with PCA, then t-SNE and UMAP as the modern crowd-pleasers.

## Diagnosing high-dimensional pain

- Curse of dimensionality: distance metrics and density estimates get wobbly as features explode.
- Visual inspection of 10K-dimensional vectors? Not happening without projection.
- Downstream perks: faster clustering, sanity-checking labels, spotting outliers before they bite.
- Orange gives us a playground: drag, drop, tweak parameters, and instantly see the scatterplot.

## What makes a good projection?

- Preserve neighbourhoods: points close in high dimensions should stay nearby.
- Maintain global structure when possible (clusters, gradients).
- Be reproducible enough for storytelling — "trust but verify" with multiple runs.
- Keep computation reasonable so you can iterate live in class.

## What is PCA, anyway?

- Principal Component Analysis (PCA) rotates the coordinate system to line up with directions of maximum variance.
- Each principal component is a weighted combination of the original features — think "synthetic features".
- Components are orthogonal, so information is not double-counted; later components explain diminishing variance.
- We'll use PCA as a baseline projection and a warm-up for nonlinear techniques.

## Computing PCA in practice

- Standard workflow:
  1. Standardise features so variance comparisons are fair.
  2. Compute the covariance matrix and find its eigenvectors.
  3. Project the data onto the leading eigenvectors to get lower-dimensional coordinates.
- Scree plots (variance explained per component) help justify how many components to keep.
- Linear method: great for spotting broad trends, but it cannot unwrap curved manifolds.

## Orange workflow and Python mirror: PCA

### Orange steps

1. **File** → load sms_spam.csv.
2. **Corpus** → **Bag of Words** → **TF-IDF**.
3. Add **PCA** and connect to **Scatter Plot** and **Scree Plot**.
4. Colour by the spam/ham label; hover to inspect messages.

```
import pandas as pd
from
    sklearn.feature_extraction.
    import TfidfVectorizer
from sklearn.decomposition
    import PCA

data =
    pd.read_csv("sms_spam.csv")
tfidf =
    TfidfVectorizer(stop_words=
X =
    tfidf.fit_transform(data["m

pca = PCA(n_components=2,
    random_state=0)
coords =
    pca.fit_transform(X.toarray
```

## Interpreting the PCA view

- Use the scree plot or `explained` array to decide if two components capture enough variance.
- Clusters here often signal that linear classifiers will have an easier job.
- Overlap reminds us PCA is linear — it can flatten directions of variance but not separate curved structures.
- Treat PCA as the “control” view before trying nonlinear magic.

## t-SNE essentials

- Designed for visualising local neighbourhoods with heavy-tailed Student t-distribution.
- Key knob: **perplexity** — roughly how many neighbours to trust.
- Stochastic optimiser: each run can look different, especially on small datasets.
- Best for highlighting cluster islands, not for reading distances between islands.

## t-SNE in Orange

- Replace the PCA widget with **t-SNE**; feed the same data pipeline.
- Start with perplexity 30, 1,000 iterations; tick "**initialisation: PCA**" for stability.
- Watch the live optimisation — Orange shows points settling into clusters.
- Encourage students to run multiple seeds and compare; discuss why shapes shift yet clusters persist.

## Python workflow: t-SNE

```python
from sklearn.manifold import TSNE

tsne = TSNE(
    n_components=2,
    perplexity=30,
    init="pca",
    n_iter=1000,
    random_state=42,
    learning_rate="auto",
)
tsne_coords = tsne.fit_transform(X)
```

- Use the same `X` TF-IDF matrix so Orange and Python plots are comparable.
- Run multiple seeds (change `random_state`) to discuss stability.

## Why UMAP is the new hotness

- Balances local fidelity with more global structure via fuzzy simplicial sets.
- Much faster than t-SNE on medium-to-large datasets, so great for live demos.
- Produces embeddings you can reuse for downstream models (less random drift).
- Handles sparse input directly — perfect for text vectors without dense PCA pre-step.

## UMAP workflow in Orange

- Drag in **UMAP** (found under "Unsupervised" tools) and wire it to the same data source.
- Tweak **n neighbours** (15–50) and **min distance** to control cluster tightness.
- Colour by label or prediction to spot where the model is confused.
- Export the embedding table directly for scikit-learn if you want to hand off to Python.

## Python workflow: UMAP

```
import umap

umap_model = umap.UMAP(
    n_neighbors=15,
    min_dist=0.1,
    metric="cosine",
    random_state=42,
)
umap_coords = umap_model.fit_transform(X)
```

- UMAP handles sparse matrices directly; no need to densify `X`.
- Export the embedding via `umap_model.transform` for reuse in downstream models.

## What to discuss live

- Step through PCA → t-SNE → UMAP on the same dataset; ask students what changed.
- Highlight runtime differences — UMAP should finish first, t-SNE takes patience.
- Talk through interpretation traps: cluster size is arbitrary, axes meaningless in t-SNE/UMAP.
- Use data subsets (spam vs ham, or topic labels) to show how supervision can guide colour choices.

## Documenting the workflow

- Capture screenshots for Canvas so students can rebuild the pipeline step-by-step.
- Save the Orange workflow as `week10-dimensionality-reduction.ows` and drop it in Git.
- Encourage experimentation: swap in image embeddings, try distance metrics, explore 3D scatter.
- Reinforce that projections are exploratory — always validate insights with domain knowledge.

## Key messages to take away

- PCA gives a linear baseline; it's new to us today, so pause to unpack the mechanics.
- t-SNE shines for local structure but demands tuning and interpretive caution.
- UMAP offers speed and stability, making it ideal for Orange demos and rapid iteration.
- Next steps: bake embeddings into Assignment 2 ideas and show how they feed clustering/classification.

## Before we go

- Upload the Orange workflow and screenshots after class so students can follow along.
- Flag office hours and the forum — dimensionality reduction rabbit holes are more fun together.
- Next lecture: pulling these embeddings into evaluation pipelines.
- Thanks everyone — shout if you need help getting Orange installed before the prac.