Lecture 6

Dimension Reduction

Contents

When we use chemical features to train a machine learning model, we typically work with a large number of descriptors. Consider the molecular fingerprints discussed in the previous lecture: MACCS keys contain 166 bits, whereas the Klekota–Roth fingerprint expands to 4,860 bits. Each bit encodes a specific structural fragment or property, called a **feature** in ML, and mathematically, each feature corresponds to a **dimension** in the feature space.

This high-dimensional feature space presents three major challenges:

1. **Computational cost.** Higher dimensionality directly increases computational expense in both training and inference.
2. **Redundancy.** Many chemically derived features are correlated. For example, molecular weight depends strongly on the number of heavy atoms. Including both features introduces redundancy, which can degrade model stability and interpretability.
3. **Curse of dimensionality.** As the number of dimensions increases, the ability of a model to generalize decreases unless the amount of training data increases accordingly. Maintaining good model performance therefore requires substantially more data in high-dimensional spaces.

In this lecture, we introduce two methods to prune the chemical feature space.

1 Feature Selection

Before performing more complex transformations of the features, we can often improve learning by simply removing features that contribute little to the task.

There are two practical rules to follow:

- **Remove features with zero or near-zero variance.**
  Intuitively, if the data shows very little variance along a feature axis, that feature carries little information. Conversely, features with larger variance capture more diversity in the data and are therefore more relevant for analysis or modeling. See Fig. 1 for an example.
- **Occam's Razor.**[1]
  The number of features should be significantly smaller than the number of samples. Including too many features relative to the dataset size can lead to overfitting and unstable models.

You can use `scikit-learn` to do feature selection

```python
from sklearn.feature_selection import VarianceThreshold

# Remove features with variance below 0.01
selector = VarianceThreshold(threshold=0.01)
X_selected = selector.fit_transform(X)
```

Feature selection is a simple way to reduce redundancy, but it can be ineffective when features have different scales. PCA provides a more systematic approach by combining correlated features into principal components.

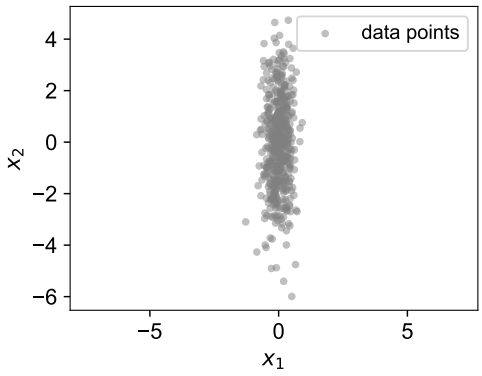

Figure 1: An example of features along which the data has small variance (x_1).

2 Principal Component Analysis (PCA)

Consider two features, x_1 and x_2; for example, molecular weight (MW) and number of rings (NoR) of a molecule. Given a dataset of molecules, we can visualize them in the $x_1 - x_2$ plane. How would the plot look if the two features were uncorrelated versus correlated?

Figure 2 illustrates this:

- **Left:** Distribution of data with uncorrelated features. Each feature individually follows a roughly Gaussian distribution, and the joint probability factorizes as $p(x_1, x_2) = p(x_1)\, p(x_2)$. The scatter forms an axis-aligned cloud with no apparent slope.
- **Right:** Distribution of data with correlated features. A clear trend emerges: as x_1 increases, x_2 tends to increase as well. The scatter forms an elongated ellipse along the diagonal, reflecting positive correlation.

[1] Also called the principle of parsimony: among competing hypotheses, the one with the fewest assumptions and simplest explanation is usually preferred.

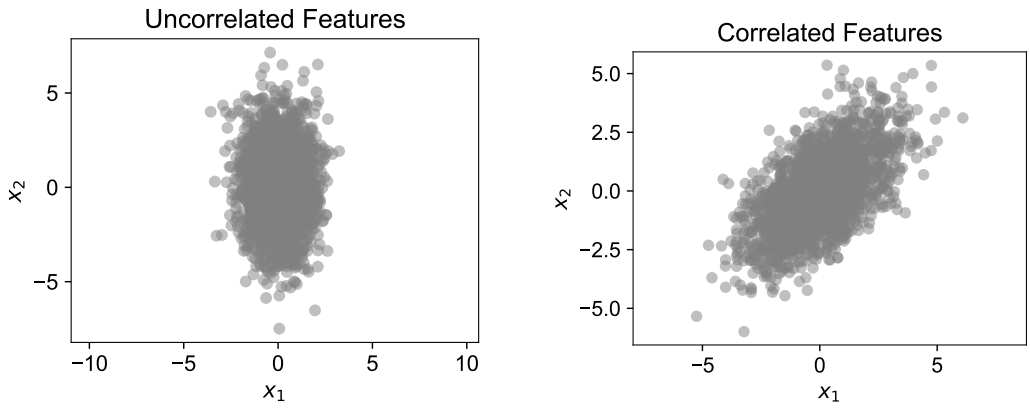

Figure 2: Data distribution with uncorrelated feature (left) and correlated features (right).

Now, if we want to pick additional features from the right distribution, which one should we choose? We use what we learned from last section: Choose the feature along which the data exhibits the most variance.

2.1 Rescaling

Before applying feature selection, the first step is to **rescale** the features. For example, molecular weight (MW) may range from 50–1000, while the number of rings (NoR) only ranges from 0–10. To ensure all features contribute equally, we standardize them by subtracting the mean and dividing by the standard deviation:

$$\text{Standardization:} \quad x_i \to \frac{x_i - \bar{x}_i}{\sigma_i}$$

This can be implemented easily in `scikit-learn`:

```python
from sklearn.preprocessing import StandardScaler

# Initialize the StandardScaler object
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)  # X is the original feature data
```

2.2 PCA Algorithm

PCA identifies the directions of maximum variance in correlated features, which correspond to the **principal components**.

The blue lines denoted by "PC1" in Fig. 3 illustrate the principal components of the previous example. While the red lines are the less "principal" ones.

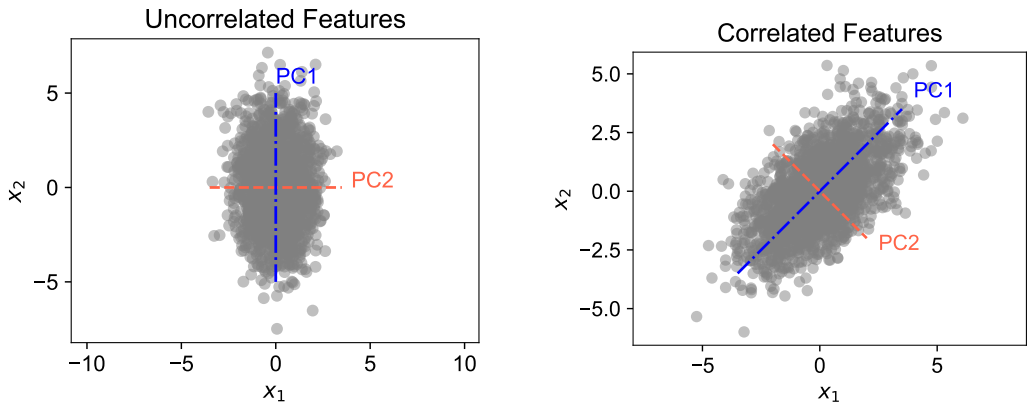

Figure 3: Data distribution with uncorrelated features (left) and correlated features (right). The blue lines show the first principal component (PC1).

Just like in a CSV file, we represent *columns* as features and *rows* as data points, forming a **feature matrix** of size $n \times p$, where n is the number of data points and p is the number of features:

$$X = \begin{array}{c} \\ \text{data}_1 \\ \text{data}_2 \\ \vdots \\ \text{data}_n \end{array} \begin{array}{c} \begin{array}{cccc} \text{feature}_1 & \text{feature}_2 & \cdots & \text{feature}_p \end{array} \\ \begin{bmatrix} X_{11} & X_{12} & \cdots & X_{1p} \\ X_{21} & X_{22} & \cdots & X_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ X_{n1} & X_{n2} & \cdots & X_{np} \end{bmatrix} \end{array}$$

- **Step 1: Construct covariance matrix.**

$$\text{Covariance matrix: } \Sigma = \frac{1}{n-1} X^T X \tag{1}$$

- **Step 2: Diagonalize Σ.** Solve for eigenvalues and eigenvectors:

$$\Sigma \mathbf{v}_k = \lambda_k \mathbf{v}_k, \tag{2}$$

  where $\mathbf{v}_k$ are eigenvectors and λ_k are eigenvalues. Since Σ is symmetric, all $\lambda_k \geq 0$.

- **Step 3: Choose principal components.** Sort eigenvalues in descending order: $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p$. The eigenvector $\mathbf{v}_1$ corresponding to λ_1 is the first principal component, $\mathbf{v}_2$ the second, and so on. Optionally, select the top m components based on a cutoff:

$$V = [\mathbf{v}_1, \ldots, \mathbf{v}_m] \tag{3}$$

- **Step 4: Project original features onto the principal components.** Each new feature $\tilde{f}_k$ is obtained by:

$$\tilde{f}_k = \mathbf{f}^T \mathbf{v}_k = \sum_{i=1}^{p} f_i v_{ki} \tag{4}$$

The transformed data matrix becomes:

$$Z = XV \tag{5}$$

where Z is of size $n \times m$.

We can now train machine learning models using Z, which is typically better conditioned. Another advantage of PCA is that we can quantify how much variance each component explains:

$$\text{Explained Variance Ratio: } r_k = \frac{\lambda_k}{\sum_{i=1}^{p} \lambda_i} \tag{6}$$

2.3 Implementation with `scikit-learn`

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Standardize features
X_scaled = StandardScaler().fit_transform(X)  # X is the original feature matrix

# Perform PCA
pca = PCA(n_components=2) # choose 2 components
Z = pca.fit_transform(X_scaled) # new feature matrix is Z
```

In the above example, we explicitly chose 2 PCs from the PCA process. We can also set a threshold on the λ_k values, e.g.

```
pca = PCA(tol=1e-3)
```

`scikit-learn` also provides other dimension reduction methods. Feel free to explore them on your own: `https://scikit-learn.org/stable/api/sklearn.decomposition.html`

3 Lab

We will see a simple example:

https://colab.research.google.com/drive/1lsFFE71fB7mLxtPQDMGXsK4noVGz0QaF?usp=sharing