Principal Component Analysis

General Principles

Principal Component Analysis (PCA) (Tipping and Bishop 1999) is a technique used to reduce the dimensionality of a dataset by transforming it into a new coordinate system where the greatest variance of the data is projected onto the first coordinates (called principal components). This method helps capture the underlying structure of high-dimensional data by identifying patterns based on variance.

In **Bayesian PCA**, uncertainty in the model parameters is explicitly taken into account by using a probabilistic framework. This allows us to not only estimate the principal components but also quantify the uncertainty around them and avoid overfitting by incorporating prior knowledge.

PCA is employed for *dimensionality reduction*, particularly in scenarios involving high-dimensional datasets, as it effectively reduces complexity while explicitly accounting for uncertainty in the underlying latent structure. This approach also plays a crucial role in *data visualization*, enabling the projection of intricate, high-dimensional data into more interpretable 2D or 3D representations. Additionally, PCA excels in *feature extraction*, where the latent variables it identifies can be repurposed as informative features for subsequent tasks, such as classification or clustering. By modeling latent variables, it further enhances the interpretability and utility of the data for a variety of analytical applications.

Considerations

Note

- In **Bayesian PCA**, we assume prior distributions for the latent variables Z and the principal component loadings W. We place Gaussian priors on both Z and W and learn their posterior distributions using the observed data X.
- **Robustness to Outliers:** Standard PCA are sensitive to outliers due to the assumption of Gaussian noise. Robust variants of **Robust Bayesian PCA** ad-

dress this by employing heavy-tailed distributions for the noise model, such as the Student's t-distribution, which reduces the influence of outliers (Archambeau, Delannay, and Verleysen 2006).

- **Automatic Dimensionality Selection:** Through techniques like Automatic Relevance Determination (ARD), Bayesian PCA can automatically determine the effective dimensionality of the latent space. Priors are placed on the relevance of each principal component, and components that are not supported by the data are effectively "switched off' (Bishop 1998).
- **Sparsity for High-Dimensional Data:** In high-dimensional settings, it is often desirable for the principal components to be influenced by only a subset of the original features, leading to more interpretable results. Sparse Bayesian PCA achieves this by placing sparsity-inducing priors (e.g., Laplacian or spike-and-slab priors) on the loading (Sigg and Buhmann 2008).

Example

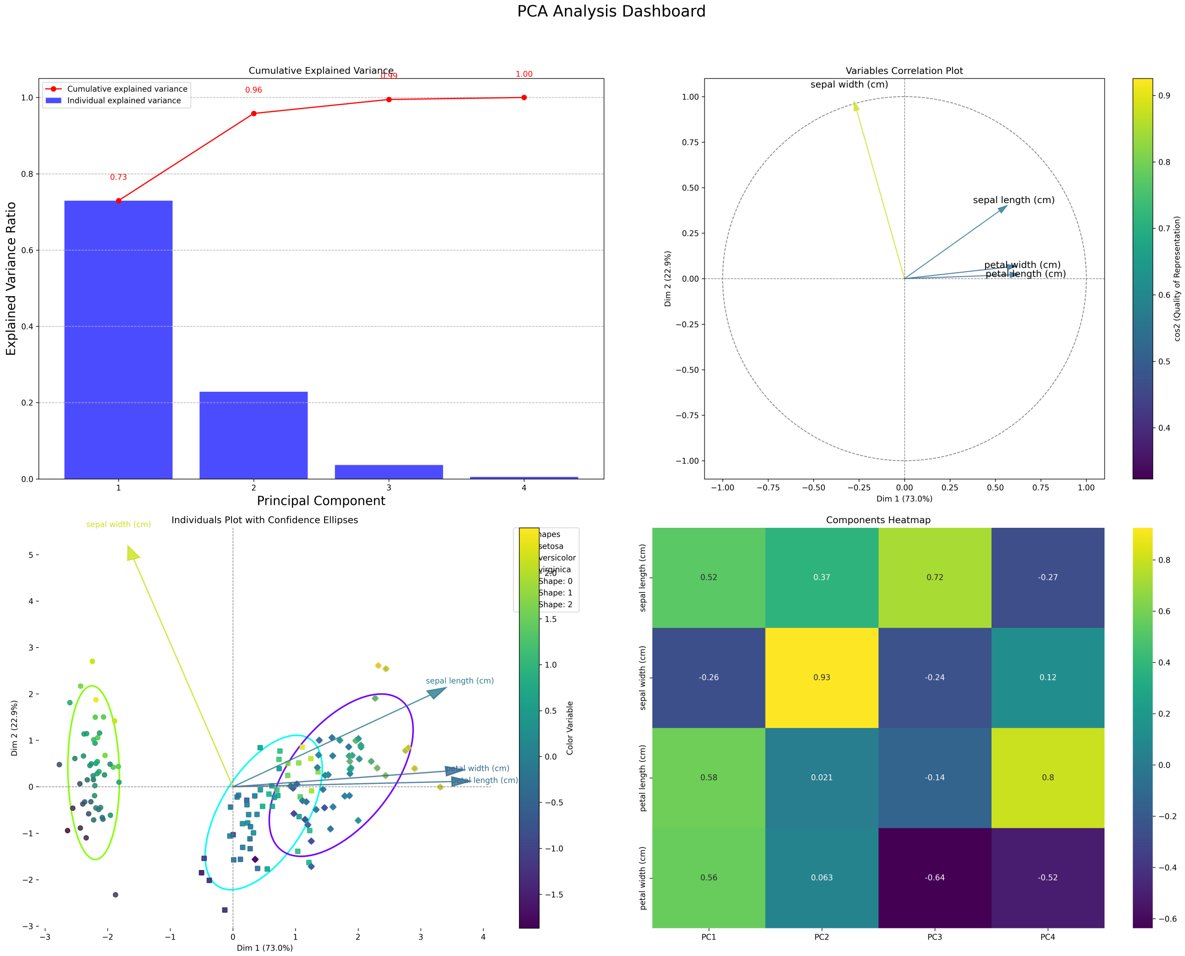
Here is an example code snippet demonstrating Bayesian PCA using BI:

Build in functions

```python
from BI import bi,jnp
m=bi()
m.data('iris.csv', sep=',') # Data is already scaled
m.data_on_model = dict(
    X=jnp.array(m.df.iloc[:,0:-2].values)
)
m.fit(m.models.pca(type="ARD"), progress_bar=False) # or robust, sparse, classic, sparse_robu
m.models.pca.plot(
    X=m.df.iloc[:,0:-2].values,
    y=m.df.iloc[:,-2].values,
    feature_names=m.df.columns[0:-2],
    target_names=m.df.iloc[:,-1].unique(),
    color_var=m.df.iloc[:,0].values,
    shape_var=m.df.iloc[:,-2].values
)
```

```
jax.local_device_count 16
```

```
/home/sosa/work/BI/BI/Models/PCA.py:417: UserWarning: color is redundantly defined by the 'co
  ax.plot(indices, cumulative, 'ro-', color=color_cumulative_line, label='Cumulative explaine
```



Standard

```
def model(x_train, data_dim, latent_dim, num_datapoints):
    # Gaussian prior for the principal component 'W'.
    w = m.dist.normal(0, 1, shape=(data_dim, latent_dim), name='w')

    # Gaussian prior on the latent variables 'Z'
    z = m.dist.normal(0, 1, shape=(latent_dim, num_datapoints), name='z')

    # Exponential prior on the noise variance 'epsilon'
    epsilon = m.dist.exponential(1, name='epsilon')

    # Likelihood
    m.dist.normal(w @ z , epsilon, obs = x_train)

m.data_on_model = dict(
    x_train=x_train_h,
    data_dim=data_dim,
    latent_dim=4,
    num_datapoints=num_datapoints
)
m.fit(model)
```

With ARD

```

from BI import bi
import jax.numpy as jnp

m = bi(platform='cpu')

def model_ARD(X, data_dim, latent_dim, num_datapoints):
    # --- Automatic Dimensionality Selection using an ARD Prior ---
    # There is one 'alpha' for each latent dimension.
    # This 'alpha' acts as a prior on the "relevance" of each principal component. A large al
    alpha = m.dist.gamma(.05, 1e-3, shape=(latent_dim,), name='alpha')

    # Gaussian prior for the principal component 'W'.
    # The precision (1 / variance) of the weights for each component is controlled by its co
    w = m.dist.normal(0, 1. / jnp.sqrt(alpha)[None, :], shape=(data_dim, latent_dim), name='w

    # Gaussian prior on the latent variables 'Z'
    z = m.dist.normal(0, 1., shape=(latent_dim, num_datapoints), name='z')

    # Noise model
    # We assume the noise is Gaussian, but we don't know its variance. So, we place a Gamma p
    precision = m.dist.gamma(1.0, 1.0, name='precision')
    # We convert the learned precision into a standard deviation for use in the likelihood.
    stddv = 1. / jnp.sqrt(precision)

    # Likelihood
    # 'X' is modeled as a linear projection of the latents (w @ z) plus Gaussian noise.
    m.dist.normal(w @ z, stddv, obs=X)

m.data_on_model = dict(
    x_train=x_train_h,
    data_dim=data_dim,
    latent_dim=4,
    num_datapoints=num_datapoints
)
m.fit(model_ARD)
m.summary()
```

Robust

```
def model_robust(x_train, data_dim, latent_dim, num_datapoints):
    """
    Robust Bayesian PCA model using a Student's t-distribution for the likelihood.
    """
    # --- Standard Priors for W and Z ---
    w = m.dist.normal(0, 1., shape=(data_dim, latent_dim), name='w')
    z = m.dist.normal(0, 1., shape=(latent_dim, num_datapoints), name='z')

    # --- Robustness to Outliers via a Heavy-Tailed Noise Model ---
    # This defines the prior on the scale (similar to standard deviation) of the noise.
    sigma = m.dist.halfcauchy(1.0, name='sigma')

    # This is a prior on the "degrees of freedom" ('nu') of the Student's t-distribution.
    # This parameter controls the "heaviness" of the tails. A small 'nu' means
    # heavier tails, making the model more robust to outliers. By learning this
    # parameter, the model can adapt its robustness to the data.
    nu = m.dist.gamma(2.0, 0.1, name='nu')

    # This is the key line for this model. The likelihood is a Student's t-distribution.
    # As your description states, this "heavy-tailed distribution... reduces the
    # influence of outliers" by treating them as more plausible events than a
    # Gaussian distribution would, thus preventing them from skewing the results.
    m.dist.studentt(df=nu, loc=w @ z, scale=sigma, obs=x_train)

m.data_on_model = dict(
    x_train=x_train_h,
    data_dim=data_dim,
    latent_dim=4,
    num_datapoints=num_datapoints
)
m.fit(model_robust)
m.summary()
```

Sparse

```
def model_sparse(x_train, data_dim, latent_dim, num_datapoints):
    """
    Sparse Bayesian PCA model using a Laplace prior on the weights (w).
    """
    # --- Sparsity for High-Dimensional Data via a Sparsity-Inducing Prior ---

    # This is the first part of a hierarchical prior (known as the Bayesian Lasso).
    # We place a prior on 'lambda_', which will control the scale of our Laplace prior.
    # This allows the model to learn the appropriate level of sparsity from the data.
    lambda_ = m.dist.gamma(1.0, 1.0, shape=(latent_dim,), name='lambda')

    # This is the key line for this model. We place a Laplace prior on the loadings 'W'.
    # As your description states, this is a "sparsity-inducing prior". The Laplace
    # distribution is sharply peaked at zero, which encourages many of the weight
    # values in 'W' to be exactly zero, leading to "more interpretable results".
    w = m.dist.laplace(0., 1. / lambda_[None, :], shape=(data_dim, latent_dim), name='w')

    # --- Standard Model Components ---

    # We place a standard Gaussian prior on the latent variables 'Z', as described
    # in the note: "We place Gaussian priors on both Z and W...".
    z = m.dist.normal(0., 1., shape=(latent_dim, num_datapoints), name='z')

    # This section defines the standard Gaussian noise model, which is separate
    # from the sparsity-inducing prior on the weights.
    precision = m.dist.gamma(1.0, 1.0, name='precision')
    stddv = 1. / jnp.sqrt(precision)

    # This is the standard likelihood, same as in the ARD model. The generative story
    # is unchanged, but the prior on 'W' now enforces the desired sparsity property.
    m.dist.normal(w @ z, stddv, obs=x_train)

m.data_on_model = dict(
    x_train=x_train_h,
    data_dim=data_dim,
    latent_dim=4,
    num_datapoints=num_datapoints
)
m.fit(model_sparse)
m.summary()
```

Sparse robust with ARD

```
def model_sparse_robust_ard(x_train, data_dim, latent_dim, num_datapoints):
    """
    A combined Sparse, Robust Bayesian PCA model with Automatic Relevance Determination (ARD

    - Sparsity: Achieved with a Laplace prior on the weights 'w'.
    - Robustness: Achieved with a Student's t-distribution for the likelihood.
    - ARD: Achieved by placing a hierarchical prior on the scale of the Laplace
      distribution, allowing entire latent dimensions to be pruned.
    """
    # --- ARD and Sparsity-Inducing Prior on w ---

    # This is the ARD component. We define a relevance parameter 'lambda_' for each
    # latent dimension. A large lambda will signal that the corresponding
    # component is not relevant and should be shrunk away.
    # The Gamma prior ensures lambda_ is positive.
    lambda_ = m.dist.gamma(1.0, 1.0, shape=(latent_dim,), name='lambda')

    # This is the Sparsity component. We use a Laplace prior for the weights 'w'.
    # The COMBINED effect happens here: the scale of the Laplace distribution is
    # controlled by the ARD parameter 'lambda_'. If a component is irrelevant
    # (large lambda_), the scale becomes small, and the Laplace prior aggressively
    # forces the weights in that column of 'w' to zero.
    # This gives us both sparsity and automatic dimensionality selection.
    w = m.dist.laplace(0., 1. / lambda_[None, :], shape=(data_dim, latent_dim), name='w')

    # --- Standard Prior for Z ---

    # The prior on the latent variables 'Z' remains a standard Gaussian.
    z = m.dist.normal(0., 1., shape=(latent_dim, num_datapoints), name='z')
    # --- Robustness to Outliers via a Heavy-Tailed Noise Model ---

    # Prior on the scale of the Student's t-distribution.
    sigma = m.dist.halfcauchy(1.0, name='sigma')

    # Prior on the degrees of freedom 'nu', which controls the robustness.
    nu = m.dist.gamma(2.0, 0.1, name='nu')

    # The likelihood is the Student's t-distribution, which makes the entire
    # model robust to outliers in the observed data 'x_train'.
    m.dist.studentt(df=nu, loc=w @ z, scale=sigma, obs=x_train)

m.data_on_model = dict(
    x_train=x_train_h,
    data_dim=data_dim,
    latent_dim=4,
    num_datapoints=num_datapoints
)
m.fit(model_sparse_robust_ard)
m.summary()
```

Mathematical Details

We assume the observed data matrix X is centered and arranged with features as rows and samples as columns, $X \in \mathbb{R}^{D \times N}$. The generative model projects the data into a lower-dimensional space with K latent variables, $K \ll D$. :

$$X_{dn} \sim \text{Normal}((WZ)_{dn}, \sigma)$$

where :

- X is the observed data matrix.
- $Z \in \mathbb{R}^{N \times K}$ is the latent variable matrix (latent features) with $K \ll D$. Z is defined by a Normal distribution with mean 0 and variance 1.
- $W \in \mathbb{R}^{D \times K}$ is the matrix of principal components (*projection matrix*). W is defined by a Normal distribution with mean 0 and variance 1.
- $\epsilon \in \mathbb{R}^{N \times D}$ is Gaussian noise matrix, assumed to be normally distributed centered at 0 as data is assumed to be centered.

The likelihood and priors are defined element-wise for $d = 1...D$, $n = 1...N$, and $k = 1...K$. for the following models:

Standard PCA

$$
\begin{aligned}
X_{dn} &\sim \text{Normal}((WZ)_{dn}, \sigma) \\
Z_{kn} &\sim \text{Normal}(0, 1) \\
W_{dk} &\sim \text{Normal}(0, 1) \\
\sigma &\sim \text{Exponential}(1)
\end{aligned}
$$

PCA with ARD

In PCA with Automatic Relevance Determination (ARD), the variance of the prior on each column of W is controlled by a unique relevance parameter α_k.

$$
\begin{aligned}
X_{dn} &\sim \text{Normal}((WZ)_{dn}, \sigma) \\
Z_{kn} &\sim \text{Normal}(0, 1) \\
W_{dk} &\sim \text{Normal}(0, \alpha_k^{-1}) \\
\alpha_k &\sim \text{Gamma}(0.05, 10^{-3})
\end{aligned}
$$

$$\sigma \sim \text{Normal}(0, \tau^{-1})$$

$$\tau \sim \text{Gamma}(1, 1)$$

Where:

- $\alpha \in \mathbb{R}^K$ is the **relevance parameter vector**. Each element α_k controls the precision (inverse variance) of the weights for the k-th principal component. If a component is not supported by the data, its corresponding α_k will become large, forcing the weights in the k-th column of W towards zero and effectively "switching off" that component.
- τ is the **noise precision** (inverse variance) of the data.

Robust PCA

In Robust PCA, the model is modified to handle outliers by using a heavy-tailed distribution for the likelihood:

$$
\begin{aligned}
X_{dn} &\sim \text{StudentT}(\nu, (WZ)_{dn}, \sigma) \\
Z_{kn} &\sim \text{Normal}(0, 1) \\
W_{dk} &\sim \text{Normal}(0, 1) \\
\nu &\sim \text{Gamma}(2, 0.1) \\
\sigma &\sim \text{HalfCauchy}(1)
\end{aligned}
$$

Where:

- ν is the **degrees of freedom** of the Student's t-distribution. It controls the "heaviness" of the tails. Small values of ν lead to greater robustness. By placing a prior on it, we allow the model to learn the appropriate level of robustness from the data.
- σ is the **scale parameter** of the Student's t-distribution, which is analogous to the standard deviation in a Normal distribution.

Sparse PCA

In Sparse PCA, a Laplace prior is placed on the weights W to induce sparsity.

$$
\begin{aligned}
X_{dn} &\sim \text{Normal}((WZ)_{dn}, sigma) \\
Z_{kn} &\sim \text{Normal}(0, 1) \\
W_{dk} &\sim \text{Laplace}(0, \lambda_k^{-1}) \\
\lambda_k &\sim \text{Gamma}(1, 1) \\
\\
\sigma &\sim \text{Normal}(0, \tau^{-1}) \\
\tau &\sim \text{Gamma}(1, 1)
\end{aligned}
$$

where:

- $\lambda \in \mathbb{R}^K$ is the **sparsity control vector**. Each element λ_k is the rate parameter (inverse scale) for the Laplace prior on the k-th column of W. A larger λ_k enforces stronger sparsity on that component.
- τ is the **noise precision** (inverse variance) of the data.

Sparse Robust PCA with ARD

This model combines all three features: a sparsity-inducing prior, a robust likelihood, and automatic relevance determination.

$$\begin{aligned}
X_{dn} &\sim \text{StudentT}(\nu, (WZ)_{dn}, \sigma) \\
Z_{kn} &\sim \text{Normal}(0, 1) \\
W_{dk} &\sim \text{Laplace}(0, \lambda_k^{-1}) \\
\lambda_k &\sim \text{Gamma}(1, 1) \\
\nu &\sim \text{Gamma}(2, 0.1) \\
\sigma &\sim \text{HalfCauchy}(1)
\end{aligned}$$

Note

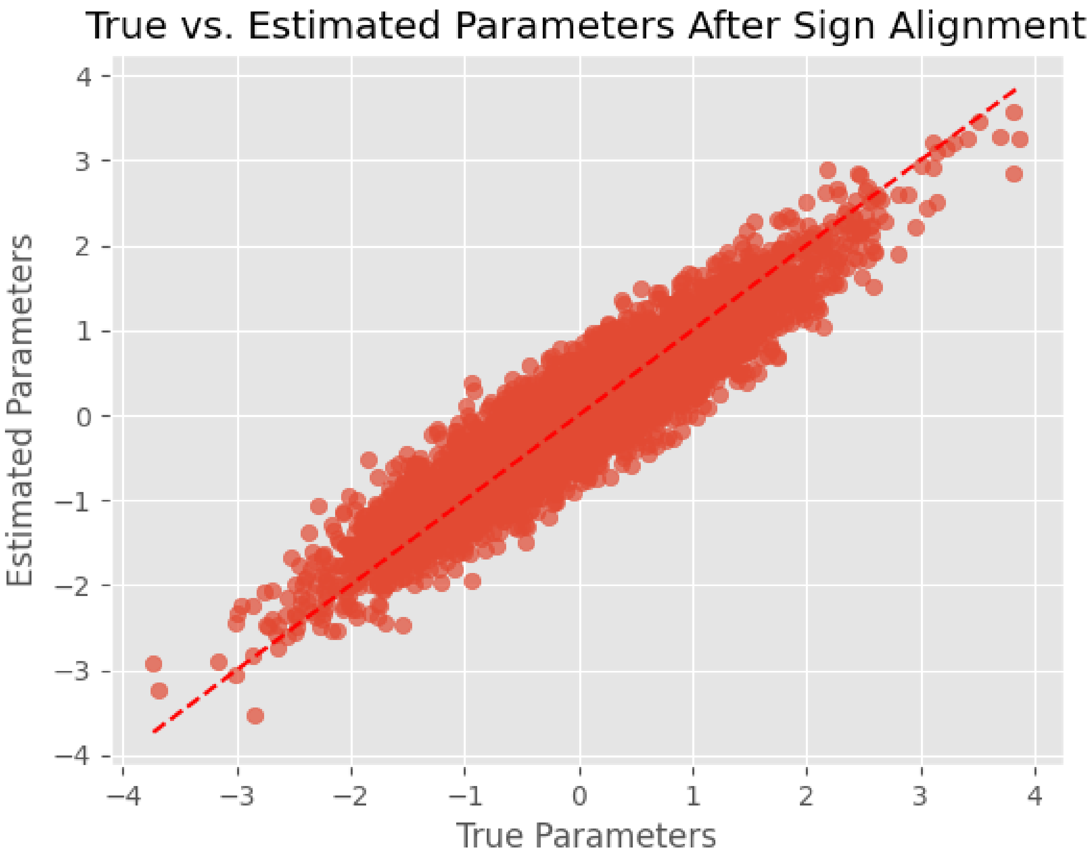
- To account for **sign ambiguity** in PCA, we can align the signs of the estimated parameters with the true parameters before comparison. To do this, calculate the dot product between the true and estimated parameters. If it is negative, multiply the estimated parameters by -1 to align them with the true parameters. Below, a code snippet highlights how to do this:

```
true_params = jnp.array(real_data)
estimated_params = jnp.array(m.posteriors)

# Compute dot product
dot_product = jnp.dot(true_params, estimated_params)

# Align signs if necessary
if dot_product < 0:
    estimated_params = -estimated_params

# Plot the aligned parameters
plt.scatter(true_params, estimated_params, alpha=0.7)
plt.plot([min(true_params), max(true_params)], [min(true_params), max(true_params)], 'r--')
plt.xlabel('True Parameters')
plt.ylabel('Estimated Parameters')
plt.title('True vs. Estimated Parameters After Sign Alignment')
plt.show()
```



Reference(s)

Archambeau, Cédric, Nicolas Delannay, and Michel Verleysen. 2006. "Robust Probabilistic Projections." In *Proceedings of the 23rd International Conference on Machine Learning*, 33–40.

Bishop, Christopher. 1998. "Bayesian Pca." *Advances in Neural Information Processing Systems* 11.

Sigg, Christian D, and Joachim M Buhmann. 2008. "Expectation-Maximization for Sparse and Non-Negative PCA." In *Proceedings of the 25th International Conference on Machine Learning*, 960–67.

Tipping, Michael E, and Christopher M Bishop. 1999. "Probabilistic Principal Component Analysis." *Journal of the Royal Statistical Society Series B: Statistical Methodology* 61 (3): 611–22.