Some title containing the words 'k-means', 'images', and 'compression', e.g., this one

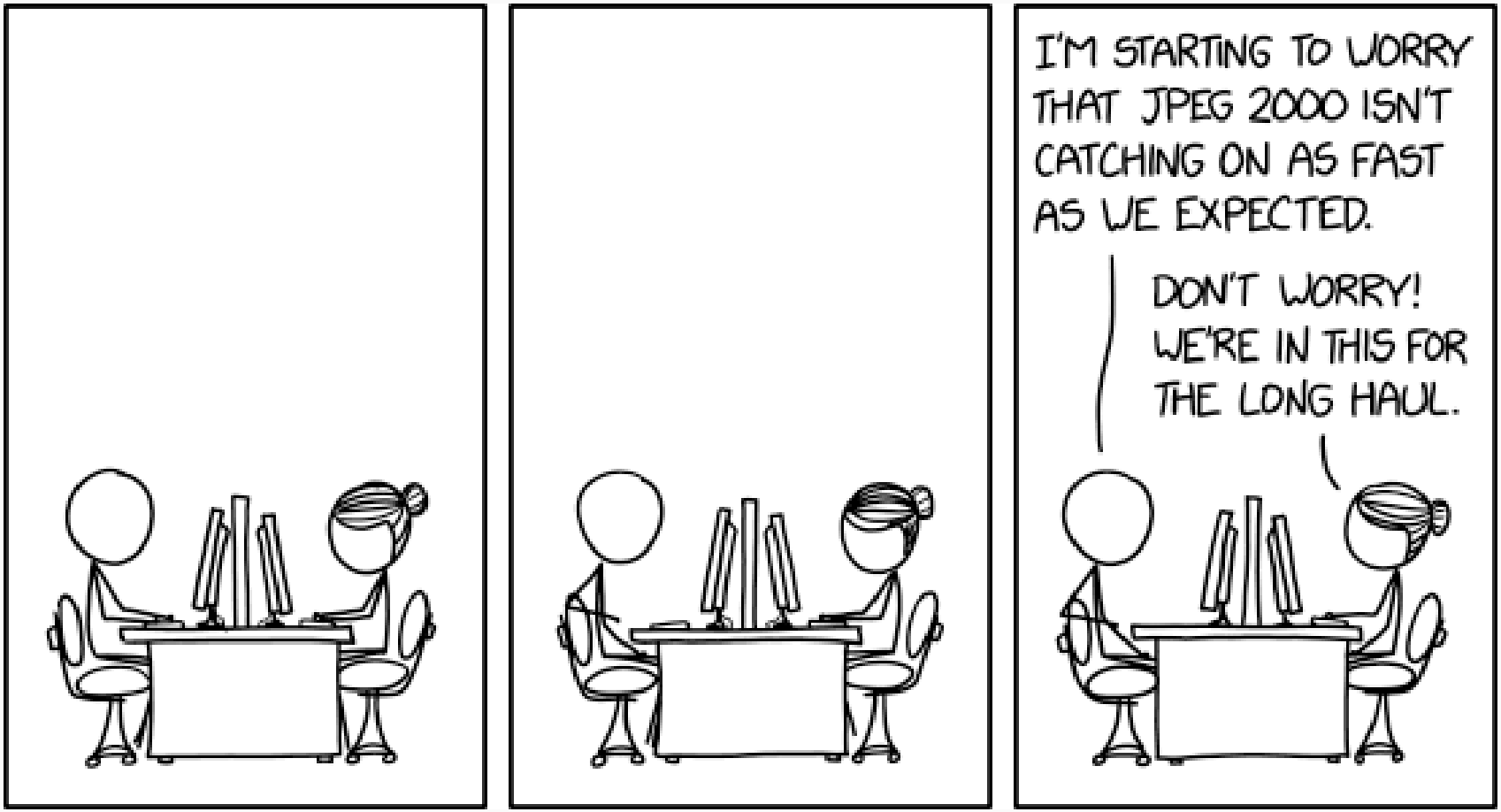
Geoffrey Thompson

5/14/2020

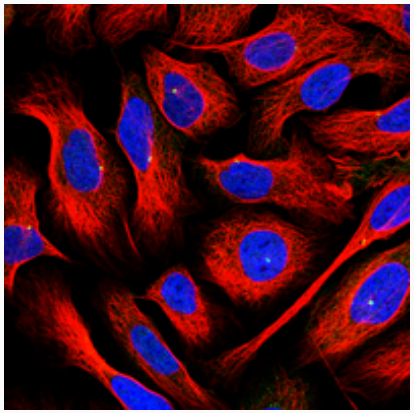
What we're talking about

- You have a bunch of images
- You want to make them smaller
- Can we use **data science**? Maybe.
- Honestly, just use JPEG or something if you're confronted with this problem.
- However, we can make this work.

A brief interlude



Example Images



The top three are small images (512 x 512, 2378 x 3560, and 2048 x 2048), the bottom three are large images (5325 x 6000, 4941 x 4795, and 7703 x 6795).

Lossless Compression

- What it sounds like: encoding same information using less storage space. Done by exploiting statistical redundancy in the data.
- Relevance to statistical learning:
  - A function that predicts the posterior probabilities of a sequence given its history can be used for optimal data compression
  - An optimal compressor can be used for prediction.
- Note: if there is no statistical redundancy, there is no compression!

Lossless Compression: Theory and Examples

- Entropy of an alphabet of symbols X, with each symbol x_i having probability p_i of occurring:

$$H(X) = -\sum_i p_i \log_2 p_i$$

- This is a theoretical lower bound for bits per symbol in symbol-by-symbol encoding.
- Huffman (1952) developed an algorithm for optimal symbol-by-symbol input coding for a known probability distribution.
- Essential idea: assign short codes to high probability symbols, longer codes to low probability symbols.

Lossless Compression: other ideas

- Higher efficiency is often possible if the symbol-by-symbol restriction is dropped.
- One idea: run-length encoding. Example:

$$aaaaaabbbaaaaaccaaaaaa$$

- Another idea: referring back to repetitions in the data. Example:

$$abc|zxy|abc|ddd|abc$$

LZ77:

The LZ77 (Lempel and Ziv, 1977) algorithm is based on these ideas: when it encounters a symbol or string of symbols it has recently encountered, it instead codes an *offset* (distance) and a *length* instead of the symbol. So an example like this can be encoded efficiently:

$$abc|zxy|abc|ddd|abc$$

Lossy Compression

- Noise is not compressible
- Sometimes a signal is structure plus noise - denoise the signal, then compress
- In audio and video, more information is contained in the data than is necessary for the human perceptual system
- One example: vector quantization. Replace colors in an image with a fixed palette of colors.
  - can be done with k-means

What other lossy image compression algorithms do:

- JPEG (see http://needsmorejpeg.com/) cuts the image into 8x8 blocks, performs a DCT on the blocks, discards some high-frequency coefficients, and then has a clever method of encoding the results using a run-length encoder and a dynamic Huffman encoder.
- JPEG-2000 is the next generation of JPEG and uses wavelets instead of a DCT and arithmetic coding (not discussed) instead of Huffman coding.
- Generally: transform to a domain where efficient approximate representation is possible, then losslessly encode that representation.

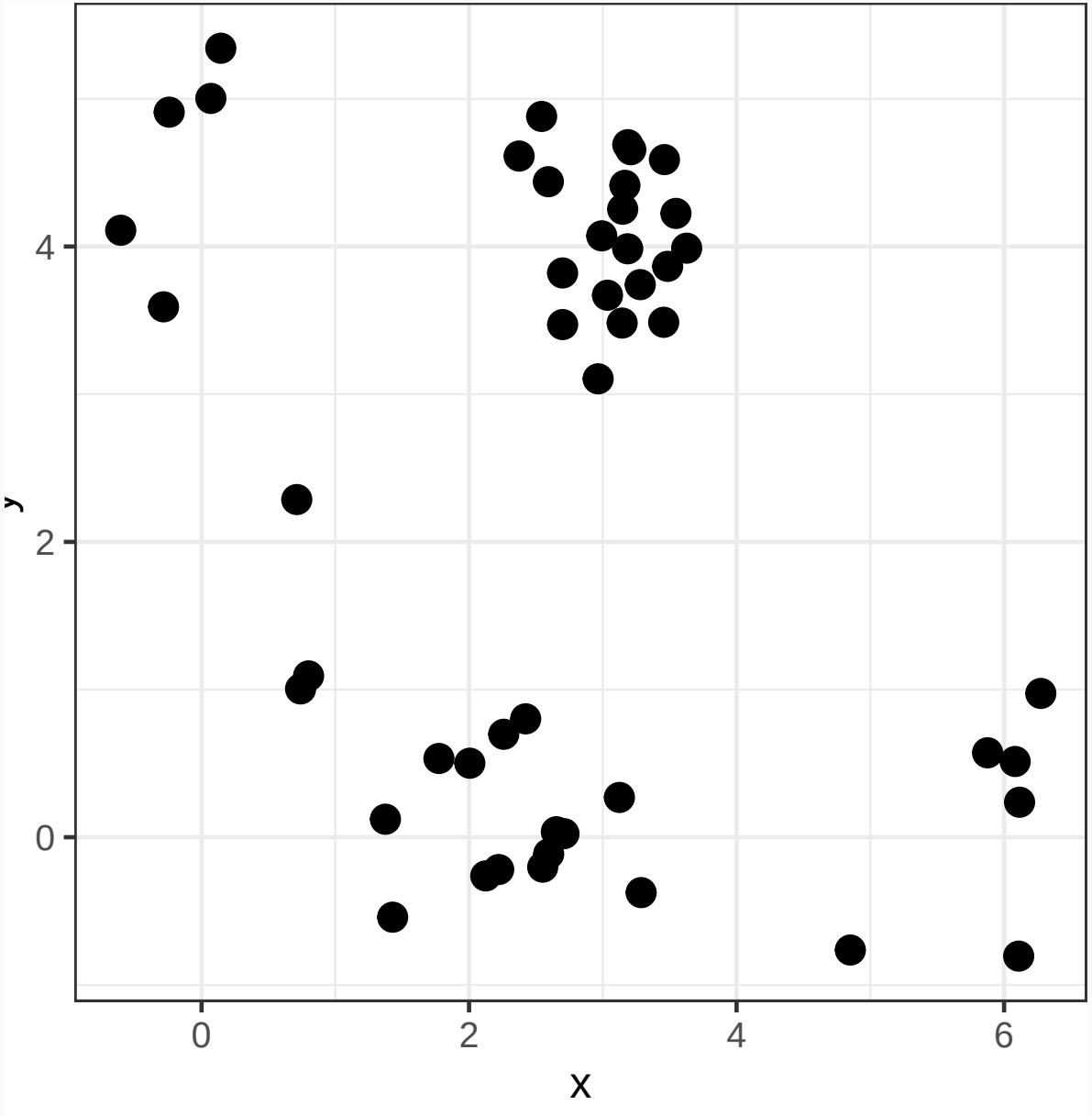
K-means: Introduction

The goal of k-means is to partition a dataset into K distinct groups such that the following objective function is minimized:

$$\sum_{k=1}^{K} \sum_{i=1}^{n} \|\boldsymbol{x}_i - \mu_k\|^2 \mathcal{I}(\boldsymbol{x} \in \mathcal{C}_k)$$

Where $\mathcal{C}_k$ are the partitions, $\mathcal{I}$ is the indicator function, and μ_k are the cluster centers.

K-means Example



Lloyd's Algorithm

Lloyd proposed the following algorithm:

1. Initialize: Start with K initial centers.
2. Assign: Assign each observation to the cluster with the nearest center.
3. Update: Update the centers to the new means:

$$\mu_k^{new} = \frac{1}{n_k} \sum_{\boldsymbol{x}_i \in \mathcal{C}_k} \boldsymbol{x}_i$$

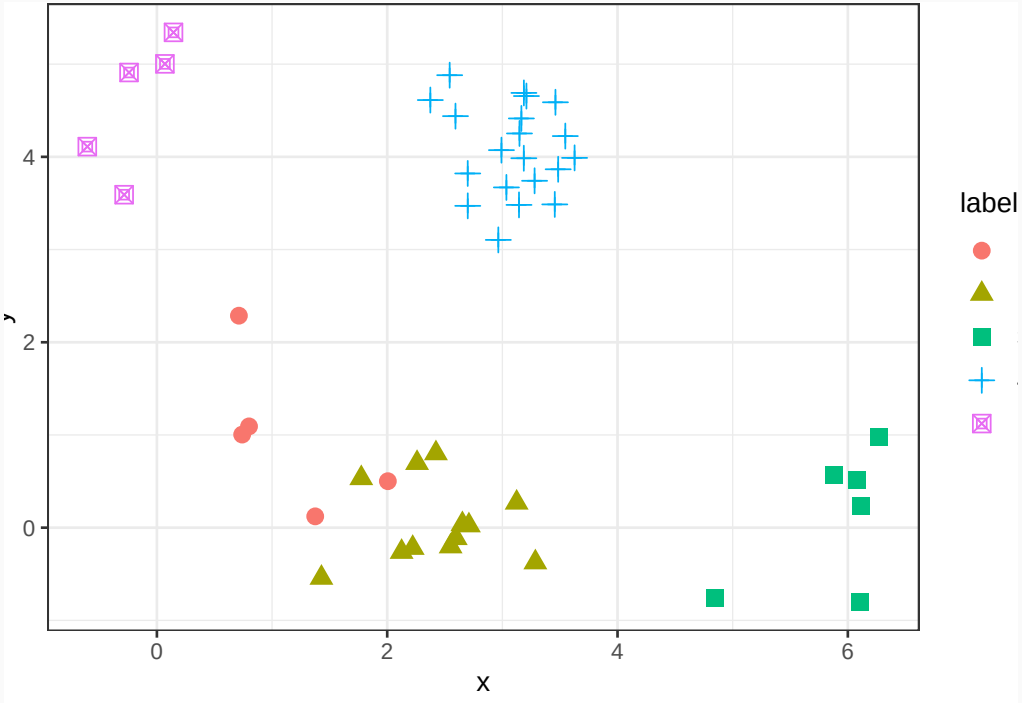
4. Repeat the middle steps until the centers (or group assignments) do not change.

More on Lloyd's

- This requires a choice of K and a method of initializing the centers.
- This method can be very inefficient; this paper uses a refinement by Hartigan and Wong (1979) which reduces the number of comparisons necessary at the assignment step and updates the centers as points are reassigned.

Ideal result:

- Ideally, the procedure will produce the "true" underlying structure of the data.
- However, this is not always possible if clusters have some overlap.



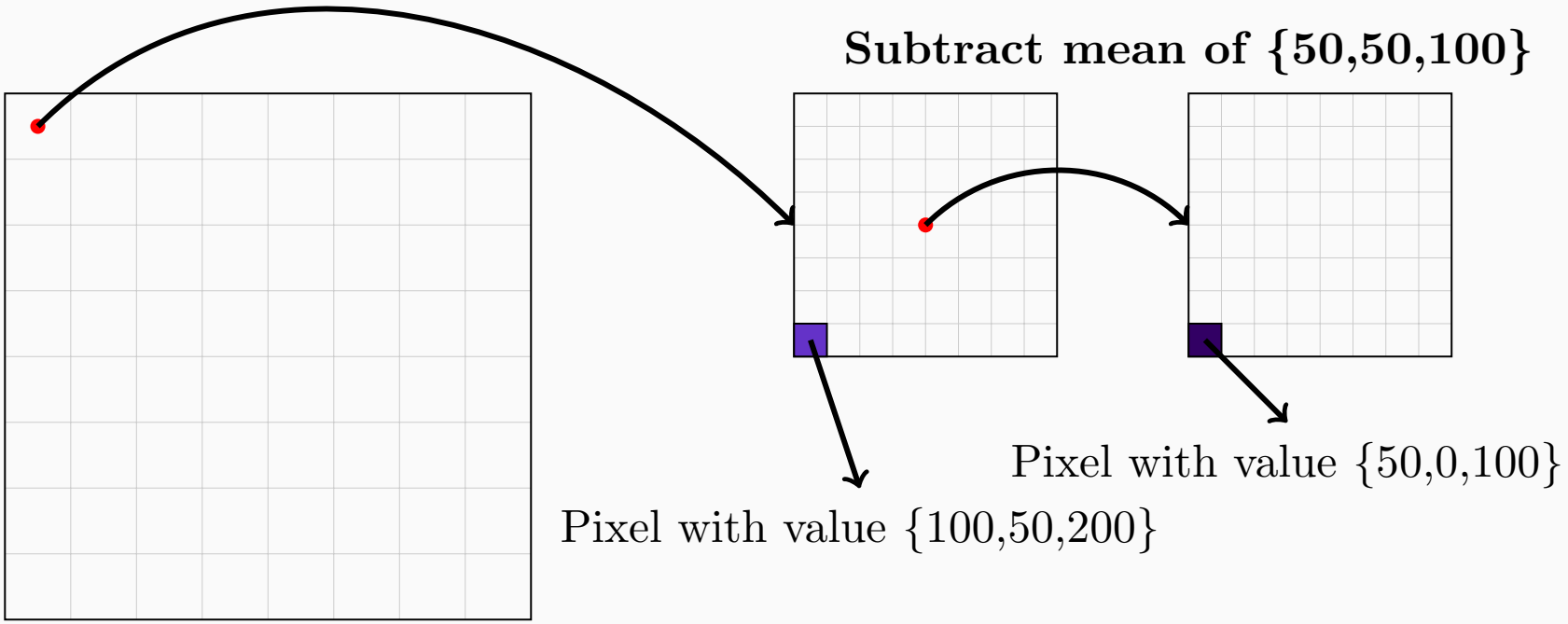
Preliminary notes on images:

- For simplicity, the presentation will be restricted to square images with 24-bit color depth.
- Each pixel has three "channels" - red, green, and blue - and uses one byte to store the value for each channel.
- Then, if the image is $N \times N$, the image can be stored in $3N^2$ bytes uncompressed.

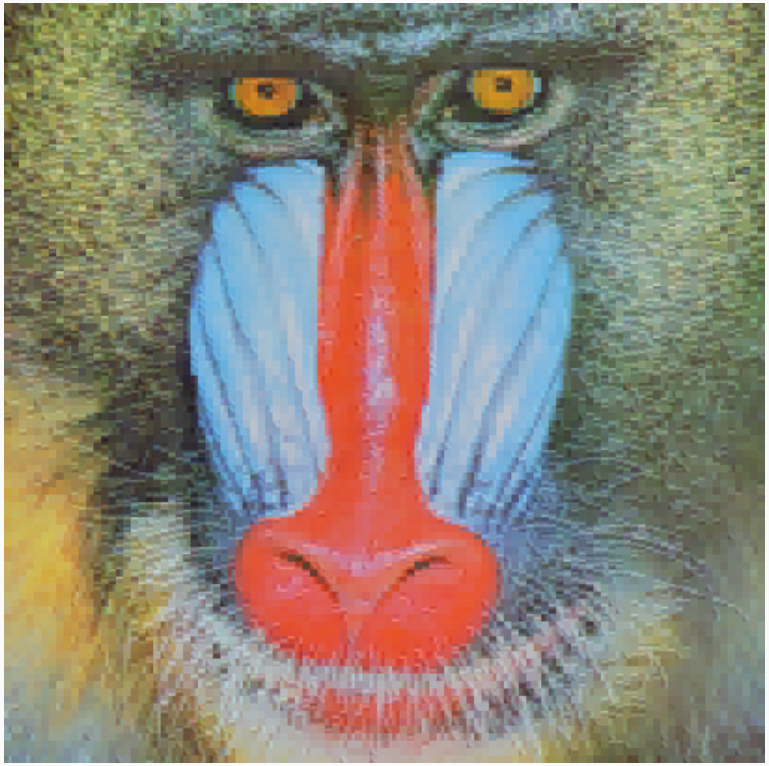
Image compression method

- Idea:
  - Choose some p - a "block size" - and cut the image into $p \times p$ blocks.
  - The mean value in each channel (ie, red, green, or blue) in the block is subtracted off and stored
  - k-means clustering is performed for some k on the residuals in the $p \times p$ blocks. Each block can be treated as a $3 \cdot p \cdot p$ vector.
  - The block means are then clustered using $k = 256$.
  - What is stored for each $p \times p$ block: the class label for the residuals, the class label for the mean.
  - Additional storage: the cluster centers for the means and residuals.

Illustration



Examples of compression



Baboon with $p = 5, k = 8$

Baboon with $p = 7, k = 50$

Method (continued)

- If $k < 256$, the class identifiers for the blocks themselves can be stored in N/p^2 bytes (one byte per block).
- The cluster means for the blocks can be stored in $8 \cdot 3 \cdot kp^2$ bytes if they are stored as double precision floating point numbers. You can cheat and make it $3 \cdot kp^2$.
- The class identifiers for the means of the blocks can be stored in N/p^2 bytes (one byte per block). The cluster means for the means of the blocks can be stored in $3 \cdot 256$ bytes. (don't need extra precision)
- For $N^2 \gg kp^2$, the compression ratio is approximately $\frac{2}{3p^2}$.

Other Options:

1. Clustering each channel (RGB) separately. This increases image quality at the expense of some compression. It essentially doubles the space requirement.
2. Perform the same sort of "blocking" on the block means instead of only quantization.
3. Cut the image into regions and perform clustering on each region. This speeds things up.

Next step: storage of the information

We've generated a lossy version of the image.

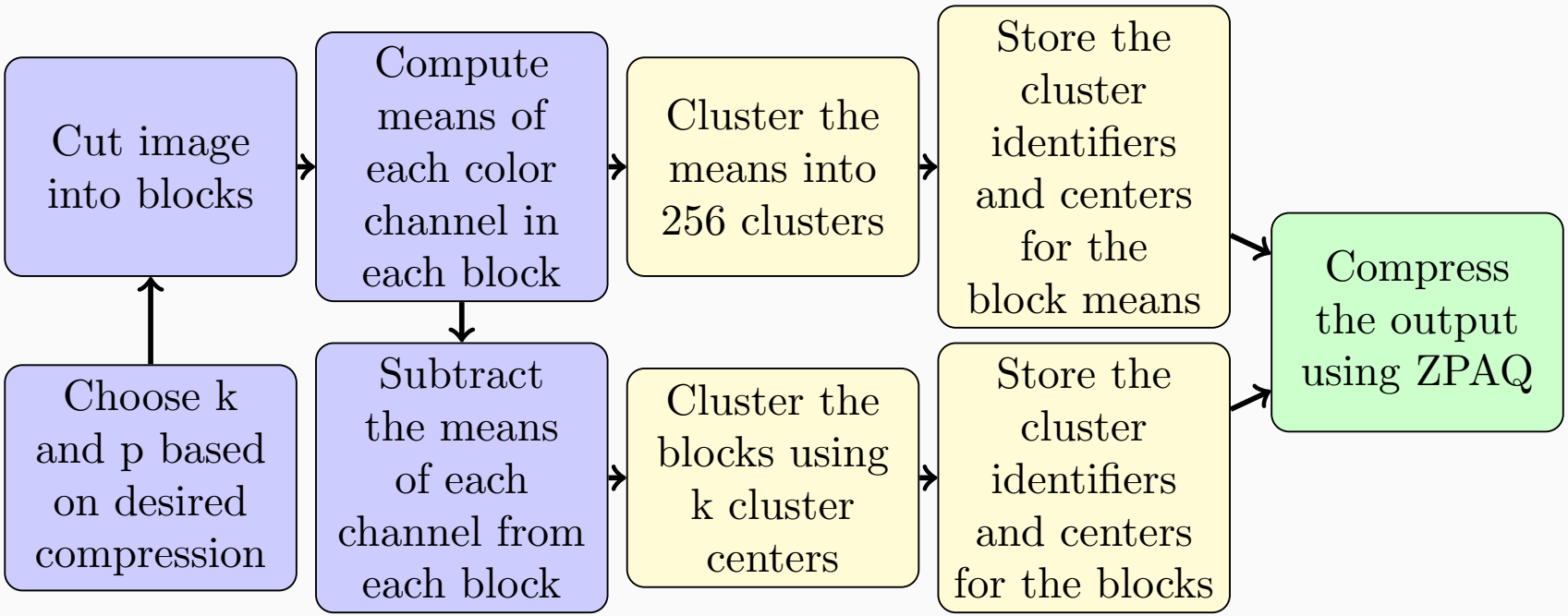
Now we need to store that representation in a compact fashion using some kind of lossless compression method.

Method - Compressed Storage

The cluster identifiers and cluster centers for the blocks and for the means are stored and then compressed further using the lossless encoding method LZMA (Lempel-Ziv-Markov Chain Algorithm), which is a modified version of LZ77 with some alterations to improve efficiency, such as efficiently storing the most recent matches and attempting to predict the next byte based on the previous byte.

Or, they can be stored slightly more efficiently using another, slightly more complicated arithmetic encoder, ZPAQ (not described here).

Illustration



Things left unsaid:

- Choosing k and p: based on the desired level of compression and some of the later results
- Initialization: use any standard method, why not k-means++?

Initialization Method

The examples were initialized using k-means++ (Arthur and Vassilvitsky (2007), which guarantees with high probability an initialization within $O(\log k)$ of the global optimum.

1. Select a point $x_1 \in X$ uniformly at random.
2. Compute the squared distances between x_1 and the other points in X.
3. Sample a point $x_2 \in X$ with probability proportional to the squared distance from x_1.
4. For each point in X, compute the minimum squared distance to the set of already selected points.
5. Continue sampling points in this manner until k points have been selected.

Metrics for Evaluating Performance

There are three things to look at when evaluating performance:

- Qualitative: how does it look?
- Size: what is the ratio of original size to compressed size?
- Quality (in a quantitative fashion): how close is it to the original?

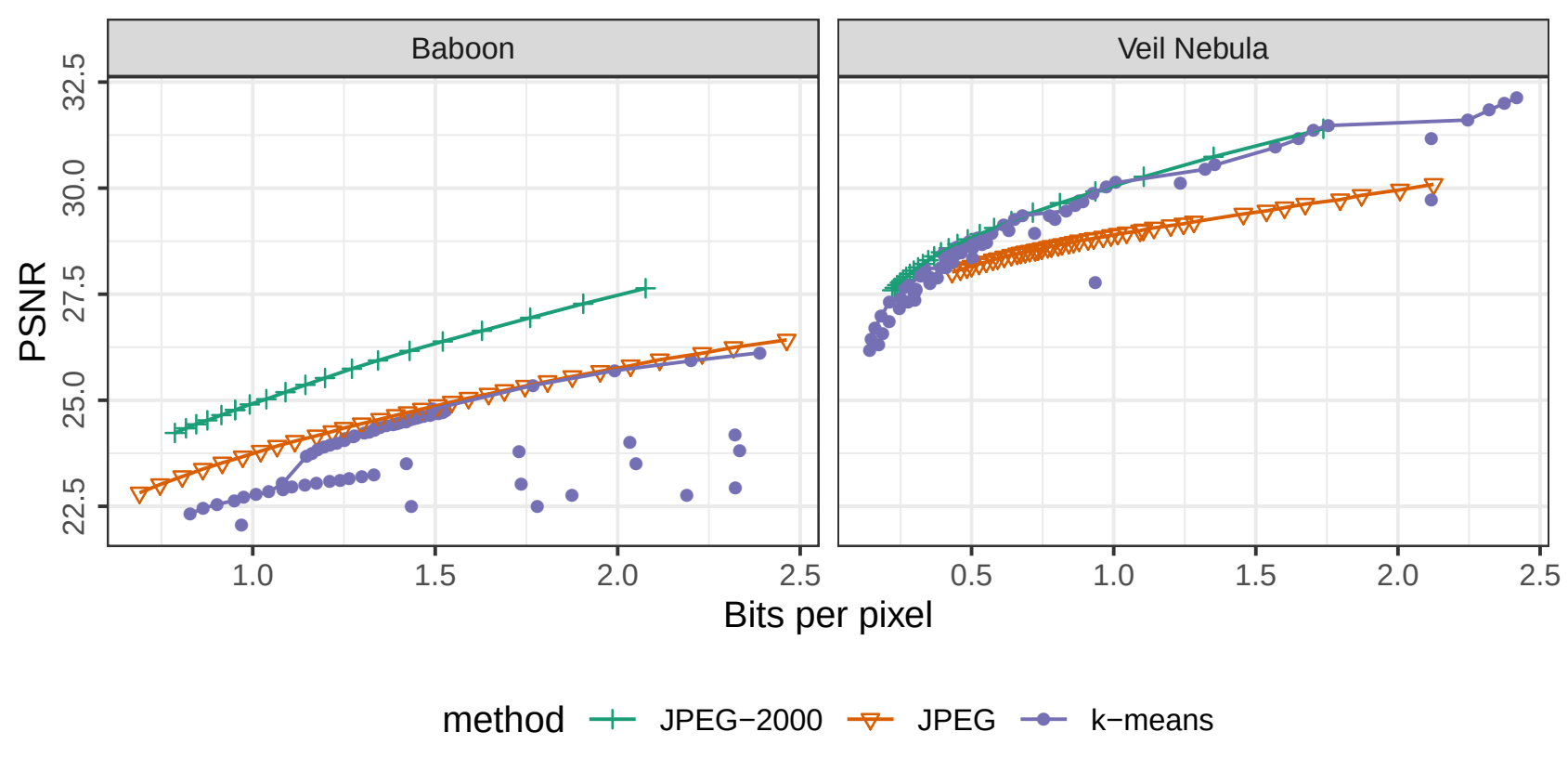
A common, but old-fashioned, metric based on the mean square error, the Peak Signal to Noise Ratio:

$$PSNR = 10 \cdot \log_{10} \frac{MAX_I^2}{MSE}$$

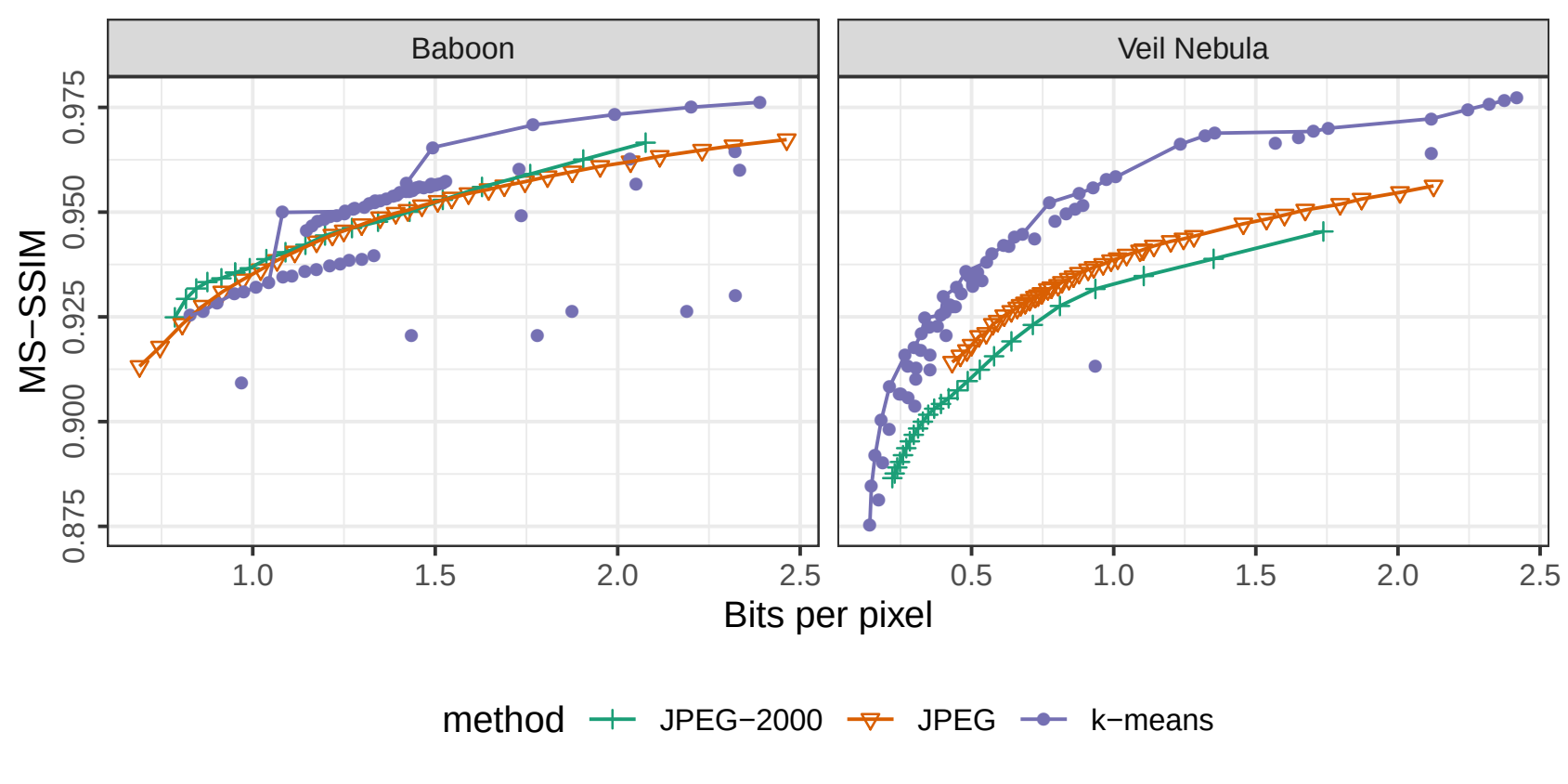
Metrics (continued):

- PSNR, though frequently used, does not relate well to human perception of image quality.
- Multi-Scale Structural Similarity Index Measure (MS-SSIM) (Wang et al, 2004) correlates highly with ratings of visual quality given by human subjects.
- The MS-SSIM looks at 8×8 patches of the original and the compressed image and compares them using the means, variances, and covariances between the images at a several different scales to come up with a measure of the similarity between the two images.
- The metric ranges from 0 to 1 with 0 being no similarity at all and 1 being a perfect match.

Comparison to Original Images (PSNR)



Comparison to Original Images (MS-SSIM)



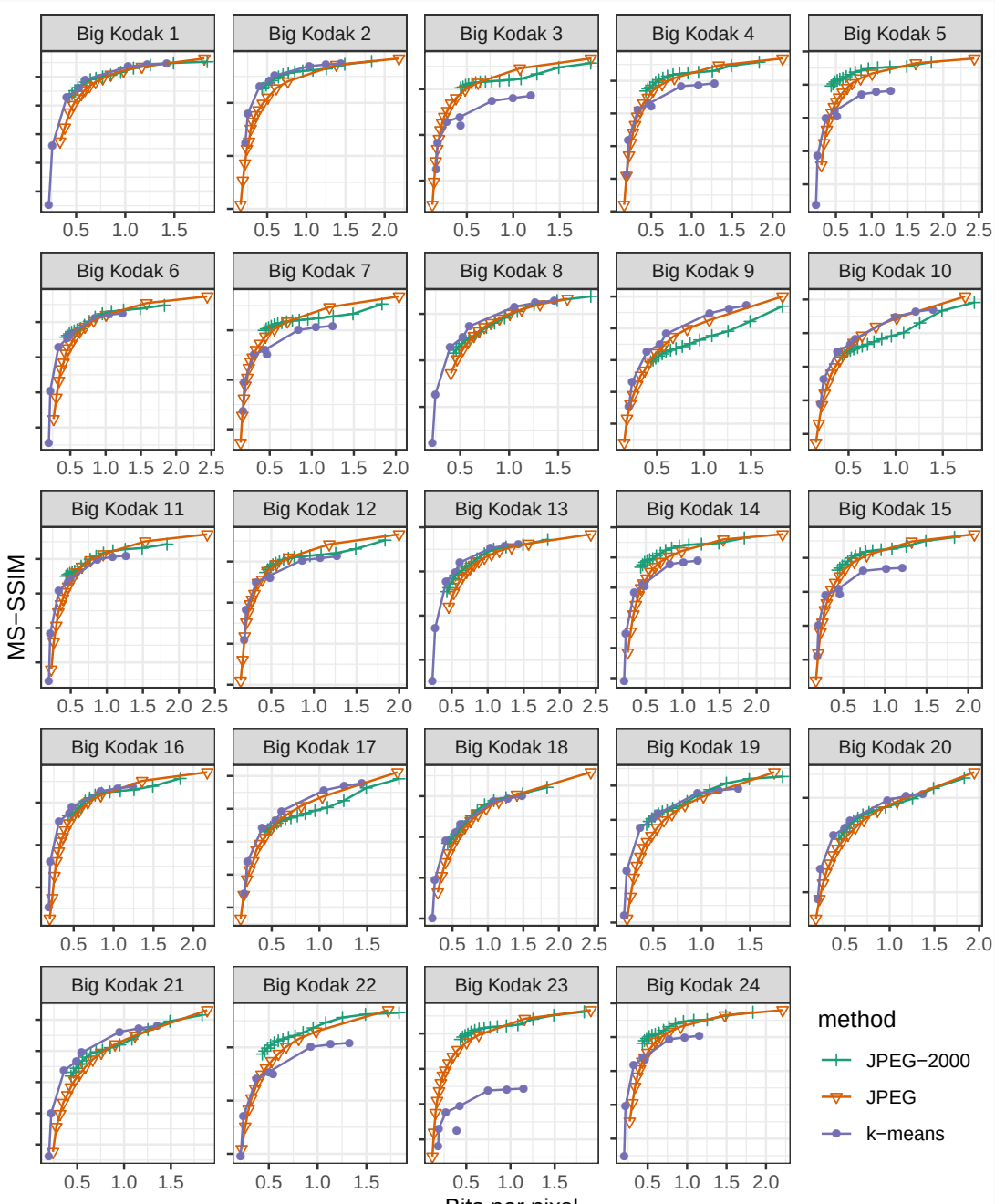
So it works!

Now let's try it on some larger sets of images

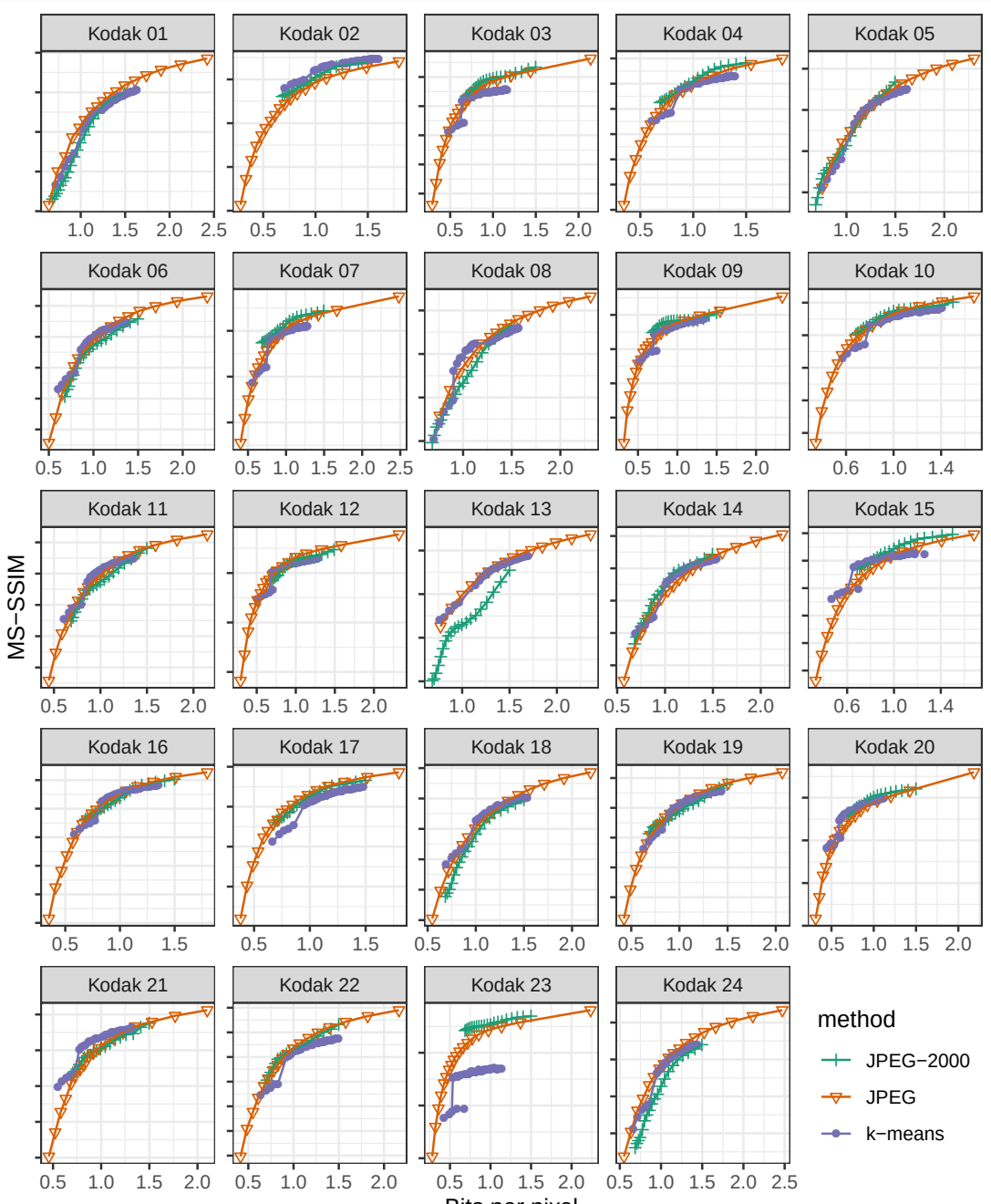
- Kodak: 24 images, a small version (768×512) and large version (3072×2048)
- Aerial photos from USC-SIPI: twelve 512×512 images and twenty-four 1024×1024 images.
- An assortment of mostly larger images (mostly astronomical)
- MR-TIP: a collection of 15 colored MRI images (512×512)

Kodak: Examples

Large Kodak Results:

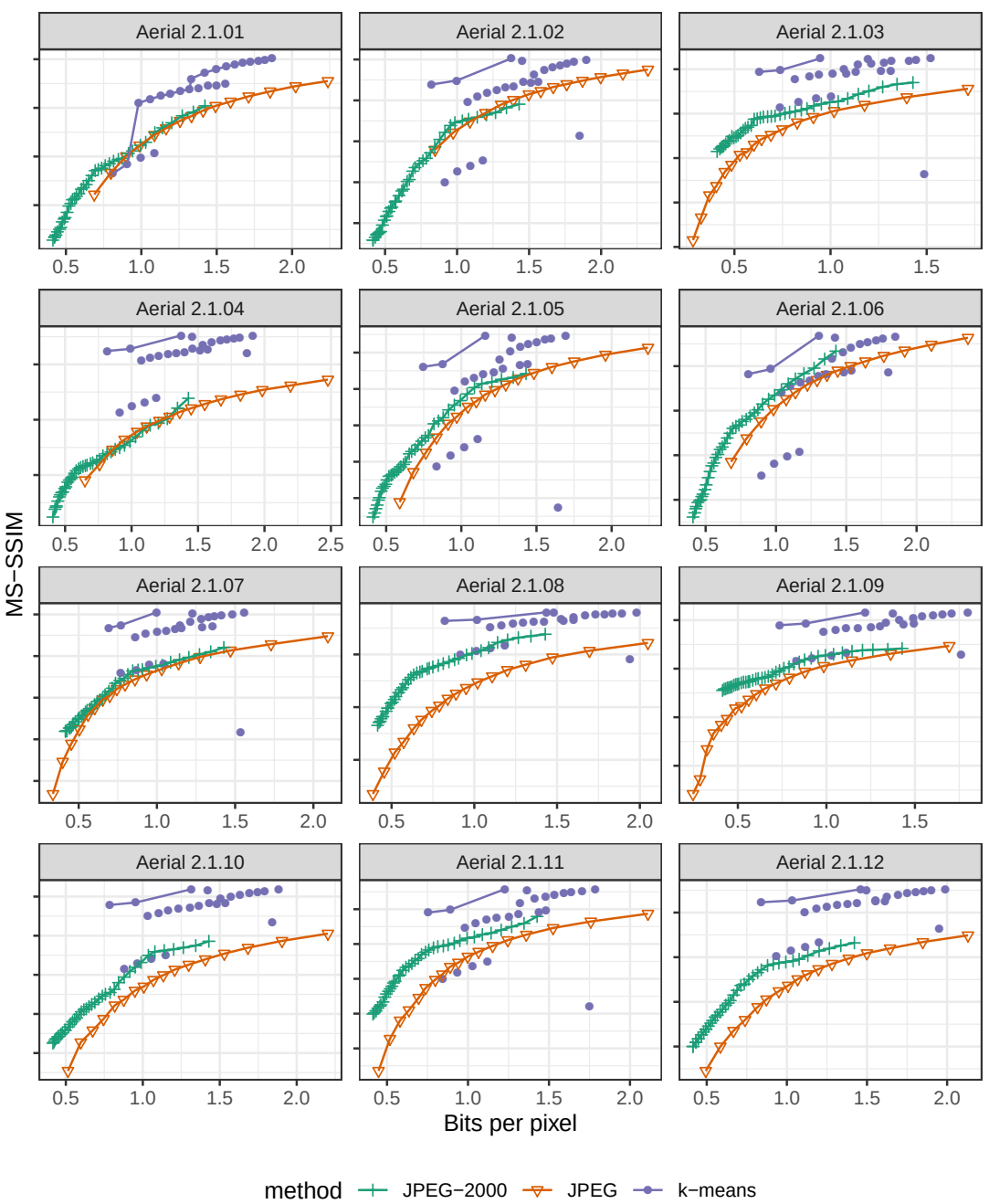


Smaller Kodak Results:

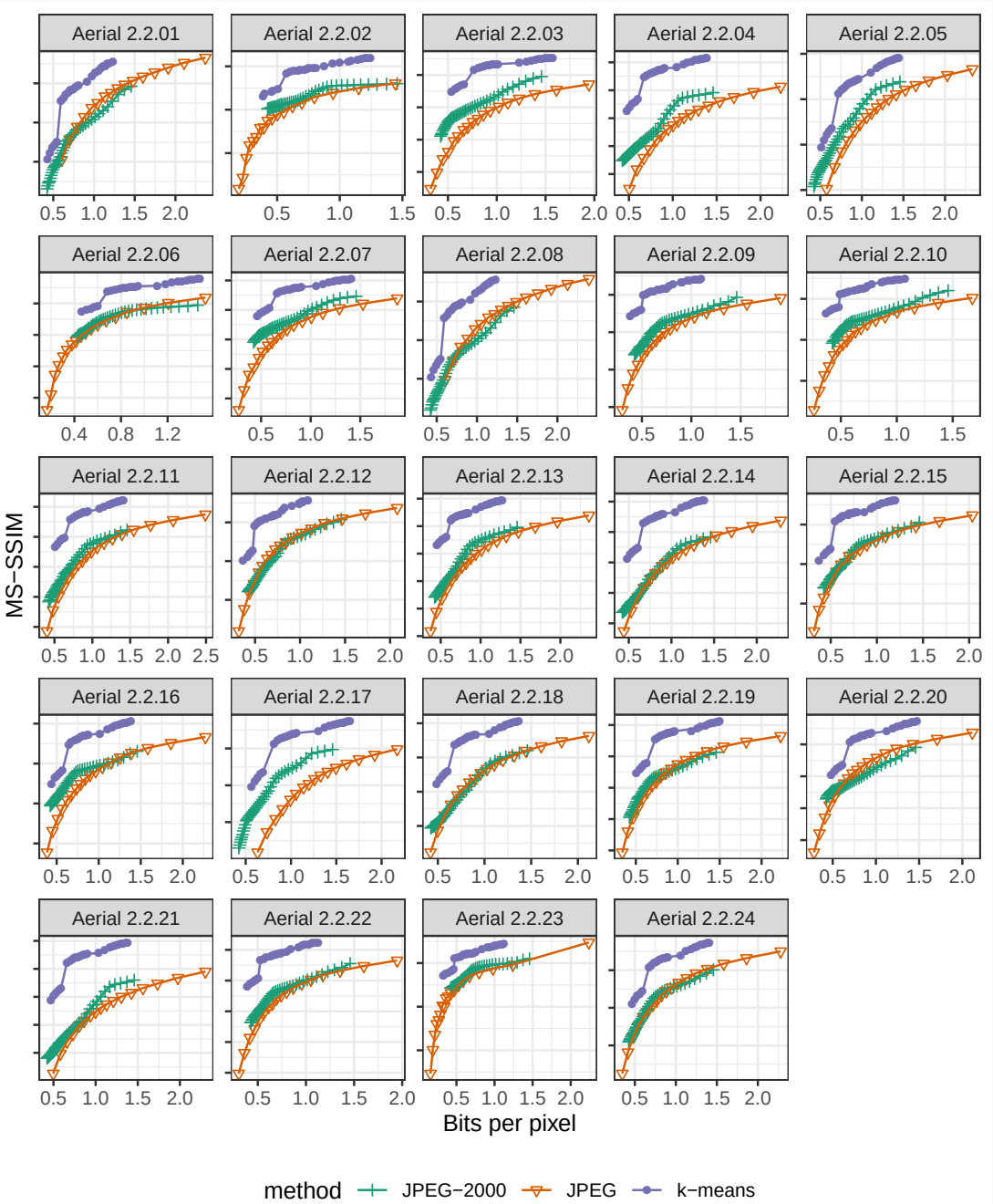


Aerial Photo Examples:

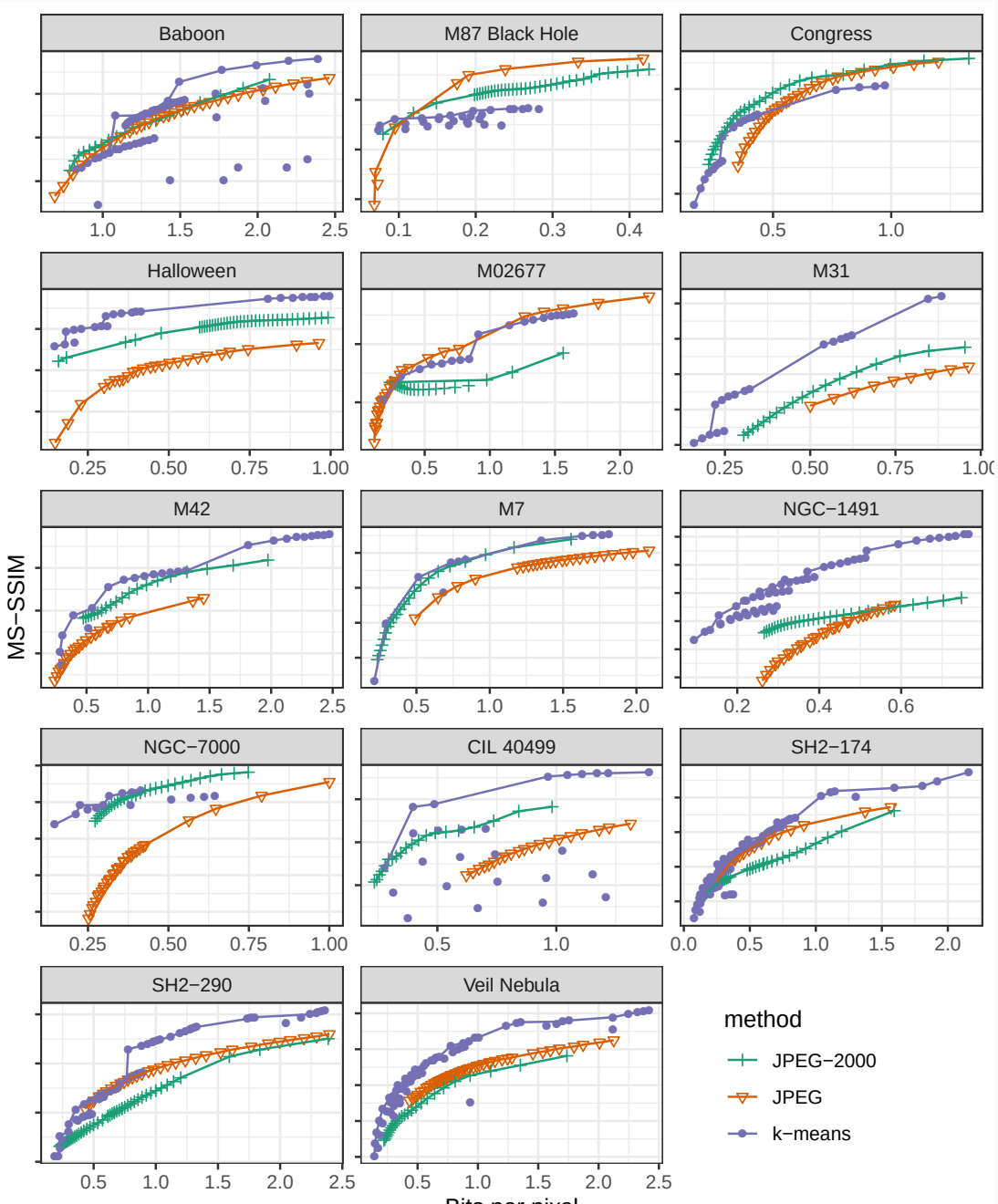
Smaller Aerial Results:



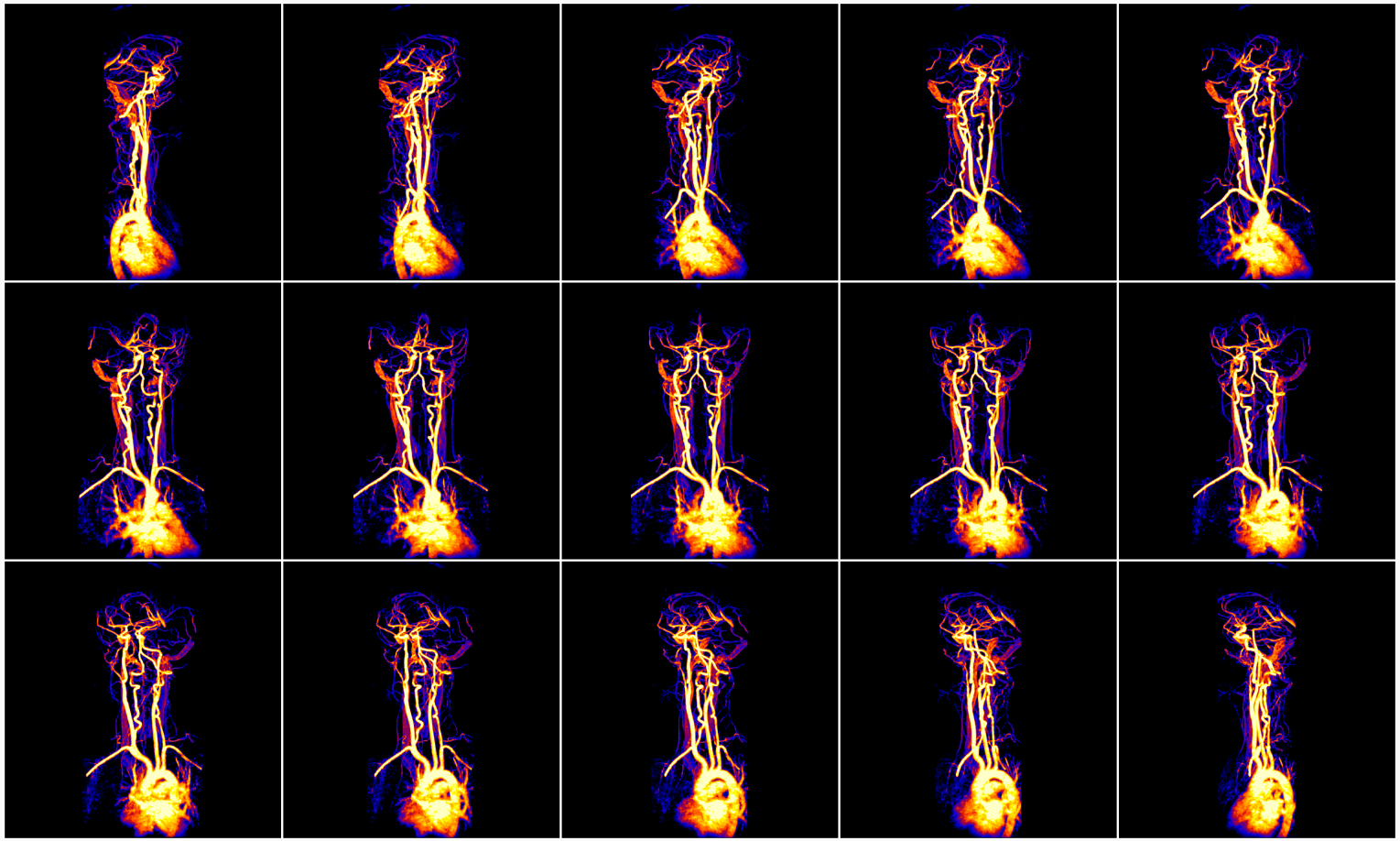
Large Aerial Results:



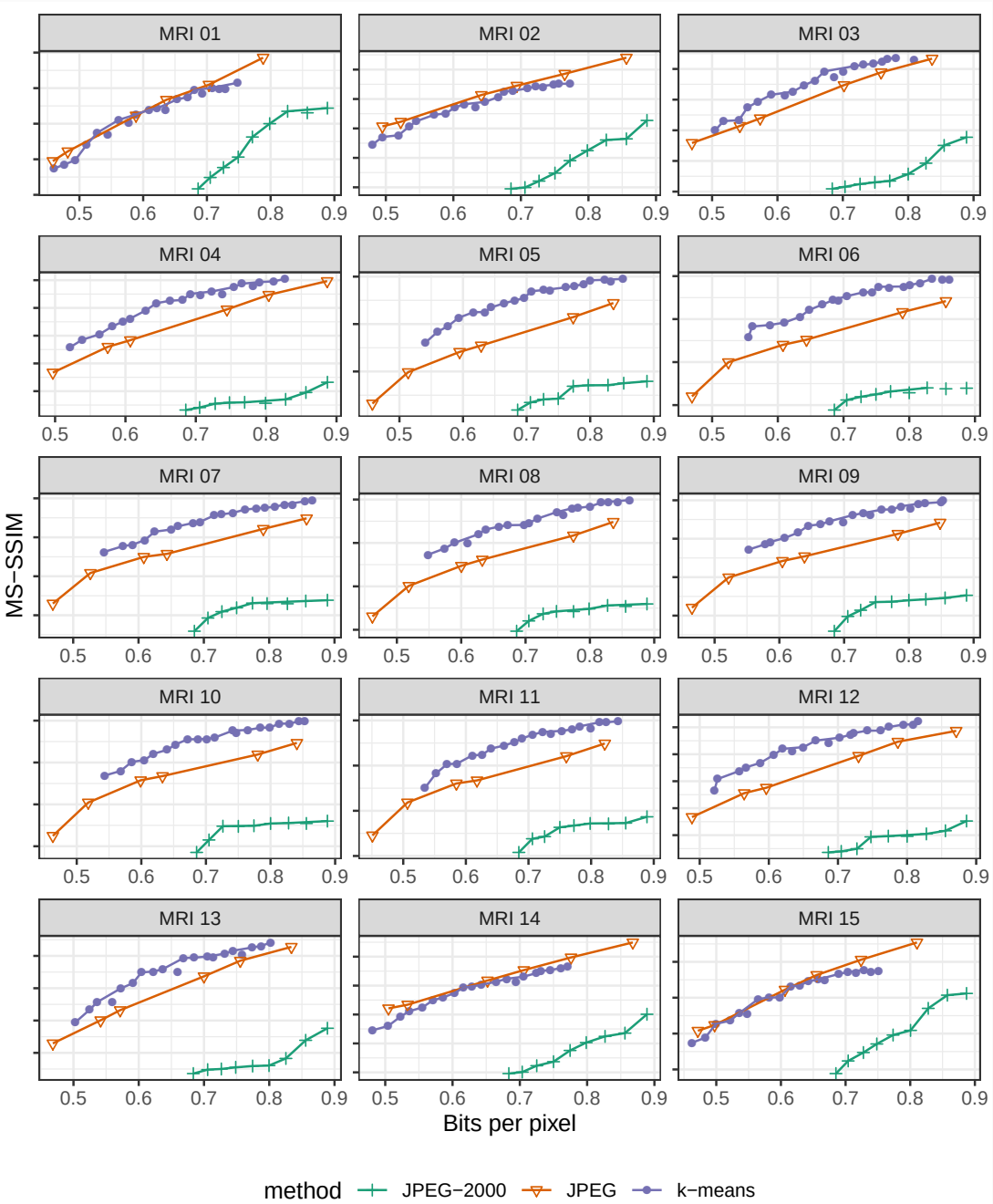
Assorted images of varying (mostly big) sizes:



MRI Example:



MRI Results:


MRI 01
MRI 02
MRI 03
MRI 04
MRI 05
MRI 06
MRI 07
MRI 08
MRI 09
MRI 10
MRI 11
MRI 12
MRI 13
MRI 14
MRI 15
MS−SSIM
Bits per pixel
0.5
0.6
0.7
0.8
0.9
method
JPEG−2000
JPEG
k−means

When does it work?

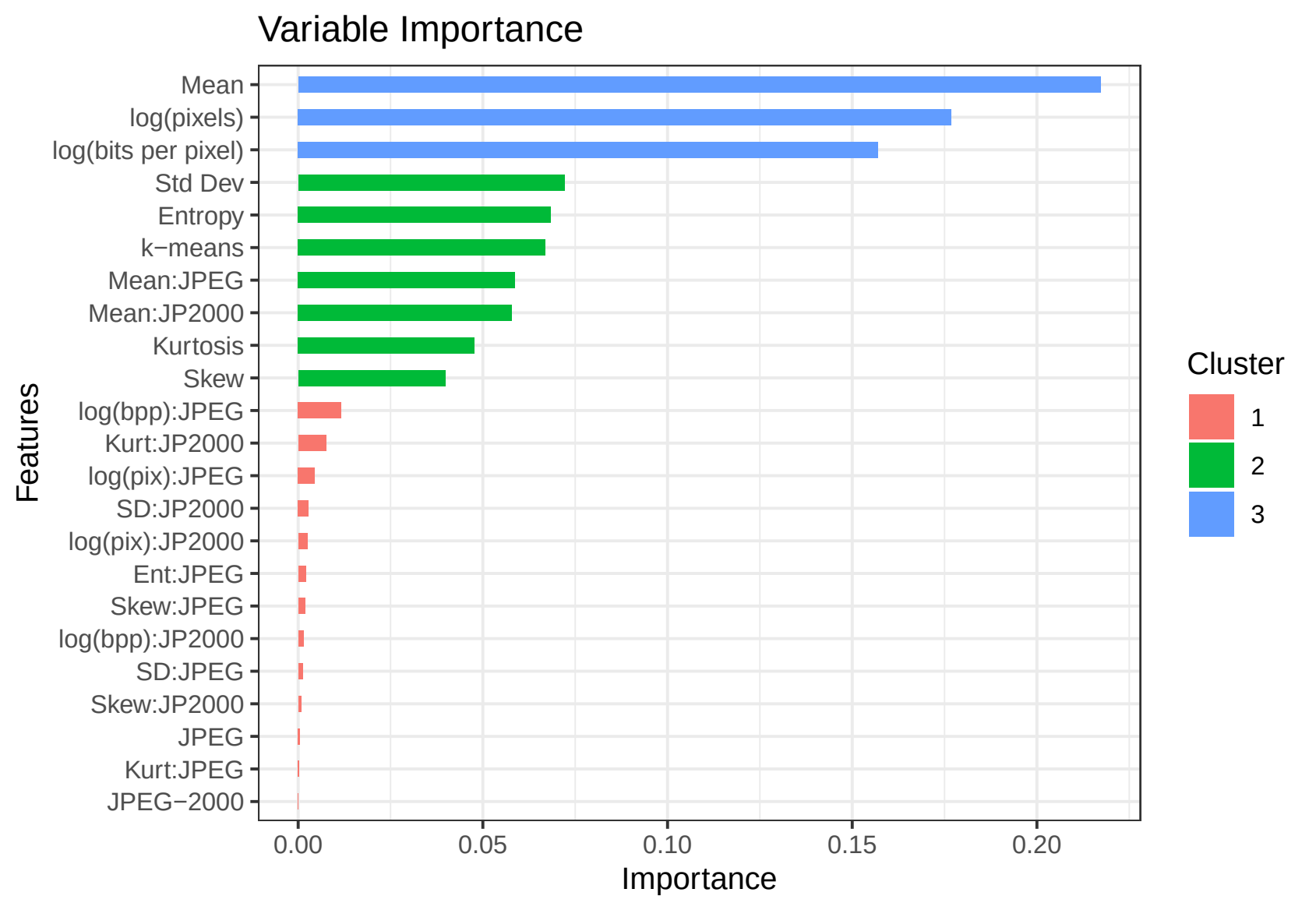
We can try to build a model to predict when this method does better.

Estimating growth curves based on image characteristics.

We can do this with fancy methods, like XGBoost (gradient boosted regression trees) with a monotonicity constraint, though interpretation is then difficult.

Look at summary statistics for each image and their interactions with the compression method as inputs to predict MS-SSIM performance.

XGBoost importance results



Conclusions:

- This relatively simple and parsimonious method achieves competitive performance across a variety of images at various sizes.
- For large images, it is possible in some compression ranges to outperform standard image compression software (in terms of both PSNR and modern image quality metrics) using the block k-means method and intelligent choices of k and the block size, p.
- No clear story about when the k-means method performs better.

References