LAB #6: CLASSIFICATION

CS 109A, STAT 121A, AC 209A: Data Science

Fall 2016

Harvard University

ANNOUNCEMENTS

1. Thank you for your feedback (they are welcomed at any time)!
2. **Reiterating our philosophy:** to really understand something is to build it from scratch **at least once** (even if the process is tedious and painful: like learning arithmetic). This level of understanding makes you a more educated user (even if you don't want to be a maker).

   HWs in the first half of the semester will be naturally more time consuming/implementation heavy, b/c we have few tools and are seeing everything for the first time.

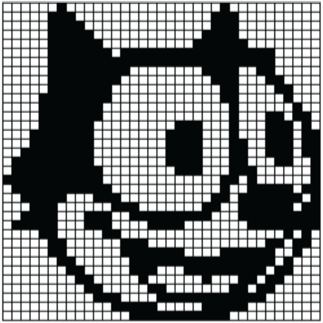
   HWs in the second half of the semester will naturally be more applications based and interpretation heavy, b/c you now have lots of tools and the implementation of many of the algos are out of the scope of the math/programming background we assume.

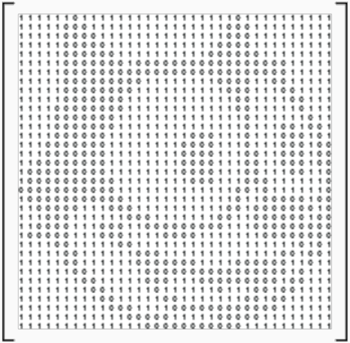
   We are going to be very cognizant of your time commitment b/c you now have projects to think about. We will keep the HWs shorter.
3. **Advice for doing HW:** Do it with your study group (but don't copy). Do it during office hours (the instructor OHs are particularly underused!).
4. **Labs will have more interpretation:** rather than releasing code all at once, we pre-filled some cells in your notebook with starter code, so you can concentrate on experimentation and observation.
5. **Hint for Problem 2:** This problem sound familiar from Lab 5

IMAGE PROCESSING

REPRESENTING IMAGES AS MATRICES AND ARRAYS

An image can be represented digitally as a grid of illuminated pixels (small area of illumination on display device). We can represent this as a matrix of numbers, each number encoding the intensity of the corresponding pixel.





35x35

We can flatten the image matrix into an image vector:

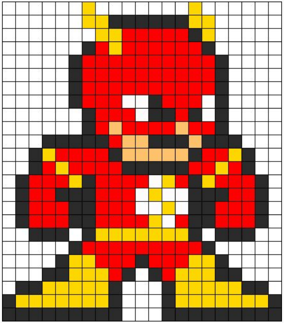


Image as matrix of pixels

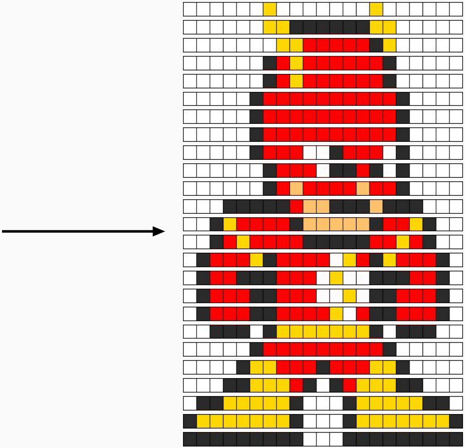


Image matrix split along rows

REPRESENTING IMAGES AS MATRICES AND ARRAYS

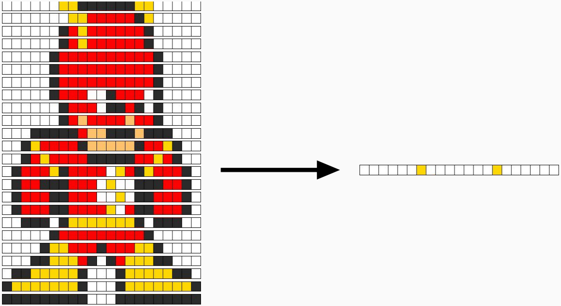


Image as matrix of pixels

Image as single vector of all the rows concatenated

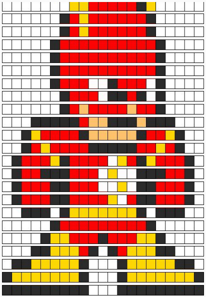


Image as matrix of pixels

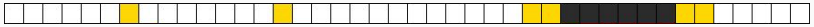


Image as single vector of all the rows concatenated

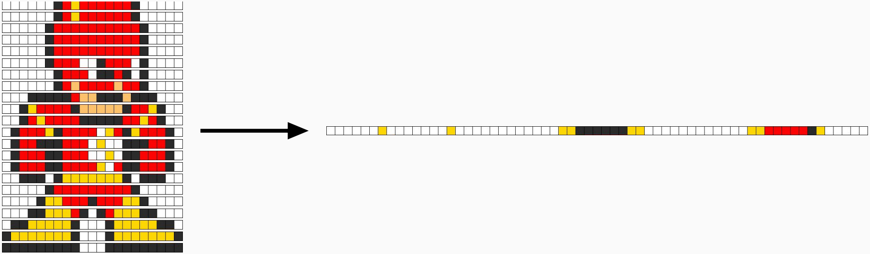
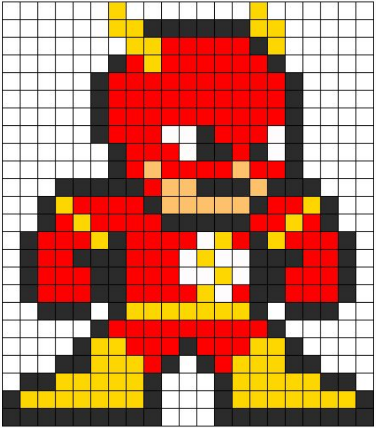


Image as matrix of pixels

Image as single vector of all the rows concatenated

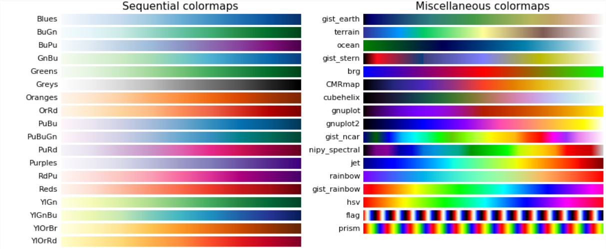


This image, when flattened, is represented as a `numpy` array of shape `(441, )`.

Given an image array, we can reshape it into an image matrix; and then display the image, by mapping each number to an intensity on a colormap.

plt.imshow(flash_vector.reshape(21, 21), cmap=color_map)

where `color_map` is an appropriately chosen `matplotlib` color map:



DIMENSION REDUCTION

Now that we know how to convert digital images into numpy arrays, we can feed them as input into any of our fancy statistical models!

In particular, we can perform classification on digital images (automatically classify images as “containing you” or “containing your friends” on Facebook for example).

Today, you'll be working with a dataset of digital images of hand-written digits. Each data point is an 8x8 gray-scale image of a single hand-written digit, flattened into a 64-length vector.

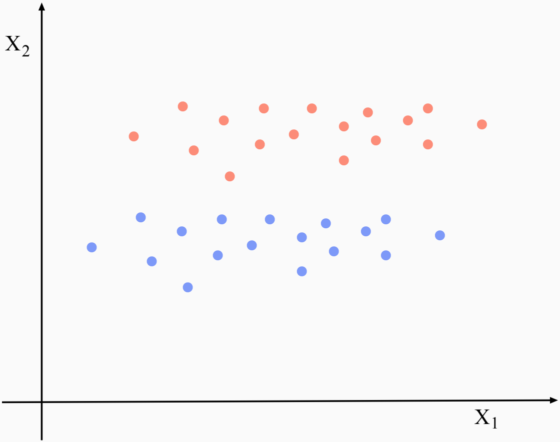
You will perform a task that major tech companies have been trying to perfect for ages: correctly **classify** handwritten symbols.

Question: what's a potential difficulty with working with images as arrays?

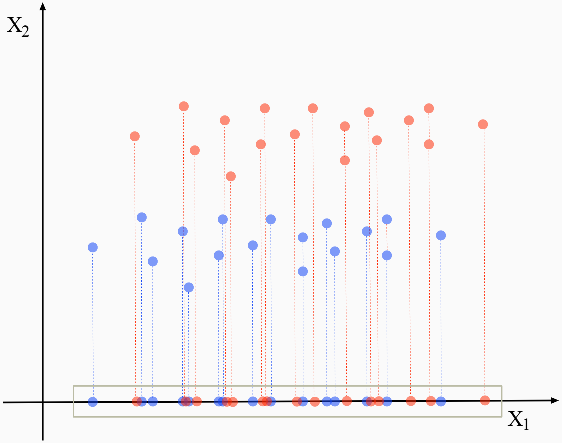
Dimension reduction is the task of transforming data with a large number of features (or predictors) into a data set with much fewer number of features.

Dimension reduction can be done in many ways. The most naïve approach would be to simply select a couple of features (or predictors) and chuck the rest - **variable selection**.

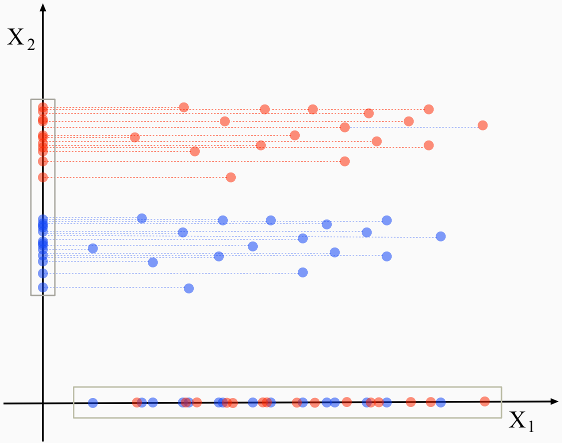
A data set with two classes and two predictors, X_1 and X_2.



We chuck X_2, and only keep X_1 (projection onto X_1).

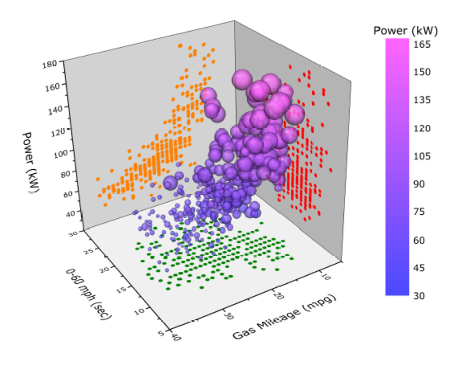


We chuck X_1, and only keep X_2 (projection onto X_2).



Question: Which projection is better? Why?

We can do the same for data with three predictors.



Question: What is the problem with doing variable selection this way?

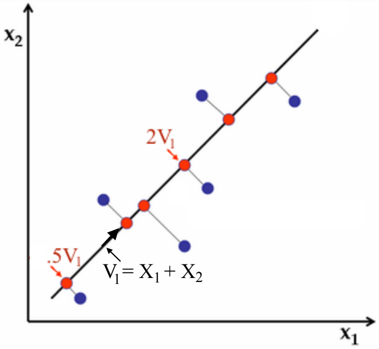
Recall that PCA finds the (orthogonal) directions in which the data exhibits the maximum variance.

The top PCA component is the direction (given by a vector) along which the data has maximum variance; the second component of the PCA is the direction (orthogonal to the first) along which the data exhibits the "second greatest" amount of variance, etc.

Each component of the PCA is a linear combination of the original set of predictors, for example

$$\underbrace{v_1 = x_1 + x_2}_{\text{component 1}}, \quad \underbrace{v_2 = x_1 - 3x_2}_{\text{component 2}}$$

Recall that PCA finds the (orthogonal) directions in which the data exhibits the maximum variance.



When we project the data onto the axes formed by the components, each data point is now a linear combination of the components.

To perform PCA in `sklearn`:

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(x)
x_reduced = pca.transform(x)
```

To get the components out of PCA:

```
#gets the vector for the 0-th component
comp_0 = pca.components_[0]
#gets the vector for the 1-st component
comp_1 = pca.components_[1]
```

WHAT DO WE WANT OUT OF THIS?

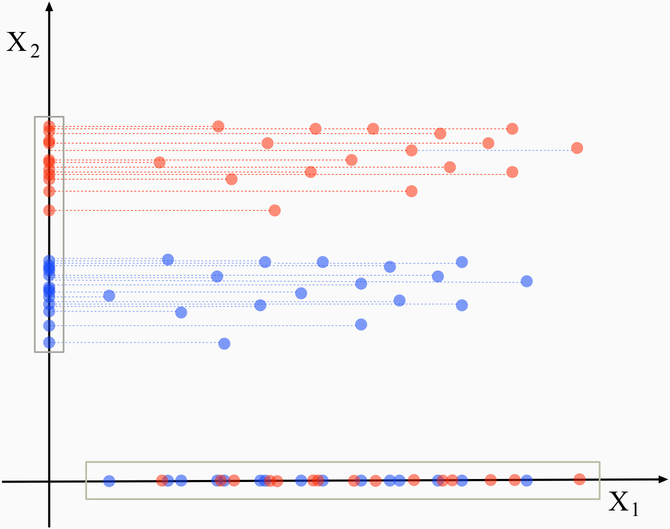
We want to reduce the dimension (the number of features) of the data.

PCA allows us to perform automated dimension reduction - we project the data onto the top PCA components.

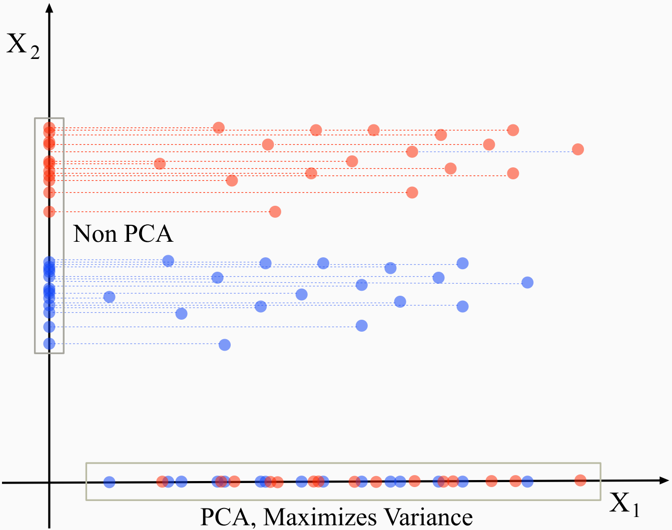
Question: Is performing PCA for dimension reduction helpful for our classification task?

Try it on your dataset! Do Steps 1 and 2.

IS PCA ALWAYS GOOD FOR CLASSIFICATION?



IS PCA ALWAYS GOOD FOR CLASSIFICATION?



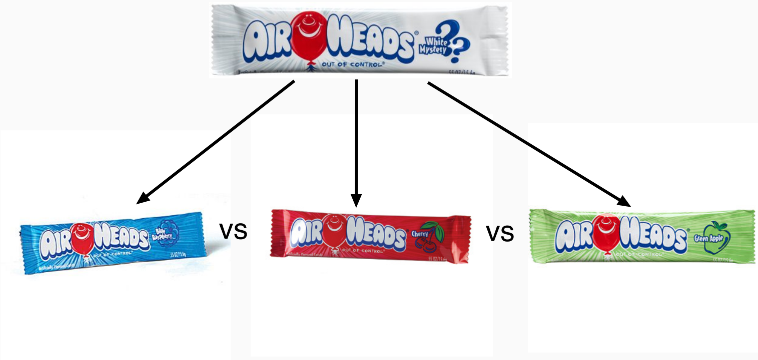
CLASSIFICATION

Implementing logistic regression using `sklearn`:

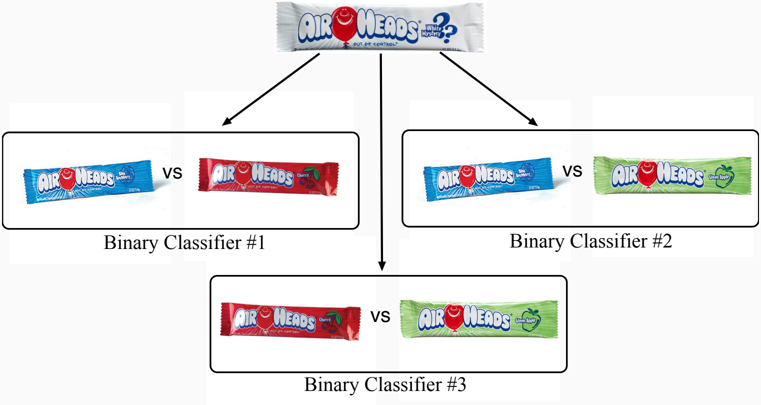
```
from sklearn.linear_model import LogisticRegression as LogReg
log = LogReg() #by default regularization param is 1
log.fit(x, y)
log.predict(x)
```

The coefficients for the predictors are given by `log.coef_`. The intercept is given by `log.intercept_`.

NAIVE THREE-CLASS CLASSIFICATION



NAIVE THREE-CLASS CLASSIFICATION



We tally up the predictions from all the classifiers: 2 votes red, 1 vote green, means mystery = red!

TRY IT OUT

Implement a classifier for you handwriting dataset. (Step 3)

How good was your classifier?

How do we assess the quality of our model? What are the strength and draw-backs of each of the following?

1. Correct Classification Rate
2. Visualize decision boundaries
3. Other metrics?