NOTES ON INDEPENDENT COMPONENT ANALYSIS

A VERY ROUGH DRAFT WITH MUSINGS

NIVAS

1. Uncorrelatedness, Independence and Orthogonality

1.1. **The Pearson Correlation Coefficient.** Suppose x, y are distributions on $\mathbb{R}$. Then (x, y) may be defined as the covariance between the distributions. If $X, Y \in \mathbb{R}^n$ are n draws from these distributions respectively, then the estimate of the covariance is simply $X^T Y/n$, after centering both X, Y. So the pearson-correlation coefficient is

$$\frac{X^T Y/n}{||X||/\sqrt{n} \cdot ||Y||/\sqrt{n}} = \frac{X^T Y}{||X||\,||Y||} = \hat{X}^T \hat{Y}.$$

Clearly, the pearson correlation coefficient is not an inner product since it is not positive homogeneous.

1.2. **Gram Schmidt With The Covariance Inner Product.** All that the GS orthogonalization process needs is an inner product definition, and it can take in a bouquet of vectors and make it orthogonal with respect to this inner product. If we supply the covariance inner product, then it returns uncorrelated vectors as independent vectors.

1.3. **Independence as Strong Uncorrelatedness.** If x, y are independent, then $f(x), g(y)$ are uncorrelated for all f, g. So if we pick candidate $f_i, 1 \leq i \leq N$ we can take $x_j, 1 \leq j \leq m$ and make $f_i(x_j)$ orthogonal to $f_k(x_l)$ for all $j \neq l$. Clearly, if there are enough draws of x_i (i.e. $x_i \in \mathbb{R}^n$ for a large enough n) compared to the number of m, then we can fill up the matrix with functions $f_i(x_j)$ without suffering a dimension-suffocation problem.

1.4. **A GS-like IC-estimation algorithm.** At each step, the vanilla covariance-GS algorithm computes a single $O(n)$ orthogonalization of x_j with every vector, x_l for $l \leq j$. So in $O(nm^2)$ time, we have an uncorrelated set of vectors. Clearly, independent vectors are always uncorrelated but not otherwise. So the correlation-rank of the data is an upper bound on the number of independent components. Now consider computing N functions, f_i, on the data. Since every function of x_j has to be uncorrelated with every function of x_k for $k \neq j$, it follows that instead of an $O(n)$ orthogonalization for each pair x_j, x_k, we have an $O(nN^2)$ orthogonalization, resulting in an $O(nm^2N^2)$ algorithm for returning uncorrelatedness. Now the correlation rank is likely to be lesser and still an upper bound. So with $N \to \infty$, we get exact independence. With a clever choice of functions (orthogonal functions, possibly, since they are maximally independent of each other), we may get very close to the number of independent components using a small N.

1.5. **Comments on Functions to Choose.** Immediate candidates are: the exponential family of functions with appropriate parameters to make them orthogonal; the orthogonal polynomial basis; most other orthogonal function bases. The exponential seems to be a good idea, and this will have to be pursued to a larger extent.

1.6. **A Final Comment On GS.** It seems like GS is the method that ICA uses to obtain independent components, with a clever choice of the functions f_i. However, the initialization is seemingly random. So the entire method is hinged upon the provision of an externally guessed number of independent components. The analysis by itself does not provide an upper bound for this number.

2. The Basic GS Algorithm for Uncorrelation

Let $X \in \mathbb{R}^{n,m}$, with $Y = \{y_i\}_1^m$ (y_i are data points represented as column vectors in Y). Let $\langle y \rangle$ denote the ensemble average of the components of y. Now define, for all i,

$$x_i = y_i - \langle y_i \rangle ,$$

and $X = \{x_i\}_1^m$. So x_i are the centered versions of y_i. Clearly, if x_i, x_j are independent, then they're uncorrelated and $\mathbb{E}[x_i x_j] = \mathbb{E}[x_i]\,\mathbb{E}[x_j] = 0$. Since the best estimate for the covariance between x_i, x_j is $x_i^T x_j/(n-1)$, we have that x_i, x_j are uncorrelated if and only if $x_i^T x_j = 0$, i.e. if and only if x_i, x_j are mutually orthogonal. So a natural way of determining the number of uncorrelated columns in X is just to orthogonalize X. We can do this with regular Gram-Schmidt, modified for numerical stability:

(1) For $i = 1 \to m$
   (a) if $||x_i|| < tol$, reject x_i and move on to the next column (just set $i = i + 1$).
   (b) $u_i = x_i / ||x_i||$,
   (c) For $j = i + 1 \to m$
   $$x_j = x_j - u_i u_i^T x_j$$
(2) Count the number of columns in X (some of them will be rejected in general) and declare this as the number of independent components.

2.1. **Note on deciding the tolerance.** One good way of coming up with a tolerance is to take the squared norms, and take the top ninety percent or so of the norms and throw out the bottom ten percent or so. For this, you cannot reject the columns as and when you compute their norms, because you don't know this cutoff yet till you've seen all the norms. So you'll have to store the norms and throw away columns at the very end based on which norms fall in the lowest ten percent.

3. The Souped-Up GS Algorithm For near-Independence

Find a function, $\phi : \mathbf{C} \to \mathbf{C}$, such that f is orthogonal to the identity function. Any fourier basis will work, really. So take $\phi = e^{ix/2\pi}$. Compute $H = \phi(X)$ such that

$$H(i,j) = \phi(x_j(i)) = \phi(X_{ij}),$$

and compute F from H just like we computed X from Y:

$$F = \{f_i\}_1^m, \quad f_i = h_i - \langle h_i \rangle , \quad h_i(j) = \phi(x_i(j)).$$

Basically, F contains columns which are centered versions of the corresponding columns in H. Now:

(1) For $i = 1 \to m$
   (a) if $||x_i|| < tol$ or if $||f_i|| < tol$, reject x_i, f_i and move on to the next column (just set $i = i + 1$).
   (b) $u_i = x_i / ||x_i||$,
   (c) $v_i = f_i / ||f_i||$,
   (d) For $j = i + 1 \to m$
      (i) $x_j = x_j - u_i u_i^T x_j - v_i v_i^T x_j$,
      (ii) $f_j = f_j - u_i u_i^T f_j - v_i v_i^T f_j$,
(2) Count the number of x_is (or f_is; either one will do since both are rejected if one of them is too small in norm) and declare this as the number of independent components.

3.1. **Note on deciding tolerance.** Similar to the previous version, it would help to have a distribution of norms which you can cut-off at the top ninety percent or so. For this, take all the norms of f_i, x_i, plot them and take the bottom ten percent. Throw away the vectors that have norms in this ten percent, but be sure to throw away the vector's partner too (i.e., if you discard x_i, be sure to discard f_i even if $||f_i||$ is appreciably large, and vice versa). That pretty much does it.

3.2. **A note on runtime.** The runtime of this algorithm only suffers a linear cost in the number of functions computed on the data vectors. Let's say N functions are computed (including the identity function which is computed by default). Then N orthogonalizations are performed at every step, one for x_j, one for each of $f_j^{(1)}, f_j^{(2)}, \cdots, f_j^{(n-1)}$, to give a total runtime of $O(nm^2N)$. We can even include two more fourier basis functions and suffer only a 4-fold increase in runtime for this step. This might be worth exploring if it doesn't take too long, to see if this reduces the number of declared independent components. Finally, note that this runtime is linearly dependent on N and not quadratically dependent on N as mentioned earlier in the document. The previous mention was based on an immediate and inefficient algorithm. This will be corrected later, in a more clean version.

4. Some additional comments

First, it makes sense to do all of this with any transform: the wavelet transform might work, but it might be even better to figure out a transform that is tailor-made for the data. There might be an iterative way to do this - I'm tossing this around in my head and will work out specifics sometime soon. The basic idea is: in the untransformed case, find the component closest to the seizure, orthogonalize starting with this component and use the resulting orthonormal system as a transform, on which you do ICA again, take the component closest to the seizure, orthogonalize starting with this component now, and so on. We can terminate the algorithm when the 'closeness' of an independent component to the seizure (as measured by the similarity metric I had sent you earlier) is below a certain threshold, or stops decreasing quickly enough.

If we do this, the point will not be to exactly single out the seizure component. In general, the rough seizure-component we pick out will have non-seizure activity too. But this activity will hopefully be predicted well by the predictor we learn based on the rest of the data. So the non-seizure component of the seizure data might act as the 'testing set' for the predictor we learn. Provided this predictor is good enough, the error distribution due to the non-seizure part mixed in to the seizure component will be acceptable. It is only the seizure-part of the seizure-component that'll throw off the predictor and help raise a seizure alarm. So the primary engine for seizure-alarm is still transfer learning; just that it is given a slightly higher octane fuel with the iterative method.

In any case, this iteration idea is something that will have to be considered a bit more carefully before thinking about its implementation.