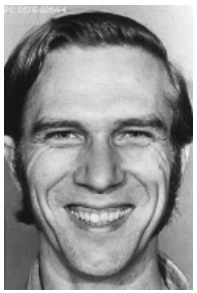
Project Algorithm Engineering: PCA

Maximilian Felde, Frank Nußbaum

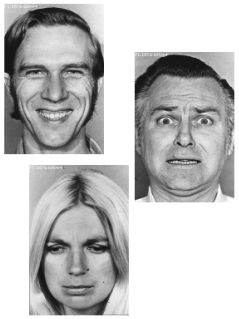
08.02.2016

Table of contents

Idea: Principal Component Analysis (PCA) and Classification



→ projection into the space of the k most important features → classification



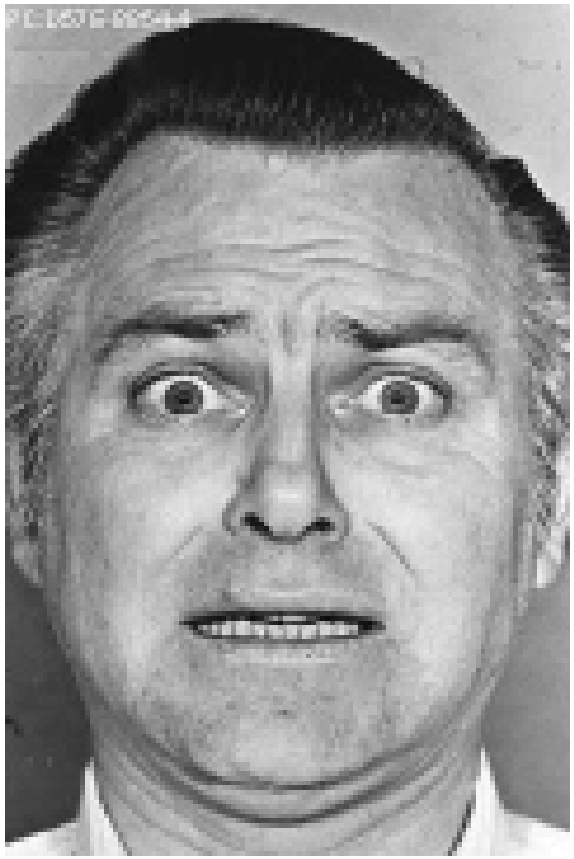
openCV $\longrightarrow$ $X = \begin{pmatrix} 1 & 0 & 1 & \dots \\ 0 & 0 & 1 & \dots \\ & \dots & & \dots \end{pmatrix}$ $\longrightarrow$ computation: empirical covariance matrix $(\frac{1}{n} X^t X - \mu^t \mu)$

QR-decomposition to get the k largest eigenvalues $\longrightarrow$ projection matrix from corresponding eigenvectors $U^t = \begin{pmatrix} -v_1^t- \\ -v_2^t- \\ \dots \end{pmatrix}$ $\longrightarrow$ projection of training data for future classification

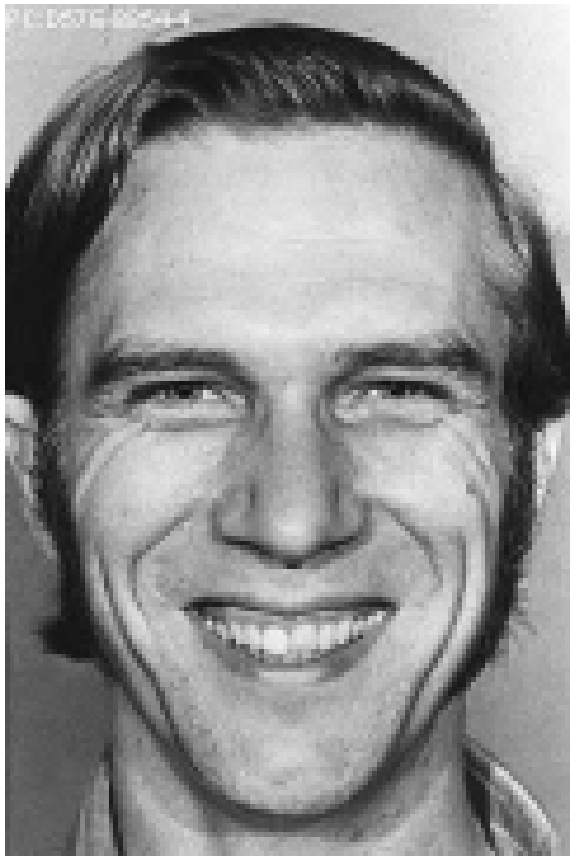
Examples: Eigenfaces

(a) first (b) second (c) third

Examples: Sample Reconstruction 1



Examples: Sample Reconstruction 2



Potential for Vectorization

```
#pragma omp simd aligned(A, A:64)
for (k=0;k<=j; ++k) /
    A[j*ldim+k] -= A[i*ldim+k];
```

- use of vectorization reports
- Padding, boost aligned vectors
- force vectorization if required

Potential for Vectorization

```
1     int const BLOCK_SIZE = 128;
2     int const M = links.get_anz_zeilen();
3     int const N = rechts.get_anz_spalten();
4     int const DIM_C = links.get_anz_spalten();
5
6     #pragma omp parallel for
7     for(int kk=0; kk< links.get_anz_spalten(); kk+=BLOCK_SIZE){
8       for(int jj=0; jj < N; jj+=BLOCK_SIZE){
9         for(int i=0; i < M; ++i){
10          for(int j=jj; j < std::min(jj+BLOCK_SIZE, N); ++j){
11            double sum = c[i*ldc+j];
12            #pragma omp simd aligned(a,b:64) reduction (+:sum)
13            for(int k= kk; k < std::min(kk + BLOCK_SIZE,DIM_C); ++k){
14              sum += a[i*lda+k]*b[j*ldb+k];
15            }
16            c[i*ldc+j]=sum;
17          }
18        }
19      }
20    }
```

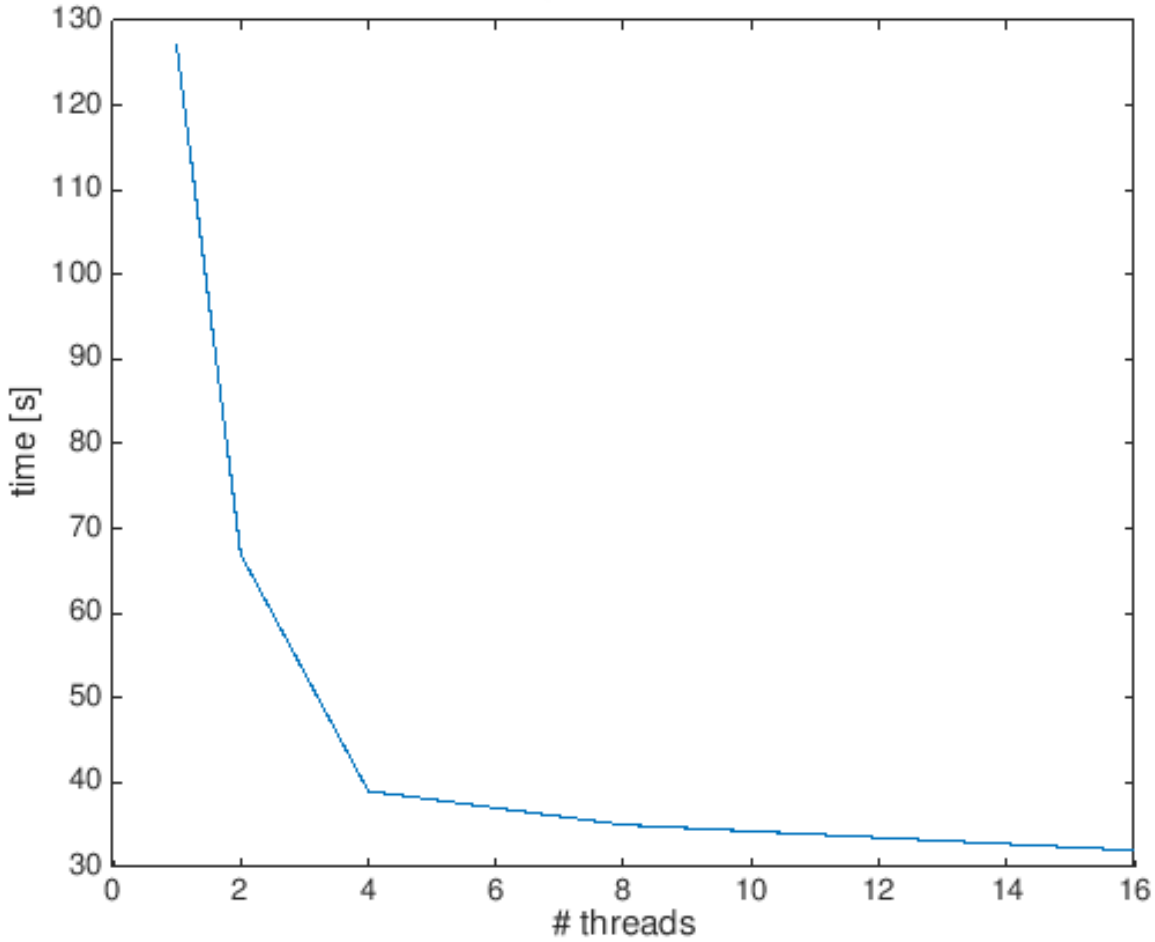
- and blockwise data access

Hessenberg: Parallelization using OpenMP

```
#pragma omp parallel default(none) private (j,...) shared(f, A, l, ...) if (l>2000)
{
                #pragma omp for schedule(static, 64) reduction (+:f)
                for (j=0;j<=l; ++j) {
                    f++;
                    // do more stuff, work per iteration O(l)
                }
                #pragma omp single
                // do some stuff only once
                #pragma omp for schedule(dynamic, 64)
                for (j=0;j<=l; ++j) {
                    // do stuff, work per iteration O(j)
                }

} // end parallel section
```

Hessenberg: Effect of Parallelization



Distribution using MPI

```
1   int init_data[2]{0,0}; // enthält gesamtgröße der matrix und anzahl spalten
2   double* buffer;
3   if(0==rank){
4         init_data[0]=U.size();
5         init_data[1]=nfeatures;
6         buffer = U.data();
7         MPI_Bcast(&init_data, 2, MPI_INT, 0, MPI_COMM_WORLD);
8         MPI_Bcast(buffer, init_data[0], MPI_DOUBLE, 0, MPI_COMM_WORLD);
9   }else{
10        MPI_Bcast(&init_data, 2, MPI_INT, 0, MPI_COMM_WORLD);
11        std::vector<double> buf(init_data[0]);
12        buffer = buf.data();
13        MPI_Bcast(buffer, init_data[0], MPI_DOUBLE, 0, MPI_COMM_WORLD);
14
15        projection = matrix(init_data[0]/init_data[1],init_data[1],false,buf);
16  }
```

- distribution of the projection matrix and the classifier

Demo and Conclusion

- it was a lot of hard work
- we know much more about programming in C++ ... and Git
- in particular our awareness for good vs. bad code has improved

Demo and Conclusion

- it was a lot of hard work
- we know much more about programming in C++ ... and Git
- in particular our awareness for good vs. bad code has improved

DEMO ◡̈