```cpp
#include <vector>
#include <bitset>
#include <fstream>
#include <sstream>
#include "LinearAlgebra.h"
#include "Counter.h"

#define DIM 28
#define TRAIN_SIZE 42000
#define TEST_SIZE 28000
#define MAX_ITERATIONS 2000 // tolerancia de iteraciones para el metodo de la potencia

typedef double Label;

typedef enum {
    KNN,
    PCA_KNN
} SolutionMethod;

/*
* Devuelve un número del 0 al 9 que representa el dígito reconocido
*/
Label kNN(int k, const Matrix &trainingSet, const std::vector<Label> &trainingLabels, Matrix &evSet, int i1,
const DistanceF &f) {
    Timer timer("kNN Timer");
    min_queue<std::pair<double, Label>> distances;

    for (int i = 0; i < trainingSet.rows(); ++i) {
        distances.push(std::make_pair(f(trainingSet, i, evSet, i1), trainingLabels[i]));
    }

    int i = 0;
    int labels[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

    while (!distances.empty() && i < k) {
        labels[(int) distances.top().second]++;
        distances.pop();
        ++i;
    }

    int maximum = 0;

    for (int j = 0; j < 10; ++j) {
        if (labels[j] > labels[maximum]) {
            maximum = j;
        }
    }

    return (double)maximum;
}

void loadTrainingSet(std::string path, Matrix &trainingSet, std::vector<Label> &trainingLabels) {
    Timer timer("Load Training Dataset Timer");

    std::fstream train(path + "train.csv", std::ios_base::in);
    // Obtenemos el header, así después el algorítmo sólo levanta los datos.
    std::string line;
    std::getline(train, line);

    // Se refiere a la linea del csv que estamos leyendo. O sea, a la imagen que estamos levantando.
    int l = 0;

    // Levantamos una linea del csv y la separamos por coma
    while (getline(train,line) && !train.eof() && !train.bad()) {
        // Buscamos cuál es el Label
        std::string::size_type prev = line.find_first_of(',');

        if (prev == std::string::npos) {
            std::stringstream fmt;
            fmt << "Label no encontrado en la linea " << l << " en el archivo de training";
            train.close();
            throw new std::invalid_argument(fmt.str());
        }

        Label lbl = line.substr(0, prev).c_str()[0] - 48;

        if (lbl > 9) {
            std::stringstream fmt;
```

```
78              fmt << "Label erroneo en la linea " << l << " en el archivo de training: " << lbl;
79              train.close();
80              throw new std::out_of_range(fmt.str());
81          }
82
83          trainingLabels[l] = lbl;
84
85          // Este contador es el número de pixel que estamos procesando en la imagen
86          int i = 0;
87
88          while (prev != std::string::npos) {
89              // Encontramos la próxima coma
90              std::string::size_type cur = line.find_first_of(',', prev + 1);
91
92              if (cur != std::string::npos) {
93                  trainingSet(l, i) = std::stod(line.substr(prev + 1, cur-(prev+1)));
94              }
95
96              prev = cur;
97              ++i;
98          }
99
100         ++l;
101     }
102
103     if (train.bad()) {
104         train.close();
105         throw new std::runtime_error("Error de lectura en el archivo de training");
106     }
107
108     train.close();
109 }
110
111 void loadTestingSet(std::string path, Matrix &testingSet) {
112     Timer timer("Load Testing Dataset Timer");
113
114     std::fstream test(path + "test.csv", std::ios_base::in);
115     // Obtenemos el header, así después el algorítmo sólo levanta los datos.
116     std::string line;
117     std::getline(test, line);
118
119     // Se refiere a la linea del csv que estamos leyendo. O sea, a la imagen que estamos levantando.
120     int l = 0;
121
122     // Levantamos una linea del csv y la separamos por coma
123     while (getline(test,line) && !test.eof() && !test.bad()) {
124         // Buscamos cuál es el primer pixel
125         std::string::size_type prev = line.find_first_of(',');
126         testingSet(l, 0) = std::stod(line.substr(0, prev));
127
128         // Este contador es el número de pixel que estamos procesando en la imagen
129         int i = 1;
130
131         while (prev != std::string::npos) {
132             // Encontramos la próxima coma
133             std::string::size_type cur = line.find_first_of(',', prev + 1);
134
135             if (cur != std::string::npos) {
136                 testingSet(l, i) = std::stod(line.substr(prev + 1, cur-(prev+1)));
137             }
138
139             prev = cur;
140             ++i;
141         }
142
143         ++l;
144     }
145
146     if (test.bad()) {
147         test.close();
148         throw new std::runtime_error("Error de lectura en el archivo de testing");
149     }
150
151     test.close();
152 }
153
154 template <std::size_t K>
155 std::pair<Matrix, std::vector<Label>> filterDataset(const Matrix &A, const std::vector<Label> &labels, const
```

```
    std::bitset<K> &filter) {
156     Timer timer("Filter Dataset Timer");
157 
158     if (K < labels.size() || filter.count() <= 1) {
159         std::stringstream fmt;
160         fmt << "Filtro para el dataset tiene tamaño " << K << " cuando el dataset tiene tamaño " <<
    labels.size();
161         throw new std::invalid_argument(fmt.str());
162     }
163 
164     std::vector<Label> output(filter.count(), 0.0);
165     int last = 0;
166 
167     for (int i = 0; i < A.rows(); ++i) {
168         if (filter.test((std::size_t) i)) {
169             output[last] = labels[i];
170             last++;
171         }
172     }
173 
174     return std::pair<Matrix, std::vector<Label>>(Matrix(A, filter), output);
175 }
176 
177 void PCAKNN(std::string path, std::string output, std::string append, int alpha, int neighbours, int tests,
    std::vector<std::bitset<TRAIN_SIZE>> &masks,
178             Matrix &trainingSet, std::vector<Label> &trainingLabels, Matrix &testingSet, std::vector<Label>
    &predictions) {
179     std::cerr << "Comenzando kNN con PCA para las particiones" << std::endl;
180 
181     for (int k = 0; k < tests; ++k) {
182         std::cerr << "Procesando partición " << k << std::endl;
183 
184         std::pair<Matrix, std::vector<Label>> fTrain = filterDataset(trainingSet, trainingLabels, masks[k]);
185         std::cerr << "Casos de train: " << masks[k].count() << std::endl;
186         std::cerr << "Casos de train concretos: " << fTrain.first.rows() << std::endl;
187 
188         masks[k].flip();
189         std::pair<Matrix, std::vector<Label>> fTest = filterDataset(trainingSet, trainingLabels, masks[k]);
190         std::cerr << "Casos de test: " << masks[k].count() << std::endl;
191         std::cerr << "Casos de test concretos: " << fTest.first.rows() << std::endl;
192 
193         Timer timer("kNN Partition Timer");
194         std::cerr << "Calculando promedio de variables para la particion " << k << std::endl;
195 
196         Matrix mean(1, DIM*DIM);
197 
198         Timer PCAMean("PCA Mean CV");
199 
200         // TODO: Seria util que la propia matriz tenga una funcion que te de el vector promedio. La
    sumatoria la va generando cada vez que se cambian las filas.
201         for (int i = 0; i < fTrain.first.rows(); i++) {
202             for (int j = 0; j < fTrain.first.columns(); j++) {
203                 mean(0,j) += fTrain.first(i,j);
204             }
205         }
206 
207         for (int j = 0; j < mean.columns();j++)
208             mean(0,j) /= fTrain.first.rows();
209 
210 
211         PCAMean.stop();
212 
213         Timer PCANormalizeTraining("PCA Normalize CV");
214 
215         for (int j = 0; j < fTrain.first.columns(); ++j) {
216             for (int i = 0; i < fTrain.first.rows(); ++i) {
217                 fTrain.first(i, j) -= mean(0, j);
218                 fTrain.first(i, j) /= sqrt(fTrain.first.rows()-1);
219             }
220         }
221 
222         PCANormalizeTraining.stop();
223 
224         Matrix covariance(fTrain.first.columns(), fTrain.first.columns());
225         std::string covFileName = path + "covariance" + std::to_string(k) + append;
226         std::fstream inCov(covFileName, std::ios_base::in);
227 
228         if (inCov.good()) {
```

```
229             std::cerr << "Levantando matriz de covarianza para la partición " << k << " del training
    dataset" << std::endl;
230             inCov >> covariance;
231             inCov.close();
232         } else {
233             std::cerr << "Calculando matriz de covarianza la partición " << k << " training dataset" <<
    std::endl;
234             Timer PCACov("PCA Covariance CV");
235 
236             for (int i = 0; i < covariance.columns(); i++) {
237                 if (i % 10 == 0) {
238                     std::cerr << "Progreso: " << i << "/" << covariance.columns() << '\r';
239                 }
240 
241                 for (int j = 0; j <= i; j++) {
242                     // j es la columna de X_t, que resulta ser la fila j-esima de X
243                     for (int k = 0; k < fTrain.first.rows(); k++) {
244                         covariance(i, j) += fTrain.first(k, i) * fTrain.first(k, j);
245                     }
246 
247                     covariance(j, i) = covariance(i, j);
248                 }
249             }
250 
251             std::cerr << std::endl;
252 
253             PCACov.stop();
254 
255             std::fstream outCov(covFileName, std::ios_base::out);
256 
257             if (outCov.good()) {
258                 std::cerr << "Guardando matriz de covarianza para la partición " << k << " del training
    dataset" << std::endl;
259                 outCov << covariance;
260                 outCov.close();
261             } else {
262                 std::cerr << "Error guardando la matriz de covarianza, siguiendo igualmente" << std::endl;
263             }
264         }
265 
266         std::cerr << "Calculando los autovalores de la matriz de covarianzas" << std::endl;
267 
268         // Obtenemos los autovalores y autovectores de la matriz de covarianzas
269         std::list<EigenPair> eigenPair = decompose(covariance, alpha, N2, MAX_ITERATIONS);
270 
271         std::fstream eigenvalues(output, std::ios_base::out | std::ios_base::app);
272 
273         if (eigenvalues.good()) {
274             std::cerr << "Guardando autovalores para los test." << std::endl;
275             eigenvalues.precision(6);
276 
277             for (const EigenPair& ep : eigenPair) {
278                 eigenvalues << std::scientific << std::sqrt(ep.first) << std::endl;
279             }
280 
281             eigenvalues.close();
282         } else {
283             std::cerr << "Error guardando los autovalores, siguiendo igualmente" << std::endl;
284             std::cerr << "ALERTA: LOS TESTS NO VAN A PASAR" << std::endl;
285         }
286 
287         std::cerr << "Haciendo cambio de base para el training dataset" << std::endl;
288         Matrix trainChangeBasis(fTrain.first.rows(), alpha);
289         // En este paso vamos a realizar un cambio de espacio a todos los vectores
290         dimensionReduction(fTrain.first, trainChangeBasis, eigenPair);
291 
292         // a cada imagen del testing set debemos restarle mean y dividirlos por sqrt(fTrain.first.rows()-1)
293         // segun diapositivas de la clase.
294         Timer PCANormalizeTesting("PCA Normalize Testing CV");
295 
296         for (int i = 0; i < fTest.first.rows(); i++) {
297             for (int j = 0; j < fTest.first.columns(); j++) {
298                 fTest.first(i, j) -= mean(0, j);
299                 fTest.first(i, j) /= std::sqrt(fTrain.first.rows()-1);
300             }
301         }
302 
303         PCANormalizeTesting.stop();
```

```
304
305         std::cerr << "Haciendo cambio de base para el testing dataset" << std::endl;
306         Matrix testChangeBasis(fTest.first.rows(), alpha);
307         dimensionReduction(fTest.first, testChangeBasis, eigenPair);
308
309         // ya tenemos los vectores en sus respectivos cambios de bases
310         Counter hit("kNN Hit CV");
311         Counter miss("kNN Miss CV");
312         Counter total("kNN Total CV");
313         Timer kNNPartitionTimer("kNN Partition Timer CV");
314
315         std::cerr << "Corriendo kNN" << std::endl;
316
317         for (int i = 0; i < testChangeBasis.rows(); ++i) {
318             Label l = kNN(neighbours, trainChangeBasis, fTrain.second, testChangeBasis, i, L2);
319
320             if (l == fTest.second[i]) {
321                 ++hit;
322             } else {
323                 ++miss;
324             }
325
326             ++total;
327
328             if (i % 100 == 0) {
329                 std::cerr << "Progreso: " << i << "/" << testChangeBasis.rows() << '\r';
330             }
331         }
332
333         std::cerr << std::endl;
334     }
335
336     Matrix mean(1, DIM*DIM);
337
338     Timer PCAMean("PCA Mean");
339
340     for (int i = 0; i < trainingSet.rows(); i++) {
341         for (int j = 0; j < trainingSet.columns(); j++) {
342             mean(0,j) += trainingSet(i,j);
343         }
344     }
345
346     for (int j = 0; j < mean.columns();j++)
347         mean(0,j) /= trainingSet.rows();
348
349
350     PCAMean.stop();
351
352     Timer PCANormalizeTraining("PCA Normalize Training");
353
354     for (int j = 0; j < trainingSet.columns(); ++j) {
355         for (int i = 0; i < trainingSet.rows(); ++i) {
356             trainingSet(i, j) -= mean(0, j);
357             trainingSet(i, j) /= sqrt(trainingSet.rows()-1);
358         }
359     }
360
361
362     PCANormalizeTraining.stop();
363
364     Matrix covariance(trainingSet.columns(), trainingSet.columns());
365     std::string covFileName = path + "covariance" + append;
366     std::fstream inCov(covFileName, std::ios_base::in);
367
368     if (inCov.good()) {
369         std::cerr << "Levantando matriz de covarianza para el training dataset" << std::endl;
370         inCov >> covariance;
371         inCov.close();
372     } else {
373         std::cerr << "Calculando matriz de covarianza del training dataset" << std::endl;
374         Timer PCACov("PCA Covariance");
375
376         for (int i = 0; i < covariance.columns(); i++) {
377             if (i % 10 == 0) {
378                 std::cerr << "Progreso: " << i << "/" << covariance.columns() << std::endl;
379             }
380
381             for (int j = 0; j <= i; j++) {
```

```
382                 // j es la columna de X_t, que resulta ser la fila j-esima de X
383                 for (int k = 0; k < trainingSet.rows(); k++) {
384                     covariance(i, j) += trainingSet(k, i) * trainingSet(k, j);
385                 }
386                 covariance(j, i) = covariance(i, j);
387             }
388         }
389 
390         PCACov.stop();
391 
392         std::fstream outCov(covFileName, std::ios_base::out);
393 
394         if (outCov.good()) {
395             std::cerr << "Guardando matriz de covarianza para el training dataset" << std::endl;
396             outCov << covariance;
397             outCov.close();
398         } else {
399             std::cerr << "Error guardando la matriz de covarianza, siguiendo igualmente" << std::endl;
400         }
401     }
402 
403     std::cerr << "Calculando los autovalores de la matriz de covarianzas" << std::endl;
404     // Obtenemos los autovalores y autovectores de la matriz de covarianzas
405     std::list<EigenPair> eigenPair = decompose(covariance, alpha, N2, MAX_ITERATIONS);
406 
407     std::fstream eigenvalues(output, std::ios_base::out | std::ios_base::app);
408 
409     if (eigenvalues.good()) {
410         std::cerr << "Guardando autovalores para los test." << std::endl;
411         eigenvalues.precision(6);
412 
413         for (const EigenPair& ep : eigenPair) {
414             eigenvalues << std::scientific << std::sqrt(ep.first) << std::endl;
415         }
416 
417         eigenvalues.close();
418     } else {
419         std::cerr << "Error guardando los autovalores, siguiendo igualmente" << std::endl;
420         std::cerr << "ALERTA: LOS TESTS NO VAN A PASAR" << std::endl;
421     }
422 
423     std::cerr << "Haciendo cambio de base para el training dataset" << std::endl;
424     Matrix trainChangeBasis(trainingSet.rows(), alpha);
425     // En este paso vamos a realizar un cambio de espacio a todos los vectores
426     dimensionReduction(trainingSet, trainChangeBasis, eigenPair);
427 
428     // a cada imagen del testing set debemos restarle mean y dividirlos por sqrt(testingSet.rows()-1)
429     // segun diapositivas de la clase.
430     Timer PCANormalizeTesting("PCA Normalize Testing");
431 
432     for (int i = 0; i < testingSet.rows(); i++) {
433         for (int j = 0; j < testingSet.columns(); j++) {
434             testingSet(i, j) -= mean(0, j);
435             testingSet(i, j) /= std::sqrt(trainingSet.rows()-1);
436         }
437     }
438 
439     PCANormalizeTesting.stop();
440 
441     std::cerr << "Haciendo cambio de base para el testing dataset" << std::endl;
442     Matrix testChangeBasis(testingSet.rows(), alpha);
443     dimensionReduction(testingSet, testChangeBasis, eigenPair);
444 
445     Counter total("kNN Total");
446     Timer kNNPartitionTimer("kNN Testing Timer");
447 
448     std::cerr << "Corriendo kNN" << std::endl;
449 
450     for (int i = 0; i < testChangeBasis.rows(); ++i) {
451         Label l = kNN(neighbours, trainChangeBasis, trainingLabels, testChangeBasis, i, L2);
452         predictions[i] = l;
453 
454         ++total;
455 
456         if (i % 100 == 0) {
457             std::cerr << "Progreso: " << i << "/" << testChangeBasis.rows() << '\r';
458         }
459     }
```

```
460
461     std::cerr << std::endl;
462 }
463
464 void NORMALKNN(int neighbours, int tests, std::vector<std::bitset<TRAIN_SIZE>> &masks,
465                Matrix &trainingSet, std::vector<Label> &trainingLabels, Matrix &testingSet,
    std::vector<Label> &predictions) {
466     std::cerr << "Comenzando kNN para las particiones" << std::endl;
467
468     for (int k = 0; k < tests; ++k) {
469         std::cerr << "Procesando partición " << k << std::endl;
470
471         std::pair<Matrix, std::vector<Label>> fTrain = filterDataset(trainingSet, trainingLabels, masks[k]);
472         std::cerr << "Casos de train: " << masks[k].count() << std::endl;
473         std::cerr << "Casos de train concretos: " << fTrain.first.rows() << std::endl;
474
475         masks[k].flip();
476         std::pair<Matrix, std::vector<Label>> fTest = filterDataset(trainingSet, trainingLabels, masks[k]);
477         std::cerr << "Casos de test: " << masks[k].count() << std::endl;
478         std::cerr << "Casos de test concretos: " << fTest.first.rows() << std::endl;
479
480         Counter hit("kNN Hit CV");
481         Counter miss("kNN Miss CV");
482         Counter total("kNN Total CV");
483         Timer kNNPartitionTimer("kNN Partition Timer");
484
485         for (int i = 0; i < fTest.first.rows(); ++i) {
486             Label l = kNN(neighbours, fTrain.first, fTrain.second, fTest.first, i, L2);
487
488             if (l == fTest.second[i]) {
489                 ++hit;
490             } else {
491                 ++miss;
492             }
493
494             ++total;
495
496             if (i % 100 == 0) {
497                 std::cerr << "Progreso: " << i << "/" << fTest.first.rows() << '\r';
498             }
499         }
500
501         std::cerr << std::endl;
502     }
503
504     std::cerr << "Comenzando kNN para el testing dataset" << std::endl;
505
506     Timer kNNTestingTimer("kNN Testing Timer");
507
508     for (int i = 0; i < testingSet.rows(); ++i) {
509         Label l = kNN(neighbours, trainingSet, trainingLabels, testingSet, i, L2);
510         predictions[i] = l;
511
512         if (i % 100 == 0) {
513             std::cerr << "Progreso: " << i << "/" << testingSet.rows() << '\r';
514         }
515     }
516
517     std::cerr << std::endl;
518 }
519
520 std::string basename(const std::string &pathname) {
521     return std::string(std::find_if( pathname.rbegin(), pathname.rend(), [](char ch) -> bool { return ch ==
    '/'; }).base(), pathname.end());
522 }
523
524 int main(int argc, char *argv[]) {
525     Timer programTimer("Program Timer");
526     SolutionMethod method = SolutionMethod::KNN;
527
528     if (argc < 3) {
529         std::cerr << "Faltan argumentos." << std::endl;
530         return 0;
531     }
532
533     if (*argv[3] == '1') {
534         method = SolutionMethod::PCA_KNN;
535     }
```

```
536
537     std::cerr << "Input file: " << argv[1] << std::endl;
538     std::cerr << "Output file: " << argv[2] << std::endl;
539     std::cerr << "Basename: " << basename(std::string(argv[2])) << std::endl;
540
541     // Levantamos los datos de entrada al programa
542     std::string path;
543     int neighbours, alpha, tests;
544     std::fstream input(argv[1], std::ios_base::in);
545
546     input >> path >> neighbours >> alpha >> tests;
547
548     std::cerr << "Input path: " << path << std::endl;
549     std::cerr << "Neighbours: " << neighbours << std::endl;
550     std::cerr << "Alpha: " << alpha << std::endl;
551     std::cerr << "Tests: " << tests << std::endl;
552
553     // Levantamos las mascaras para definir el training set y el test set
554     std::vector<std::bitset<TRAIN_SIZE>> masks((unsigned long) tests);
555
556     for (int i = 0; i < tests; ++i) {
557         for (int j = 0; j < TRAIN_SIZE; ++j) {
558             bool in = false;
559             input >> in;
560             masks[i][j] = in;
561         }
562     }
563
564     input.close();
565
566     Matrix trainingSet(TRAIN_SIZE, DIM*DIM);
567     std::vector<Label> trainingLabels(TRAIN_SIZE, 0.0);
568     loadTrainingSet(path, trainingSet, trainingLabels);
569
570     Matrix testingSet(TEST_SIZE, DIM*DIM);
571     loadTestingSet(path, testingSet);
572
573     std::vector<Label> predictions(TEST_SIZE, 0.0);
574
575     if (method == PCA_KNN) {
576         PCAKNN(path, std::string(argv[2]), basename(std::string(argv[2])), alpha, neighbours, tests, masks,
    trainingSet, trainingLabels, testingSet, predictions);
577     } else {
578         NORMALKNN(neighbours, tests, masks, trainingSet, trainingLabels, testingSet, predictions);
579     }
580
581     Timer timer("Output Dataset Timer");
582
583     std::fstream output(path + "clasification" + basename(std::string(argv[2])), std::ios_base::out);
584     output << "ImageId,Label" << std::endl;
585
586     for (int i = 0; i < predictions.size(); ++i) {
587         output << i + 1 << "," << predictions[i] << std::endl;
588     }
589
590     output.close();
591     timer.stop();
592
593     Logger::getInstance().dump(path + "statistics" + basename(std::string(argv[2])));
594
595     return 0;
596 }
```