

Курсов проект

по Изкуствен Интелект

**Тема:**

Разпознаване на цифри (digit recognizer)

Изготвен от:

Даниел Делчев, фн:81211, Компютърни Науки,

4 курс, 2 поток, 6 група

Зимен семестър 2017/2018г

Съдържание

[Мотивация, Задача на курсовата работа 3](#_Toc506588089)

[Кратък обзор 3](#_Toc506588090)

[Моето решение 5](#_Toc506588091)

[Защо ни е тази трансформация? 6](#_Toc506588092)

[Програмна реализация 7](#_Toc506588093)

[Резултати 8](#_Toc506588094)

[Заключение 9](#_Toc506588095)

[Декларация за липса на плагиатство 10](#_Toc506588096)

[Библиография: 11](#_Toc506588097)

# Мотивация, Задача на курсовата работа

Настоящата задача се състои в това да се разпознаят цифри по стойностите на сивотата на пикселите в техните изображения. Подробно описание може да се намери на [[1]](#_Библиография:).

За данни са използвани предварително обработени данни от kaggle. Данните представляват greyscalе(дава информация само за количеството светлина и интензитед) изображения 28x28 пиксела.Всеки пиксел е представен с интензитед (0-255).

Въпреки това в течението на разработка бе приложена и трансформация на данните, със спорни резултати описани в текущата документация.

Данните са вариация на прословутото множество данни MNIST [[2]](#_Библиография:). Върху него са прилагани доста различни алгоритми с различни хиперпараметри. От [[ 2]](#_Библиография:) , както и от самите резултати в състезанието в kaggle[[1]](#_Библиография:) се вижда, че най-успешни за задачата са конвулационните невронни мрежи.

В текущата работа, обаче бе решено да се използва по-простоват алгоритъм в няколко различни форми – KNN , както и един прийом за смяна на векторният базис на данните и намаляване на броя характеристики (PCA [[5]](#_Библиография:) [[6]](#_Библиография:) [[7]](#_Библиография:)). Целта на курсовата работа е не да опровергае фaкта, че конвулационните невронните мрежи се справят изключително добре при разпознаване на изображения, а да провери до каква степен и по-простoват алгоритъм се справя със задачата. В имплементацията е заложено максимално на собствени имплементации. Все пак за пресмятане на собствените вектори и собствени стойности (eigen vectos, eigen values) на коварационната матрица от пикселите на изображенията, са използвани две библиотеки за линейна алгебра [[4]](#_Библиография:). Въпреки това имплементацията е сравнително експлицитна (за сравнение при използване нa sklearn библиотекта за python, имплементацията на целия проект не би надвишила 50 реда код [9], но такава имплементация би била изключително имплицитна и методите в нея биха били черни кутии.)

# Кратък обзор

Както вече бе споменато, данните, които са използвани са вариация на MNIST , което е hello world множество от данни за разпознаване на изображение. Съответното състезание от kaggle[[1]](#_Библиография:)  също е популярно, така че за към конкретната задача са прилагани доста решения.

На [[2]](#_Библиография:) както и на дискусионните форуми в kaggle могат да се намерят доста от тях.

Някои примери са:

|  |  |  |  |
| --- | --- | --- | --- |
| CLASSIFIER | PREPROCESSING | TEST ERROR RATE (%) | Reference |
| **Linear Classifiers** | | | |
| linear classifier (1-layer NN) | none | 12.0 | [LeCun et al. 1998](http://yann.lecun.com/exdb/publis/index.html#lecun-98) |
| linear classifier (1-layer NN) | deskewing | 8.4 | [LeCun et al. 1998](http://yann.lecun.com/exdb/publis/index.html#lecun-98) |
| pairwise linear classifier | deskewing | 7.6 | [LeCun et al. 1998](http://yann.lecun.com/exdb/publis/index.html#lecun-98) |
| **K-Nearest Neighbors** | | | |
| K-nearest-neighbors, Euclidean (L2) | none | 5.0 | [LeCun et al. 1998](http://yann.lecun.com/exdb/publis/index.html#lecun-98) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Non-Linear Classifiers** | | | |
| 40 PCA + quadratic classifier | none | 3.3 | [LeCun et al. 1998](http://yann.lecun.com/exdb/publis/index.html#lecun-98) |
| 1000 RBF + linear classifier | none | 3.6 | [LeCun et al. 1998](http://yann.lecun.com/exdb/publis/index.html#lecun-98) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Neural Nets** | | | |
| 2-layer NN, 300 hidden units, mean square error | none | 4.7 | [LeCun et al. 1998](http://yann.lecun.com/exdb/publis/index.html#lecun-98) |
| 2-layer NN, 300 HU, MSE, [distortions] | none | 3.6 | [LeCun et al. 1998](http://yann.lecun.com/exdb/publis/index.html#lecun-98) |
| 2-layer NN, 300 HU | deskewing | 1.6 | [LeCun et al. 1998](http://yann.lecun.com/exdb/publis/index.html#lecun-98) |
| 2-layer NN, 1000 hidden units | none | 4.5 | [LeCun et al. 1998](http://yann.lecun.com/exdb/publis/index.html#lecun-98) |
| 2-layer NN, 1000 HU, [distortions] | none | 3.8 | [LeCun et al. 1998](http://yann.lecun.com/exdb/publis/index.html#lecun-98) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Convolutional nets** | | | |
| Convolutional net LeNet-1 | subsampling to 16x16 pixels | 1.7 | [LeCun et al. 1998](http://yann.lecun.com/exdb/publis/index.html#lecun-98) |
| Convolutional net LeNet-4 | none | 1.1 | [LeCun et al. 1998](http://yann.lecun.com/exdb/publis/index.html#lecun-98) |
| Convolutional net LeNet-4 with K-NN instead of last layer | none | 1.1 | [LeCun et al. 1998](http://yann.lecun.com/exdb/publis/index.html#lecun-98) |
| Convolutional net LeNet-4 with local learning instead of last layer | none | 1.1 | [LeCun et al. 1998](http://yann.lecun.com/exdb/publis/index.html#lecun-98) |
| Convolutional net LeNet-5, [no distortions] | none | 0.95 | [LeCun et al. 1998](http://yann.lecun.com/exdb/publis/index.html#lecun-98) |
| Convolutional net LeNet-5, [huge distortions] | none | 0.85 | [LeCun et al. 1998](http://yann.lecun.com/exdb/publis/index.html#lecun-98) |

# Моето решение

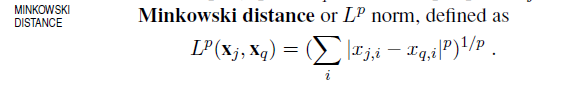
За решението на задачата е използван KNN алгоритъма.

Използвани са две негови разновидности – претеглена, и с краен брой съседи, като се е ескпериментирало с теглата, броя съседи и мярката за разстояние. Поначало са използвани директно данните от kaggle, а в последствие чрез PCA е сменен базиса на данните и са направени няколко опита с новите данни.

Успеваемостта на алгоритъма е измерена от броя правилно класифицирани примери. Измервана е върху тестовото множество, както и с крос валидация с 6 части при някои случай.

Считаме , че читателят е пределно запознат с алгоритъма KNN. Все пак отбелязваме , че начина на определяне на разстоянието от един запис(точка) до друг е от съществено значение.

В решението е използванo



С различни стойности на p.

За претегленият вариант на алгоритъма , в който участват всички съседи са използвани тегла от вида

**(1/dist(I,j))^eps**

Тоест тегла обратно пропорционални на разстоянието, повдигнати на някаква степен.

Тъй като отклонението в теглата между точките е сравнително малко, то е нужно повдигане на реципрочното на разстоянието на степен , като дори и при повдигане на 10 степен, успеваемостта на алгоритъма е ниска.

В последствие бе решено да се приложи PCA (Principal Component Analysis) за смяна на базиса на входните данни и евентуално намаляване на броя характеристики. Алгоритъма по извършване нa PCA е следният:

1) Тренировъчните данни се центрират (от всяка характеристика на всеки запис се изважда средната стойност за тази характеристика от всички записи).

2) Пресмята се ковариацонната матрица от характеристиките на центрираното тренировъчно множество.

3) намират се собствените вектори и собствените стойности (eigen vectors, eigen values) на ковариацонната матрица. (за тази част от алгоритъма в имплементацията е използвана библиотека за линейна алгебра [[4]](#_Библиография:) )

4) сортират се собствените вектори, спрямо съответните им собствени стойности и се поставят като вектор стълбове в нова матрица E.

5) след като имаме E, то всеки вектор от пространството може да се изрази в базиса от собствените вектори по следният начин :

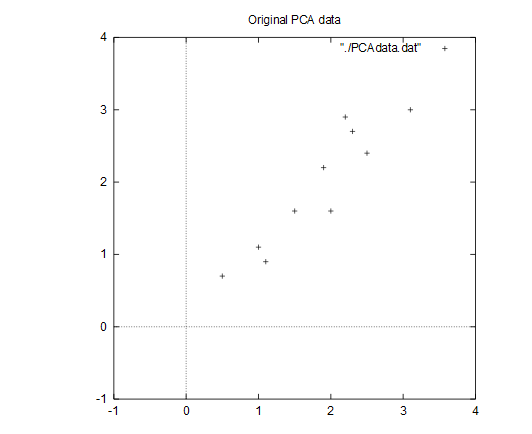
**(E)t x vt = v’t**

Тоест чрез умножение на транспонираната матрица Е по трнспонирания вектор v получаваме транспонирания вектор v’ отговарящ на v ,представен в базиса на собствените вектори.

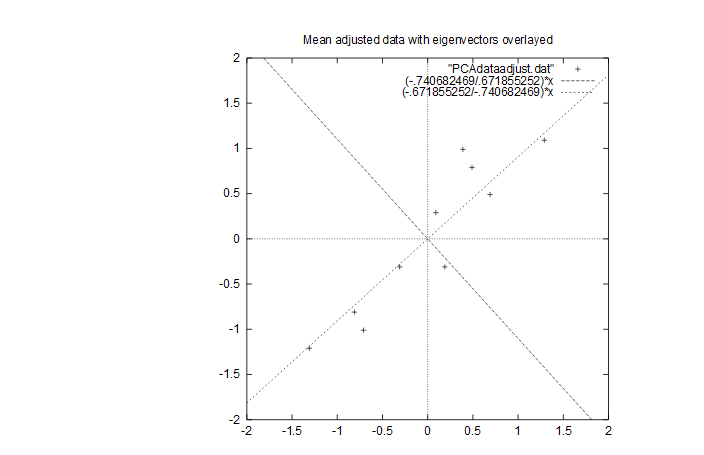
[[5]](#_Библиография:)

## Защо ни е тази трансформация?

Нека имаме следните двумерни данни:



Явно е, че те не са особено пригодени за конкрентата координатна система.Представни в базис намерен от PCA ,те биха изглеждали така.



Тоест базисът от собствените вектори от PCA описва много по-добре данните. Тоест чрез PCA си осигуряваме по-добри характеристики , линейни комбинации от изходните. Още повече , оказва се, че собствените вектори с по-високи собствени стойности са много по-значими за данните, отколкото тези с по-ниски стойности. Тоест можем да намалим броя координати на точките (броя характеристики на записите), като запазим само част от собствените вектори при трансформацията – тези с най-високи собствени стойности. По този начин намаляваме пространството , като губим само малка част от информацията.

Мотивът за прилагане на PCA бе да се открият по-ясни характеристики ,отколкото чистите пикслеи на изображението, както и да се провери как намалянето на размерността на точките (характеристиките на записите) влияе на успеваемостта както и н производителността на програмата.

# Програмна реализация

Програмната реализация до голяма степен е продиктувана от всичко казано дотук. Проекта се състои от два изпълними файла, съставени от файловете knn.cpp и pca.cpp . Първият извършва предвиждането за тестовото множество било с класически Knn или притеглен, както и крос-валидации . Изпълнимият файл приема входни аргументи и изпълнението е параметризирано спрямо изплозваната L-норма, броя съседи, броя характеристики (ако се използват данни обработени с pca) които се използват и други.

Вторият изпълним файл произлизащ от pca.cpp също е параметризиран, той приема тренировъчно и тестово множество, и ги обръща в базис на собствените вектори на ковариационната матрица на характеристиките на тренировъчното множество.

Имплементацията на претегленият и класическият KNN е стандартна, а имплементацията на pca следва описания алгоритъм по-рано, като за стъпката на пресмятане на собствените вектори и стойности се ползва външна билбиотека [[4]](#_Библиография:).

# Резултати

Крос валидациите са извършвани на 6 части върху цялото тренировъчно множество на 24 ядрен процесор. Превиждането на тестовото множествно също е извършвано с 24 нишки. (Това има отношение относно производителността, която е засегната по-доло).

|  |  |  |  |
| --- | --- | --- | --- |
| Части на крос-валиация | К | L-норма разстояние | успеваемост |
| 6 | 11 | 2 | 0.963262 |
| 6 | 11 | 3 | 0.965714 |
| 6 | 11 | 4 | 0.966476 |
| 6 | 11 | 5 | 0.967881 |
| 6 | 11 | 6 | 0.968119 |
| 6 | 11 | 7 | 0.968357 |
| **6** | **11** | **8** | **0.968929** |
| 6 | 11 | 9 | 0.968762 |
| 6 | 11 | 10 | 0.968190 |
| **6** | **11** | **11** | **0.969000** |
| 6 | 11 | 12 | 0.968762 |
| **6** | **4** | **8** | **0.972714** |
| 6 | 6 | 8 | 0.971310 |
| 6 | 8 | 8 | 0.970857 |
| 6 | 10 | 8 | 0.968810 |
| 6 | 12 | 8 | 0.968000 |
| 6 | 4 | 3 | 0.968952 |
| 6 | 4 | 5 | 0.971095 |
| **6** | **4** | **7** | **0.972119** |
| 6 | 4 | 9 | 0.971548 |
| **6** | **4** | **11** | **0.972405** |

Подчертаните резултати са сметнати за добри и съответните параметри също.

Относно претегленият KNN следните резултати диктуваха той да не се изследва повече

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| K | L-норма | Време (сек) | тегло | успеваемост |
| всички | 3 | 3691 | (1/dist)^10 | 0.84557 |
| всички | 8 | 3897 | (1/dist)^10 | ~ 0.75 |

|  |  |  |  |
| --- | --- | --- | --- |
| К | L-норма | Време (сек) | успеваемост |
| **3** | **5** | **3600** | **0.97285** |
| 4 | 8 | 3700 | 0.97257 |
| 11 | 3 | 3999 | 0.96600 |
| 11 | 4 | 3972 | 0.96828 |
| 11 | 8 | 3951 | 0.97085 |
| 11 | 11 | 3571 | 0.97028 |
| 30 | 3 | 3881 | 0.95714 |

Относно данни получени с PDA се получиха следните резултати:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| К | L-норма | Брой компоненти  (характеристики) | Време(сек) | успеваемост |
| 3 | 5 | 256 | 2789 | 0.97171 |
| 3 | 5 | всички | 4923 | 0.97100 |

# Заключение

Проекта разгледа способностите на простоват алгоритъм KNN за решаване на поставената задача. Резултатите показват , че при добри хипер параметри класическият KNN (непретегленият) се справя задоволителнно с успеваемост около 97% . Претегленият KNN не се представя добре. KNN приложен върху, данни над, които е бил приложен PCA не се представят по-добре от KNN върху стандартните данни, но при използване само на част от характеристиките след PCA, производителността се подобравя, без да намалее осезамо успеваемостта – тя отново е около 97%.

Сравнение с други алгоритми постигащи около 98.5% успеваемост като конвулационни невронни мрежи, KNN не се представпя особено по-зле, въпреки, че разлика от 1.5% все пак е респектираща.

Значи ли това, че „не е нужно да се ходи на лов с топ за патици” ?

-Не, въпреки, че KNN е едно лигитимно решение за класификационната задача, което дава доста добър резултат на успеваемост, той е мързеливо учещ се алгоритъм. Предвижданията му са бавни.

Обръщаме внимание, че за предвиждането на 28000 записа тестово множество с 42000 записа тренировъчно множествно с 24 нишки на 24 ядрен процесор е нужно около час. Тоест една нишка предвижда средно 1166 записа на час, тоест около 3 в секунда. В реална ситуация 0.3 секунди за предвиждане на един единствен запис, при положение , че тези 0,3 секунди не включват време за предварителна обработка на изображение са прекалено много.

При използване на PCA и намаляване на броя характеристики от 784 на 256, успеваемостта е около 97.1%, а времето нужно за предвижданена тестовото множество при същите условия е около 45 мин. Тоест малко по-добре, но все още доста бавно.

В крайна смета извода е ,че KNN е пригоден за задачата откъм успеваемост, но ако откъм производителност, той е приложим ,само ако не е нужно предвужданията да се случват в реално време.

# Декларация за липса на плагиатство

Тази курсова работа е моя работа, като всички изречения, илюстрации и програми от други хора са изрично цитирани.

Тази курсова работа или нейна версия не са представени в друг университет или друга учебна институция.

Разбирам, че ако се установи плагиатство в работата ми ще получа оценка “Слаб”.

Даниел Николаев Делчев

.....................................

# Библиография:

* [[1]Kagle състезание](https://www.kaggle.com/c/digit-recognizer) [https://www.kaggle.com/c/digit-recognizer]
* [[2] MNIST](http://yann.lecun.com/exdb/mnist/index.html) [http://yann.lecun.com/exdb/mnist/index.html]
* [[3] C++ for Mathematicians: An Introduction for Students and Professionals](https://books.google.bg/books?id=qsfzxExOVTsC&pg=PA278&lpg=PA278&dq=JAMA+c%2B%2B+eigenValue&source=bl&ots=FtChEmXl24&sig=cNo-WCin37TbQ_MROMEq3E6QjcA&hl=en&sa=X&ved=0ahUKEwjQiKup_6nZAhWEmbQKHSfWAFUQ6AEIRzAD#v=onepage&q&f=false) [https://books.google.bg/books?id=qsfzxExOVTsC&pg=PA278&lpg=PA278&dq=JAMA+c%2B%2B+eigenValue&source=bl&ots=FtChEmXl24&sig=cNo-WCin37TbQ\_MROMEq3E6QjcA&hl=en&sa=X&ved=0ahUKEwjQiKup\_6nZAhWEmbQKHSfWAFUQ6AEIRzAD#v=onepage&q&f=false]
* [[4] JAMA and TNT libraries](https://math.nist.gov/tnt/download.html) [https://math.nist.gov/tnt/download.html]
* [[5] Tutorial on PCA](http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf) [http://www.cs.otago.ac.nz/cosc453/student\_tutorials/principal\_components.pdf]
* [[6] Reference code](https://github.com/liyanghua/Principal-component-analysis-) [https://github.com/liyanghua/Principal-component-analysis-]
* [[7] Video on Eigen vectors and values](https://www.khanacademy.org/math/linear-algebra/alternate-bases/eigen-everything/v/linear-algebra-introduction-to-eigenvalues-and-eigenvectors) [https://www.khanacademy.org/math/linear-algebra/alternate-bases/eigen-everything/v/linear-algebra-introduction-to-eigenvalues-and-eigenvectors]
* [8] Artificial intelligence a modern approach 3rd edition
* [[9] Алтернативна имплементация с sklearn](https://www.kaggle.com/rbk1988/digit-recognizer-using-pca-and-knn/code) [https://www.kaggle.com/rbk1988/digit-recognizer-using-pca-and-knn/code]
* [[10] предишни разработки по задачата](http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf) [http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf]