САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-Механический факультет

Кафедра Системного Программирования

Сайфутдинов Руслан Айдарович

Исследование алгоритмов уменьшения размерности данных для задачи классификации

Курсовая работа

Научный руководитель:
Невоструев К. Н.

Санкт-Петербург
2014 г.

Содержание

Введение **3**
1.1 Мотивация . . . 3
1.2 Постановка задачи . . . 3
1.3 Доступные программные средства . . . 3
1.4 Обзор существующих работ . . . 3

2 Алгоритмы уменьшения размерности **4**
2.1 PCA . . . 4
2.2 ICA . . . 6
2.2.1 FastICA для одной компоненты . . . 7
2.2.2 FastICA для нескольких компонент . . . 7
2.2.3 Свойства алгоритма FastICA . . . 8
2.3 SVD . . . 8

3 SVM **10**
3.1 Бинарная классификация . . . 10
3.2 Случай линейной неразделимости . . . 10
3.3 Мультиклассификация . . . 11

4 Сравнения **12**
4.1 Методика тестирования . . . 12
4.2 Результаты . . . 12

Заключение **14**

Введение

1.1 Мотивация

В современном мире данные, такие как аудио, цифровые изображения, тексты, обычно имеют большую размерность. Для обработки таких данных, их размерность должна быть уменьшена. *Уменьшение размерности* — преобразование исходных данных с большой размерностью в новое представление меньшей размерности, сохраняющее основную информацию. В идеальном случае размерность преобразованного представления соответствует внутренней размерности данных. Внутренняя размерность данных — минимальное число переменных, необходимое, чтобы выразить все возможные свойства данных.

Уменьшение размерности помогает смягчить влияние проклятия размерности и других нежелательных свойств пространств большой размерности. В результате уменьшение размерности способствует, среди прочего, классификации, визуализации и компрессии данных большой размерности. Традиционно, для уменьшения размерности используются алгоритмы, такие как *Principal Component Analisys (PCA)*, *Independent Component Analisys (ICA)*, *Singular Value Decomposition (SVD)* и другие.

1.2 Постановка задачи

Цель работы состоит в обзоре существующих подходов и алгоритмов для уменьшения размерности данных. А также в проведении исследования зависимостей между количеством компонент, получившимся после уменьшения размерности, и эффективностью различных алгоритмов. В этой работе планировалось решить следующие задачи:

- Реализовать несколько алгоритмов уменьшения размерности данных
- Установить зависимость эффективности алгоритмов от количества получившихся компонент

1.3 Доступные программные средства

Все вычисления производились с использованием языка программирования Python 2.7 и библиотек numpy, scipy, libsvm.

1.4 Обзор существующих работ

При выполнении данной работы был проведён анализ существующих публикаций по схожим темам [10], [11], [12]. Но ни в одной из них не исследовалось поведение алгоритмов уменьшения размерности данных в зависимости от числа получивших после преобразования компонент. Поэтому было решено провести данное исследование.

2 Алгоритмы уменьшения размерности

2.1 PCA

PCA — метод, который проецирует данные на новую координатную систему меньшей размерности, определяемую собственными векторами и собственными числами матрицы. Хотя существует несколько алгоритмов, работающих таким образом, PCA является наиболее популярным из них. По этой причине он включен в данное сравнение.

PCA включает в себя вычисление ковариационной матрицы данных, чтобы минимизировать избыточность и максимизировать дисперсию. Математически PCA определяется как ортогональное линейное преобразование и полагает все базисные векторы ортонормированной матрицей [1] [2].

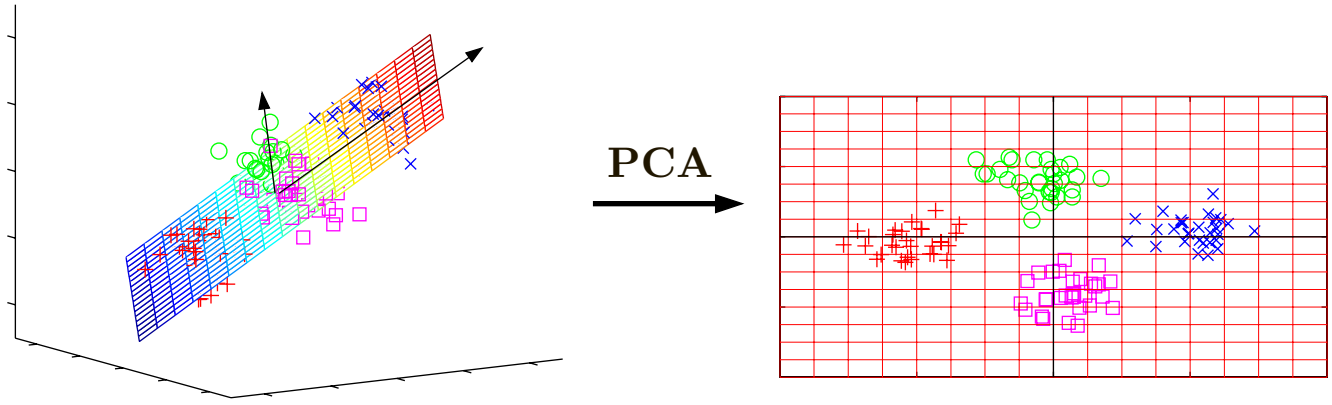

Рис. 1: Principal Component Analisys

PCA работает с помощью нахождения собственных чисел и собственных векторов матрицы ковариации. Матрица ковариации используется, чтобы измерить, как много размерностей отличаются от среднего по отношению друг к другу. *Ковариация* двух случайных величин (размерностей) — мера их линейной зависимости:

$$\operatorname{cov}(X, Y) = \mathbb{E}\left[(X - \mathbb{E}X)(Y - \mathbb{E}Y)\right],$$

где $\mathbb{E}[X]$ и $\mathbb{E}[Y]$ — математическое ожидание случайной величины X и Y соответственно. Для определенного набора данных мы можем записать:

$$\operatorname{cov}(X, Y) = \sum_{i=1}^{N} \frac{(x_i - \bar{x})(y_i - \bar{y})}{N},$$

где $\bar{x}$ — среднее X, $\bar{y}$ — среднее Y, а N — размерность данных. *Матрица ковариации* — матрица A с элементами $A_{ij} = \operatorname{cov}(i, j)$. Она центрирует данные, вычитая среднее из каждого объекта.

В ковариационной матрице точное значение не так важно, как его знак (т.е. положительное или отрицательное). Если значение положительное, это означает, что обе координаты возрастают, то есть если значение координаты X растёт, то же самое происходит с координатой Y. Если значение отрицательное, тогда если одна координата растёт, то другая уменьшается. В таком случае координаты в итоге примут противоположные значения. В последнем случае, если ковариация равна нулю, две координаты

независимы друг от друга. По свойству коммутативности, ковариация X и Y ($\mathrm{cov}(X, Y)$) равна ковариации Y и X ($\mathrm{cov}(Y, X)$).

После того, как собственные векторы и собственные числа найдены, собственные числа сортируются в порядке убывания. Благодаря этому можно получить компоненты в порядке уменьшения значимости. Собственный вектор с самым большим собственным числом —самая главная компонента набора данных. Он выражает самые существенные отношения между координатами. Поэтому главные компоненты получаются умножением строк из собственных векторов на отсортированные собственные значения.

Как было сказано ранее, PCA используется для уменьшения размерности с помощью нахождения главных компонент входных данных. Для этого с помощью собственных чисел и собственных векторов матрицы ковариации определяется лучшее пространство меньшей размерности. Лучшим пространством меньшей размерности называется пространство, имеющее минимальную ошибку между исходным набором данных и полученным по следующему критерию:

$$\frac{\sum_{i=1}^{K} \lambda_i}{\sum_{i=1}^{N} \lambda_i} > \Theta,$$

где K — выбранная новая размерность, Θ — порог, а λ_i — собственные числа.

Основываясь на этом критерии, матрица $M \times N$ линейно преобразуется в матрицу $M \times K$. Хотя размерность матрицы уменьшилась после PCA, разница между исходной и полученной матрицами невелика.

PCA пытается найти линейное отображение M, которое максимизирует оценочную функцию $\mathrm{tr}(M^T \mathrm{cov}(X)M)$. Таким образом, PCA решает задачу:

$$\mathrm{cov}(X)M = \lambda M$$

Представление уменьшенной размерности y_i точек x_i находится с помощью линейного отображения M, $Y = XM$.

Можно также искать отображение M, которое минимизирует оценочную функцию $\varphi(Y)$.

$$\varphi(Y) = \sum_{i=1}^{N} \sum_{j=1}^{N} (d_{ij}^2 - \|y_i - y_j\|^2),$$

где d_{ij} — евклидово расстояние между точками x_i и x_j, $\|y_i - y_j\|$ — евклидово расстояние между точками y_i и y_j, $y_i = x_i M$, $\|m_j\|^2 = 1 \ \forall j$. Можно показать, что минимум этой оценочной функции получается из спектрального разложения матрицы Грама $K = XX^T$ данных большой размерности. Значения матрицы Грама могут быть получены двойным центрированием квадратов попарных расстояний d_{ij}. То есть вычислением:

$$k_{ij} = -\frac{1}{2} \left(d_{ij}^2 - \frac{1}{n} \sum_{l=1}^{N} d_{il}^2 - \frac{1}{n} \sum_{l=1}^{N} d_{jl}^2 + \frac{1}{N^2} \sum_{l=1}^{N} \sum_{m=1}^{N} d_{lm}^2 \right)$$

Минимум оценочной функции φ теперь может быть получен умножением собственных векторов дважды центрированной матрицы евклидовых расстояний (то есть собственных векторов матрицы Грама) на корень из соответствующих собственных чисел.

Похожесть между этими двумя постановками задачи PCA связана с отношением между собственными векторами матрицы ковариации и матрицы Грама: можно показать, что собственные векторы u_i и v_i матриц $X^T X$ и XX^T удовлетворяют равенству $\sqrt{\lambda_i} v_i = X v_i$ [3]. Эта связь описывается более подробно, например, в [4].

PCA может быть успешно использован для распознования лиц [5], анализа движений, классификации [6]. Но у PCA есть два основных недостатка.

Во-первых, в PCA размер ковариационной матрицы прямо пропорционален размерности входных данных. Из-за этого поиск собственных векторов может быть невозможен для очень больших размерностей данных.

Во-вторых, оценочная функция φ показывает, что PCA фокусируется в основном на сохранении больших попарных расстояний d_{ij}^2, вместо того чтобы фокусироваться на сохранении маленьких попарных расстояний, которые являются более важными.

2.2 ICA

Метод независимых компонент получил широкое распространение при обработке сигналов. Это статистическая техника общего назначения, которая пытается линейно преобразовать исходные данные в новые компоненты, которые будут максимально независимы друг от друга в статистическом смысле. Независимые компоненты не обязательно ортогональны друг другу, но статистически независимы. Это более строгое условие, чем статистическая некоррелированность, которая используется в PCA. В данной работе используется реализация FastICA [7]. FastICA известен как надежный и эффективный алгоритм обнаружения лежащих в основе независимых компонент данных для широкого диапазона распределений. Математические детали FastICA могут быть найдены в [7]. В практическом применении FastICA требуется два шага предварительной обработки. Во-первых, центрирование, то есть вычитание среднего из данных. Во-вторых, «отбеливание» (whitening), которое означает линейное преобразование вектора x в новый вектор, такой что его координаты не коррелированы, а дисперсия координат равна единице.

Критерием независимости в FastICA является негауссовость, которая измеряется с помощью коэффициента эксцесса:

$$\mathrm{kurt}(v) = \mathbb{E}[v^4] - 3(\mathbb{E}[v^2]^2)$$

Для гауссовых случайных величин он равна нулю, поэтому FastICA пытается максимизировать это значение. Если $\widetilde{x}$ — «отбеленные» данные, то матрица ковариации «отбеленных» данных — единичная матрица.

$$\mathbb{E}\left[\widetilde{x}\widetilde{x}^T\right] = I$$

Такое преобразование всегда возможно. Популярный метод «отбеливания» использует спектральное разложение матрицы ковариации $\mathbb{E}[xx^T] = EDE^T$, где E — ортогональная матрица собственных векторов, а D — диагональная матрица собственных чисел, $D = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$. Теперь «отбеливание» может быть выполнено как

$$\widetilde{x} = ED^{-1/2}E^T x,$$

где матрица $D^{-1/2}$ вычисляется простой покомпонентной операцией, как

$$D^{-1/2} = \mathrm{diag}(\lambda_1^{-1/2}, \ldots, \lambda_n^{-1/2})$$

2.2.1 FastICA для одной компоненты

Итеративный алгоритм ищет направление весового вектора w, максимизируя негауссовость проекции $w^T x$ данных x. Функция $g(u)$ — производная неквадратичной нелинейной функции $f(u)$. В [7] автор даёт хорошие функции f:

$$f(u) = \log\cosh u$$
$$f(u) = -e^{-u^2/2}$$

Их первые и вторые производные:

$$g(u) = \tanh(u) \qquad g'(u) = 1 - \tanh^2(u)$$
$$g(u) = ue^{-u^2/2} \qquad g'(u) = (1-u^2)e^{-u^2/2}$$

Первая функция хорошо подходит для общего назначения, тогда как вторая более устойчива к различным выбросам.

Базовый алгоритм FastICA заключается в следующем:

1. Выбрать начальный (случайный) вектор w.
2. $w^+ = \mathbb{E}\left[xg(w^T x)\right] - \mathbb{E}\left[g'(w^T x)\right] w$.
3. $w = \frac{w^+}{\|w^+\|}$.
4. Если последовательность w не сошлась, перейти к шагу 2.

Сходимость означает, что старые и новые значения вектора w совпадают, то есть их скалярное произведение почти равно единице. На практике математические ожидания должны быть заменены их оценками. Естественными оценками являются средние значения. В идеальном случае все доступные данные должны использоваться, но часто это не очень хорошая идея, потому что вычисления могут стать слишком требовательными к ресурсам. Тогда могут быть получены усредненные результаты с использованием только части данных, размер которой значительно влияет на итоговый результат. Если сходимость неудовлетворительна, нужно увеличить размер рассматриваемых данных.

2.2.2 FastICA для нескольких компонент

Алгоритм в предыдущем пункте находит только одну из независимых компонент или одно направление наилучшей проекции. Для того, чтобы получить несколько независимых компонент алгоритм должен быть повторён несколько раз. Чтобы избежать схождения разных векторов к одному и тому же максимуму, полученные векторы $w_1^T x, \ldots, w_n^T x$ должны быть декоррелированы после каждой итерации. Существует три метода, чтобы это сделать. Простой способ достижения декорреляции основан на ортогонализации Грама-Шмидта. Независимые компоненты вычисляются последовательно. Когда найдено p независимых компонент или p векторов $w_1, \ldots, w_p$, запускается алгоритм для одной компоненты w_{p+1}, и после каждой итерации вычитается из w_{p+1} «проекции» $w_{p+1}^T w_j w_j$, $j = 1, \ldots, p$ предыдущих найденных векторов, после чего w_{p+1} нормализуется:

1. $w_{p+1} = w_{p+1} - \sum_{j=1}^{p} w_{p+1}^T w_j w_j$
2. $w_{p+1} = w_{p+1} / \sqrt{w_{p+1}^T w_{p+1}}$

Иногда, однако, возникает желание использовать симметричную декорреляцию, когда ни один вектор не «привилегирован» по сравнению с другими. Это может быть осуществлено, например, классическим методом с использованием матричных квадратных корней:

$$W = (WW^T)^{-1/2}W,$$

где W матрица $(w_1, \ldots, w_n)^T$ векторов, а обратный квадратный корень $(WW^T)^{-1/2}$ получается из спектрального разложения $WW^T = FDF^T$, как $(WW^T)^{-1/2} = FD^{-1/2}F^T$. Более простая альтернатива — следующий итерационный алгоритм:

1. $W = \frac{W}{\sqrt{\|WW^T\|}}$
2. $W = \frac{3}{2}W - \frac{1}{2}WW^TW$
3. Повторять 2 до тех пор, пока не сойдётся.

Норма в шаге 1 может быть практически любой обыкновенной матричной нормой, например наибольшей по модулю суммой строки или столбца.

2.2.3 Свойства алгоритма FastICA

Алгоритм FastICA имеет несколько важных свойств по сравнению с другими существующими алгоритмами ICA:

1. Кубическая сходимость (или хотя бы квадратичная), в отличие от других алгоритмов ICA, основанных на методе градиентного спуска, где сходимость только линейная.

2. По сравнению с алгоритмами, основанными на градиентном спуске, не нужно выбирать параметр размера шага. Это делает FastICA легким в использовании.

3. Алгоритм находит независимые компоненты практически любого негауссового распределения, используя любую нелинейную функцию g.

4. Независимые компоненты вычисляются одна за другой, что приблизительно эквивалентно методу поиска наилучшей проекции.

5. Производительность алгоритма может быть оптимизирована выбором подходящей нелинейной функции g.

2.3 SVD

В линейной алгебре сингулярное разложение — факторизация вещественной или комплексной матрицы с множеством полезных применений в обработке сигналов и статистике.

Пусть матрица M порядка $m \times n$ состоит из элементов из поля K, где K — либо поле вещественных чисел, либо поле комплексных чисел.

Неотрицательное вещественное число σ называется *сингулярным числом* матрицы M тогда и только тогда, когда существуют два вектора единичной длины $u \in K^m$ и $v \in K^n$ такие, что:

$$Mv = \sigma u,$$
$$M^*u = \sigma v$$

Такие векторы u и v называются, соответственно, *левым сингулярным вектором* и *правым сингулярным вектором*, соответствующим сингулярному числу σ.

Сингулярным разложением [8] матрицы M порядка $m \times n$ является разложение следующего вида:

$$M = U\Sigma V^*,$$

где Σ — размера $m \times n$, у которой элементы, лежащие на главной диагонали — это сингулярные собственные числа (а все элементы, не лежащие на главной диагонали, являются нулевыми), а матрицы U (порядка m) и V (порядка n) — это две унитарные матрицы, состоящие из левых и правых собственных сингулярных векторов соответственно (а V^* — это сопряжённо-транспонированная матрица к V). Причем диагональные элементы σ_i матрицы Σ упорядоченны следующим образом:

$$\Sigma = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n)$$

$$\sigma_1 \geqslant \sigma_2 \geqslant \ldots \geqslant \sigma_r = \ldots = \sigma_n = 0,$$

где $r = \mathrm{rank}(A)$. В частности, $\|A\| = \sigma_1$.

Пусть матрице A поставлен в соответствие линейный оператор. Сингулярное разложение можно переформулировать в геометрических терминах. Линейный оператор, отображающий элементы пространства $\mathbb{R}^n$ в себя, представим в виде последовательно выполняемых линейных операторов вращения и растяжения. Поэтому компоненты сингулярного разложения наглядно показывают геометрические изменения при отображении линейным оператором A множества векторов из векторного пространства в себя или в векторное пространство другой размерности.

Для уменьшения размерности данных требуется приближение заданной матрицы M некоторой другой матрицей M_k с заранее заданным рангом k [9]. Известна следующая теорема, которую иногда называют теоремой Эккарта – Янга.

Если потребовать, чтобы такое приближение было наилучшим в том смысле, что норма Фробениуса разности матриц M и M_k минимальна при ограничении $\mathrm{rank}(M_k) = k$, то оказывается, что наилучшая такая матрица M_k получается из сингулярного разложения матрицы M по формуле:

$$M_k = U\Sigma_k V^*,$$

где Σ_k – матрица Σ, в которой заменили нулями все диагональные элементы, кроме k наибольших элементов.

$$\Sigma = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_k, 0, \ldots, 0)$$

Если элементы матрицы Σ упорядочены по невозрастанию, то выражение для матрицы M_k можно переписать в такой форме:

$$M_k = U_k\Sigma_k V_k^*$$

где матрицы U_k, Σ_k и V_k получаются из соответствующих матриц в сингулярном разложении матрицы M обрезанием до ровно k первых столбцов.

Таким образом видно, что приближая матрицу M матрицей меньшего ранга, мы выполняем своего рода сжатие информации, содержащейся в M: матрица M размера $m \times n$ заменяется меньшими матрицами размеров $m \times k$ и $k \times n$ и диагональной матрицей с k элементами. При этом сжатие происходит с потерями — в приближении сохраняется лишь наиболее существенная часть матрицы M.

3 SVM

В данной работе для решения задачи классификации был применён метод опорных векторов (SVM). Была использована реализация из библиотки libsvm языка Python 2.7.

3.1 Бинарная классификация

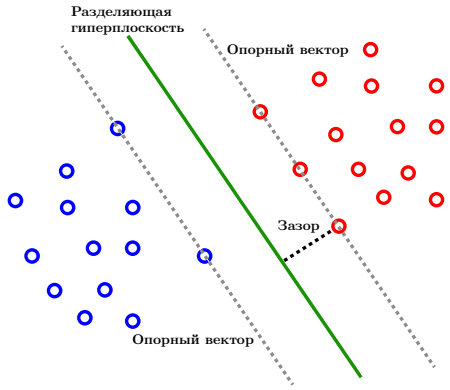

Рис. 2: Разделяющая гиперплоскость

Метод опорных векторов основан на поиске разделяющей гиперплоскости с наибольшим зазором [13]. Две параллельных гиперплоскости строятся по обеим сторонам гиперплоскости, разделяющей наши классы. Разделяющей гиперплоскостью будет гиперплоскость, максимизирующая расстояние до двух параллельных гиперплоскостей. Алгоритм работает в предположении, что чем больше разница или расстояние между этими параллельными гиперплоскостями, тем меньше будет средняя ошибка классификатора.

3.2 Случай линейной неразделимости

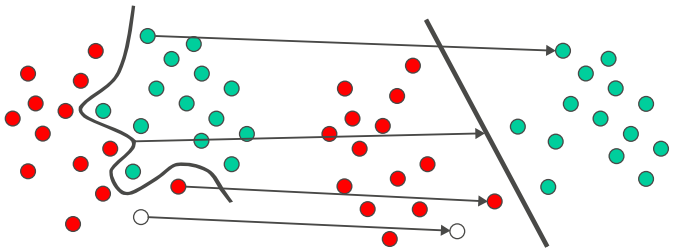

Рис. 3: Преобразование пространства с помощью ядра

Не всегда данные бывают линейно разделимы. Поэтому часто используется преобразование пространства с помощью ядра. Используя такое преобразование, можно получить новое пространство, в котором данные будут линейно разделимы. В данной работе использовалось ядро RBF:

$$K(x, x') = e^{\frac{-\|x-x'\|_2^2}{2\sigma^2}},$$

где $\|x - x'\|_2^2$ — евклидова норма, а σ — параметр.

3.3 Мультиклассификация

Для решения задачи мультиклассфикации применяют сведение её к задаче бинарной классификации. Для этого существует два основных метода: один-против-всех и каждый-против-каждого [14]. В первом случае строятся классификаторы, которые отличают объекты своего класса от всех остальных. Для классификации произвольного объекта нужно выбрать классификатор с максимальным результатом. Во втором случае же строится по классификатору для каждой пары классов. Для классификации произвольного объекта все классификаторы голосуют за один из классов, после чего выбирается класс, набравший наибольшее число голосов.

4 Сравнения

4.1 Методика тестирования

Все описанные выше алгоритмы уменьшения размерности были реализованы на языке Python 2.7 с использованием библиотек numpy, scipy. В качестве задачи классификации было выбрано распознавание рукописных цифр из базы MNIST [15], которая состоит из 60 000 изображений в тренировочной выборке и 10 000 изображений в тестовой выборке. Каждое изображение имеет одну компоненту цвета и размер 28×28 пикселей (всего 784 признака). Для тестирования был использован компьютер с процессором Intel Core i7 с тактовой частотой 2.9 ГГц и 8 ГБ оперативной памяти.

В качестве метрики точности была использована оценка F_1 [16]:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}},$$

где precision — отношение количества верно распознанных объектов заданного класса к количеству объектов распознанных как заданный класс, а recall — отношение количества верно распознанных объектов заданного класса к общему количеству объектов этого класса.

4.2 Результаты

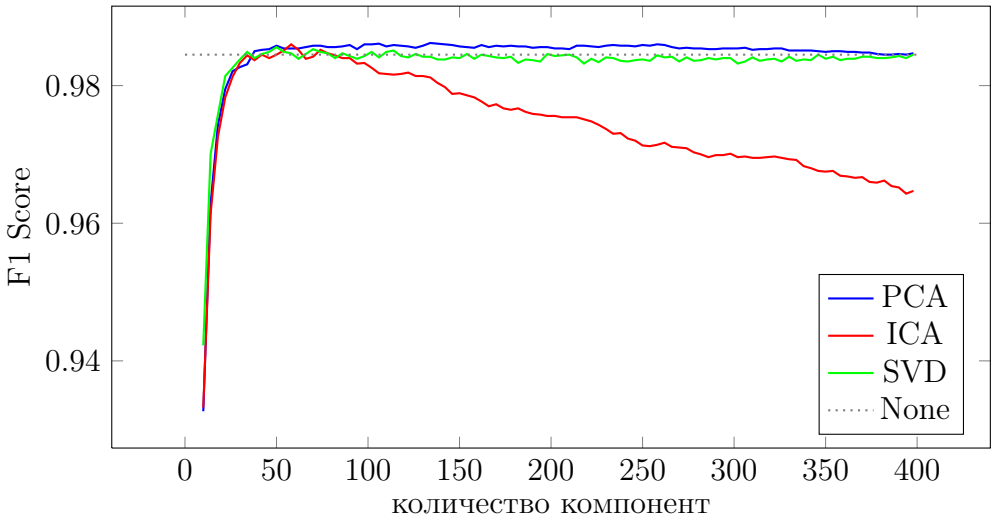

Рис. 4: Точность

На рисунке 4 показано изменение оценки F_1 в зависимости от получившейся после преобразования размерности данных. Видно, что все алгоритмы показывают наибольшую точность при достаточно маленьком количестве оставшихся компонент, порядка 10%. С увеличением количества компонент точность после применения алгоритмов PCA и SVD практически не изменяется, а после применения ICA начинает линейно падать. Это связано с тем, что FastICA не может выделить такое большое количество независимых компонент и перестаёт сходиться, из-за чего данные становятся линейно неразделимыми. Серой линией показана точность классификации без использования алгоритмов предварительной обработки данных. PCA также помогает немного увеличить точность.

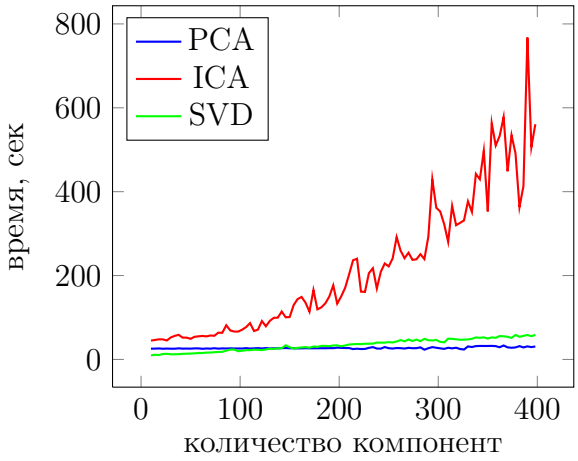

Рис. 5: Время работы

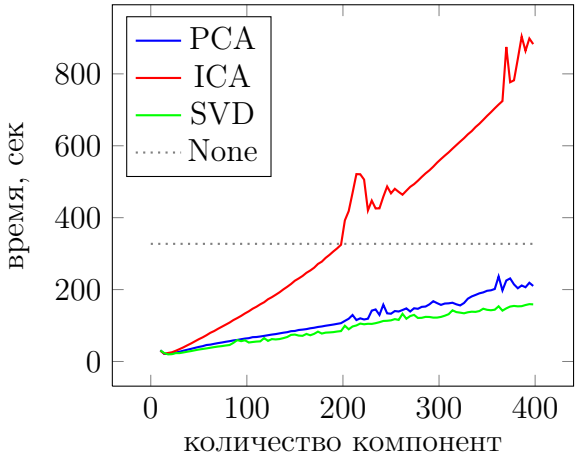

Рис. 6: Время обучения

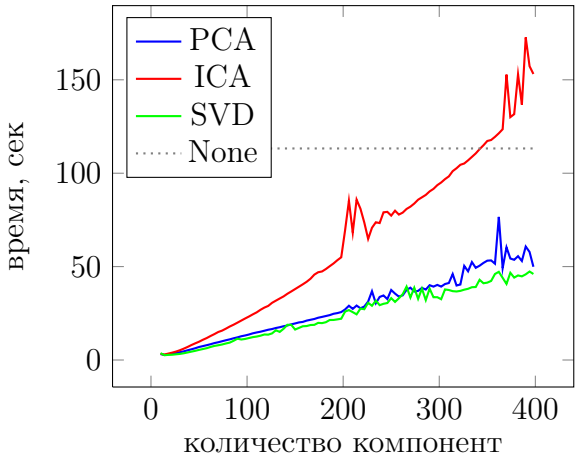

Рис. 7: Время классификации

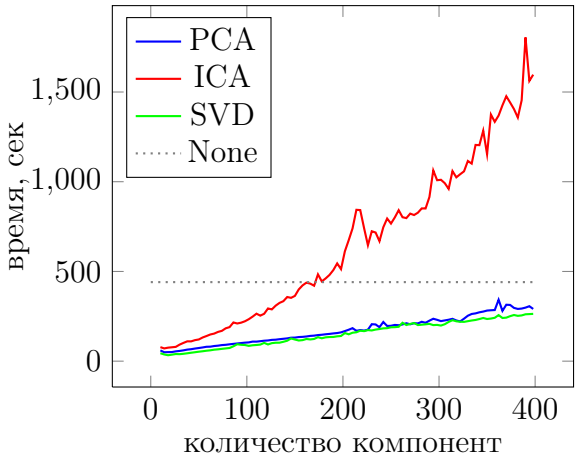

Рис. 8: Суммарное время работы

Время работы PCA не зависит от количества получившихся компонент, только от размера исходных данных. Время работы SVD растёт линейно, как и время работы SVM после его применения. В отличие от них время работы FastICA растёт нелинейно. Более того, начиная с некоторого момента SVM после FastICA начинает работать дольше, чем без него. Это связано, опять же, с тем, что FastICA не может выделить необходимое число независимых компонент.

Заключение

В данной работе были реализованы алгоритмы PCA, FastICA и SVD, было рассмотрено применение SVM для решения задачи классификации. Была исследована их эффективность при решении задачи классификации в зависимости от количества компонент, оставшихся после уменьшения размерности пространства данных. Было обнаружено, что алгоритм FastICA деградирует с увеличением количества компонент, причем нелинейно. PCA же способен немного повысить точность классификации. Точность же, полученная после применения алгоритмов PCA и SVD, практически не зависит от количества компонент, а время их работы растёт линейно. Поэтому оптимальным выбором для новой размерности данных будет небольшое количество компонент, порядка 10% от первоначального, когда все рассмотренные алгоритмы показывают наилучшие результаты.

Список литературы

[1] I. T. Jolliffe. *Principal Component Analysis, Second Edition*. Springer, 2002.

[2] H. Hotelling. Analysis of a complex of statistical variables into components. *Journal of Educational Psychology*, 1933.

[3] C. Chatfield and A.J. Collins. *Introduction to Multivariate Analysis*. Chapman and Hall, 1980.

[4] C.K.I. Williams. On a connection between Kernel PCA and metric multidimensional scaling. *Machine Learning*, 2002.

[5] M. Turk and A. Pentland. Face recognition using eigenfaces. *Computer Vision and Pattern Recognition*, 1991.

[6] Yehuda Koren and Liran Carmel. Visualization of labeled data using linear transformations. *IEEE Computer Society*, 2003.

[7] A. Hyvärinen and E. Oja. *Independent Component Analysis: Algorithms and Applications*. Cambridge University Press, 1991.

[8] P. C. Hansen. *Numerical Recipes in C. 2nd edition*, chapter Singular Value Decomposition. BIT, 1987.

[9] P. C. Hansen. The Truncated SVD as a method for regularization. *BIT*, 1987.

[10] L. van der Maaten and E. Postma. Dimensionality reduction: A Comparative Review. *TiCC TR*, 2009.

[11] R. Ohbuchi, J. Kobayashi, A. Yamamoto, and T. Shimizu. Comparison of dimension reduction methods for database-adaptive 3D model retrieval. *AMR*, 2007.

[12] L.J. Cao, K.S. Chua, W.K. Chong, H.P. Lee, and Q.M. Gu. A comparison of PCA, KPCA and ICA for dimensionality reduction in support vector machine. *Neurocomputing*, 2003.

[13] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000.

[14] J. Milgram, M. Cheriet, and R. Sabourin. "one against one"or “one against all” fo: Which One is Better for Handwriting Recognition with SVMs. *Tenth International Workshop on Frontiers in Handwriting Recognition*, 2006.

[15] Набор данных MNIST. `http://yann.lecun.com/exdb/mnist/` (Дата обращения 20 мая 2014).

[16] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.