Решение реальной прикладной задачи «Topical Classification of Biomedical Research Papers»

отчёт преподавателя практикума на ЭВМ (317гр. ф-та ВМК МГУ) Дьяконова Александра Геннадьевича о проделанной работе

Данный документ родился в результате следующего эксперимента. Весной 2012 года 317 группа факультета ВМК получила очередное задание по практикуму на ЭВМ: решать реальную прикладную задачу Международного конкурса[1]. Преподаватель (автор этих строк) также решал эту задачу вместе с группой (имея, правда, гораздо меньше возможностей показать высокий результат, поскольку в это же время был занят в более крупном конкурсе[2]). Тем не менее, в конце эксперимента студенты попросили преподавателя сделать отчёт по проделанной работе (в том формате, в котором делают они).

Автор не уверен, что отчёт получится «классным», но, возможно, он поможет некоторым при решении реальных задач. В отчёте не всё объясняется подробно, поскольку автор уже рассказал своё решение студентам. В рамках приводится код на языке системы MATLAB (комментарии в коде позволяют понять детали).

ПОСТАНОВКА ЗАДАЧИ

Необходимо написать алгоритм для автоматической классификации биомедицинских научных статей. Это стандартная задача классификации с 83 признаками, 10000 обучающими и 10000 контрольными объектами, ~25000 признаками. Признаковая матрица описывает что-то вроде чисел вхождений терминов и является сильно разреженной (организаторы оставили в секрете природу признаков). В качестве функционала качества используется F-мера.

Фактически, задача представляется **обычной задачей текстовой классификации**. Опыт подсказывает, что **алгоритм решения** здесь **должен быть предельно простым** (линейный классификатор или kNN, но точно не случайный лес или SVM с экзотическим ядром). Функционал качества не должен как-то повлиять на выбор модели алгоритмов, но на последнем этапе решающее правило придётся подкорректировать, оптимизируя функционал качества (сами модели лучше не трогать). Разреженность признаковой матрицы позволяет применять некоторые модели «быстрых» алгоритмов (эти алгоритмы заточены на

[1] http://tunedit.org/challenge/JRS12Contest?m=summary
[2] http://www.kaggle.com/c/WhatDoYouKnow/

разреженные данные), но больше она нигде не пригождается (поскольку, в отличие от задач коллаборативной фильтрации, нулевое значение признака здесь означает не отсутствие информации, а нулевое значение какого-то показателя).

ХОД РЕШЕНИЯ

Были исследованы следующие модели алгоритмов:

1. k Ближайших соседей (kNN).
2. Центроидный (замена объектов класса одним представителем – центром класса).
3. Линейное разделение после SVD-разложения признаковой матрицы.
4. LIBSVM (SVM-подобный алгоритм)

Выбор моделей определился желанием использовать лишь простые модели (не сложнее kNN и линейного классификатора). При выборе была допущена страшная ошибка: вместо пакета (LIBLINEAR для разреженных данных) была использована библиотека LIBSVM. Из-за чего SVM обучался и тестировался около 1 часа!

Была предложена следующая схема применения алгоритма. Если алгоритм порождает матрицу оценок[3] `Y` для обучающих объектов и матрицу оценок `Y2` – для контрольных, то ответ алгоритма вычисляется по формулам

```
L2 = Y2*((Y'*Y)\(Y'*L)) % в новом MATLABe - Y2*(Y\L)
bsxfun(@ge, L2, min(theta, max(L2,[],2)))
```

где `theta` – порог, который выбирается при оптимизации F-меры на контроле.
Эксперименты показали, что такая схема может на ~2% повысить качество классификации.

В результате алгоритм (2) был отвергнут (из-за низкого качества как самого алгоритма, так и суперпозиций с участием этого алгоритма), а остальные были использованы в решении.

Первоначально решение искалось в виде линейной комбинации алгоритмов, потом были применены более сложные комбинации (см. ниже при описании финального решения) с нормировками.

Ниже подробно описаны отчёты по исследованиям.

[3] Термин из курса «АМА». См. также ниже при описании финального решения.

ФИНАЛЬНОЕ РЕШЕНИЕ

Решение, которое показало результат **0.53242** (по F-мере) и заняло 3 место на соревновании является смесью линейного классификатора над SVD разложением, метода ближайших соседей и двух LIBSVM-алгоритмов[4]. Весь код написан в системе MATLAB[5].

Каждый из перечисленных алгоритмов получает матрицу размера 10000×83, в которой *ij*-й элемент равен[6] степени принадлежности *i*-го контрольного объекта к *j*-му классу (матрицу оценок). Затем эти матрицы нормируются: каждый элемент делится на корень максимального элемента в строке, и складываются с определёнными коэффициентами (которые подобраны методом покоординатного спуска). Все элементы итоговой матрицы сравниваются с порогом: если *ij*-й элемент больше порога, то относим *i*-й объект к *j*-му классу.

ПРАГМАТИКА:

1. **Выбор «суперпозиции»** определяется опытом автора, верой в то, что любая нелинейная задача решается «суммой нелинейных слагаемых[7]». Это лежит в основе технологии LENKOR[8], разрабатываемой автором.
2. **Запись через матрицы оценок** определяется спецификой среды MATLAB (а также навеяна духом алгебраического подхода и курсом «АМА»).
3. **Пороговое решающее правило**, как мы увидим ниже, на самом деле является смесью линейной регрессии и порога. Порог выбирался с целью максимизировать F-меру на контроле.
4. **Нормировка** была выбрана «случайно», но повысила качество решения на 0.5% (является «средней стратегией» между классической нормировкой по максимуму и отсутствием нормировки).

```
% линейная комбинация алгоритмов
L = 0.5939*mynormmax(Lsvd2) + 0.0011*mynormmax(Lknn2) + 0.2970*mynormmax(Llin2) + 0.1080*mynormmax(LlinB2);

% ответ
l = bsxfun(@ge, L, min(0.345, max(L,[],2)));
```

[4] http://www.csie.ntu.edu.tw/~cjlin/libsvm/
[5] http://www.mathworks.com
[6] Скорее, «интерпретируется как».
[7] В теории интерполяции есть даже теоретическое доказательство этого факта (см. работы Пинкуса, http://www2.math.technion.ac.il/~pinkus/list.html).
[8] См. A. D'yakonov Two Recommendation Algorithms Based on Deformed Linear Combinations // Proc. of ECML-PKDD 2011 Discovery Challenge Workshop, pp 21-28 (2011). http://ceur-ws.org/Vol-770/paper5.pdf

```
% нормировка
function Y = mynormmax(Y)
Y = bsxfun(@rdivide, Y, sqrt(max(Y,[],2)));
```

Опишем, как были построены алгоритмы из линейной комбинации. Все они получают новые матрицы «обучающий объект – признак», «тестовый объект – признак», затем решают стандартную задачу линейной регрессии, в результате чего получаются матрицы оценок.

1. Линейный классификатор над SVD

Делаем SVD-разложение матрицы «обучающий объект – признак» (используем 700 максимальный собственных значений). Одну из матриц в разложении (столбцы которой являются линейной комбинацией столбцов исходной матрицы) используем как новую матрицу «обучающий объект – признак». Новую матрицу «тестовый объект – признак» формируем аналогично.

```
load S.mat % матрица обучающий объект - признак
load L.mat % матрица классификации обучающих объектов
load S2.mat % матрица тестовый объект - признак

S = bsxfun(@rdivide, S, sqrt(sum(S, 2))); % предварительная нормировка
[~,~,Sc] = svds(S, 700);
Ysvd = S*Sc; % новая матрица обучающий объект - признак
S2 = bsxfun(@rdivide, S2, sqrt(sum(S2, 2)));
Ysvd2 = S2*Sc; % новая матрица тестовый объект - признак
% решение задачи линейной регрессии
Lsvd2 = Ysvd2*((Ysvd'*Ysvd)\(Ysvd'*L));
```

2. Метод ближайшего соседа

Мы приводим код метода. Сначала делаем нормировку, которая получилась в результате перебора различных нормировок (см. далее). Затем запускаем сам метод. Его код достаточно «тяжеловесный», поскольку данные пришлось делить на «пачки» (для ускорения вычислений).

Из-за нестандартной нормировки (делим не на норму, а на корень суммы элементов вектора) используемая функция близости не является метрикой.

Отметим, что в линейной комбинации метод ближайшего соседа идёт с малым коэффициентом. Наверное, его можно вообще не использовать (мы не успели проверить это).

```
load S.mat % матрица обучающий объект - признак
load L.mat % матрица классификации обучающих объектов
load S2.mat % матрица тестовый объект - признак

% предварительная нормировка матриц
% что-то типа TF*IDF-разложения
SW = L'*S;
SW = mean(log(full(SW)+1))+0.001;

S = bsxfun(@rdivide, S, SW);
S = bsxfun(@rdivide, S, sqrt(sum(S, 2)));

S2 = bsxfun(@rdivide, S2, SW);
S2 = bsxfun(@rdivide, S2, sqrt(sum(S2, 2)));

k = 200; % число ближайших соседей
% запуск методов
Yknn  = MEDclassifierkNN1(S, L, S,  k, true);
Yknn2 = MEDclassifierkNN2(S, L, S2, k, false);

% решение задачи линейной регрессии
Lknn2 = Yknn2*((Yknn'*Yknn)\(Yknn'*L));
```

```
function L2 = MEDclassifierkNN1(S, L, S2, k, isTRAIN)
% isTRAIN - режим, когда S=S2 (и самый близкий к объекту объект - он сам)

u = 500; % размеры пачки (для быстроты вычислений)

l = size(L,2);
q = size(S2,1);
m = size(S,1);

L2 = sparse([],[],[],q,l,fix(3.7*q));

%  веса
W = (k:-1:1).^2; % квадратичная схема
W = W./sum(W);
```

```
for iu = 1:u:size(S2,1) % проход по всем пачкам
    iuend = min(iu+u-1, size(S2,1));
    uu = iuend-iu+1;

    % оценки близостей
    R = full(S*S2(iu:iuend,:)');

    if (isTRAIN)
       R(iu:(m+1):end) = 0; % устранить близость к самому себе
    end

    [R, I] = sort(R,'descend');

    Y = full(L(I(1:k,:),:));
    Y = bsxfun(@times, Y, repmat(W',uu,1));
    Y = reshape(Y, [k uu l]);
    Y = squeeze(sum(Y,1));

    Y = bsxfun(@rdivide, Y, max(Y,[],2)); % ВОЗМОЖНО, ЭТУ СТРОКУ НАДО УДАЛИТЬ ?!

    L2(iu:iuend,:) = Y; % кусочек матрицы оценок для рассматриваемой пачки
end;
```

3. LIBSVM**-методы**

Запускаем стандартный LIBSVM-алгоритм два раза. Два метода различаются предварительной нормировкой матриц «объект – признак»: в первом – все элементы делим на максимальный элемент в столбце, во втором – на максимальный элемент в строке.

Выбор нормировок ничем особо не аргументирован. Хотелось их сделать «в принципе разными», поэтому отдельно использовались нормировки по строкам и столбцам. Для использования SVM-подобных алгоритмов необходимо, чтобы признаки были «в одной шкале», поэтому выбор был невелик (по максимуму и сумме).

```
load S.mat % матрица обучающий объект - признак
load L.mat % матрица классификации обучающих объектов
load S2.mat % матрица тестовый объект - признак

% предварительная нормировка
```

```
nnorm = max([S; S2],[],1)+0.001;
S = bsxfun(@rdivide, S, nnorm);
S2 = bsxfun(@rdivide, S2, nnorm);
% обучение
A = {};
for j=1:83
A{j} = svmtrain(2*full(L(:,j))-1, S, '-t 0 -c 0.01 -h 0');
end;
% новая матрица обучающий объект - признак
Y = [];
for j=1:83
[~,~,Y(:,j)] = svmpredict(zeros(size(S(:,1))), S, A{j});
end;
% новая матрица контрольный объект - признак
Y2 = [];
for j=1:83
[~,~,Y2(:,j)] = svmpredict(zeros(size(S2(:,1))), S2, A{j});
end;

% решение задачи линейной регрессии
Llin2 = Y2*((Y'*Y)\(Y'*L));
```

```
load S.mat
load L.mat
load S2.mat
% предварительная нормировка
S = bsxfun(@rdivide, S, max(S,[],2)+0.001);
S2 = bsxfun(@rdivide, S2, max(S2,[],2)+0.001);
% обучение
A = {};
for j=1:83
A{j} = svmtrain(2*full(L(:,j))-1, S, '-t 0 -c 0.01 -h 0');
end;
% новая матрица обучающий объект - признак
Y = [];
for j=1:83
[~,~,Y(:,j)] = svmpredict(zeros(size(S(:,1))), S, A{j});
end;
% новая матрица контрольный объект - признак
```

```
Y2 = [];
for j=1:83
[~,~,Y2(:,j)] = svmpredict(zeros(size(S2(:,1))), S2, A{j});
end;

% решение задачи линейной регрессии
LlinB2 = Y2*((Y'*Y)\(Y'*L));
```

СОВЕТЫ НОВИЧКАМ

Просто решать подобные задачи. Мастерство приходит с опытом. Сама модель алгоритмов не так уж и важна, если её грамотно использовать. Например, с помощью kNN можно было войти в 15 сильнейших команд соревнования.

ЧТО Я УЗНАЛ НОВОГО

- Есть текстовые задачи, в которых центроидный алгоритм работает крайне плохо (раньше я этого не знал).
- Центроидный алгоритм нельзя использовать с контролем по фолдам и линейной регрессией (получается жуткое переобучение).
- Это первая задача, в которой у меня получилось неплохо использовать SVM-подобный алгоритм (для текстов обычно удаётся настроить kNN ничуть не хуже SVM). Правда, выводы здесь я делать не буду (нужно ещё поэкспериментировать).
- Решающее регрессионное правило оказалось достаточно мощным, я даже вижу здесь тему для научной статьи.
- Нормировка по корню из максимума также оказалась на высоте (и здесь есть неплохая тема для исследований и статьи).
- Нельзя в спешке путать пакеты прикладных программ. Видимо, здесь имеет место плохая организация работы автора и неправильное распределение пакетов по каталогам (почти всё свалено «в кучу»).

ЕСЛИ БЫ БЫЛО БОЛЬШЕ ВРЕМЕНИ

Для «побития» второго результата соревнования достаточно было сделать ещё одну-две загрузки (с другим значением порога в линейной комбинации). Для достижения уровня первого места нужно было просто настроить LIBLINEAR.

В принципе, глобальных промахов по распределению времени не было. Автор слегка недооценил соперников и думал, что лучший финальный результат будет в районе 52%, но самое главное он сделал – научился решать эту задачу.

СПАСИБО

Огромное спасибо организаторам соревнования (задача оказалась интересной и приемлемой для задания по практикуму). Большую веру в SVD автору привили его студенты 4 курса (Александр Кириллов и Михаил Фигурнов). Неплохой идеей было использование именно пакета LIBLINEAR (первый эту идею подал, видимо, Пётр Ромов, а наиболее успешное тестирование провёл Андрей Остапец, однако автор этими результатами не воспользовался).

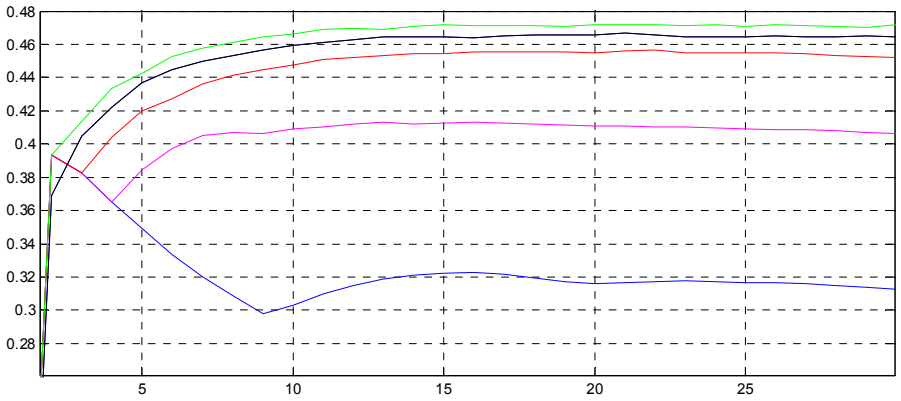
Сам автор на первом этапе соревнования пользы 317 группе не принёс[9], однако старался намекать в рамках практикума и курса AMA на «общие решающие правила», «важность выбора решающего правила» и прочие мелочи, которые сам в итоге использовал в решении.

БОЛЕЕ ПОДРОБНЫЙ ОТЧЁТ ОБ ЭКСПЕРИМЕНТАХ[10]

Начальные эксперименты с методом kNN


```
Максимальный + некий процент от его оценки
L1 - метрика
E = zeros([10000 30]);

for i = 1:10000

    RightAnswer = find(L(i,:));
    for j=1:30
        temp = squeeze(Y(i,:,1:j));
        temp = sum(temp, 2);
        % ЗДЕСЬ - МАКСИМАЛЬНЫЙ ЭЛ-Т * ПОРОГ
        MyAnswer = find(temp>max(temp)*0.50);
        E(i,j) = 2*length(intersect(RightAnswer,
MyAnswer))/(length(RightAnswer)+length(MyAnswer));
    end;
    i
```

[9] Речь о пользе зашла в связи с требованиями к отчёту (которым данный документ должен был удовлетворить).
[10] Приводятся графики и таблицы, которые получились в результате экспериментов. Изложение «чуть обрывочное». Основная цель – предоставить информацию о проделанных экспериментах.



Различие в выборе порога:

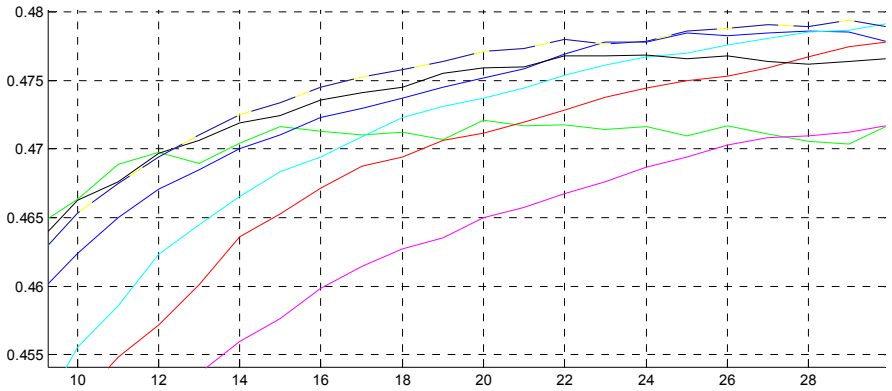
Чёрный = 0.5
Зелёный = 0.4
Красный = 0.3
Малиновый = 0.2
Синий = 0.1

```
end;
```

До этого

```
Y = zeros(m,1,30);
for i = 1:10000
    %R = full(sum(abs(bsxfun(@minus, S, S(i,:))),2));
    R = full(sum(abs( S - repmat(S(i,:),10000,1) ), 2));
    R(i) = Inf;
    [~, R]= sort(R);
    %Y(i,:) = sum(L(I(1:2),:),1);

    Y(i,:,:) = L(R(1:30),:)';
    %Y(i,:) = R'*L; % Y(:,i) = R'*L;
    i
    %toc;
end;
```

Разные весовые схемы в методе ближайшего соседа



Рекорд: 0.4794

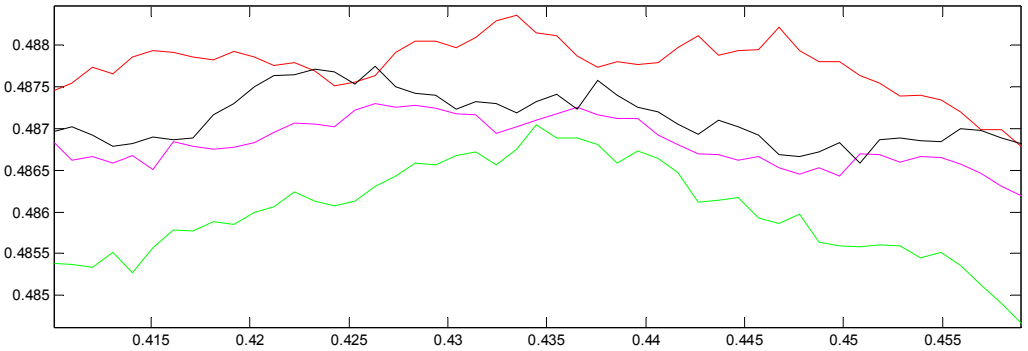
Зелёная – просто сумма
Синяя – линейная
Красная – квадратичная
Малиновая – квадратичная (с порогом 0.35)

Разные весовые схемы

```
for i = 1:10000

    RightAnswer = find(L(i,:));    for j=1:30
        temp = squeeze(Y(i,:,1:j));
        w = (j:-1:1)'; % ЛИНЕЙНАЯ СХЕМА
        w = w/sum(w);
        temp = temp*w;
        MyAnswer = find(temp>max(temp)*0.45);
        E(i,j) = 2*length(intersect(RightAnswer,
MyAnswer))/(length(RightAnswer)+length(MyAnswer));
    end;
    i
end;
```

Голубая – квадратичная (с порогом 0.45) Чёрный – корень **Синяя прерывистая - линейная** `0.45`	

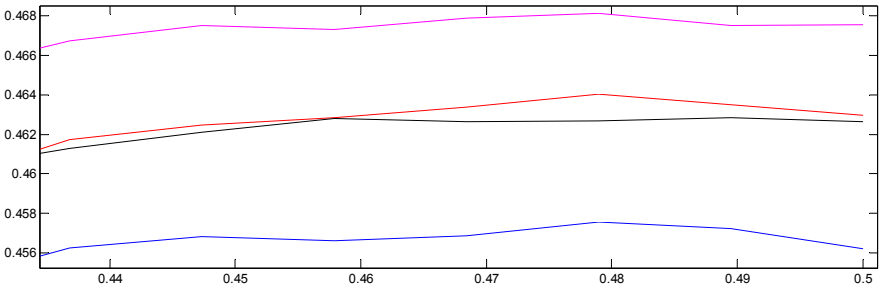
Чуть позже – после доработки kNN



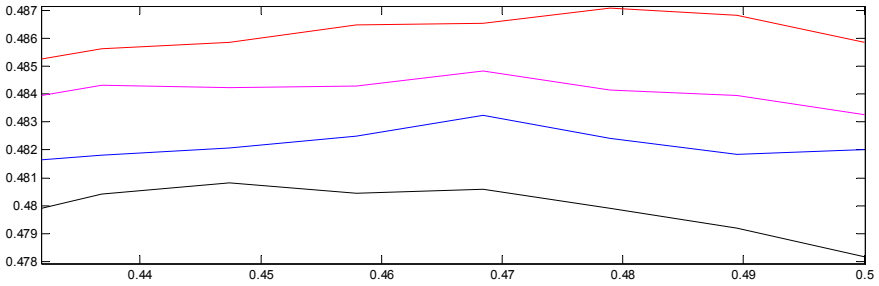
Графики от порога
Зелёный – стандартный линейный w (выше)
Красный – квадрат
Чёрный – 1.5
Малиновый – 3 степень

ВЫВОД:
решено оставить квадратичную схему весов в kNN.

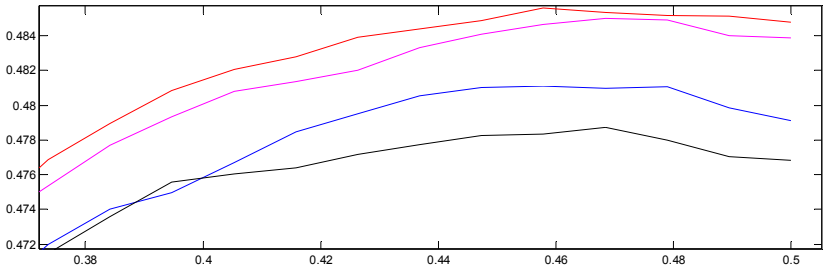
Разные нормировки данных



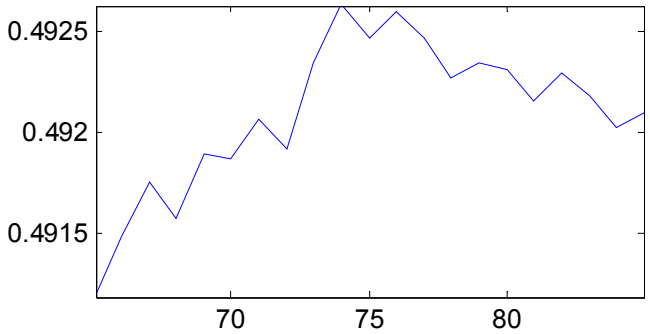

```
S = bsxfun(@rdivide, S, (sum(S, 1)+0.001));
```




```
S = bsxfun(@rdivide, S, sqrt(sum(S.^2, 2)));
```



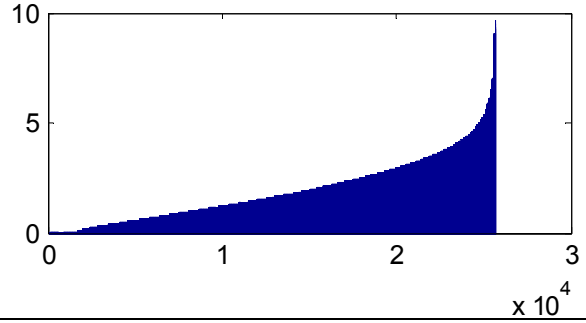
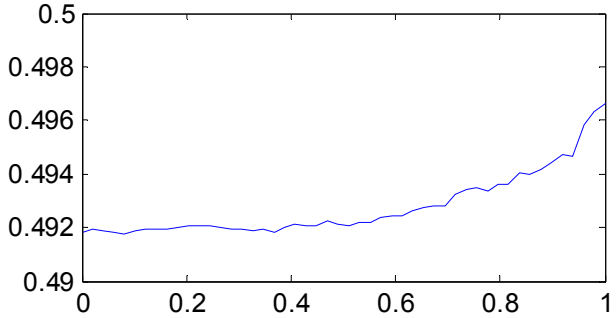

```
S = bsxfun(@rdivide, S, log(sum(S.^2, 2)+1)+0.001);
```

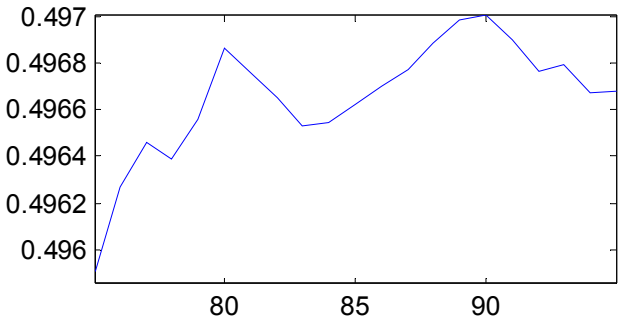
НОРМИРОВКА	КАЧЕСТВО ПО F-МЕРЕ
`S2 = bsxfun(@rdivide, S2, (sum(S, 1)+0.001));` `S2 = bsxfun(@rdivide, S2, sqrt(sum(S2, 2)));`	0.4833
`S2 = bsxfun(@rdivide, S2, sqrt(sum(S, 1)+0.001));` `S2 = bsxfun(@rdivide, S2, sqrt(sum(S2, 2)));`	0.4937
`S2 = bsxfun(@rdivide, S2, sqrt(sum(S2, 2)));`	0.4883
`% нормировки` `S = bsxfun(@rdivide, S, log(sum(S, 1)+1.001));` `S2 = bsxfun(@rdivide, S2, log(sum(S, 1)+1.001));` `S = bsxfun(@rdivide, S, sqrt(sum(S, 2)));` `% НОРМ-2` `S2 = bsxfun(@rdivide, S2, sqrt(sum(S2, 2)));` 	0.4921
`SW = L'*S;` `SW = sort(SW);` `SW = mean(abs(diff(SW)))+0.001;` `% нормировки` `S = bsxfun(@rdivide, S, SW);` `S = bsxfun(@rdivide, S, sqrt(sum(S, 2)));` `% НОРМ-2` `S2 = bsxfun(@rdivide, S2, SW);` `S2 = bsxfun(@rdivide, S2, sqrt(sum(S2, 2)));`	0.4374 0.4353 0.4391

SW = L'*S; SW = sort(SW); SW = mean(abs(diff(SW)))+0.001; % нормировки S = bsxfun(@rdivide, S, SW); S = bsxfun(@rdivide, S, sqrt(sum(S.^2, 2))); % НОРМ-2 S2 = bsxfun(@rdivide, S2, SW); S2 = bsxfun(@rdivide, S2, sqrt(sum(S2.^2, 2)));	0.4435 0.4445
SW = L'*S; SW = sort(SW); SW = mean(abs(diff(SW)))+0.001; SW = tiedrank(SW); % нормировки S = bsxfun(@rdivide, S, SW); S = bsxfun(@rdivide, S, sqrt(sum(S.^2, 2))); % НОРМ-2 S2 = bsxfun(@rdivide, S2, SW); S2 = bsxfun(@rdivide, S2, sqrt(sum(S2.^2, 2)));	0.4775
SW = L'*S; SW = sort(SW); SW = mean(abs(diff(SW)))+0.001; SW = tiedrank(SW); % нормировки S = bsxfun(@rdivide, S, SW); S = bsxfun(@rdivide, S, sqrt(sum(S, 2))); % НОРМ-2 S2 = bsxfun(@rdivide, S2, SW); S2 = bsxfun(@rdivide, S2, sqrt(sum(S2, 2)));	0.4834
SW = L'*S; SW = sum(SW>0)+0.001; %SW = sort(SW); %SW = mean(abs(diff(SW)))+0.001; %SW = tiedrank(SW); % нормировки S = bsxfun(@rdivide, S, SW); S = bsxfun(@rdivide, S, sqrt(sum(S, 2)));	0.4943

`` % НОРМ-2 S2 = bsxfun(@rdivide, S2, SW); S2 = bsxfun(@rdivide, S2, sqrt(sum(S2, 2))); ```	
`` SW = L'*S; SW = sum(SW>0)+0.001; % нормировки S = bsxfun(@rdivide, S, SW); S = bsxfun(@rdivide, S, sqrt(sum(S.^2, 2))); % НОРМ-2 S2 = bsxfun(@rdivide, S2, SW); S2 = bsxfun(@rdivide, S2, sqrt(sum(S2.^2, 2))); ```	0.4878
`` SW = L'*S; SW = sum(SW>0)+0.001; SW = tiedrank(SW); % нормировки S = bsxfun(@rdivide, S, SW); S = bsxfun(@rdivide, S, sqrt(sum(S, 2))); % НОРМ-2 S2 = bsxfun(@rdivide, S2, SW); S2 = bsxfun(@rdivide, S2, sqrt(sum(S2, 2))); ```	**0.4951**
`` SW = L'*S; SW = sum(SW>0)+0.001; SW = tiedrank(SW); % нормировки S = bsxfun(@rdivide, S, SW); S = bsxfun(@rdivide, S, sqrt(sum(S, 2))); % НОРМ-2 S2 = bsxfun(@rdivide, S2, sqrt(sum(S, 1)+0.001)); S2 = bsxfun(@rdivide, S2, sqrt(sum(S2, 2))); ```	0.4931
`` SW = L'*S; SW = sum(SW>0)+0.001; SW = tiedrank(SW); % нормировки S = bsxfun(@rdivide, S, SW); S = bsxfun(@rdivide, S, (sum(S, 2))); ```	0.4930

% НОРМ-2 S2 = bsxfun(@rdivide, S2, SW); S2 = bsxfun(@rdivide, S2, (sum(S2, 2)));	
SW = L'*S; SW = bsxfun(@rdivide, SW, sum(L)'); SW = mean(SW)+0.001; SW = tiedrank(SW); % нормировки S = bsxfun(@rdivide, S, SW); S = bsxfun(@rdivide, S, sqrt(sum(S, 2))); % НОРМ-2 S2 = bsxfun(@rdivide, S2, SW); S2 = bsxfun(@rdivide, S2, sqrt(sum(S2, 2)));	0.4902
SW = L'*S; SW = bsxfun(@rdivide, SW, sum(L)'); SW = sort(SW); SW = mean(abs(diff(SW)))+0.001; % нормировки S = bsxfun(@rdivide, S, SW); S = bsxfun(@rdivide, S, sqrt(sum(S, 2))); % НОРМ-2 S2 = bsxfun(@rdivide, S2, SW); S2 = bsxfun(@rdivide, S2, sqrt(sum(S2, 2)));	0.4430
SW = L'*S; SW = bsxfun(@rdivide, SW, sum(L)'); SW = sort(SW); SW = mean(1./(1+abs(diff(SW))))+0.001;	0.4700
SW = L'*S; SW = sort(SW); SW = mean(1./(1+abs(diff(SW))))+0.001;	0.3495
SW = L'*S; SW = sort(SW); SW = 1-mean(1./(1+abs(diff(SW))))+0.001; % нормировки S = bsxfun(@rdivide, S, SW); S = bsxfun(@rdivide, S, sqrt(sum(S, 2)));	0.4905
+ tiedrank(SW);	0.4936
SW = L'*S; SW = mean(log(full(SW)+1))+0.001;	0.4932
SW = L'*S; SW = mean(log(full(SW)+1))+0.001; SW = tiedrank(SW);	**0.4951**

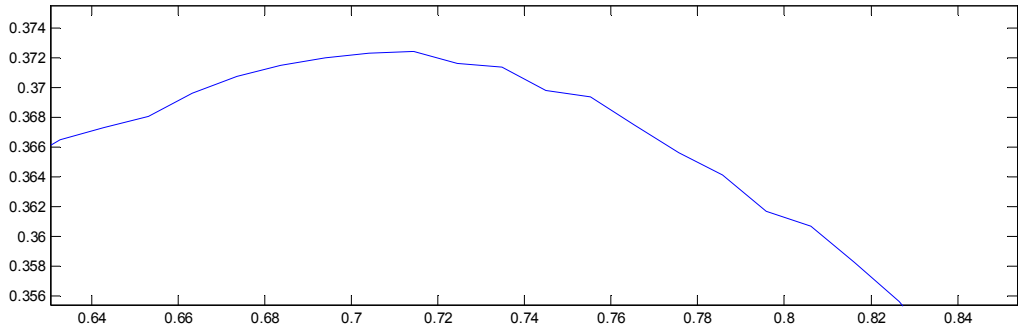
	
SW = L'*S; SW = sqrt(mean(log(full(SW)+1)))+0.001;  Если внедрять слагаемое ~ частота встречаемости (левая часть)	**0.4966**
SW = L'*S; SW = sqrt(mean(log(full(SW)+1)))+0.001; %SW = % нормировки S = bsxfun(@rdivide, S, SW); S = bsxfun(@rdivide, S, sqrt(sum(sqrt(S), 2))); % НОРМ-2 S2 = bsxfun(@rdivide, S2, SW); S2 = bsxfun(@rdivide, S2, sqrt(sum(sqrt(S2), 2)));	**0.4970**

	
0.4966 + настройка индивидуальных порогов	**0.5035**

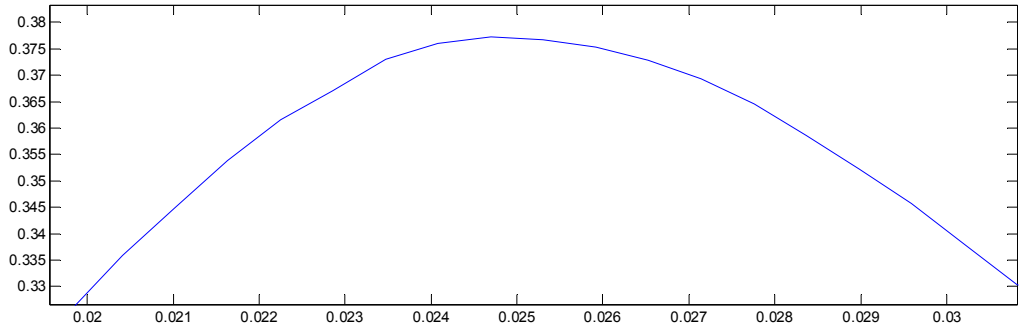
`SW = L'*S;` `SW = full(SW);` `SW = bsxfun(@rdivide, SW, sum(S)+1);` `SW = 1./(max(SW)+0.001);` `% нормировки` `S = bsxfun(@rdivide, S, SW);` `S = bsxfun(@rdivide, S, sqrt(sum(S, 2)));` `% НОРМ-2` `S2 = bsxfun(@rdivide, S2, SW);` `S2 = bsxfun(@rdivide, S2, sqrt(sum(S2, 2)));`	0.4953
+ `SW = tiedrank(SW);`	0.4911
+ `SW = SW.^2;`	0.4912
+ `SW = sqrt(SW);`	0.4940
`SW = L'*S;` `SW = full(SW);` `SW = bsxfun(@rdivide, SW, sum(S)+1);` `SW = sort(SW);` `SW = diff(SW);` `SW = 1./((1:82)*SW/sum(1:82)+0.001);` `% нормировки` `S = bsxfun(@rdivide, S, SW);` `S = bsxfun(@rdivide, S, sqrt(sum(S, 2)));`	0.4949
`SW = L'*S;` `SW = full(SW);` `SW = bsxfun(@rdivide, SW, sum(S)+1);` `SW = sort(SW);` `SW = diff(SW);` `SW = 1-40*((1:82)*SW/sum(1:82));`	0.4918

```
% нормировки
S = bsxfun(@rdivide, S, SW);
S = bsxfun(@rdivide, S, sqrt(sum(S, 2)));
```

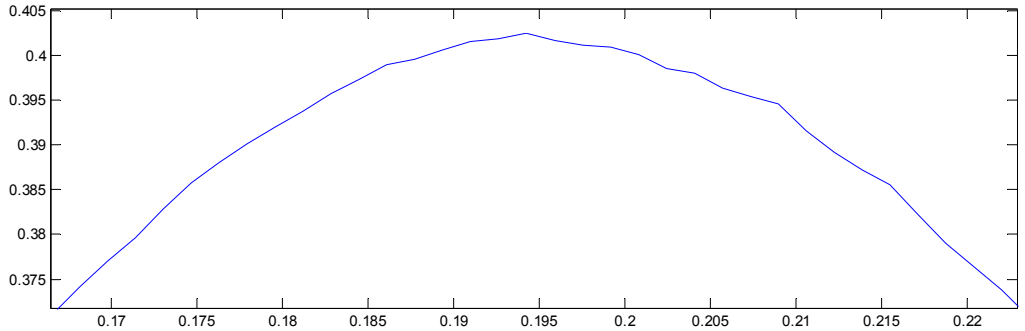
Эксперименты с центроидами



Предварительная нормировка по максимуму



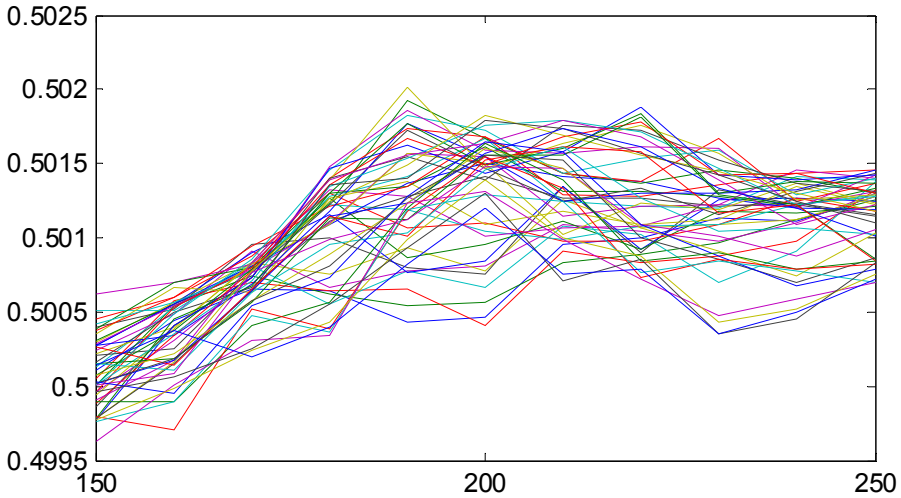
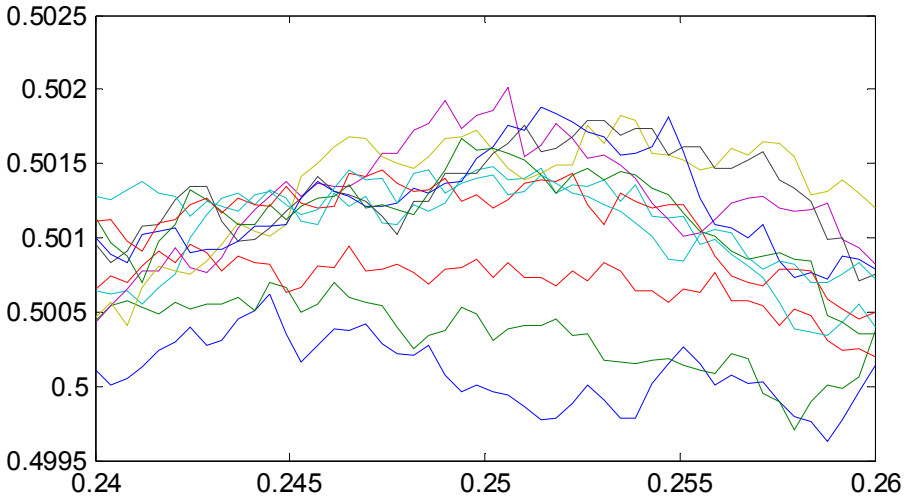
Предварительная нормировка по сумме

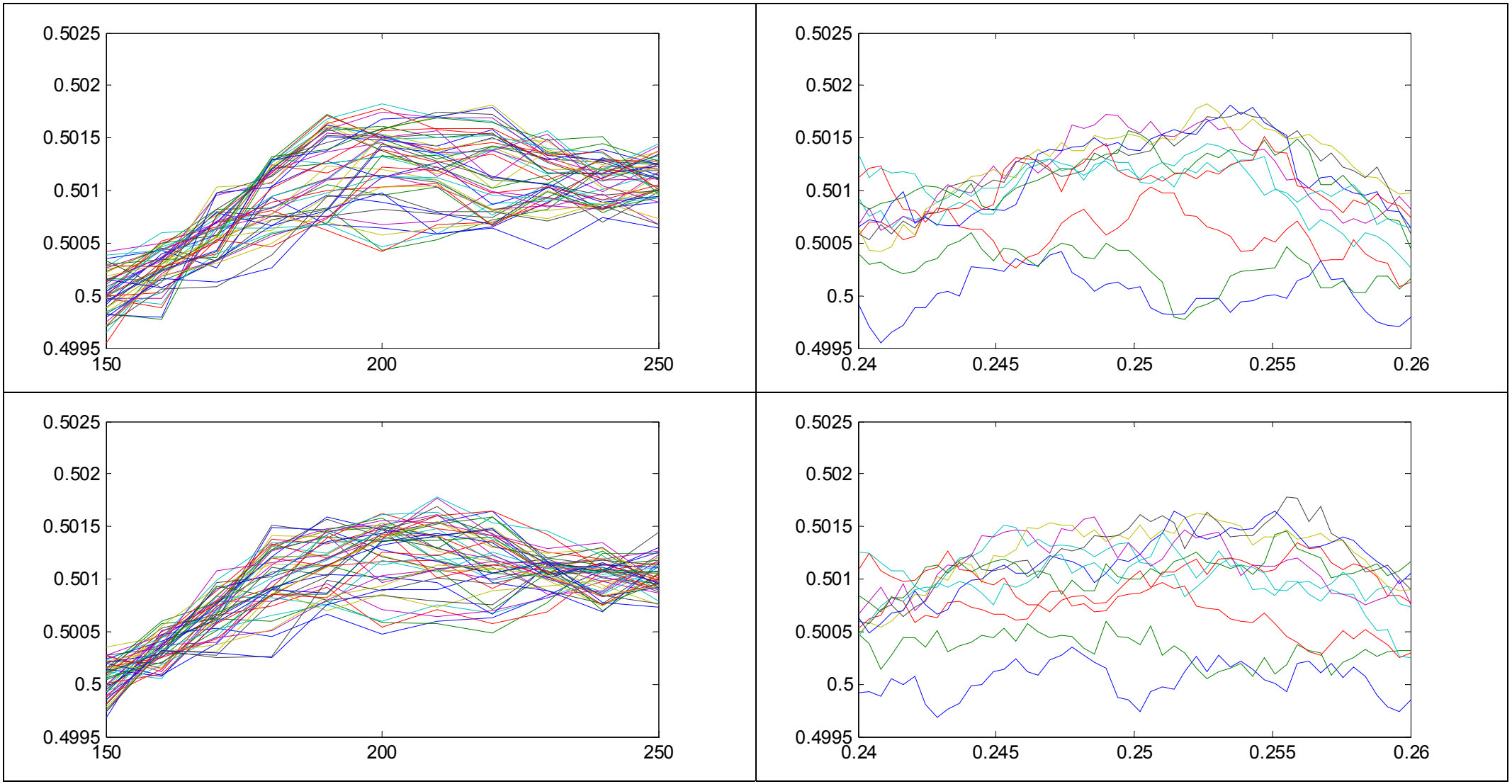


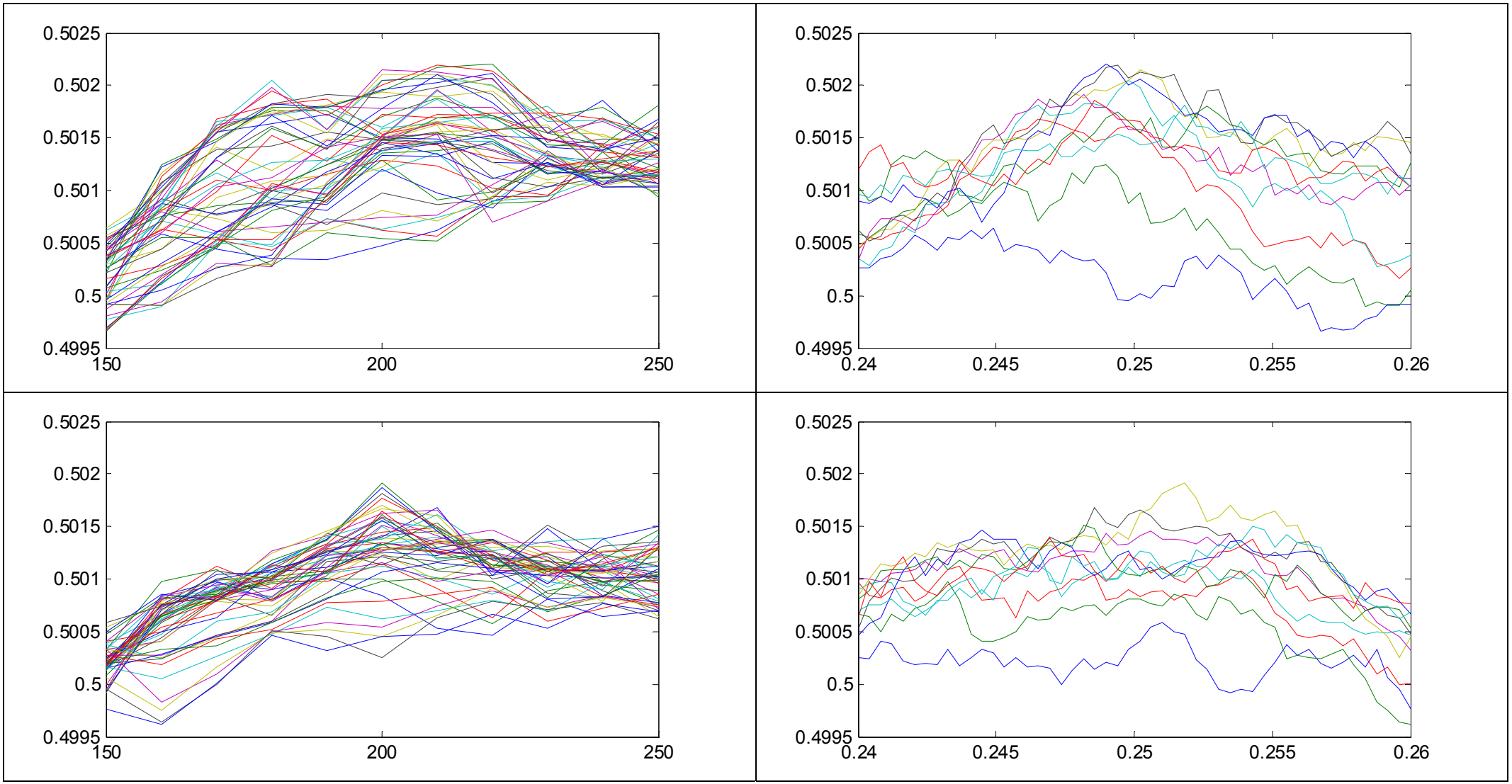
Предварительная нормировка по норме

```
% матрица эталонов
St = L'*S;
St = full(St);
St = bsxfun(@rdivide, St, sum(L,1)'); % ПРЕДВАРИТЕЛЬНАЯ НОРМИРОВКА ПО СУММЕ

St = bsxfun(@rdivide, St, sqrt(sum(St.^2,2)+0.001));

% матрица оценок
R = S*St';
R = bsxfun(@rdivide, R, max(R,[],2));

theta = linspace(0, 1, 50);
E = zeros([1  50]);
for i=1:50
    E(i) = Fperformance(L, bsxfun(@ge, R, theta(i)));
end
```

Контроль по фолдам при разных объёмах фолдов (50, 100, 250, 500, 1000)

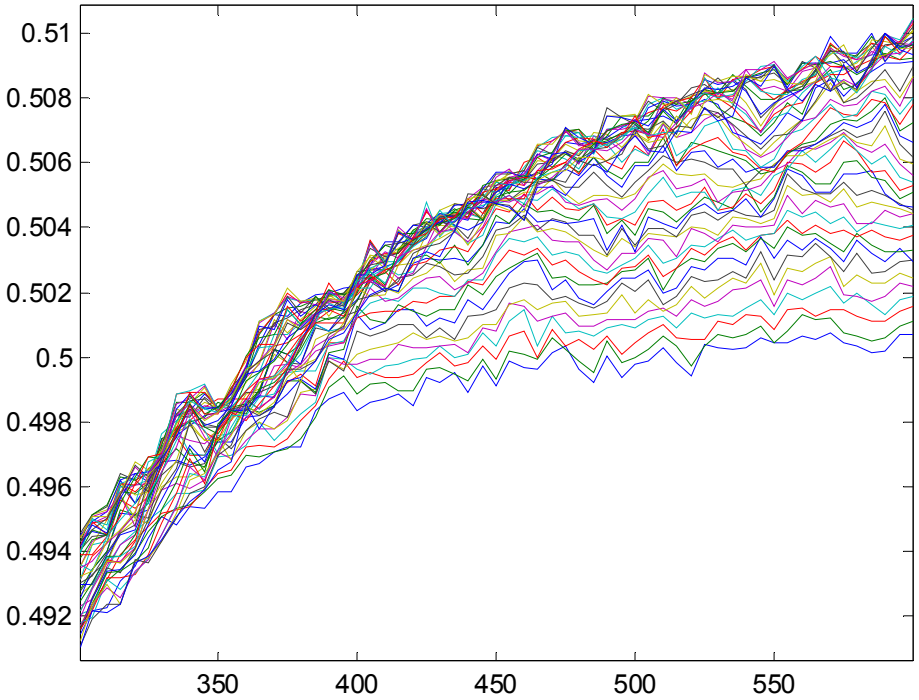
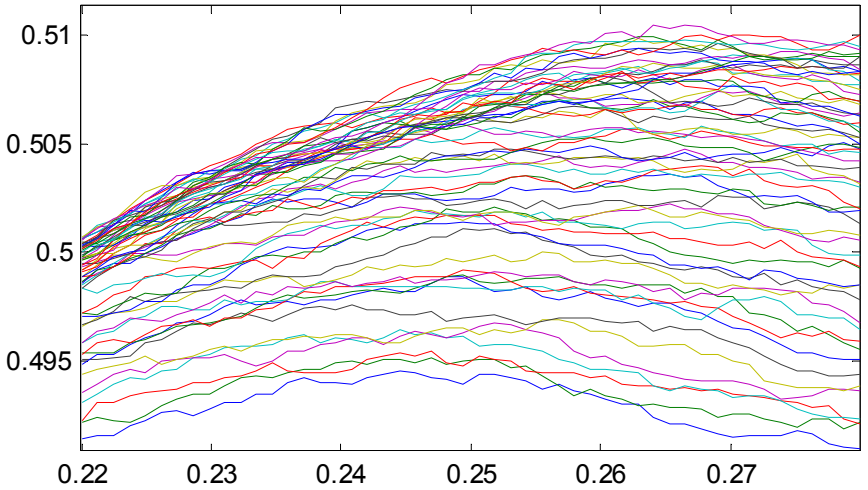
Зависимость качества от числа соседей (разным цветом показаны алгоритмы при различных значениях порога)	Зависимость качества от значения порога (разным цветом показаны алгоритмы при различном числе соседей)
	

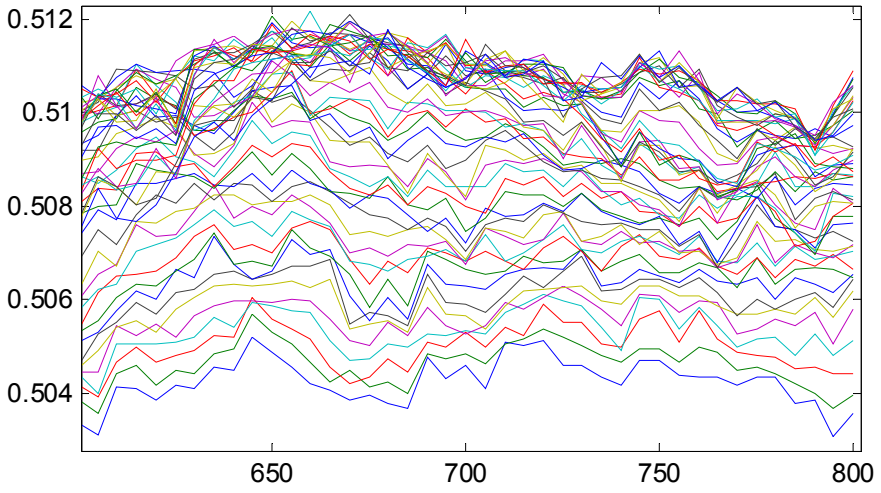

0.5025
0.502
0.5015
0.501
0.5005
0.5
0.4995
150
200
250
0.24
0.245
0.25
0.255
0.26

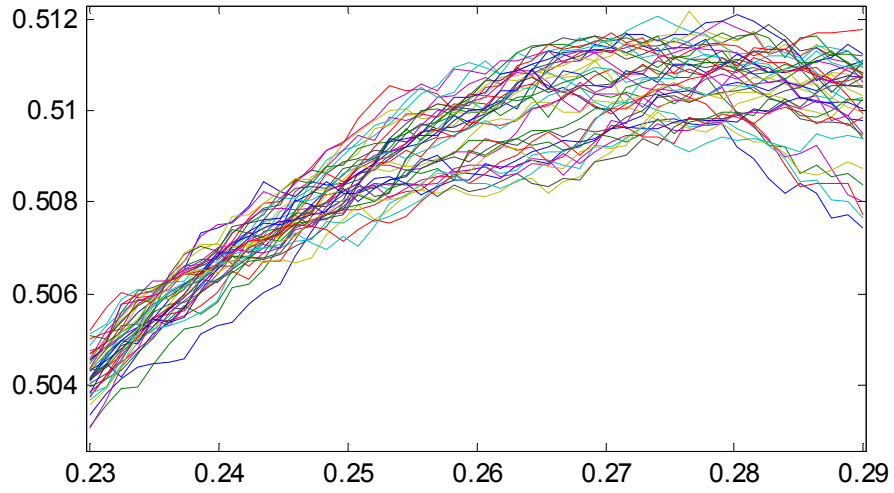


Вывод:
в целом качество не меняется. Для экспериментов решено выбрать размер фолда=500 (из-за достаточно быстрой скорости вычисления).

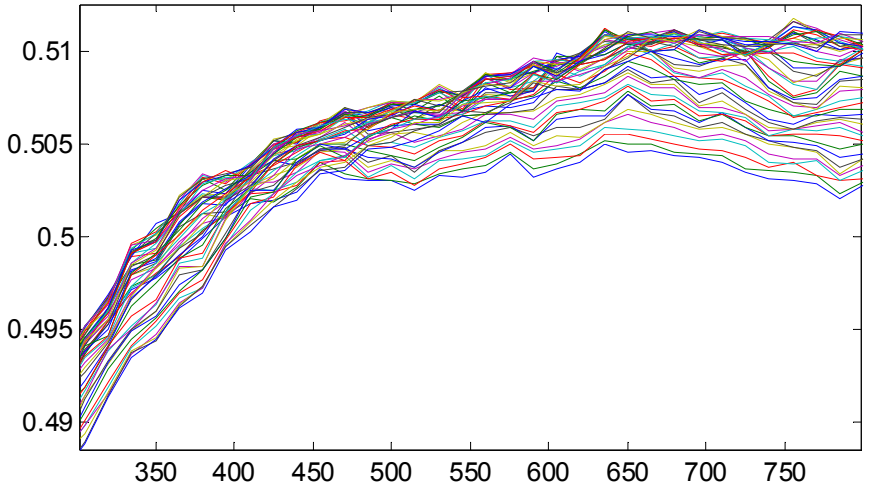
Качество алгоритма, основанного на SVD-разложении

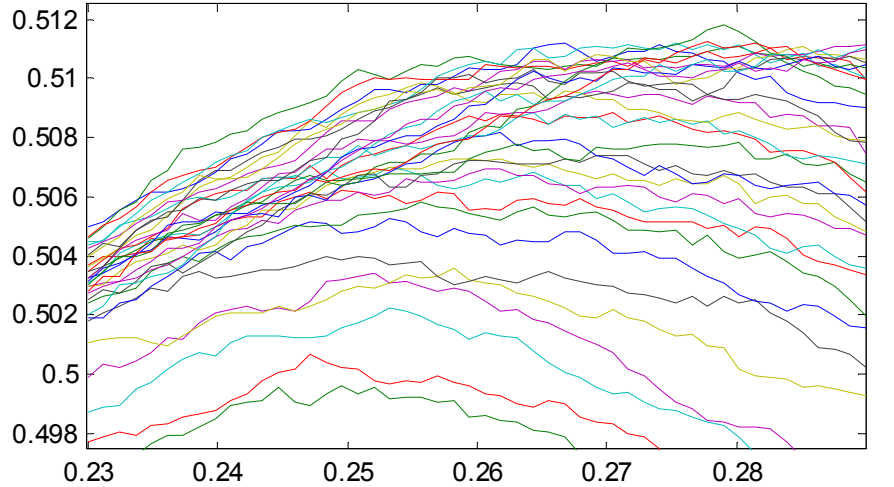
Зависимость качества от числа признаков	Зависимость качества от порога
	



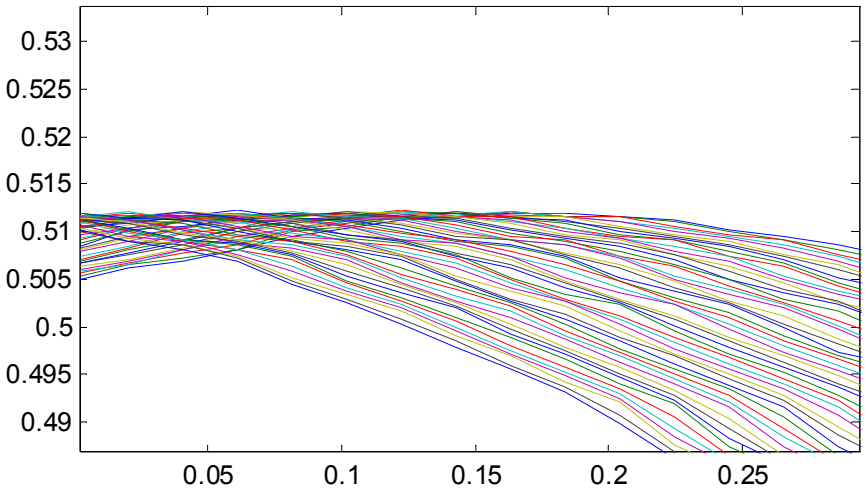
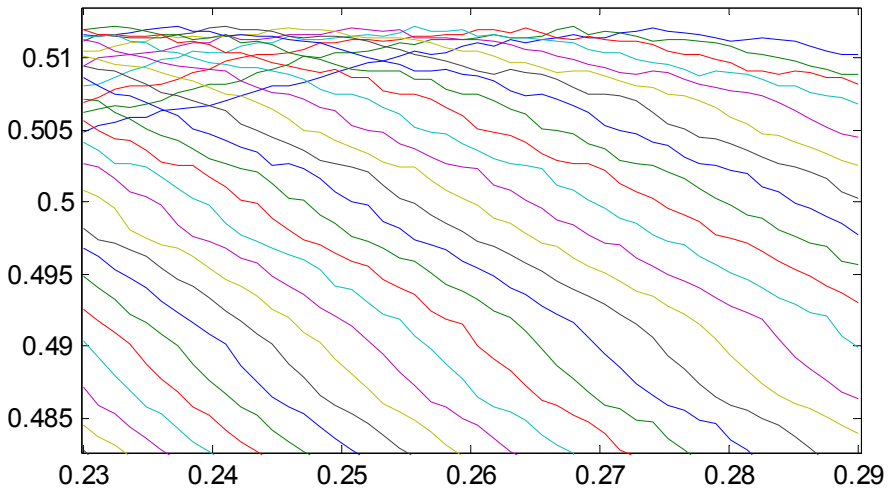


Аналогичные эксперименты на перемешанном множестве





SVD-разложение + регуляризация

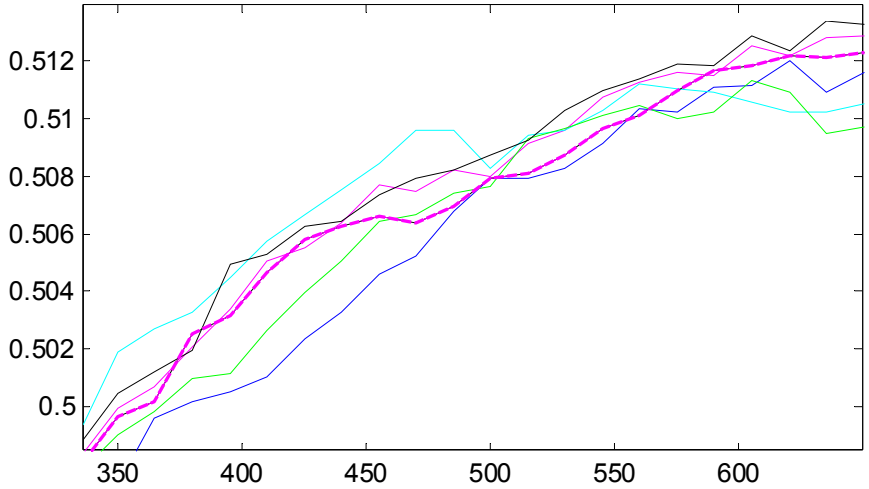
Зависимость качества от числа признаков	Зависимость качества от порога
	

Вывод:

регуляризацию делать бесполезно.

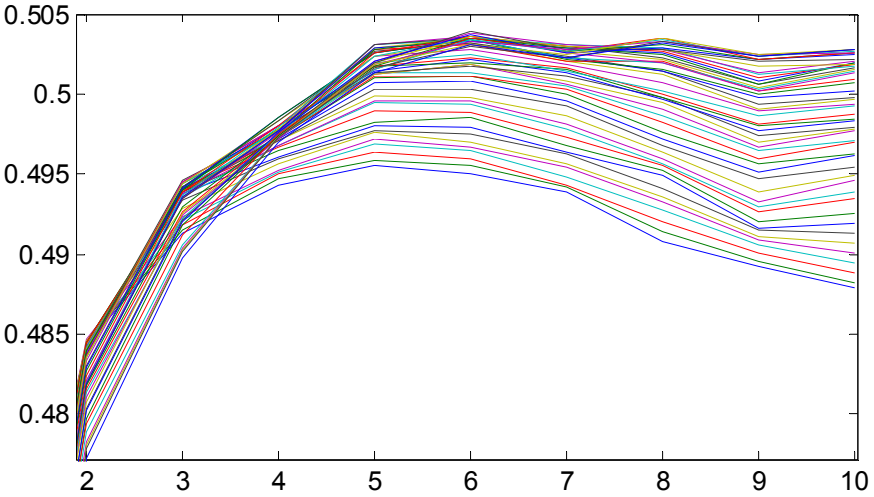
Почему-то при ней пропадает понятие «оптимального порога».

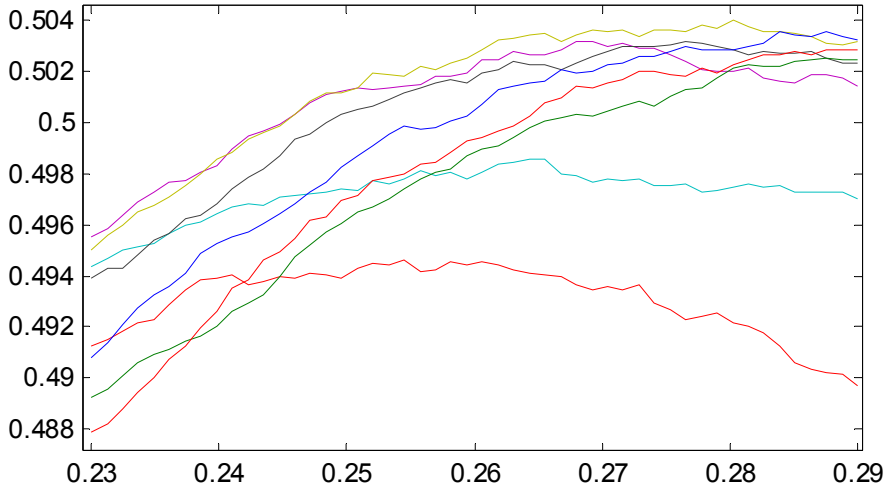
Предварительная нормировка + SVD-разложение



Синий – `S = bsxfun(@rdivide, S, (sum(S, 2)));` **Качество = 51.16**	Малиновый – `S = bsxfun(@rdivide, S, sqrt(sum(S, 2)));` **Качество = 51.29**	Голубой – `SW = L'*S;` `SW = sqrt(mean(log(full(SW)+1)))+0.001;` `S = bsxfun(@rdivide, S, SW);` `S = bsxfun(@rdivide, S, sqrt(sum(S, 2)));` **Качество = 51.05**
Зелёный – `S = sqrt(S);` **Качество = 51.13**	Утолщённый малиновый – Как и малиновый, но после перемешивания выборки. **Качество = 51.23**	Чёрный – `S = bsxfun(@rdivide, S, sqrt(sum(S, 2)));` **Но на фолдах объёма 50, а не 500.**

Попытка сделать SVD с учётом разбиения по классам






```
% + SVD преобразование ЗАВИСЯЩЕЕ ОТ КЛАССОВ
% для внешнего запуска SVD
load S.mat
load L.mat

nn = (1:10000)'; % шаблон

deltaII = 100; %50;
% это пробегают параметры
KK = 1:20; %150:10:250;
II = 1:deltaII:10000;
TT = linspace(0.23,0.29,50); % 0.24,0.26 0.24 0.265

E = zeros([length(KK), length(II), length(TT)]);
```

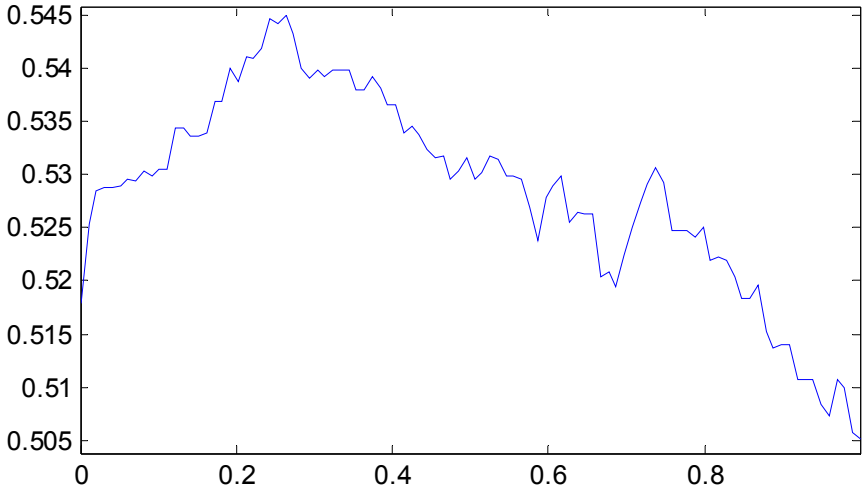
```
% матрица оценок SVD-методом
function Ltest = myMEDclassSVD(Strain, Ltrain, Stest, Nclass)

Strain = bsxfun(@rdivide, Strain, sqrt(sum(Strain, 2)));
Stest = bsxfun(@rdivide, Stest, sqrt(sum(Stest, 2)));


l = sum(Ltrain);
l(l>Nclass) = Nclass;
suml = sum(l)+Nclass;

Ttrain = zeros([size(Strain,1), suml]);
Ttest = zeros([size(Stest,1), suml]);
```

```
tic

for k = 1:length(KK)

    for i = 1:length(II)
        Itest = (nn>=II(i))&(nn<(II(i)+deltaII));
        Itrain = ~Itest;

        Ltest = myMEDclassSVD(S(Itrain,:), L(Itrain,:), S(Itest,:), KK(k));
        for t = 1:length(TT)
            E(k,i,t) = Fperformance(L(Itest,:),Ltest>=TT(t));
        end
    end;
    k
end;

toc
```

```
j = 1;
for i=1:83
    [~,~,Sc] = svds(Strain(Ltrain(:,i)~=0,:), min(Nclass,sum(Ltrain(:,i)~=0)));
    jnext = size(Sc,2);

    Ttrain(:,j:(j+jnext-1)) = Strain*Sc;
    Ttest(:,j:(j+jnext-1)) = Stest*Sc;

    j = j+jnext;
    %i
end;
[~,~,Sc] = svds(Strain, Nclass);
jnext = size(Sc,2);
Ttrain(:,j:(j+jnext-1)) = Strain*Sc;
Ttest(:,j:(j+jnext-1)) = Stest*Sc;

% решение системы линейных уравнений
Ltest = Ttest*( (Ttrain'*Ttrain)\(Ttrain'*Ltrain) );
```

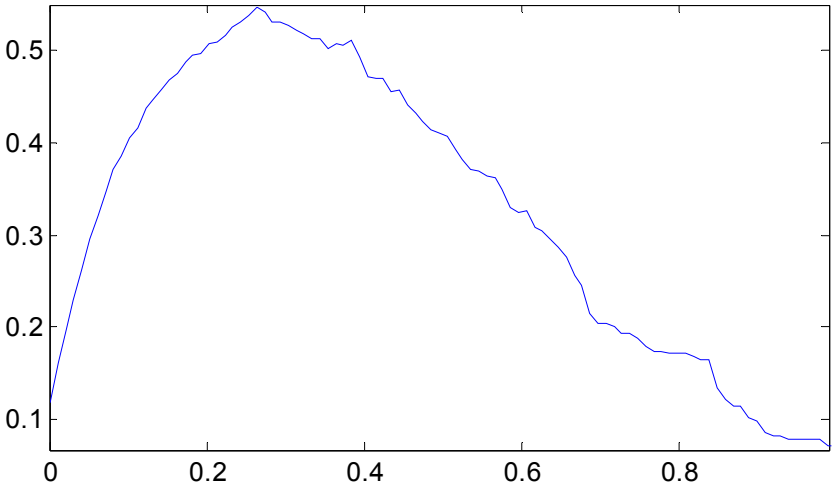
В дальнейшем от этого алгоритма автор отказался (он был хуже, чем NN), хотя…
здесь огромное поле для экспериментов.
Возможно, именно модификация этого алгоритма является наиболее удачной для этой задачи.

Смесь ближайшего соседа и SVD



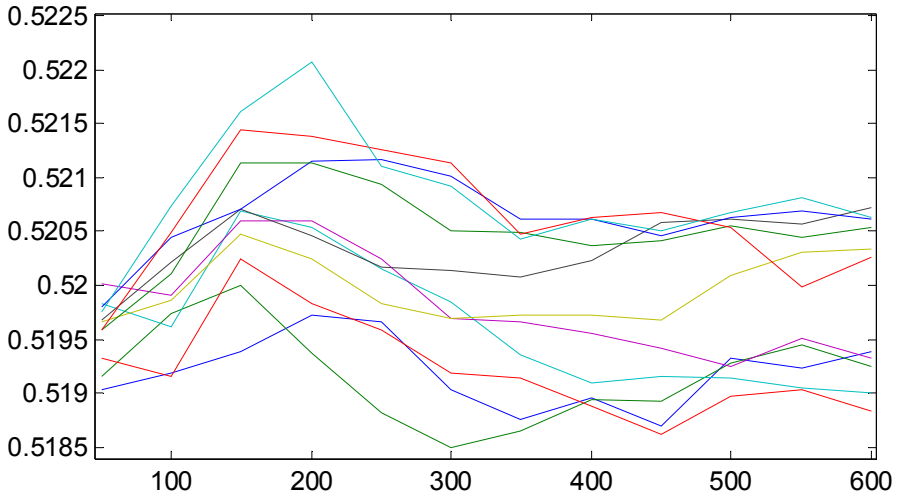
Качество от значения коэффициента при фиксированном пороге
Fperformance(L0,(коэффициент*L1+(1- коэффициент)*L3)>=0.2626)
=
0.5450

Вывод:
лучше использовать «смесь алгоритмов».

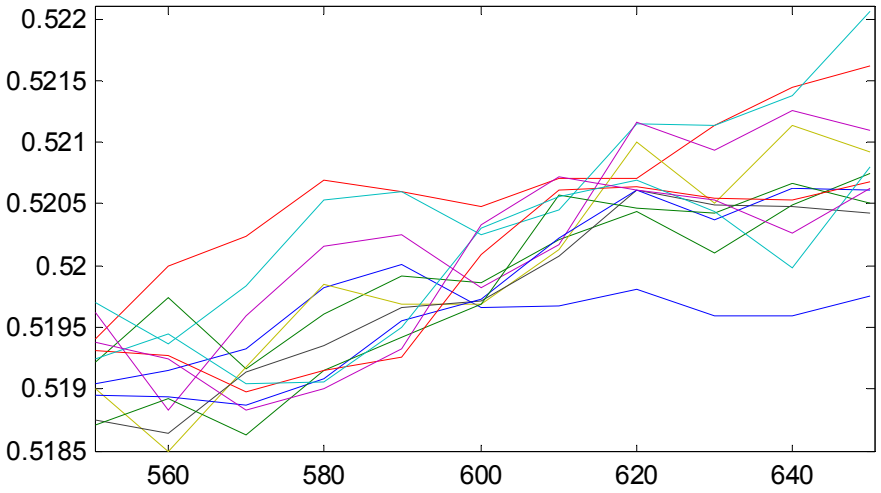


Качество от значения порога
Fperformance(L0,(0.26*L1+(1- 0.26)*L3)>= порог)

Варьирование числа соседей в kNN и числа признаков в SVD при комбинации kNN + SVD



Число соседей



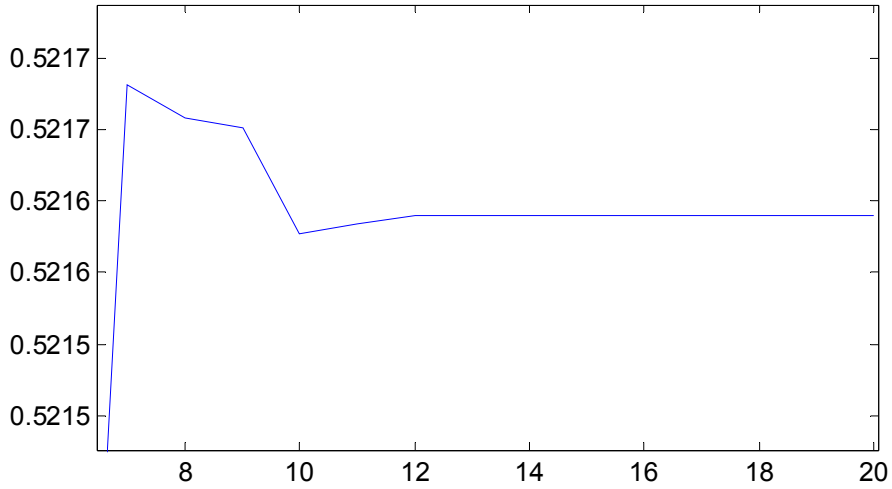
Число признаков

>=650 в SVD
~ 200 соседей в kNN

```
tic

deltaII = 100; %50;
% это пробегают параметры
KK = 50:50:600; %150:10:250;
KK2 = 550:10:650;
II = 1:deltaII:10000;

E = zeros([length(KK), length(KK2), length(II)]);

for k = 1:length(KK)

Yknn = myMEDclass2rec(S, L, S, KK(k), 0.4310);

    for i = 1:length(II)
        Itest = (nn>=II(i))&(nn<(II(i)+deltaII));
        Itrain = ~Itest;

        for k2 = 1:length(KK2)

        Lsvd = Ysvd(Itest,1:KK2(k2))*(
(Ysvd(Itrain,1:KK2(k2))'*Ysvd(Itrain,1:KK2(k2)))\(Ysvd(Itrain,1:KK2(k2))'*L(Itrain,:))
);
        Lknn = Yknn(Itest,:)*(
(Yknn(Itrain,:)'*Yknn(Itrain,:))\(Yknn(Itrain,:)'*L(Itrain,:)) );

                E(k,k2,i) = Fperformance(L(Itest,:), (0.6532*Lsvd +
0.3468*Lknn)>=0.2564 );

        end;

        %i
    end;
    k
end;

toc
```

Ограничение на число ответов сверху

Попытка поставить порог – делать классификацию не более чем на Q классов

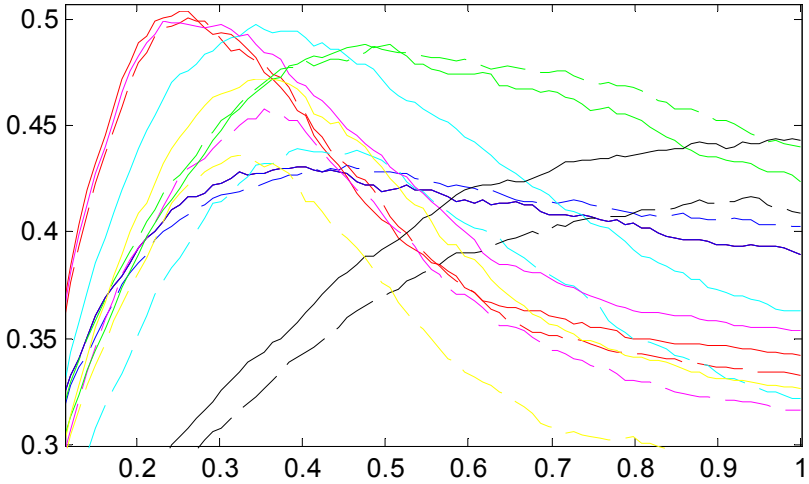


качество от Q

Вывод:
заметно улучшение при Q=7.
К сожалению, такого эффекта нет на лидерборде.

Далее мог бы быть блок с кучей графикой, но, к сожалению, была допущена ошибка… При верификации центроидного алгоритма нельзя использовать схему Leave-one-out! Этого простого факта нет в пособиях по классификации… но автор пришёл к нему в результате экспериментов. При, казалось бы, правильной схеме контроля происходит жуткое преобучение (~70% на локальных тестах, ~40% на лидерборде).

Эксперименты с LIBSVM



Качество при различных нормировках

максимум – 50.39

Синий – по сумме вертикальная

Красный – по сумме горизонтальная

Голубой – по максимуму вертикальная

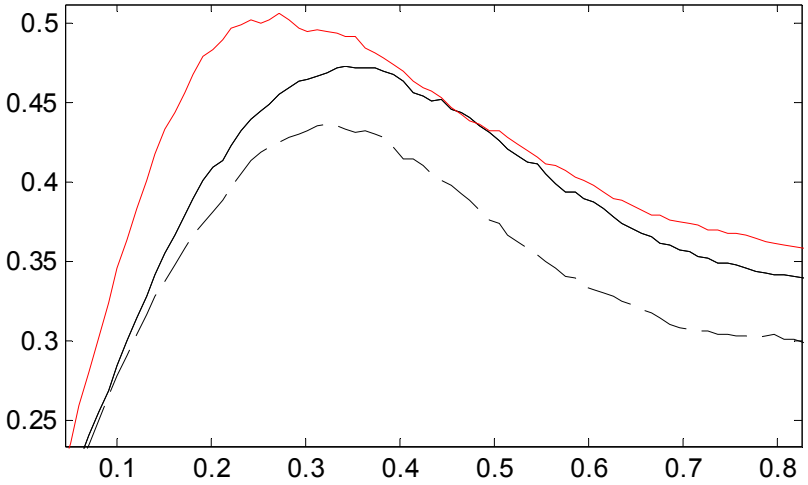
Малиновый – по максимуму горизонтальная

Зелёный –корень из суммы (вертикальная+горизонтальная)

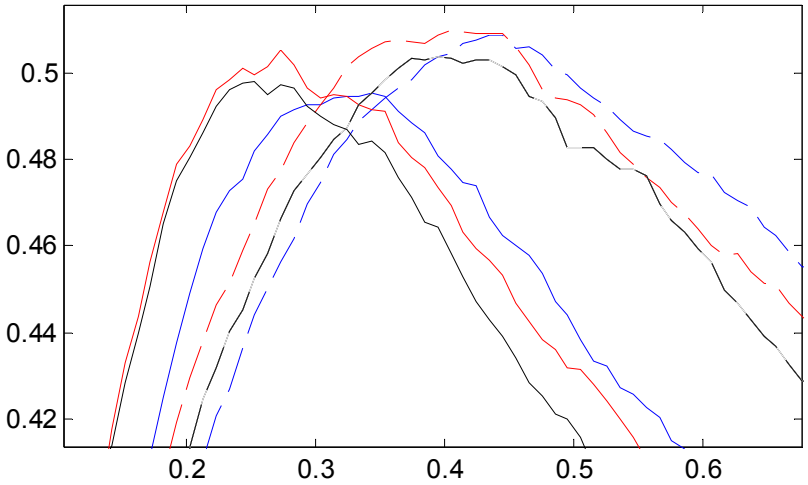
Чёрный – корень суммы вертикальная

Жёлтый – корень суммы горизонтальная

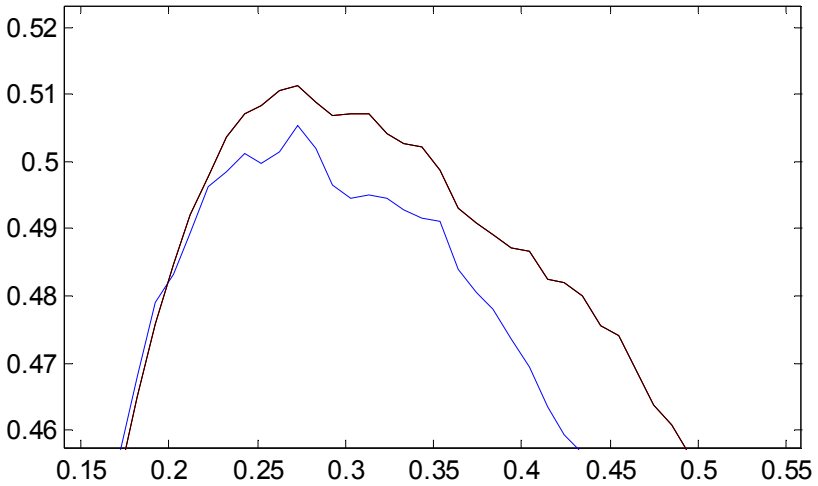
Здесь, правда, не совсем корректная нормировка по максимуму.



Чёрный и красный – разница между правильной и неправильной нормировкой по максимуму
50.5



Нормировка по максимуму по столбцам
Синий - `-c 0.1`
Красный - `-c 0.01`
Чёрный - `-c 0.001`



Чёрный – при пополнении признакового пространства модулями ответов СВМа

ТУТ интересный эффект, который не был использован в финальном решении…