МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Теоретической и прикладной информатики
(полное название кафедры)

УТВЕРЖДАЮ

Зав. кафедрой Чубич В.М.
(фамилия, имя, отчество)

(подпись)

«_____»_____ 2017 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Овчинникова Ильи Дмитриевича
(фамилия, имя, отчество студента – автора работы)

Мультиядерный метод опорных векторов наименьших квадратов для задач классификации
(тема работы)

Факультет Прикладной математики и информатики
(полное название факультета)

Направление подготовки 02.03.03. Математическое обеспечение и администрирование информационных систем
(код и наименование направления подготовки бакалавра)

Руководитель от НГТУ

Попов А.А.
(фамилия, имя, отчество)

д.т.н., профессор
(ученая степень, ученое звание)

(подпись, дата)

Автор выпускной квалификационной работы

Овчинников И.Д.
(фамилия, И.О.)

ФПМИ, ПМИ-31
(факультет, группа)

Новосибирск, 2017 г.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Теоретической и прикладной информатики
(полное название кафедры)

УТВЕРЖДАЮ

Зав. кафедрой Чубич В.М.
(фамилия, имя, отчество)

«21» марта 2017 г.

(подпись)

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

студенту ***Овчинникову Илье Дмитриевичу***
(фамилия, имя, отчество студента)

Направление подготовки *02.03.03. Математическое обеспечение и администрирование информационных систем*

Факультет Прикладной математики и информатики

Тема ***Мультиядерный метод опорных векторов наименьших квадратов для задач классификации***

Исходные данные (или цель работы):

Программная реализация алгоритма мультиядерного метода опорных векторов наименьших квадратов для задач классификации, исследование влияния параметров алгоритма, сравнение мультиядерного и одноядерного подходов на данных прикладных задач.

Структурные части работы:

Работа содержит введение, 3 главы, заключение и 1 приложение. В первой главе описана постановка задачи и алгоритм ее решения. Вторая глава содержит описание разработанной программы. Третья глава – результаты проведенных исследований. Приложение – исходных код программы.

Задание согласовано и принято к исполнению.

Руководитель от НГТУ	**Студент**
Попов А.А.	*Овчинников И.Д.*
(фамилия, имя, отчество)	(фамилия, имя, отчество)
д.т.н., профессор	*ФПМИ, ПМИ-31*
(ученая степень, ученое звание)	(факультет, группа)
21.03.2017 г.	*21.03.2017 г.*
(подпись, дата)	(подпись, дата)

Тема утверждена приказом по НГТУ № *1584/2* от «*21*» *марта 2017* г.

ВКР сдана в ГЭК № _____, тема сверена с данными приказа

21 марта 2017 г.
(подпись секретаря экзаменационной комиссии по защите ВКР, дата)

Волкова В.М.
(фамилия, имя, отчество секретаря экзаменационной комиссии по защите ВКР)

АННОТАЦИЯ

Отчет 44 с., 3 ч., 4 рис., 16 табл., 23 источника, 1 прил.

SUPPORT VECTOR MACHINES (SVM), LEAST SQUARES SUPPORT VECTOR MACHINES (LS SVM), МЕТОД ОПОРНЫХ ВЕКТОРОВ НАИМЕНЬШИХ КВАДРАТОВ, MULTIPLEKERNEL LEARNING, МУЛЬТИЯДЕРНОЕ ОБУЧЕНИЕ, RBF KERNEL, POLYNOMIAL KERNEL, КЛАССИФИКАЦИЯ, СКОЛЬЗЯЩИЙ КОНТРОЛЬ, K-FOLD CROSS-VALIDATION (K-FOLD CV).

Объектом исследования является модель классификации, построенная при использовании мультиядерного метода опорных векторов наименьших квадратов.

Цель работы – разработка программы, реализующей мультиядерный метод опорных векторов наименьших квадратов.

В процессе работы изучалось влияние параметров алгоритма на точность модели и было проведение сравнение мультиядерного и одноядерного подходов. Оценивание модели проводилось с помощью процедуры скользящего контроля.

В результате работы была выполнена поставленная цель. На основе программы были произведены исследования работоспособности рассматриваемого алгоритма и выявлены потенциальные направления дальнейшего изучения в этой области.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Классификация – раздел машинного обучения, который ставит перед собой математическую задачу распознавания объектов. Постановка задачи заключается в следующем. Имеется набор наблюдений за объектами, о которых известна их принадлежность к конкретному классу. Этот набор принято называть обучающей выборкой. Требуется построить такую модель, которая будет способна классифицировать объект максимально достоверно. Под классификацией объекта понимается определение наименования соответствующего ему класса. Для того что получить работоспособную модель классификатора, проводится его обучение. Обучение классификатора представляет собой оценивание оптимальных параметров модели на основе обучающей выборки.

С ростом информации, участвующей в анализе данных, ее обработка, необходимость которой возникает у крупных и мелких предприятий, а также в научной деятельности, становится более затруднительной. Для этого следует искать наиболее эффективные и удобные способы хранения данных и алгоритмы их обработки.

Большое количество научных и коммерческих задач сводится к задаче классификации. Примерам таких задач могут являться диагностика заболеваний, поиск месторождений ископаемых, оценивание кредитоспособности заёмщиков, распознавание символов и звуков, классификация документов и почтовых сообщений, предсказание успешности коммерческого предприятия или оттока клиентов. Необходимость анализа данных была актуальной с древних времен, но в современном мире решение этой проблемы стало широко доступно благодаря вычислительной мощности компьютеров.

Существует множество алгоритмов классификации, обладающие своими достоинствами и недостатками. Одним из алгоритмов, хорошо показавший себя в исследовательской деятельности, является метод опорных векторов (Support

Vector Machine). Основная идея метода заключается в построении оптимальной гиперплоскости между классами для минимизации эмпирической ошибки.

Метод опорных векторов является линейным алгоритмом классификации. Однако в реальности зачастую случается так, что данные не могут быть разделены линейной гиперплоскостью. Для построения классификатора нелинейной разделимости используется переход от скалярных произведений в исходном пространстве к ядерным функциям (скалярным произведением в пространстве с большей размерностью), так называемый kernel trick. И уже в этом пространстве может быть построена разделяющая гиперплоскость.

В данной работе будет рассматриваться версия алгоритма метода опорных векторов с использованием квадратичной функции потерь (Least Squares Support Vector Machine). Такая модификация позволяет найти решение задачи классификации с помощью решения системы линейных алгебраических уравнений, вместо задачи выпуклого квадратичного программирования. Возможность решения задачи через СЛАУ является большим преимуществом, поскольку требуется меньшая вычислительная мощность компьютера. Кроме того, это позволяет решать задачи больших размерностей, что является очень важным критерием при выборе метода классификации в современном мире.

Кроме модификации с квадратичной функцией потерь, рассматривается использование комбинации ядерных функций (Multiple Kernel Learning). Мультиядерный подход может быть успешно применен в ряде прикладных задач, таких как распознавание объектов на изображениях и видео, биомедицинских задачах и множестве других.

В стандартном одноядерном подходе обучения классификатора исследователи вынуждены подбирать подходящие параметры ядерной функции методом проб и ошибок, и это требует больших усилий. Мультиядерный подход решает эту проблему успешнее и обладает рядом преимуществ. Во-первых, появляется возможность выбора оптимальной ядерной функции и параметров классификатора из большого набора ядер, путем автоматической оценки их весов во время процесса машинного обучения. Во-вторых,

предлагается одновременно использовать ядерные функции разных типов, что может повысить точность модели классификатора. В-третьих, появляется возможность совмещения разнородных данных (например, разные части одного изображения или разные факторы о состоянии человека), которые имеют разные понятия сходства, в один набор. Для каждого типа данных будет использоваться своя ядерная функция. В-четвертых, – определение наиболее весомых факторов, влияющих на точность модели.

Классификации с использованием мультиядерного подхода для метода опорных векторов посвящено множество работ [8, 9, 12, 14]. Однако проблема применения этого подхода с использованием LS SVM является менее исследованной [10, 15].

Целью данной работы является программная реализация мультиядерного метода опорных векторов наименьших квадратов, исследовать влияние параметров алгоритма на точность модели и провести сравнение с одноядерным подходом на прикладных задачах, а главное, определить преимущества и недостатки рассматриваемого метода.

1. ПОСТРОЕНИЕ МОДЕЛИ КЛАССИФИКАТОРА С ИСПОЛЬЗОВАНИЕМ МК LS SVM

1.1. Постановка задачи

Метод опорных векторов (SVM) – классификатор, предложенный для задач бинарной классификации и основывается на теории минимизации среднего риска. Данные могут быть представлены как набор n наблюдений

$$(x_1, y_1),(x_2, y_2),...,(x_n, y_n) \in S = \mathsf{X} \times Y,$$

где $\mathsf{X} \in X$ – множество описаний объектов, $Y \in Y$ – множество номеров (или наименований) классов. Рассматривается случай бинарной классификации, так что $Y \in \{-1,+1\}$. Существует неизвестная целевая зависимость — отображение $y^* : X \to Y$, значения которой известны только на объектах конечной обучающей выборки S. Требуется построить модель, способную классифицировать произвольный объект $x \in X$. В формулах используется различный размер шрифта.

Признаком называется отображение $f : X \to D_f$, где D_f — множество допустимых значений признака. Если заданы признаки $f_1, f_2,..., f_m$, то вектор $\bar{F} = [\, f_1(x), f_2(x),..., f_m(x)]^T$ называется признаковым описанием объекта $x \in X$. Признаковые описания допустимо отождествлять с самими объектами. При этом множество $X = D_{f_1} \times D_{f_2} \times ... \times D_{f_n}$ называют признаковым пространством.

1.2. Ядерные функции и спрямляющее пространство

Большим прогрессом в теории SVM было предложенная Вапником в 1995 году возможность [6, 7] решать не только задачи линейной разделимости, но и нелинейной. Для того чтобы достичь этого, выполняется переход из исходного пространства факторов в пространство высокой размерности $\Psi \in R^{nf}$. Переход выполняется посредством отображения $\varphi : \mathsf{X} \to \Psi$. После этого возможно построение линейной разделяющей гиперплоскости в полученном пространстве.

Следует отметить, что для построения линейной функции в полученном пространстве признаков Ψ отсутствует необходимость рассмотрения пространства признаков в явном виде. Достаточно заменить скалярное произведение в пространстве признаков $\varphi(x_k)^T \varphi(x_l)$ соответствующим ядром $K(x_k, x_l)$, удовлетворяющим условия Мерсера, так называемый «Kernel trick».

Теорема Мерсера

Пусть $K \in L^2(C)$, $g \in L^2(C)$, где C является компактным подмножеством R^d и $K(t,z)$ описывает скалярное произведение в некотором пространстве признаков. Чтобы гарантировать, что непрерывная симметричная функция K имеет представление

$$K(t,z) = \sum_{k=1}^{\infty} \alpha_k \phi_k(t) \phi_k(z)$$

с положительными коэффициентами $\alpha_k > 0$, (т.е. $K(t,z)$ описывает скалярное произведение в некотором пространстве признаков), необходимо и достаточно, чтобы выполнялось условие

$$\int_C \int_C K(t,z) g(t) g(z) \partial t \partial z \geq 0$$

действительное для всех $g \in L^2(C)$.

Существует ряд ядер, которые могут быть использованы в моделях метода опорных векторов: линейные, полиномиальные, радиальные базисные функции (RBF) и сигмовидные. В данной работе исследования будут проводиться с использованием следующих ядер:

полиномиальное: $K(x,z) = (x^T z + c)^d$;

радиально базисная функция (RBF): $K(x,z) = \exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right)$.

1.3. Метод опорных векторов наименьших квадратов

Положительным аспектом нелинейной классификации метода опорных векторов является постановка выпуклой задачи квадратичного программирования, что означает единственное решение. Но стоит задать естественный вопрос, как можно упростить формулировку SVM без потери каких-либо преимуществ метода.

Сайкенсом был предложен [5] классификатор, формулировка которого позволяет найти решение через систему линейных уравнений вместо задачи квадратичного программирования, что для многих практических задач является более простым способом. Это достигается путем использования квадратичной функции потерь и замене ограничений в виде неравенства на равенство.

1.3.1. Формальная постановка

Оптимальная разделяющая гиперплоскость в классической формулировке SVM находится через решение следующей задачи оптимизации:

$$\min_{w,b,e} J_P(w,e) = \frac{1}{2}\|w\|^2 + \gamma \frac{1}{2}\sum_{k=1}^{N}\xi_k \qquad (1)$$

при условии

$$y_k\left[w^T\varphi(x_k)+b\right] \geq 1-\xi_k, \xi_k \geq 0, k=\overline{1,N}, \qquad (2)$$

где $b \in R$ – параметр сдвига, $\varphi(x)$ – функция перевода из исходного пространства факторов в пространство большей размерности, вектор w – перпендикуляр к разделяющей плоскости, $\gamma \in R$ – параметр регуляризации, который позволяет находить компромисс между максимизацией ширины разделяющей гиперплоскости и минимизацией суммарной ошибки. Чем больше параметр γ, тем больше идет упор на минимизацию суммарной ошибки. Чем γ меньше, тем больше упор максимизацию ширины разделяющей гиперплоскости.

Для того чтобы получить формулировку LS SVM заменим в (1) функцию потерь на квадратичную функцию потерь, и в (2) ограничения неравенства с ξ_k – ограничением равенства с переменными ошибки e_k:

$$\min_{w,b,e} J_P(w,e) = \frac{1}{2}\|w\|^2 + \gamma \frac{1}{2}\sum_{k=1}^{N} e_k^2 \tag{3}$$

при условии

$$y_k\left[w^T\varphi(x_k) + b \right] = 1 - e_k, k = \overline{1,N}. \tag{4}$$

Классификатор в исходном пространстве имеет вид:

$$y(x) = sign\left[w^T\varphi(x) + b \right]. \tag{5}$$

По теореме Каруша – Куна – Таккера (ККТ) эта задача эквивалентна двойственной задаче поиска седловой точки функции Лагранжа:

$$\mathcal{L}(w,b,e;\alpha) = J_P(w,e) - \sum_{k=1}^{N}\alpha_k\left\{ y_k\left[w^T\varphi(x_k) + b \right] - 1 + e_k \right\}, \tag{6}$$

где α – это коэффициенты Лагранжа, определяющие опорные вектора.

Условия оптимальности:

$$\begin{cases} \dfrac{\partial\mathcal{L}}{\partial w} = 0 \to w = \sum\limits_{k=1}^{N}\alpha_k y_k\varphi(x_k) \\ \dfrac{\partial\mathcal{L}}{\partial b} = 0 \to \sum\limits_{k=1}^{N}\alpha_k y_k = 0 \\ \dfrac{\partial\mathcal{L}}{\partial e_k} = 0 \to \alpha_k = \gamma e_k, & k = \overline{1,N} \\ \dfrac{\partial\mathcal{L}}{\partial \alpha_k} = 0 \to y_k\left[w^T\varphi(x_k) + b \right] - 1 + e_k = 0, & k = \overline{1,N} \end{cases} \tag{7}$$

После исключения w, e из (4) уравнение преобразуется в следующее условие ККТ:

$$y_k\left(\sum_{l=1}^{N}\alpha_l y_l K(x_k,x_l)+b\right)+\frac{\alpha_k}{\gamma}=1, k=\overline{1,N}, \tag{8}$$

которое может быть представлено в виде системы линейных алгебраических уравнений:

$$\begin{bmatrix}0 & y^T\\ y & \Omega+\frac{1}{\gamma}\mathrm{I}_n\end{bmatrix}\begin{bmatrix}b\\ \alpha\end{bmatrix}=\begin{bmatrix}0\\ 1_v\end{bmatrix}, \tag{9}$$

где $y=[y_1;...;y_N]$, $\alpha=[\alpha_1;...;\alpha_N]$, $1_v=[1;...;1]$, $\Omega_{kl}=y_k y_l K(x_k,x_l), k,l=\overline{1,N}$.

Результирующая LS SVM модель для оценки функции имеет вид:

$$y(x)=sign\left[\sum_{k=1}^{N}\alpha_k y_k K(x_k,x)+b\right]. \tag{10}$$

1.3.2. Решение системы ККТ LS SVM

Обучение LS SVM классификатора характеризуется решением квадратной системы линейных уравнений. Разработано большое множество надежных алгоритмов для решения СЛАУ.

Размер матрицы в системе растёт с увеличением наблюдений в наборе данных. Для больших наборов данных рекомендуется использовать итеративные методы. Примерами могут являться методы релаксации, сопряженных градиентов (МСГ) и другие. Тем не менее, не все итеративные методы могут быть применены для решения СЛАУ такого типа. Например, для МСГ матрица должна быть положительно определена. Из-за параметра сдвига b в модели LS SVM результирующая матрица не положительно определённая. Соответственно необходимо применить некоторые преобразования для получения положительно определенной системы.

Система ККТ LS-SVM (9) имеет вид

$$\begin{bmatrix} 0 & y^T \\ y & H \end{bmatrix} \begin{bmatrix} \xi_1 \\ \xi_2 \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \end{bmatrix}, \tag{11}$$

где $H = \Omega + \frac{1}{\gamma} \mathrm{I}_n,\ \xi_1 = b,\ \xi_2 = \alpha,\ d_1 = 0,\ d_2 = 1_v$.

Она может быть преобразована к виду

$$\begin{bmatrix} s & 0 \\ 0 & H \end{bmatrix} \begin{bmatrix} \xi_1 \\ \xi_2 + H^{-1} y \xi_1 \end{bmatrix} = \begin{bmatrix} -d_1 + y^T H^{-1} d_2 \\ d_2 \end{bmatrix}, \tag{12}$$

где $s = y^T H^{-1} y > 0 (H = H^T > 0)$. Так как S положительно и H положительно определена, то и матрица целиком положительно определена. В таком виде система решения LS-SVM классификатора может быть решена следующим алгоритмом с применением метода сопряженный градиентов.

Алгоритм решения LS SVM МСГ для больших размерностей:

1) получить η, ν из $H\eta = y,\ H\nu = 1_v$;
2) вычислить $s = y^T \eta$;
3) решение: $b = \eta^T 1_v / s, \alpha = \nu - \eta b$.

1.4. Мультиядерный подход

Мультиядерный подход заключается в использовании в качестве ядерной функции линейной комбинации других ядер:

$$K(x_i, x_j) = \sum_{d=1}^{M} \beta_d K_d(x_i, x_j), \qquad \beta_d \geq 0, \sum_{d=1}^{M} \beta_d = 1, \tag{13}$$

где M – это количество используемых ядер. Каждое ядро может использовать, как и весь набор факторов, так и какое-то его подмножество в случае если ядро предназначено только для обработки факторов определенного типа.

Подставив (13) в (8) получим:

$$y_k\left(\sum_{l=1}^{N}\alpha_l y_l\sum_{d=1}^{M}\beta_d K_d(x_k,x_l)+b\right)+\frac{\alpha_k}{\gamma}=1, k=\overline{1,N}. \quad (14)$$

При этом в (9) Ω определяется как $\Omega_{kl}=y_k y_l\sum_{d=1}^{M}\beta_d K_d(x_k,x_l), k,l=\overline{1,N}$.

Задача заключается в одновременном оценивании оптимальных коэффициентов Лагранжа и весовых коэффициентов ядерных функций. Этот подход известен, как мультиядерное обучение (Multiple Kernel Learning).

Коэффициенты ядерных функций могут быть вычислены через решения задачи минимизации функции потерь:

$$\min J(\beta,e)=\sum_{k=1}^{N}e_k^2, \beta_d\geq 0. \quad (15)$$

Подставив уравнения (4) и (7) в (15), получается следующая задача квадратичной оптимизации:

$$\min J(\beta,e)=\sum_{k=1}^{N}\left(1-y_k b-y_k\sum_{d=1}^{M}\beta_d\bar{K}_{i,d}\cdot\alpha\right)^2, \quad (16)$$

где $\bar{K}_{i,d}=\left[y_1K_d(x_i,x_1),...,y_NK_d(x_i,x_N)\right]$.

1.5. Алгоритм MK LS SVM

Зонненбургом [14] был предложен двухфазный итеративный алгоритм решения задачи мультиядерного обучения. Этот алгоритм в отличие от формулировки задачи полуопределенного программирования Ланкрейта [12] способен справляться с задачами большей размерности и ведет к лучшей точности классификации. На первой фазе выполняется обучение классификатора с фиксированными весами ядер. На второй фазе на основе полученного классификатора выполняется оценка весов ядерных функций. Действия выполняются итеративно до сходимости.

Процедура решения модели MK LS SVM для задачи классификации может быть определена как:

1) подготовить данные для обучения алгоритма, исходя из множества допустимых значений признаков D_f;
2) выбрать типы используемых ядерных функций и определить их параметры. Задать выбранные ядра как набор;
3) задать параметр регуляризации γ и начальные коэффициенты набора ядерных функций как $\left\{\beta_d^{(0)} = \frac{1}{M} \mid d = \overline{1,M}\right\}$;
4) получить коэффициенты Лагранжа $\alpha_j^{(t)}$ через решение системы линейный уравнений (9) с фиксированными коэффициентами ядер $\beta_d^{(t-1)}$ алгоритмом решения LS SVM МСГ для больших размерностей;
5) получить коэффициенты ядер $\beta_d^{(t)}$ через решение задачи квадратичного программирования (15) с фиксированными коэффициентами $\alpha_j^{(t)}$, полученных на предыдущем шаге;
6) проверить на сходимость алгоритм по невязке минимизируемой функции $\left|J^{(t-1)}(\beta,e) - J^{(t)}(\beta,e)\right| \le eps$. Если условие выполняется, то обучение модели закончено, иначе вернуться на шаг 4;
7) если результат полученной модели не удовлетворяет, то вернуться на шаг 2 или 3, задать новые параметры и повторить обучение новой модели.

1.6. Оценивание и выбор модели

Качество обучения алгоритма методом опорных векторов наименьших квадратов, зависит от множества параметров, задаваемых пользователем. Такими параметрами для LS SVM являются коэффициент регуляризации γ, тип выбранных ядерных функций и их параметры. Поэтому выбор модели, наилучшим образом аппроксимирующая предполагаемую исходную зависимость, является трудоемким и не простым занятием.

Кроме того, необходимо чтобы алгоритм обучения обладал способностью к обобщению, то есть чтобы вероятность ошибки на тестовой выборке не сильно отличалась от ошибки на обучающей выборке. Обобщающая способность тесно связана с понятиями переобучения и недообучения.

Переобучение — нежелательное явление, возникающее, когда вероятность ошибки обученного алгоритма на объектах тестовой выборки оказывается существенно выше, чем средняя ошибка на обучающей выборке.

Недообучение — нежелательное явление, возникающее, когда алгоритм обучения не обеспечивает достаточно малой величины средней ошибки на обучающей выборке.

Поэтому необходимо использовать специальные процедуры обучения алгоритмов для того чтобы получить несмещённую оценку вероятности ошибки.

Качество алгоритма оценивается по произвольной выборке прецедентов X^m с помощью функционала качества $Q(X^m)$. Для процедуры скользящего контроля не важно, как именно вычисляется этот функционал. Как правило, он аддитивен по объектам выборки:

$$Q(X^m) = \frac{1}{m} \sum_{x_i \in X^m} L(\alpha(x_i), y_i),$$

где $L(y_i)$ неотрицательная функция потерь, возвращающая величину ошибки ответа алгоритма при правильном ответе y_i.

Скользящий контроль или кросс-валидация (cross-validation, CV) — одна из основных процедура эмпирического оценивания обобщающей способности алгоритмов, обучаемых по прецедентам, применяемых при решении задач классификации.

Если выборка независима, то средняя ошибка скользящего контроля даёт несмещённую оценку вероятности ошибки. Это выгодно отличает её от средней ошибки на обучающей выборке, которая может оказаться смещённой

(оптимистически заниженной) оценкой вероятности ошибки, что связано с явлением переобучения.

Существует множество вариаций процедуры скользящего контроля такие как полный скользящий контроль (complete CV), случайные разбиения, контроль на отложенных данных (hold-out CV), контроль по отдельным объектам (leave-one-out CV), обладающие своими преимуществами и недостатками.

Контроль по k блокам (k-fold CV)

Выборка случайным образом разбивается на k непересекающихся блоков одинаковой (или почти одинаковой) длины $q_1, q_2, ..., q_k$:

$$X^L = X_1^{q_1} \cup ... \cup X_k^{q_k}, q_1 + ... + q_k = L.$$

Каждый блок по очереди становится контрольной подвыборкой, при этом обучение производится по остальным $k-1$ блокам. Для каждого разбиения n строится алгоритм $a_k = \mu(X_k^{q_k})$. Критерий определяется как средняя ошибка на контрольной подвыборке:

$$CV(\mu, X^L) = \frac{1}{k} \sum_{i=1}^{k} Q(\mu(X^L \setminus X_i^{q_i}), X_i^{q_i}). \qquad (17)$$

Этот способ кросс-валидации – компромисс между leave-one-out, hold-out и случайными разбиениями. С одной стороны, обучение производится только k раз вместо L.

2. ОПИСАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В качестве средства реализации был выбран интерпретируемый язык программирования Python. Этот язык широко используется специалистами машинного обучения для проведения исследований и решения прикладных задач. На данный момент реализовано множество библиотек для математических вычислений, позволяющих упростить процесс разработки и сделать его быстрее. Лицензия Python является открытой и позволяет использовать его для создания коммерческих продуктов. Python работает на множестве операционных систем семейств Linux, Unix, OS X, Windows и др.

2.1. Структура программы

Разработанный программный модуль состоит из трех логически разделенных файлов:

1) файл **mk_ls_svm** содержит реализацию мультиядерного LS SVM;
2) файл **kernel** содержит реализацию ядерных функций;
3) файл **crossvalidation** содержит реализацию процедуры скользящего контроля.

Рассмотрим каждый файл более подробно.

Файл **mk_ls_svm**.

class MKLSSVM – класс, реализующий мультиядерный метод опорных векторов. Отвечает за обучение алгоритма и классификацию наблюдений.

Класс содержит следующие переменные:

- *C* – параметр регуляризации;
- *tol* – предельная точность;
- *max_iter* – предельное количество итераций;
- kernel_set – набор ядер;
- *beta* – коэффициенты ядер;
- *b* – коэффицент смещения;
- *alpha* – коэффициенты Лагранжа;
- *Xfit* – набор наблюдений обучающей выборки;

- *Yfit* – набор значения классов обучающей выборки;
- *Hvec* – список ядерных матриц по обучающей выборке;
- *Kyvec* – список ядерных матриц по обучающей выборке с учетом значения класса.

Входными параметрами являются список ядер, реализованных в файле kernel, параметр регуляризации (по умолчанию – 1.0), предельная точность (по умолчанию – 10-4), предельное количество итераций (по умолчанию – 50).

Методы класса:

fit(data, target) – метод выполняющий обучение алгоритма, на основе подаваемых данных и заданных при инициализации класса входных параметрах. *data* – набор наблюдений. *target* – соответствующие наблюдениям значения класса.

Метод *fit* содержит три вложенных метода:

- *kernel_matrix()* – метод, вычисляющий значения переменных *Hvec* и *Kyvec*. Эти переменные являются постоянными в рамках одной обучающей выборки и имеет смысл вычислить их перед началом итеративного процесса обучения алгоритма;
- *lagrange_coefficient_estimation()* – метод, оценивающий значения переменных *b* и *alpha;*
- *kernel_coefficient_estimation()* – метод, оценивающий значения переменной *beta.*

predict(data) – метод, выполняющий классификацию объектов data на основе обученного алгоритма. Этот метод не может быть вызван раньше, чем метод *fit.* Метод *predict* содержит метод *y_prediction(z)*, выполняющий классификацию объектов из *data* последовательно.

to_pkl(filename) – метод выполняющий сохранение обученного классификатора в файл с расширением *.pkl.

Файл mk_ls_svm так же содержит функцию *load_clf_from_pkl(filename),* которая выполняет загрузку ранее сохраненного классификатора из файла с расширением *.pkl. Возвращаемое значение – класс MKLSSVM.

Файл **kernel**.

RBF – класс радиально базисной ядерной функции. Содержит входную переменную *sigma* и метод *compute(x, y)* для вычисления значения ядра.

Poly – класс полиномиальной ядерной функции. Содержит две входные переменные *c* и *d* и метод *compute(x, y)* для вычисления значения ядра.

Файл **crossvalidation**.

В файле содержится функция, реализующая k-fold кросс-валидацию *cross_val_score(clf, X, y, k=10)*.

Входными параметрами функции являются:

- *clf* – экземпляр класса классификатора;
- *X* – обучающая выборка;
- *Y* – значения классов обучающей выборки;
- *k* – количество разбиений обучающей выборки.

Результатом выполнения функции является средняя точность классификации по k разбиениям на тестовую и обучающие выборки.

2.2. Описание работы с программой

Результатом проделанной работы является программный модуль mk_ls_svm_lib, позволяющий использовать мультиядерный алгоритм метода опорных векторов наименьших квадратов для классификации данных. Для того чтобы использовать mk_ls_svm_lib пользователю необходимо добавить в свой проект папку с модулем.

Подключение модуля:

```
from mk_ls_svm_lib as mk
```

Создание экземпляра классификатора:

```
>>> kernel_set = [mk.kernel.RBF(10), mk.kernel.Poly(1,2)]
>>> clf = mk.mk_ls_svm.MKLSSVM(kernel_set)
```

Обучение классификатора:

```
import numpy as np
>>> X = np.array([[-1, -1], [-2, -1], [1, 1], [2, 1]])
>>> y = np.array([1, 1, 2, 2])
>>> clf = clf.fit(X,y)
```

Предсказание:

```
>>> predicted_y = clf.predict(X)
```

Сохранение обученного классификатора в файл:

```
>>> clf.to_pkl('my_clf.pkl')
```

Загрузка классификатора:

```
>>> clf = mk.mk_ls_svm.load_clf_from_pkl('my_clf.pkl')
```

Кросс-валидация:

```
>>> score = mk.crossvalidation.cross_val_score(clf, X, y)
```

3. ИССЛЕДОВАНИЯ

Целью работы является сравнение моделей классификатора, полученных с помощью одноядерного и многоядерного алгоритма LS SVM, выявление зависимостей точности классификации от параметра регуляризации, ядерных функций и различных данных с использованием процедуры скользящего контроля. Для искусственно сгенерированных данных были проведены следующие исследования влияния на качество модели:

1) влияние параметра ядерной функции;
2) влияние параметров регуляризации;
3) влияние мультиядерного подхода.

Так же сравнение мультиядерного и одноядерного подхода проводилось на данных реальных прикладных задач, взятых из репозитория UCI. Реальные данные могут иметь сложную структуру и в следствии чего требовать более сложную модель классификатора. В ходе исследования были выполнены подборы лучшей модели для всех наборах с использованием следующих типов моделей:

1) одно полиномиальное ядро;
2) одно RBF-ядро;
3) набор полиномиальных ядер;
4) набор RBF-ядер;
5) набор из полиномиальных и RBF-ядер.

Для оценки качества модели классификатора использовалась 10-fold кросс-валидация.

3.1. Исследование влияния параметров алгоритма

3.1.1. Описание данных

Для исследований используются искусственно сгенерированные данные с помощью функции sklearn.datasets.make_classification [19] из библиотеки scikit-learn.

Задается общее количество факторов. Они содержат информативные, избыточные, дублированные факторы. Задается количество факторов каждого типа. Оставшиеся факторы создаются с помощью нормального распределения N(0,1).

Каждый класс состоит из множества кластеров, каждый из которых расположен вокруг вершин гиперкуба в подпространстве заданной размерности количества информативных факторов. Для каждого кластера значения информативных факторов генерируются независимо с помощью нормального распределения N(0,1), но приобретают ковариацию через скалярное умножение на матрицу, сгенерированную с помощью равномерного распределения. Затем кластеры помещаются на вершины гиперкуба. Количество кластеров задается как параметр функции.

Избыточные факторы генерируются как скалярное произведение информативных факторов на матрицу, сгенерированную с помощью равномерного распределения.

Дублированные факторы являются копиями случайным образом выбранных информативных или избыточных факторов.

В таблице 3.1 представлена информация о параметрах, заданных при генерации выборок.

Таблица 3.1 – Параметры генерации данных.

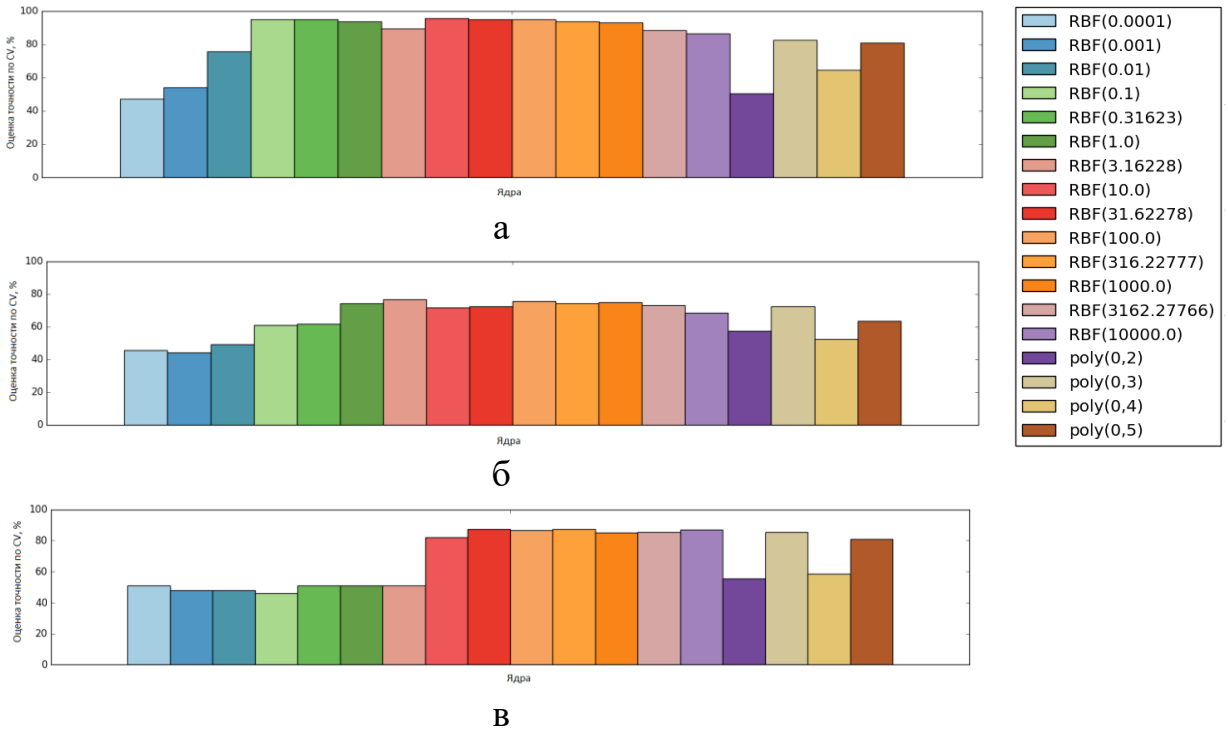
Номер выборки	1	2	3
Размер выборки	200	200	200
Количество факторов	2	10	20
Информативных факторов	2	8	2
Излишних факторов	0	0	2
Дублированных факторов	0	0	0
Кол-во кластеров каждого класса	2	3	2

3.1.2. Влияние параметра ядерной функции

В данном эксперименте изучалось качество настройки алгоритма LS SVM с одним ядром в зависимости от значения параметров ядерных функций.

Исследование проводилось на описанных выше данных с одновременной настройкой параметра регуляризации. Выбор лучшей модели осуществлялся через перебор множества значений параметров RBF-ядра: $\sigma = \left\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{-0.5}, 10^{0}, 10^{0.5}, 10^{1}, 10^{1.5}, 10^{2}, 10^{2.5}, 10^{3}, 10^{3.5}, 10^{4}\right\}$ и полиномиального ядра: $\{(c=0, d=2), (c=0, d=3), (c=0, d=4), (c=0, d=5)\}$.

Тестирование было проведено на всех сгенерированных выборках. На рисунке 3.1 можно увидеть зависимость точности классификации от выбора ядерной функции при оптимальном параметре регуляризации для каждого ядра.



а

б

в

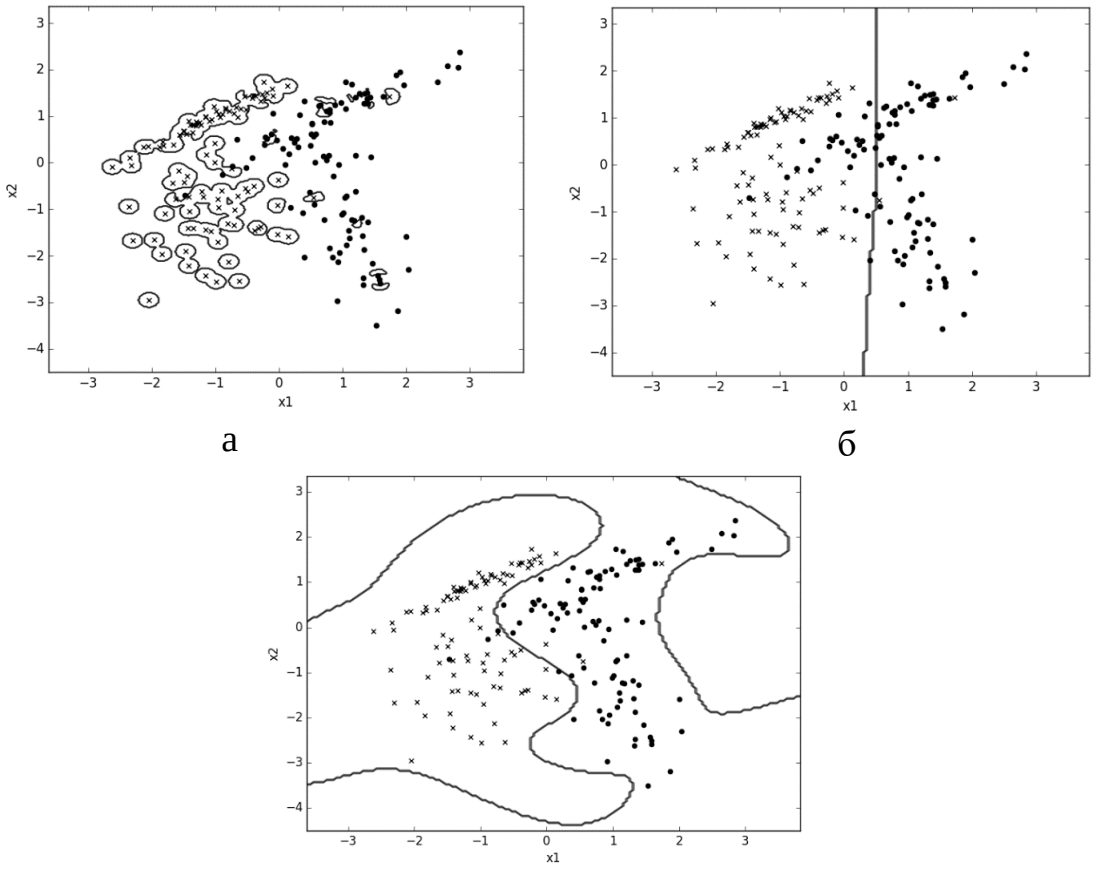
Рисунок 3.1 – Зависимость точности классификации от параметра ядра:
а – выборка 1, б – выборка 2, в – выборка 3

Максимальное значение точности и параметры лучшей модели для тестируемых выборок приведены в таблице 3.2.

Таблица 3.2 – Параметры моделей классификации.

Номер выборки	Ядро	Параметр регуляризации	Значение точности, %
1	RBF, $\sigma = 10$	10^{2}	95.5
2	RBF, $\sigma = 10^{0.5}$	10^{-1}	76.5
3	RBF, $\sigma = 10^{1.5}$	10^{-1}	87.5

Далее приведем визуальную оценку получаемых решений для выборки 1. На рисунке 3.2 представлена обучающая выборка и гиперплоскость, построенная с помощью модели с использованием RBF-ядра при $\sigma = 10$, $\sigma = 10^{-4}$ и $\sigma = 10^{4}$.



а б

в

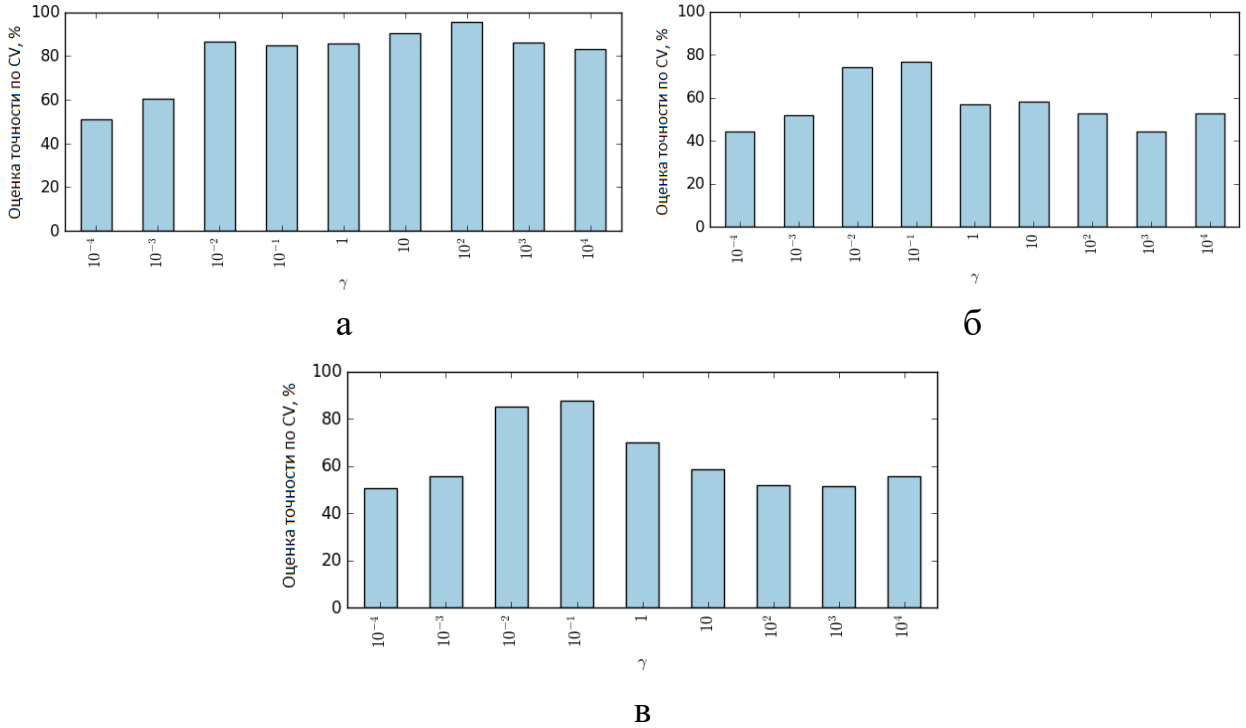
Рисунок 3.2 – RBF-ядро: а – $\sigma = 10^{-4}$, б – $\sigma = 10^{4}$, в – $\sigma = 10$

Из приведенных изображений видно, что при параметре $\sigma = 10$ (Рисунок 3.2 в) строится оптимальная модель классификатора. Так же стоит отметить, что при значениях параметра ядра, меньших оптимального (Рисунок 3.2 а) имеется тенденция к переобучению и при больших (Рисунок 3.2 б) – к недообучению.

3.1.3. Влияние параметра регуляризации

В данном эксперименте изучалось качество настройки алгоритма LS SVM с одним ядром в зависимости от параметра регуляризации γ. Исследование проводилось на описанных выше данных с одновременной настройкой оптимальной ядерной функции. Набор значений параметра $\gamma = \left\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}, 10^{4}\right\}$.

На рисунке 3.3 приведено влияния параметра регуляризации на точность классификации для каждой из тестируемых выборок при оптимальном ядре, выявленном в предыдущем подразделе.



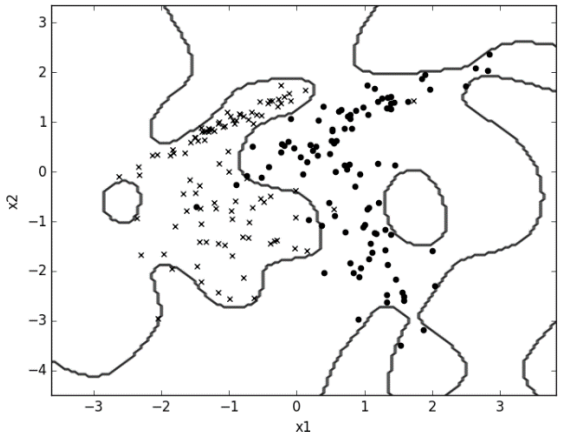
а б

в

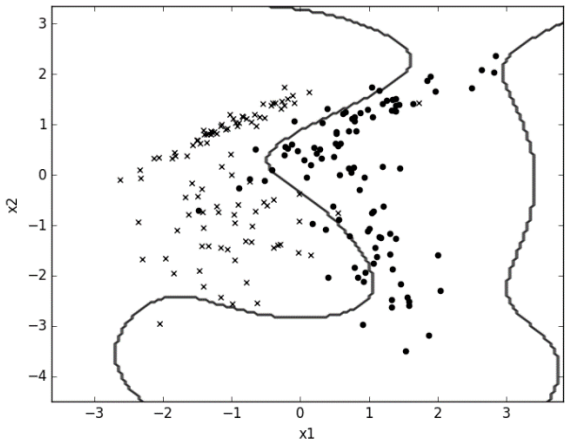
Рисунок 3.3 – Зависимости точности классификации от параметра регуляризации: а – выборка 1 (RBF-ядро при $\sigma = 10$), б – выборка 2 (RBF-ядро при $\sigma = 10^{0.5}$), в – выборка 3 (RBF-ядро при $\sigma = 10^{1.5}$)

Далее приведем визуальную оценку получаемых решений для выборки с двумя факторами при разных ядрах. На рисунке 3.4 представлена обучающая выборка с двумя факторами и гиперплоскость построенная с помощью модели с использованием RBF-ядра при $\gamma = 1$, $\gamma = 10^2$ и $\gamma = 10^4$.

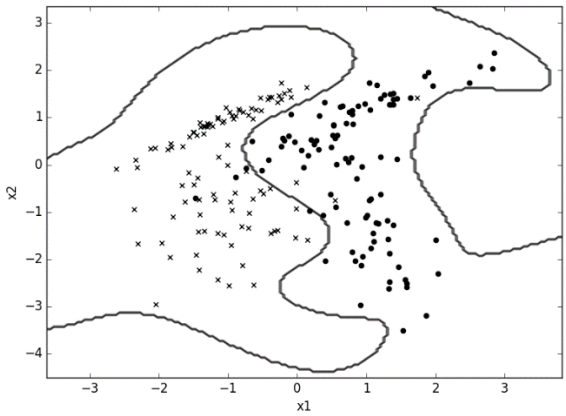
На первом изображении (Рисунок 3.4 а) прослеживается тенденция к переобучению при слишком большом значении параметра регуляризации $\gamma = 10^4$, на втором (Рисунок 3.4 б) $\gamma = 1$ – модель не переобучена, но разделяющая плоскость все еще не является оптимальной. Лучшая оценка классификации достигается при (Рисунок 3.4 в).



а



б



в

Рисунок 3.4 – RBF-ядро $\sigma = 10$: а – $\gamma = 10^4$, б – $\gamma = 1$, в – $\gamma = 10^2$

Из приведенных исследований можно утверждать, что зависимость между параметрами алгоритма присутствует и сильно влияет на обучаемую модель.

3.1.4. Влияние мультиядерного подхода

В предыдущих двух пунктах было показано, что при одновременном подборе ядра и параметра регуляризации может быть получена эффективная модель классификации. В данном эксперименте изучалось влияние использования мультиядерного подхода на качество обучаемой модели. Исследование проводилось на описанных выше данных с одновременной настройкой набора ядер и параметра регуляризации.

Для каждой тестовой выборки для исследований были выбраны несколько различных комбинаций ядер исходя из результатов, полученных выше. В таблицах 3.3, 3.4, 3.5 представлены полученные результаты. Наилучший результат классификации выделен подчеркнутым полужирным шрифтом.

Таблица 3.3 – Результаты мультиядерного обучения для выборки 1.

Тип ядра	Параметры ядер	Параметр регуляризации	Значение точности, %
RBF	$\sigma = 10$	10^2	95.5
RBF	$\sigma = \begin{Bmatrix} 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{-0.5}, 10^{0}, 10^{0.5} \\ 10^{1}, 10^{1.5}, 10^{2}, 10^{2.5}, 10^{3}, 10^{3.5}, 10^{4} \end{Bmatrix}$	150	95.5
RBF	$\sigma = \begin{Bmatrix} 10^{-1}, 10^{-0.5}, 10^{0}, 10^{0.5}, 10^{1}, 10^{1.5} \\ 10^{2}, 10^{2.5}, 10^{3}, 10^{3.5}, 10^{4} \end{Bmatrix}$	1.5	95.5
RBF	$\sigma = \begin{Bmatrix} 10^{0.5}, 10^{1}, 10^{1.5}, 10^{2} \\ 10^{2.5}, 10^{3}, 10^{3.5}, 10^{4} \end{Bmatrix}$	10^{-2}	88.0
RBF	$\sigma = \{10^{-3}, 10^{-1}, 10^{0}, 10^{0.5}, 10^{1}, 10^{1.5}, 10^{2}\}$	150	**96.5**
RBF	$\sigma = \{10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}, 10^{4}\}$	0,1	96.0
RBF	$\sigma = \{10^{1}, 10^{1.5}, 10^{3.5}\}$	100	94.5
RBF	$\sigma = \{10^{-0.5}, 10^{0}, 10^{1}, 10^{2}\}$	0.1	95.5
RBF	$\sigma = \{10^{1}, 10^{2}, 10^{3}, 10^{4}\}$	250	91.0
Poly	$\begin{Bmatrix} (c=0, d=2), (c=0, d=3) \\ (c=0, d=4), (c=0, d=5) \end{Bmatrix}$	10^{-2}	92.0
Poly	$\begin{Bmatrix} (c=0, d=2), (c=0, d=3) \\ (c=0, d=5) \end{Bmatrix}$	10^{-2}	92.5
Poly	$\begin{Bmatrix} (c=0, d=3), (c=0, d=4) \\ (c=0, d=5) \end{Bmatrix}$	0.5	92.0
Poly	$\{(c=0, d=3), (c=0, d=5)\}$	1.5	90.0

Таблица 3.4 – Результаты мультиядерного обучения для выборки 2.

Тип ядра	Параметры ядер	Параметр регуляризации	Значение точности, %
RBF	$\sigma = 10^{0.5}$	0.1	**<u>76.5</u>**
RBF	$\sigma = \begin{Bmatrix} 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{-0.5}, 10^{0}, 10^{0.5} \\ 10^{1}, 10^{1.5}, 10^{2}, 10^{2.5}, 10^{3}, 10^{3.5}, 10^{4} \end{Bmatrix}$	50	75.0
RBF	$\sigma = \begin{Bmatrix} 10^{-1}, 10^{-0.5}, 10^{0}, 10^{0.5}, 10^{1}, 10^{1.5} \\ 10^{2}, 10^{2.5}, 10^{3}, 10^{3.5}, 10^{4} \end{Bmatrix}$	0.1	75.5
RBF	$\sigma = \begin{Bmatrix} 10^{0.5}, 10^{1}, 10^{1.5}, 10^{2} \\ 10^{2.5}, 10^{3}, 10^{3.5}, 10^{4} \end{Bmatrix}$	3	**<u>76.5</u>**
RBF	$\sigma = \{10^{-3}, 10^{-1}, 10^{0}, 10^{0.5}, 10^{1}, 10^{1.5}, 10^{2}\}$	100	75.0
RBF	$\sigma = \{10^{1}, 10^{2}, 10^{3}, 10^{4}\}$	5	72.5
RBF	$\sigma = \{10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}, 10^{4}\}$	0.1	74.5
RBF	$\sigma = \{10^{1}, 10^{1.5}, 10^{3.5}\}$	0.5	72.5
RBF	$\sigma = \{10^{-0.5}, 10^{0}, 10^{1}, 10^{2}\}$	0.1	74.0
Poly	$\begin{Bmatrix} (c=0, d=2), (c=0, d=3) \\ (c=0, d=4), (c=0, d=5) \end{Bmatrix}$	10	71.0
Poly	$\begin{Bmatrix} (c=0, d=2), (c=0, d=3) \\ (c=0, d=5) \end{Bmatrix}$	0.5	72.5
Poly	$\begin{Bmatrix} (c=0, d=3), (c=0, d=4) \\ (c=0, d=5) \end{Bmatrix}$	700	71.0
Poly	$\{(c=0, d=3), (c=0, d=5)\}$	10^{-4}	73.0

Таблица 3.5 – Результаты мультиядерного обучения для выборки 3.

Тип ядра	Параметры ядер	Параметр регуляризации	Значение точности, %
RBF	$\sigma = 10^{1.5}$	0.1	87.5
RBF	$\sigma = \begin{Bmatrix} 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{-0.5}, 10^{0}, 10^{0.5} \\ 10^{1}, 10^{1.5}, 10^{2}, 10^{2.5}, 10^{3}, 10^{3.5}, 10^{4} \end{Bmatrix}$	1.5	89.5
RBF	$\sigma = \begin{Bmatrix} 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1} \\ 10^{0}, 10^{1}, 10^{2}, 10^{3}, 10^{4} \end{Bmatrix}$	20	90.0
RBF	$\sigma = \{10^{-1}, 10^{-0.5}, 10^{0}, 10^{1.5}, 10^{2}, 10^{2.5}\}$	150	87.5
RBF	$\sigma = \{10^{-1}, 10^{0}, 10^{0.5}, 10^{1}, 10^{1.5}, 10^{2}\}$	0.5	90.0
RBF	$\sigma = \{10^{-1}, 10^{-0.5}, 10^{0}, 10^{0.5}, 10^{1}, 10^{1.5}\}$	10	89.0
RBF	$\sigma = \{10^{1}, 10^{1.5}, 10^{2}\}$	1	89.5
RBF	$\sigma = \{10^{1}, 10^{1.5}\}$	3	**91.0**
RBF	$\sigma = \{10^{1.5}, 10^{2}\}$	1	87.5
Poly	$\begin{Bmatrix} (c=0, d=2), (c=0, d=3) \\ (c=0, d=4), (c=0, d=5) \end{Bmatrix}$	50	82.0
Poly	$\begin{Bmatrix} (c=0, d=2), (c=0, d=3) \\ (c=0, d=5) \end{Bmatrix}$	10	82.0
Poly	$\begin{Bmatrix} (c=0, d=3), (c=0, d=4) \\ (c=0, d=5) \end{Bmatrix}$	10^{-2}	83.5
Poly	$\{(c=0, d=3), (c=0, d=5)\}$	10	81.5

В таблицах 3.6, 3.7, 3.8 представлены веса ядерных функций для лучшей модели. Веса получены после обучения на всей выборке данных.

Таблица 3.6 – Веса ядер для выборки 1 (RBF-ядра).

	$\sigma = 10^{-3}$	$\sigma = 10^{-1}$	$\sigma = 1$	$\sigma = 10^{0.5}$	$\sigma = 10$	$\sigma = 10^{1.5}$	$\sigma = 10^{2}$
Вес	0.169	0.146	0.141	0.143	0.156	0.130	0.115

Таблица 3.7 – Веса ядер для выборки 2 (RBF-ядра).

	$\sigma = 10^{0.5}$	$\sigma = 10$	$\sigma = 10^{1.5}$	$\sigma = 10^{2}$	$\sigma = 10^{2.5}$	$\sigma = 10^{3}$	$\sigma = 10^{3.5}$	$\sigma = 10^{4}$
Вес	0.038	0.006	0.078	0.522	0	0	0.329	0.027

Таблица 3.8 – Веса ядер для выборки 3 (RBF-ядра).

	$\sigma = 10$	$\sigma = 10^{1.5}$
Вес	0.817	0.183

Для выборки 1 мультиядерное обучение дало результат на 1% лучше, чем при одноядерном подходе. Прирост может быть несущественным учитывая размер выборки, однако даже такой результат может быть важен на прикладных задачах.

Прирост точности классификации для выборки 2 не наблюдается. Точность классификации осталась на значении что и с одним ядром. Веса ядер распределились уже не равномерно. Наибольший вес имеет RBF-ядро при $\sigma = 10^{2.5}$, в то время как при одноядерном обучении оптимальным параметром был выбрана $\sigma = 10^{0.5}$.

Для выборки 3 присутствует существенный прирост точности классификации в 3.5%. Лучшей моделью классификации стала более простая модель, по сравнению с моделями для двух предыдущих выборок. В модели участвуют два RBF-ядра при $\sigma = \left\{10^{1.5}, 10^{2}\right\}$. Основной вес приходится на то же ядро, что и было получено при одноядерном подходе.

3.2. Применение алгоритма на прикладных задачах

Ранее было показано, что использование мультиядерного подхода для метода опорных векторов наименьших квадратов может привести к повышению точность классификации данных. Однако результаты были получены на искусственно сгенерированных данных, и оценка точности для них может не вполне отображать преимущества использования мультиядерного

обучения, поскольку реальные данные могут оказать более сложной формы. Были проведены исследования на реальных прикладных задачах из разных сфер деятельности человека и сравнены результаты одноядерного и мультиядерного подходов.

3.2.1. Описание данных

Данные представляют собой сбор показателей различных наблюдений из различных сфер деятельности. Sonar – поиск горных месторождений, Ionosphere – исследование атмосферы, Banknote – распознавание подлинности банкнот, Cancer и Diabet – медицинская диагностика, Credit – определение кредитоспособности заемщиков, Biodegradation – изучение разложения молекул в целях защиты окружающей среды.

В таблице 3.9 представлено описание наборов данных прикладных задач с двумя классами, взятые из репозитория UCI [16-18, 20-23], которые использовались в сравнительном анализе.

Таблица 3.9 – Описание данных прикладных задач.

Данные	Размер выборки	Количество факторов	Тип шкалы измерения
Sonar	208	60	Отношений
Ionosphere	351	34	Отношений
Banknote	1372	5	Отношений
Cancer	569	32	Отношений
Diabet	768	8	Отношений
Credit	1000	20	Отношений, номинальная, порядковая
Biodegradation	1055	41	Отношений

3.2.2. Результаты

Для исследований использовалось разбиение выборок данных с помощью 10-fold кросс-валидации. При одноядерном подходе использовались полиномиальное и RBF-ядро. При многоядерном было рассмотрены наборы из комбинации RBF-ядер, полиномиальных ядер и одновременное использования ядер обоих типов в одном наборе. Выбор ядер для MKL выполнялся на основе

результатов одноядерного обучения. За основные ядра в наборе выбирались те, при которых точность классификации при одноядерном подходе больше всего. Так же рассматривалось добавление ядер с результатами хуже. Одновременно выполнялась настройка параметра регуляризации.

В таблице 3.10 представлены результаты точности классификации рассматриваемых наборов данных. Лучший результат для каждой выборки выделен полужирным шрифтом с подчеркиванием.

Таблица 3.10 – Сравнительный результат различных подходов обучения модели.

Данные	Poly LS SVM	RBF LS SVM	Poly MK LS SVM	RBF MK LS SVM	RBF-Poly MK LS SVM
Sonar	84.39	83.41	86.34	**88.29**	86.82
Ionosphere	92.85	93.28	93.71	**95.43**	93.71
Banknote	96.42	**99.85**	98.58	99.61	98.64
Cancer	94.88	94.16	**95.75**	95.04	95.22
Diabet	77.78	73.86	**78.17**	70.53	76.73
Credit	77.60	70.10	**77.80**	70.10	75.90
Biodegradation	**86.63**	83.56	86.06	85.69	85.83

Как видно из результатов мультиядерный подход дает прирост в точности классификации. Однако в некоторых случаях одноядерный показал себя лучше. Это может быть связано с тем что большое количество ядер в модели ведет к переобучению или недообучению. В таблице 3.11 представлены параметры лучших моделей.

Таблица 3.11 – Параметры лучших моделей классификации.

Данные	Тип ядра	Параметры ядер	Параметр регуляризации
1	2	3	4
Sonar	RBF	$\sigma = \{10^{-1}, 10^{-0.5}, 10^{0}\}$	1.5
Ionosphere	RBF	$\sigma = \{10^{-1}, 10^{0.5}, 10^{1}\}$	0.75

Продолжение таблицы 3.11

1	2	3	4
Banknote	RBF	$\sigma = 10^3$	10^4
Cancer	Poly	$\{(c=4,d=2),(c=0,d=3)\}$	10^{-5}
Diabet	Poly	$\{(c=0,d=2),(c=2,d=2)\}$	10^{-3}
Credit	Poly	$\{(c=0,d=2),(c=2,d=2)\}$	10^{-4}
Biodegradation	Poly	$c=3,d=2$	10^{-3}

В таблицах 3.12-3.16 представлены веса ядерных функций для лучшей модели. Веса получены после обучения на всей выборке данных.

Таблица 3.12 – Веса ядер для выборки Sonar.

	RBF, $\sigma = 10^{-1}$	RBF, $\sigma = 10^{-0.5}$	RBF, $\sigma = 10^{0}$
Вес	0.027	0.667	0.306

Таблица 3.13 – Веса ядер для выборки Ionosphere.

	RBF, $\sigma = 10^{-1}$	RBF, $\sigma = 10^{0.5}$	RBF, $\sigma = 10^{1}$
Вес	0.288	0.331	0.381

Таблица 3.14 – Веса ядер для выборки Cancer.

	Poly, $c=4,d=2$	Poly, $c=0,d=3$
Вес	0.61	0.39

Таблица 3.15 – Веса ядер для выборки Diabet.

	Poly, $c=0,d=2$	Poly, $c=2,d=2$
Вес	0.52	0.48

Таблица 3.16 – Веса ядер для выборки Credit.

	Poly, $c=0,d=2$	Poly, $c=2,d=2$
Вес	0.459	0.541

3.3. Выводы

В ходе проведенных исследований изучалась степень влияния параметра регуляризации, типов и параметров ядер на результат классификации, а также возможность получения более точных моделей при использовании алгоритма мультиядерного обучения.

Для изучения влияния параметров ядерных функций было проведено отдельное исследование. Было установлено, что использование различных параметров RBF-ядер сильно влияет на результат классификации. Для различных данных могут быть определены оптимальные ядра с различными параметрами. При этом использование значений параметра меньше чем оптимальное имеет тенденцию к переобучению, а больших – к недообучению.

Важным фактором при настройке модели так же является параметр регуляризации. При фиксированном ядре этот параметр производит аналогичный эффект, что и параметр ядерной функции. Использование значений параметра меньше чем оптимальное ведет к неодообучению, а больших – к переобучению.

При обучении алгоритма необходимо проводить одновременную настройку по каждому из параметров, иначе будет получена недостоверная модель. Переобучение возникает при использовании избыточно сложных моделей, а недообучение при недостаточно сложных.

В результате исследования мультиядерного обучения на искусственных данных было выявлено, что этот подход не всегда ведет к улучшению качества модели, однако на определённых данных присутствует значительный прирост в точности. Выбор ядерных функций в наборе напрямую влияет на результат. Поэтому выбор ядер в набор необходимо осуществлять исходя из поставленной задачи.

В ходе исследований также было продемонстрировано влияние мультиядерного подхода на точность классификации данных прикладных задач различных размерностей и размеров выборки. В пяти из семи случаях мультиядерный подход привел к повышению точности. Таким образом, было подтверждено предположение, что реальные данные имеют более сложную форму в пространстве факторов и порой требуют более сложную структуру модели классификатора для получения лучшего решения.

Таким образом можно сформулировать следующие преимущества и недостатки мультиядерного метода опорных векторов наименьших квадратов.

Преимущества:

1) настройка алгоритма с помощью параметра регуляризации позволяет избежать переобучения или недообучения;
2) решение задачи оценки коэффициентов Лагранжа через СЛАУ;
3) двухфазный подход позволяет отделить задачи оценки коэффициентов Лагранжа и коэффициентов ядер;
4) применение множества ядер в модели классификатора позволяет компенсировать недостаточную гибкость других ядер.

Недостатки:

1) точность классификации сильно зависит от выбранных параметров алгоритма;
2) ьинарная классификация;
3) высокая вычислительная сложность алгоритма;
4) большое время обучения алгоритма.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы бакалавра были выполнены ее задачи, а также достигнута цель.

На первом этапе была осуществлена постановка решаемой задачи, разобраны принципы работы метода опорных векторов наименьших квадратов для бинарной классификации, а также разработан алгоритм решения с использованием мультиядерного подхода.

На втором этапе была выполнена программная реализация разработанного алгоритма. Реализация представляет собой программный модуль для языка Python и имеет удобный интерфейс для работы, используемый во многих других библиотеках машинного обучения. Разработкой может воспользоваться для своих нужд любой желающий, поскольку исходный код находится в открытом доступе.

На третьем этапе проводилось исследование влияния параметров алгоритма на точность классификации и сравнение мультиядерного и одноядерного подхода. Было получено, что мультиядерное обучение во многих задачах ведет к повышению точности модели, что несомненно является большим плюсом. Сравнение было проведено и на данные реальных прикладных задач и так же показало улучшение результатов. Таким образом разработанная программа может быть использована в научных и коммерческих целях для решения актуальных проблем.

В дальнейшем возможно применение других подходов мультиядерного обучения для получения еще более точных результатов:

1) применение ядер не на все пространство факторов, а только на определенные, логически сгруппированные, разнородные данные. Так будет использоваться каждое ядро для определенного типа данных;
2) оценивание важности факторов позволяет определить те, которые больше всего влияют на точность классификации. Отбросив несущественные факторы, результат решения может улучшиться.

СПИСОК ЛИТЕРАТУРЫ

1. Вапник В. Н. Восстановление зависимостей по эмпирическим данным. / В. Н. Вапник, – М.: Наука, 1979. – 448 с.
2. Вьюгин В.В. Математические основы машинного обучения и прогнозирования / Вьюгин В.В. – М.: 2013. – 387 с
3. Прикладная статистика: классификация и снижение размерности: в 3 т. / сост.: Айвазян С. А., Бухштабер В. М., Енюков И. С., Мешалкин Л. Д. – М.: Финансы и статистика, 1989. – Т. 3. – 586 с.
4. Воронцов К. В. Комбинаторный подход к оценке качества обучаемых алгоритмов [Электронный ресурс] – Режим доступа: http://www.ccas.ru/frc/papers/voron04mpc.pdf (дата обращения: 01.05.2017).
5. Suykens A. K. Least Squares Support Vector Machines / J. A. K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, J. Vandewalle – Singapore: World Scientific Pub. Co., 2002 – 294 p.
6. Vapnik V. Statistical Learning Theory / V. Vapnik – NY: John Wiley, 1998. – 768 p.
7. Vapnik V. The Nature of Statistical Learning Theory / V. Vapnik – NY: Springer-Verlag, 1995. – 314 p.
8. Chapelle O. Choosing multiple parameters for support vector machines / O. Chapelle, V. Vapnik, O. Bousquet and S. Mukherfee // Machine Learning. – 1999. – vol. 46, – no. 1 – pp. 131–159.
9. Chen Z. A multiple kernel support vector machine scheme for feature selection and rule extraction from gene expression data of cancer / Z. Chen, J. Li, L. Wei // Artificial Intelligence in Medicine. – 2007. – vol. 41, Iss. 2. – pp. 161—175.
10. Jian L. Design of a multiple kernel learning algorithm for LS-SVM by convex programming / L. Jian, Z. Xia, X. Liang, C. Gao // Neural networks. – 2011. – vol. 24, – no. 5 – pp. 476 – 483.
11. Kohavi R. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection / Kohavi R. // 14th International Joint Conference on

Artificial Intelligence, Palais de Congres Montreal, Quebec, Canada. — 1995. — pp. 1137-1145.

12. Lanckriet G. R. G. Learning the kernel matrix with semidefinite programming / G. R. G. Lanckriet, N. Cristianini, P. Bartlett, L. El Ghaoui, M. Jordan // Journal of Machine Learning Research. – 2004. – vol. 32, – no. 5 – pp. 27 – 72.
13. Li J. Feature selection via Least Squares Support Feature Machine / J. Li, Z. Chen, L. Wei, W. Xu, G. Kou // International Journal of Information Technology and Decision Making. – 2007. – vol. 6, – no. 4 – pp. 671 – 686.
14. Sonnenburg S. Large scale multiple kernel learning / S. Sonnenburg, G. Ratsch, C. Schafer, B. Scholkopf // Journal of Machine Learning Research. – 2006. – vol. 7 – pp. 1531–1565
15. van Gestel T. Benchmarking Least Squares Support Vector Machine Classifiers / T. van Gestel, J.A. Suykens, B. Baesens // Machine Learning. – 2004. – vol. 54, – no. 1 – pp. 5 – 32.
16. Banknote authentication Data Set [Электронный ресурс] – Режим доступа: https://archive.ics.uci.edu/ml/datasets/banknote+authentication (дата обращения: 15.05.2017).
17. Breast Cancer Wisconsin (Diagnostic) Data Set [Электронный ресурс] – Режим доступа: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29 (дата обращения: 15.05.2017).
18. Connectionist Bench (Sonar, Mines vs. Rocks) Data Set [Электронный ресурс] – Режим доступа: https://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+%28Sonar%2C+Mines+vs.+Rocks%29 (дата обращения: 15.05.2017).
19. Documentation for scikit-learn version 0.18.1 [Электронный ресурс] – Режим доступа: http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html (дата обращения: 15.05.2017).
20. Ionosphere Data Set [Электронный ресурс] – Режим доступа: https://archive.ics.uci.edu/ml/datasets/Ionosphere (дата обращения: 15.05.2017).

21. Pima Indians Diabetes Data Set [Электронный ресурс] – Режим доступа: https://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes (дата обращения: 15.05.2017).
22. QSAR biodegradation Data Set [Электронный ресурс] – Режим доступа: https://archive.ics.uci.edu/ml/datasets/QSAR+biodegradation (дата обращения: 15.05.2017).
23. Statlog (German Credit Data) Data Set [Электронный ресурс] – Режим доступа: https://archive.ics.uci.edu/ml/datasets/Statlog+%28German+Credit+Data%29 (дата обращения 15.05.2017).

ПРИЛОЖЕНИЕ. Текст программы

Файл kernel.py

```
import numpy as np


class RBF:
    def __init__(self, sigma):
        '''
        RBF-ядро
        :param sigma: float
            Параметр ядра.
        '''
        self._sigma = sigma

    def compute(self, x, y):
        '''Метод вычисляет значение ядра между двумя векторами

        :param x: array-like
            Первый вектор
        :param y: array-like
            Второй вектор
        :return: float
        '''
        sqr_dist = sum([(it1 - it2)**2 for it1, it2 in zip(x,y)])
        return np.e**(-1.0 * (sqr_dist ** 2) / (2 * (self._sigma ** 2)))

class Poly:
    def __init__(self, c, d):
        ''' Полиномиальное ядро

        :param c: float
            Параметр смещения
        :param d: float
            Параметр степени
        '''
        self.__c = c
        self.__d = d

    def compute(self, x, y):
        '''Метод вычисляет значение ядра между двумя векторами

        :param x: array-like
            Первый вектор
        :param y: array-like
            Второй вектор
        :return: float
        '''
        return (np.dot(x,y) + self.__c)**self.__d
```

Файл crossvalidation.py

```
import numpy as np
from sklearn.metrics import accuracy_score

def cross_val_score(clf, X, y, cv=10):
    '''K-fold Cross-Validation

    :param clf: object
        Модель классификатора
    :param X: array-like, shape = [n_samples, n_features]
        Обучающая выборка. Значение факторов наблюдений.
    :param y: array-like, shape = [n_samples]
        Значения классов обучающей выборки, соответсвующие X.
    :param cv: int, optional (default=10)
        Количество разбиений кросс-валидации
    :return: float
        Точность классификации по кросс-валидациив
```

```
    '''
    data = list(zip(X, y))
    np.random.shuffle(data)
    data = np.array(data)
    test_size = int(len(X) / cv)
    scores = []
    for slice in range(cv):
        test_start = slice*test_size
        test_data = data[test_start:test_start+test_size]

        a = data[:test_start]
        b = data[test_start+test_size:]
        train_data = np.concatenate((a, b), axis=0)

        test_X = test_data[:, 0]
        test_y = test_data[:, 1]
        train_X = train_data[:, 0]
        train_y = train_data[:, 1]

        clf.fit(np.array(train_X),np.array(train_y))
        score = accuracy_score(list(test_y), list(clf.predict(test_X)))
        scores.append(score)
    return np.mean(scores), np.std(scores)
```

Файл mk_ls_svm.py

```
import pickle
import numpy
import scipy
from scipy.optimize import (minimize)
from functools import reduce

class MKLSSVM:
    def __init__(self, kernel_set, C=1.0, tol=1e-4, max_iter=50):
        '''Мультияерный метод опорных векторов наименьших квадратов.
        Автоматаческая настройка коэффициентов Лагранжа и коэффицентов ядерных функций
        для задач бинарной классификации.

        :param kernel_set: list of instances from kernel
            Набор ядерных функций
        :param C: float, optional (default=1.0)
            Параметр ргуляризации
        :param tol: float, optional (default=1e-4)
            Достигаемая точность критерия останова
        :param max_iter: int, optional (default=50)
            Максимальное количество итераций
        '''
        self.C = C
        self.tol = tol
        self.max_iter = max_iter
        self.kernel_set = kernel_set
        self.beta = numpy.array([1.0 / len(kernel_set) for _ in kernel_set])

    def fit(self, data, target):
        '''Обучение модели на основе выборки

        :param data: array-like, shape = [n_samples, n_features]
            Обучающая выборка. Значение факторов наблюдений.
        :param target: array-like, shape = [n_samples]
            Значения классов обучающей выборки, соответсвующие data.
        :return self: object
            Returns self.
        '''
        def kernel_matrix():
            trainSeqLen = len(target)
            H_vec = []
            for K in self.kernel_set:
                H = numpy.matrix(numpy.zeros(shape=(trainSeqLen, trainSeqLen)))
                for i in range(trainSeqLen):
                    for j in range(i, trainSeqLen):
```

```
                        val = K.compute(data[i], data[j])
                        H[i, j] = val
                        H[j, i] = val
                H_vec.append(H)
            return H_vec

        def kernel_matrix_y():
            Ky_vec = []
            for H in self.__Hvec:
                Ky = []
                for i, _ in enumerate(H):
                    Ky.append(numpy.asarray([y * H[i, j] for j, y in
enumerate(target)], dtype=float))
                Ky_vec.append(Ky)
            return Ky_vec

        def lagrange_coefficient_estimation():
            trainSeqLen = len(target)
            weighted_H = map(lambda h, beta: h * beta, self.__Hvec, self.beta)
            H = reduce(lambda p_h, h: p_h + h, weighted_H)
            for i in range(trainSeqLen):
                for j in range(i, trainSeqLen):
                    H[i, j] *= target[i] * target[j]
                    H[j, i] *= target[j] * target[i]
                    if i == j:
                        H[i, j] += 1.0 / self.C

            d = numpy.ones(trainSeqLen)
            eta = scipy.sparse.linalg.cg(H, target, maxiter=1000)[0]
            nu = scipy.sparse.linalg.cg(H, d, maxiter=1000)[0]
            s = numpy.dot(target.T, eta)
            b = numpy.dot(eta.T, d) / s
            alpha = nu - eta * b
            return b, alpha

        def kernel_coefficient_estimation():
            def score_func(beta_vec):
                def K_sum(i):
                    weighted_kernels = [b_c * K[i] for b_c, K in zip(beta_vec,
self.__Kyvec)]
                    return numpy.array(reduce(lambda l, m: l + m, weighted_kernels))

                loss_func_vec = []
                for i, y in enumerate(target):
                    weighted_kernels_sum = K_sum(i)
                    loss_func_vec.append(1.0 - y * self.b - y *
numpy.dot(weighted_kernels_sum, self.alpha))

                loss_func = reduce(lambda e1, e2: e1 + e2 ** 2, loss_func_vec)
                return loss_func

            cons = ({'type': 'eq', 'fun': lambda x: sum(x) - 1.0})
            bnds = [(0.0, 1.0) for _ in self.beta]
            betaopt = minimize(score_func, self.beta,
                               bounds=bnds, constraints=cons,
                               method='SLSQP',
                               options={'maxiter': 1000, 'disp': False})

            return betaopt.x, betaopt.fun

        classes = numpy.unique(target)

        if len(classes) == 1 or len(classes) != 2:
            raise Exception('Количество классов должно быть равно двум.')
        self.class_dict = {
            '1.0': classes[0],
            '-1.0': classes[1]}
        target = numpy.array(list(map(lambda y: 1.0 if y == classes[0] else -1.0,
target)))

        self.__Xfit = data
        self.__Yfit = target
```

```
        self.__Hvec = kernel_matrix()
        self.__Kyvec = kernel_matrix_y()

        prev_score_value = 0
        prev_beta_norm = numpy.linalg.norm(self.beta)
        cur_iter = 0
        while True:
            self.b, self.alpha = lagrange_coefficient_estimation()
            if len(self.kernel_set) == 1:
                break
            self.beta, score_value = kernel_coefficient_estimation()
            # выход по количеству итераций
            if cur_iter >= self.max_iter:
                break
            # выход по невязке функции
            if abs(prev_score_value - score_value) < self.tol:
                break
            # выход по невязке нормы коэфициентов
            beta_norm = numpy.linalg.norm(self.beta)
            if abs(prev_beta_norm - beta_norm) < self.tol:
                break
            prev_score_value = score_value
            prev_beta_norm = beta_norm
            cur_iter += 1

        return self

    def predict(self, data):
        '''Предсказание принадлежности классу на основе ранее обученной модели

        :param data: array-like, shape = [n_samples, n_features]
            Тестовая выборка.
        :return target: array-like, shape = [n_samples]
            Значения классов, соответсвующие data.
        '''
        def y_prediction(z):
            support_vectors_sum = sum([alpha * y *
                                       sum([beta * K.compute(z, x) for beta, K in
zip(self.beta, self.kernel_set)])
                                       for alpha, x, y in zip(self.alpha, self.__Xfit,
self.__Yfit)])

            p = support_vectors_sum + self.b
            if p == 0.0:
                p = 1.0;
            return self.class_dict[str(numpy.sign(p))]

        return [y_prediction(test_x) for test_x in data]

    def to_pkl(self, filename):
        '''Сохранение ранее обученной модели в файл *.pkl

        :param filename:
            Название файла с расширением *.pkl, куда будет сохранена модель
        :return:
        '''
        with open(filename, 'wb') as output:
            pickle.dump(self, output, pickle.HIGHEST_PROTOCOL)


def load_clf_from_pkl(filename):
    '''Загрузка модели из файла *.pkl

    :param filename:
        Название файла с расширением *.pkl, откуда будет загружена модель.
    :return: MKLSSVM
        Модель классификатора.
    '''
    with open(filename, 'rb') as input:
        return pickle.load(input)
```