Санкт-Петербургский государственный университет

Выпускная квалификационная работа

общей образовательной программы бакалавриата
«Прикладная математика и информатика»

студента 4 курса, группы 17.Б05-мм
Владислава Сергеевича Трошина

на тему

«Объединение различных методов классификации изображений для повышения качества распознавания»

Научный руководитель:
к.ф.-м.н., М. С. Ананьевский
доцент кафедры теоретической кибернетики

Рецензент:
д.т.н., Игорь Борисович Фуртат
в.н.с. лаб. УСС ИПМаш РАН

г. Санкт-Петербург
2021 год

Saint Petersburg State University

Graduation Project

Applied Mathematics and Computer Science

Control and Processing of Information in Cybernetical and Robotic Systems

Vladislav Troshin

Combining different methods of image classification to improve the quality of recognition

Scientific Supervisor:

Mikhail Ananyevskiy

Candidate of Physico-Mathematical Sciences

Reviewer:

Igor Furtat

Doctor of Engineering Sciences

Saint Petersburg

2021

Содержание

1 Введение

Классификация с учителем играет центральную роль в анализе данных дистанционного зондирования, например, для составления карт землепользования или земного покрова, инвентаризации лесов или мониторинга городских территорий [4]. Классификаторы, основанные на методе опорных векторов (SVM, Support Vector Machine), в последнее время привлекают значительное внимание как из-за их способности к обобщению даже с входными данными большой размерности, так и из-за их точных результатов во многих приложениях [5, 6]. Однако теория SVM была разработана с акцентом на независимые и одинаково распределенные выборки и метки классов. С точки зрения классификации изображений этот фокус приводит к изначально неконтекстному подходу, то есть каждый пиксель классифицируется как таковой, независимо от соседних пикселей [4]. Этот подход не учитывает информацию, связанную с корреляциями между отдельными пикселями изображения, и представляет собой сильное ограничение на использование классификаторов на основе SVM для анализа изображений.

В этой работе я воспользовался интегрированной структурой, предложенной в статье "Combining Support Vector Machines and Markov Random Fields in an Integrated Framework for Contextual Image Classification" [1], которая объединяет подходы SVM и марковского случайного поля (MRF, Markov Random Field) [7] к классификации и, следовательно, позволяет ввести строгое контекстуальное обобщение SVM. MRF — это очень общее семейство вероятностных моделей, которые позволяют включать пространственную информацию в байесовские схемы анализа изображений с точки зрения минимизации подходящих "энергетических функций" [7, 8]. Соответственно, комбинирование байесовских схем обработки с моделями MRF довольно просто, но SVM —

это небайесовские методы классификации, что делает проблему интеграции подходов SVM и MRF нетривиальной.

Эта проблема может быть выражена в общих рамках обработки изображений и распознавания образов, но играет особенно важную роль в дистанционном зондировании, особенно в отношении проблемы размерности и специфических характеристик пространственного контекста в изображениях Земли. Классификация таких изображений часто требует работы с данными большой размерности. Типичными примерами являются задачи классификации, которые включают гиперспектральные изображения и/или извлечение нескольких дополнительных признаков (например, текстового описания). И, напротив, работа с многомерными данными обычно не требуется в других областях обработки изображений (например, биомедицинская визуализация или компьютерное зрение). Проблема размерности данных критически влияет на точность контролируемых классификаторов из-за "проклятия размерности". Это явление вызывает серьезную потерю точности, когда количество обучающих выборок недостаточно для вычисления надежных оценок параметров классификатора в многомерном пространстве признаков. Чтобы избежать таких проблем, при дистанционном зондировании часто применяется сокращение признаков перед классификацией. Однако этот предварительный шаг нетривиален и, возможно, требует много времени. Методы опорных векторов не требуют предварительного сокращения признаков, поскольку они доказали свою устойчивость к "проклятию размерности"(если размер обучающей выборки не очень мал). Поэтому в настоящее время они признаны мощными инструментами в решении проблем размерности и вызвали большой интерес к классификации изображений дистанционного зондирования.

Точно так же контекстную информацию на изображениях дистанционного зондирования трудно смоделировать с точки зрения формы объекта (в отличие от других приложений распознавания изображений) из-за наличия естественного покрова и большого разнообразия форм и размеров искусственного покрова на Земле [9]. Напротив, модели MRF, которые характеризуют локальную и глобальную пространственную статистику изображений на попиксельной основе, обычно достаточно гибки, чтобы захватывать контекстную информацию, связанную с изображениями дистанционного зондирования. Эти свойства объясняют успех моделей MRF во многих задачах классификации, в том числе связанных с мультисенсорными [10], мультитемпоральными [11, 12, 13] или крупномастштабными спутниковыми изображениями [14].

Вышеупомянутые соображения мотивируют полезность интеграции SVM и MRF в уникальную структуру для совместного использования как спектральной, так и пространственной информации, связанной с подлежащим классификации изображением дистанционного зондирования. Предложенная структура [1] разработана путем определения аналитической взаимосвязи между марковским правилом принятия решений с минимальной энергией и применением SVM в соответствующем преобразованном пространстве, индуцированном определенным ядром (которое будет называться "марковским ядром"). Предыдущие методы были разработаны для включения пространственной информации в функции ядра для классификации изображений дистанционного зондирования (см., например, [16] или [17]). Однако, в отличие от этих методов, предлагаемая стратегия направлена на полную интеграцию подходов SVM и MRF в уникальную структуру. Эта формулировка представляет собой строгий и гибкий инструмент для включения контекстной информации, моделируемой MRF, в классификацию опорных векторов и охватывает как ядра, индуцирующие конечномерные собственные пространства (например, линейные или полиномиальные ядра), так и ядра, индуцирующие бесконечномерные собственные пространства (например, гауссово ядро). В последнем случае интеграция SVM-MRF достигается путем формализации задачи классификации в терминах подходящих случайных процессов с непрерывным и дискретным временем и путем расширения методологических аргументов, которые изначально были введены в цифровых коммуникациях для разработки оптимальных приемников для связи по каналам с аддитивным белым гауссовским шумом.

В этой работе будут рассмотрены разные модификации марковского ядра. Будут представлены экспериментальные результаты работы новых моделей и сравнение с классическими.

2 Постановка задачи

В этой работе будут описаны такие методы классификации, как метод опорных векторов (SVM) и классификатор на основе марковского случайного поля (MRF). Затем будет предложен способ объединения SVM и MRF, после чего полученный метод с разными схемами соседей будет протестирован на наборе данных. Результаты тестирования будут сравнены с работой классического SVM. В качестве данных взяты изображения поверхности Луны [2].

3 Предложенная SVM-MRF модель

3.1 Введённые обозначения

Пусть изображение состоит из d каналов, и пусть $\mathcal{I}$ будет соответствующей решеткой пикселей, $x_i \, (x_i \in \mathbb{R}^d)$ будет вектором признаков i-го пикселя, а $\mathcal{X}$ будет компактным подмножеством $\mathbb{R}^d$, содержащим все образцы изображений, состоящих из x_i ($i \in \mathcal{I}$). Будем проводить бинарную классификацию, т. е. предполагаем, что в изображенной сцене присутствуют два тематических класса, и обозначим через $\mathcal{L} \subset \mathcal{I}$ набор обучающих пикселей для таких классов. Двоичная метка y_i прикрепляется к каждому i-му пикселю, так что $y_i = 1$ или $y_i = -1$, если пиксель связан с некоторым классом ($i \in \mathcal{I}$). Истинная метка класса известна для каждой обучающей выборки (т. е. $i \in \mathcal{L}$). Следовательно, на решетке $\mathcal{I}$ определено непрерывнозначное случайное поле $\{x_i\}_{i \in \mathcal{I}}$ векторов признаков и дискретнозначное случайное поле $\{y_i\}_{i \in \mathcal{I}}$ меток классов.

Если $\{f_i\}_{i \in \mathcal{I}}$ - произвольное поле на $\mathcal{I}$, а $\mathcal{A} \subset \mathcal{I}$ - подмножество пикселей, мы обозначим через $\mathbf{f}_{\mathcal{A}}$ вектор, собирающий все выборки полей f_i, $i \in \mathcal{A}$. Обозначим через $P(\cdot)$ и $p(\cdot)$ функцию вероятности дискретных случайных величин и функцию плотности вероятности непрерывных случайных величин соответственно.

3.2 Модель SVM

Подход SVM заключается в вычислении линейной разделяющей функции f между двумя классами в нелинейно преобразованном пространстве, минимизируя верхнюю границу ошибки обобщенного классификатора и принимая формулировку на основе ядра. Пусть K — ядерная функция, т.е. пусть существуют вещественное векторное пространство $\mathcal{F}$, снабженное скалярным произведением $(\cdot,\cdot)_{\mathcal{F}}$, и отображение $\Phi : \mathcal{X} \to \mathcal{F}$ такие, что $K(x, x') = (\Phi(x), \Phi(x'))_{\mathcal{F}}$ для всех x, $x' \in \mathcal{X}$. Разделяющая функция на основе SVM может быть эквивалентно выражена как линейная функция от $\mathcal{F}$, т.е. ($x \in \mathbb{R}^d$)

$$f(x) = (w, \Phi(x))_{\mathcal{F}} + b$$

или как взвешенная сумма нелинейных ядер с центром на подмножестве обучающих признаков, т. е.

$$f(x) = \sum_{j \in S} \alpha_j y_j K(x_i, x_j) + b,$$

где w и b ($w \in \mathcal{F}, b \in \mathbb{R}$) — вес и смещение линейной разделяющей функции в $\mathcal{F}$. $S \subset \mathcal{L}$ — подмножество обучающих признаков, элементы которых называются опорными векторами. Неизвестному признаку x классификатор присваивает оценочную метку класса $\hat{y} = \operatorname{sgn} f(x)$. Как выбор признаков в S, так и вычисление набора коэффициентов $\{\alpha_j\}_{j \in S}$ и b получаются путем решения следующей задачи квадратичного программирования:

$$\begin{cases} \min\limits_{\alpha \in \mathbb{R}^n} \left(\frac{1}{2}\alpha^T Q \alpha - \mathbf{1}^T \alpha\right) \\ \alpha \in [0, C]^n, \quad y_{\mathcal{L}}^T \alpha = 0, \end{cases}$$

где l — количество обучающих признаков (т.е. мощность $\mathcal{L}$); $\mathbf{1}$ — l-мерный вектор с унитарными компонентами; Q — $l \times l$ симметричная матрица, (i, j)-ый элемент которой равен $Q_{ij} = y_i y_j K(x_i, x_j)\,(i, j \in \mathcal{L})$; C — параметр регуляризации метода. В частности, $S = \{i \in \mathcal{L} : \alpha_i > 0\}$. В ядро K могут быть включены дополнительные параметры (например, дисперсия гауссова ядра или порядок полиномиального ядра).

3.3 Контекстная классификация изображений на основе MRF

Марковское случайное поле (MRF) представляет собой широкое семейство стохастических моделей для контекстной информации, связанной с данными изображения. MRF предлагает мощный и эффективный в вычислительном отношении подход к формализации этой информации в терминах распределений априорной вероятности в байесовском анализе изображений. Пусть $\{\partial i\}_{i\in\mathcal{I}}$ — система окрестностей на $\mathcal{I}$, где $\partial i \subset \mathcal{I}$ — множество соседей каждого i-го пикселя. Например, окрестность первого и второго порядка i-го пикселя состоит из 4 и 8 окружающих пикселей соответственно. Поле меток $\{y_i\}_{i\in\mathcal{I}}$ является марковским случайным полем по отношению к этой системе окрестностей, если его совместная вероятность $P(y_{\mathcal{I}})$ строго положительна и выполняется следующее марковское свойство:

$$P(y_i|y_{\mathcal{I}-\{i\}}) = P(y_i|y_{\partial i}) \quad \forall i \in \mathcal{I}.$$

Предположим, что совместная плотность $x_{\mathcal{I}}$ с заданным $y_{\mathcal{I}}$ может быть факторизована как $p(x_{\mathcal{I}}|y_{\mathcal{I}}) = \Pi_{i\in\mathcal{I}}\, p(x_i|y_i)$, где $p(x_i|y_i)$ — плотность x_i при заданном y_i (условно — гипотеза независимости), и что $\{y_i\}_{i\in\mathcal{I}}$ является марковским случайным полем. Затем, с помощью теоремы Хаммерсли — Клиффорда, максимизация совместного апостериорного распределения $P(y_{\mathcal{I}}|x_{\mathcal{I}})$ всех меток с учетом всех векторов признаков (т. е. байесовское «максимальное апостериорное» правило принятия решения) эквивалентна минимизации глобальной энергетической функции, определенной согласно системе окрестностей. Более того, апостериорное распределение $P(y_{\mathcal{I}}|x_{\mathcal{I}})$ однозначно определяется совокупностью всех краевых апостериорных распределений $P(y_i|x_i, y_{\partial i})\,(i \in \mathcal{I})$ меток каждого пикселя, обусловленных его вектором признаков и соседними метками. Эти краевые распределения могут быть записаны (с точностью до мультипликативного множителя) как $\exp[-U_i(y_i|x_i, y_{\partial i})]$, где

$$U_i(y_i|x_i, y_{\partial i}) = -\ln p(x_i|y_i) + \beta\mathcal{E}_i(y_i|y_{\partial i})$$

— локальная апостериорная функция энергии, $\mathcal{E}_i$ — локальная априорная функция энергии, характеризующая модель MRF, выбранную для $\{y_i\}_{i\in\mathcal{I}}$, а β — положительный параметр сглаживания. С точки зрения объединения данных, локальная апостериорная функция энергии аддитивно объединяет два энергетических вклада, связанных с неконтекстной условной статистикой класса и контекстной информацией, соответственно. Параметр β регулирует обратные веса этих двух членов и, следовательно, влияет на свойства пространственного сглаживания MRF.

Многие широко используемые методы минимизации вышеупомянутой глобальной энергии могут быть выражены через следующую функцию разности энергий ($i \in \mathcal{I}$):

$$\Delta U_i(x_i, y_{\partial i}) = U_i(-1|x_i, y_{\partial i}) - U_i(1|x_i, y_{\partial i}).$$

3.4 MRF-ядро для модели SVM

В дальнейшим будем использовать в качестве ядра для классического SVM функцию Гаусса $K(x, x') = \exp(-\gamma||x-x'||^2)$. Это ядро порождает бесконечномерное собственное подпространство, поэтому потребуется утверждение теоремы 2 из [1]. Теперь, чтобы объединить MRF с SVM, введём следующие предположения.

Предположение 1. *$\{y_i\}_{i\in\mathcal{I}}$ — марковское случайное поле с системой соседей $\{\partial i\}_{i\in\mathcal{I}}$, $\mathcal{E}_i$ и β обозначают априорную локальную энергию и параметр сглаживания, соответственно. Также имеется условная независимоть плотности вероятности $\{x_i\}_{i\in\mathcal{I}}$ при $\{y_i\}_{i\in\mathcal{I}}$.*

Предположение 2. *Размерность преобразованного пространства $\mathcal{F}$, индуцированного ядром K, бесконечна.*

Предположение 3. *Для каждого i-го пикселя $\tilde{x}_i(t)$, обусловленный либо $y_i = 1$, либо $y_i = -1$, является гауссовским случайным процессом со средним значением $m^+(t) = E\{\tilde{x}_i(t)|y_i = 1\}$ или $m^-(t) = E\{\tilde{x}_i(t)|y_i = -1\}$ со-*

ответственно и одна и та же функция автоковариации для обоих классов. Кроме того, эта автоковариационная функция непрерывна и положительно определена.

Наконец, согласно теореме 2 из [1] под предположениями 1, 2 и 3 функция энергии разности может быть переписана в терминах ядра для метода опорных векторов:

$$\Delta U_i(x_i, y_{\partial i}) = \sum_{j \in S} \alpha_j y_j K_{MRF}(x_i, \varepsilon_i, x_j, \varepsilon_j) + b,$$

где $K_{MRF}(x_i, \varepsilon_i, x_j, \varepsilon_j) = K(x_i, x_j) + \beta \varepsilon_i \varepsilon_j$, $\varepsilon_i = \mathcal{E}_i(-1|y_{\partial i}) - \mathcal{E}_i(1|y_{\partial i})$, а $\alpha = (\alpha_1, \ldots, \alpha_n)$ можно найти из задачи квадратичного программирования.

4 Реализация предложенного классификатора

4.1 Поиск оптимального параметра β для MRF-ядра

Пусть $\{\hat{y}^0_{\partial i}\}_{i \in \mathcal{I}}$ — начальное множество предсказываемых меток для признаков $\{x_i\}_{i \in \mathcal{I}}$ (например, результат работы SVM на обучаемом множестве). Обучаемое множество корректно классифицировано тогда и только тогда, когда выполняется следующее неравенство:

$$y_i \Delta U^0_i(x_i, \hat{y}^0_{\partial i}) \geq 0$$

Это условие можно эквивалентно переформулировать в следующей форме:

$$E\zeta \geq 0$$

где E - матрица размера $l \times 2$, $\zeta = \begin{pmatrix} \zeta_1 & \zeta_2 \end{pmatrix}^T$ — такой вектор, что $\beta = \zeta_2/\zeta_1, \zeta_1 > 0$. E_{i1}, E_{i2} вычисляются следующим образом:

$$E_{i1} = \sum_{j \in S} \alpha_j y_i y_j K(x_i, x_j) + y_i b$$

$$E_{i2} = \sum_{j \in S} \alpha_j y_i y_j \varepsilon_i^0 \varepsilon_j^0$$

Согласно [4], оцениваем β как численное решение линейной системы неравенств, чтобы определить вектор ζ, который способствует правильной классификации обучающих выборок.

4.2 Схемы соседей

Рассмотрим следующие схемы соседей:

Тип 1. *Всего 4 соседа: левый, верхний, правый, нижний.*

Тип 2. *Все пиксели внутри квадрата с произвольной длиной стороны, кроме рассматриваемого.*

Тип 3. *Все пиксели внутри квадрата с произвольной длиной стороны, кроме рассматриваемого, и пиксели на произвольном удалении от квадрата.*

В качестве ядра для классического SVM будет использована функция Гаусса $K(x, x') = \exp(-\gamma ||x - x'||^2)$, а в качестве локальной априорной функции энергии $\mathcal{E}_i(y_i | y_{\partial i}) = -\sum_{j \in \partial i} \delta(y_i, y_j)$. Таким образом, SVM-MRF ядро можно переписать как:

$$K_{MRF}(x_i, y_{\partial i}, x_j, y_{\partial j}) = \exp(-\gamma ||x_i - x_j||^2) + \\ + \beta \left(\sum_{k \in \partial i} \delta(1, y_k) - \delta(-1, y_k) \right) \left(\sum_{k \in \partial j} \delta(1, y_k) - \delta(-1, y_k) \right).$$

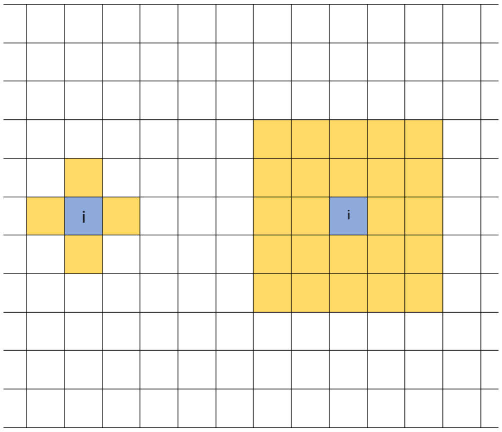


Рис. 1: Тут представлены 2 схемы соседей: тип 1 и тип 2. Сами соседи выделены жёлтым. У схемы типа 1 соседями являются только левый, верхний, правый и нижний пиксели. У схемы типа 2 соседи расположены внутри квадрата со стороной в 5 пикселей.

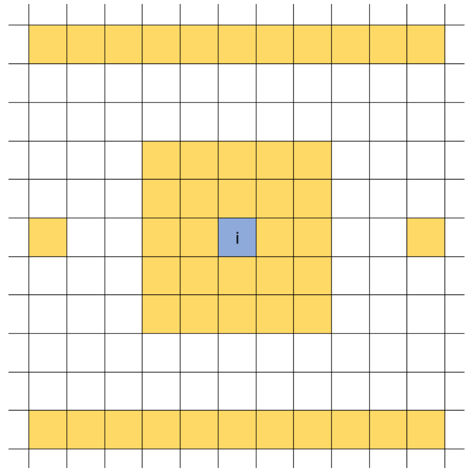


Рис. 2: У схемы типа 3 соседи расположены внутри квадрата со стороной в 5 пикселей, а также "соседями" являются пиксели на удалении от квадрата на 2 пикселя.

Как видно, ядру необходимо иметь всю карту меток для построения предсказания. Однако при использовании модели для построения предсказания для неизвестного признака карта меток заведома неизвестна. Но в качестве начальной инициализации можно брать карту меток, заполненную нулями, тогда классификатор справится как обычный SVM. После чего полученные метки можно использовать в качестве приближенной карты и затем итерационно улучшать карту меток.

Параметр γ находится с помощью выборочной дисперсии обучающей выборки. Параметр β находится численным решеним системы неравенств из секции 4.1

4.3 Мультиклассовая стратегия классификации

Таким образом, был получен бинарный классификатор, однако будем работать с числом классов большим двух. Для этого будет использована мультиклассовая стратегия "каждый против каждого" ("one-vs-one"). То есть будут построены $n(n-1)/2$ классификаторов, где n — число классов, которые будут работать только с двумя классами, пытаясь отличить один от другого.

5 Полученные результаты

Использовался следующей набор данных "Artificial Lunar Landscape Dataset" из [2] с разрешением 720×480. Для простоты вычислений было решено промасштабировать изображения до разрешения 90×60 и брать пять изображений для обучения, а тест проводить на пяти других случайных изображениях из оставшихся изображений.

Из полученных результатов видно, что MRF-SVM справляется на этом наборе данных не лучше обычного SVM, однако некоторые схемы соседей могут улучшить распознование наиболее плохо предсказанных классов. Улучшению распознования отдельного класса способствует схождение параметра β, из классификаторов модели, соответствующих этому классу, к своему оптимальному значению. Графики параметра β представлены в приложении.

	Небо	Грунт	Камни	Общая точность
SVM	91%	81%	26%	88%
MSVM (тип 1)	91%	81%	26%	88%
MSVM (тип 2, a = 1)	91%	80%	28%	88%
MSVM (тип 2, a = 2)	91%	81%	26%	88%
MSVM (тип 2, a = 3)	91%	79%	38%	87%
MSVM (тип 2, a = 5)	91%	78%	32%	87%
MSVM (тип 3, a = 1, bs = 5)	91%	81%	26%	88%
MSVM (тип 3, a = 3, bs = 5)	91%	78%	32%	87%
MSVM (тип 3, a = 5, bs = 5)	91%	78%	32%	87%

Таблица 1: Результаты полученные на наборе данных "Artificial Lunar Landscape Dataset". В зависимости от схемы соседей модификации MRF-SVM справляются по-разному, однако общая точность существенно не отличается от обычного SVM. Здесь a — сторона квадрата, bs — смещение от квадрата.

6 Заключение

В ходе этой работы был написан класс, реализующий модель MRF-SVM и различные настраиваемые схемы соседей, на языке Python. Различные вариации этой модели были протестированы на наборе данных "Artificial Lunar Landscape Dataset". Разные способы охвата контекстной информации были сравнены и было выяснено, что локальные априорные функции энергии, охватывающие большие множества соседей, способны собрать достаточное количество контекстной информации и тем самым повлиять на принятие решения в пользу хуже представленных классов, что улучшает распознование экземпляров таких классов. Рассмотренный подход позволяет успешно решать задачу распознавания неба (точность 91%), грунта ($\sim 80\%$) и камней ($\sim 30\%$)

на изображениях поверхности Луны. При этом удачный выбор схемы позволяет существенно поднять точность распознавания камней — с 26% для SVM до 38%. Таким образом, подобное объединение методов имеет практический смысл и может быть полезно для решения ряда практических задач распознавания объектов на фотографиях.

7 Список литературы

[1] Gabriele Moser and Sebastiano B. Serpico, “Combining Support Vector Machines and Markov Random Fields in an Integrated Framework for Contextual Image Classification,” IEEE Geosci. Remote Sens., vol. 51, no. 5, pp. 2734–2752, May 2013.

[2] Kaggle [Электронный ресурс]: Artificial Lunar Landscape Dataset. URL: https://www.kaggle.com/romainpessia/artificial-lunar-rocky-landscape-dataset (дата обращения: 22.05.2021).

[3] Chih-Chung Chang and Chih-Jen Lin, “A Library for Support Vector Machines,” ACM TIST, vol. 2, no. 3, article 27, Apr. 2011.

[4] D. A. Landgrebe, “Signal Theory Methods in Multispectral Remote Sensing,” Hoboken, NJ: Wiley, 2003.

[5] G. Camps-Valls and L. Bruzzone, “Kernel Methods for Remote Sensing Data Analysis,” Hoboken, NJ: Wiley, 2009.

[6] P. Mantero, G. Moser, and S. B. Serpico, “Partially supervised classification

of remote sensing images using SVM-based probability density estimation," IEEE Trans. Geosci. Remote Sens., vol. 43, no. 3, pp. 559–570, Mar. 2005.

[7] S. Li, "Markov Random Field Modeling in Image Analysis," Berlin, Germany: Springer-Verlag, 2009.

[8] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration," IEEE Trans. Pattern Anal. Mach. Intell., vol. PAMI-6, no. 6, pp. 721–741, Nov. 1984.

[9] F. Melgani and S. B. Serpico, "A statistical approach to the fusion of the spectral and spatio-temporal contextual information for the classification of remote sensing images," Pattern Recognit. Lett., vol. 23, no. 9, pp. 1053–1061, Jul. 2002.

[10] A. H. S. Solberg, T. Taxt, and A. K. Jain, "A Markov random field model for classification of multisource satellite imagery," IEEE Trans. Geosci. Remote Sens., vol. 34, no. 1, pp. 100–113, Jan. 1996.

[11] F. Melgani and S. B. Serpico, "A Markov random field approach to spatio-temporal contextual image classification," IEEE Trans. Geosci. Remote Sens., vol. 41, no. 11, pp. 2478–2487, Nov. 2003.

[12] G. Moser and S. B. Serpico, "Unsupervised change detection from multichannel SAR data by Markovian data fusion," IEEE Trans. Geosci. Remote Sens., vol. 47, no. 7, pp. 2114–2128, Jul. 2007.

[13] G. Moser, S. B. Serpico, and G. Vernazza, "Unsupervised change detection from multichannel SAR images," IEEE Geosci. Remote Sens. Lett., vol. 4, no. 2, pp. 278–282, Apr. 2007.

[14] G. Storvik, R. Fjortoft, and A. H. S. Solberg, “A Bayesian approach to classification of multiresolution remote sensing data,” IEEE Trans. Geosci. Remote Sens., vol. 43, no. 3, pp. 539–547, Mar. 2005.

[15] G. Moser and S. B. Serpico, “Automatic parameter optimization for support vector regression for land and sea surface temperature estimation from remote-sensing data,” IEEE Trans. Geosci. Remote Sens., vol. 47, no. 3, pp. 909–921, Mar. 2009.

[16] G. Camps-Valls, L. Gomez-Chova, J. Munoz-Mari, J. Vila-Frances, and J. Calpe-Maravilla, “Composite kernels for hyperspectral image classification,” IEEE Geosci. Remote Sens. Lett., vol. 3, no. 1, pp. 93–97, Jan. 2006.

[17] F. Lafarge, X. Descombes, and J. Zerubia, “Textural kernel for SVM classification in remote sensing: Application to forest fire detection and urban area extraction,” in Proc. ICIP, Genoa, Italy, 2005, pp. III-1096–III-1099.

8 Приложение

8.1 Код

```
from sklearn.svm import SVC
import numpy as np
import numexpr as ne


class MSVC(SVC):


    def __init__(self, width, height, C=1.0, kernel='rbf', gamma='scale',
                 beta = None, neigh_type = 0, deep = 1, bias = 0,
                 degree=3, coef0=0.0, shrinking=True, probability=False,
                 tol=1e-3, cache_size=200, class_weight=None,
                 verbose=False, max_iter=-1, break_ties=False,
                 random_state=None):

        self.neigh_type = neigh_type
        self.deep = deep
        self.bias = bias
        self.mrbf = False
        self.beta = beta
        self.C = C
        self.kernel = kernel
        self.gamma = gamma
        self.degree = degree
        self.coef0 = coef0
        self.shrinking = shrinking
        self.probability = probability
        self.tol = tol
        self.cache_size = cache_size
        self.class_weight = class_weight
        self.verbose = verbose
        self.max_iter = max_iter
        self.break_ties = break_ties
```

```
        self.random_state = random_state
        self.width = width
        self.height = height

        if kernel == "mrbf":
            self.mrbf = True


    def fit(self, X, y):
        self.classes = np.unique(y)
        self.models = []
        if self.beta == None:
            self.beta = [0 for i in range(int(len(self.classes)*(len(self.classes)-1)/2))]
        if self.gamma == 'scale':
            self.gamma = 1/(X.var()*X.shape[1])
        elif self.gamma == 'auto':
            self.gamma = 1/(X.shape[1])
        for i in range(len(self.classes)):
            for j in range(i+1, len(self.classes)):
                X_1, y_1, index = self.div_cl(X, y, fir_cl = self.classes[i], sec_cl = self.classes[j])

                if self.mrbf:
                    self.kernel = self.my_kernel(self.gamma, self.beta[int((2*len(self.classes)-1-i)*i/2)+j-i-1], index = X.shape[1])

                model = SVC(C=self.C, kernel=self.kernel, gamma=self.gamma,
                 degree=self.degree, coef0=self.coef0, shrinking=self.shrinking, probability=self.probability,
                 tol=self.tol, cache_size=self.cache_size, class_weight=self.class_weight,
                 verbose=self.verbose, max_iter=self.max_iter,
                 break_ties=self.break_ties, random_state=self.random_state)

                if self.mrbf:
                    self.models.append(model.fit(self.merge(X_1,self.neighbor_labels(y, width = self.width, height = self.height, cl1 = self.
```

```
classes[i], cl2 = self.classes[j], neigh_type = self.neigh_type,
                    deep = self.deep, bias = self.bias)[index]), y_1))
                else:
                    self.models.append(model.fit(X_1, y_1))

                self.beta[int((2*len(self.classes)-1-i)*i/2)+j-i-1] = self.
beta_search(
                    self.models[int((2*len(self.classes)-1-i)*i/2)+j-i-1],
X_1, y, index = index, gamma = self.gamma, width = self.width, height =
self.height,
                    neigh_type = self.neigh_type, deep = self.deep, bias =
self.bias, fir_cl = self.classes[i], sec_cl = self.classes[j])

        return self

    def predict(self, X, width, height, y = None):
        y_pred = []
        for i in range(len(self.classes)):
            for j in range(i+1, len(self.classes)):
                if self.mrbf:
                    y_pred.append(self.models[int((2*len(self.classes)-1-i)*i
/2)+j-i-1].predict(self.merge(X,self.neighbor_labels(y, width, height, cl1
 = self.classes[i], cl2 = self.classes[j], neigh_type = self.neigh_type,
                    deep = self.deep, bias = self.bias))))
                else:
                    y_pred.append(self.models[int((2*len(self.classes)-1-i)*i
/2)+j-i-1].predict(X))

        y_ans = np.zeros([len(y_pred[0])])
        for i in range(len(y_ans)):
            count = [0 for k in range(max(self.classes)+1)]
            for j in range(int(len(self.classes)*(len(self.classes)-1)/2)):
                count[int(y_pred[j][i])] += 1
            for k in range(len(count)):
                if count[k] == max(count):
                    y_ans[i] = k
                    break

```

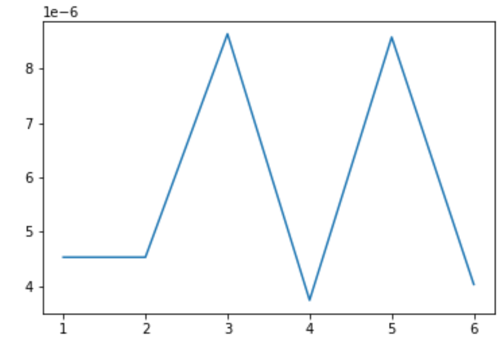
```
        return y_ans

    @staticmethod
    def beta_search(self, x, y, index, gamma, width, height, neigh_type = 0, deep = 1, bias = 0, fir_cl = 0, sec_cl = 1):

        alpha = self.dual_coef_
        b = self.intercept_
        supp = self.support_

        eps = MSVC.neighbor_labels(y, width, height, cl1 = fir_cl, cl2 = sec_cl, neigh_type = neigh_type,
            deep = deep, bias = bias)[index]
        y = y[index]
        mx = 0
        mn = 0
        xj = x[supp]
        E10 = alpha
        E20 = np.sum(alpha*eps[supp])
        for i in range(len(y)):
            if y[i] == fir_cl:
                yi = 1
            else:
                yi = -1
            E1 = (np.sum(E10*(np.sum(np.exp(-gamma*(x[i]-xj)**2),axis = 1))+b)*yi
            E2 = E20*yi*eps[i]
            if E2 > 0 and -E1/E2 > mx:
                if mn >= -E1/E2:
                    mx = -E1/E2
            elif E2 < 0 and -E1/E2 < mn:
                if -E1/E2 >= mx:
                    mn = -E1/E2
            elif E2 < 0 and mn == 0:
                mn = -E1/E2
        return mn
```

```
    @staticmethod
    def my_kernel(gamma = 1, beta = 0, index = 1):
        def MRF_kernel(X1, X2):
            return ne.evaluate('exp(-gamma*(A+B-2*C)) + beta * D', {
                    'A' : np.einsum('ij,ij->i',X1[:,:index],X1[:,:index])[:,None],
                    'B' : np.einsum('ij,ij->i',X2[:,:index],X2[:,:index])[None,:],
                    'C' : np.dot(X1[:,:index], X2[:,:index].T),
                    'D' : (X1[:,index]*X2[:,index,None]).T,
                    'gamma' : gamma,
                    'beta' : beta
                   })
        return MRF_kernel

    @staticmethod
    def neigh_index(i, width, height, neigh_type, deep = 1, bias = 0):
        neighs = []
        if neigh_type == 0:
            if (i % width) > 0:
                neighs.append(i-1)
            if (i % width) < width-1:
                neighs.append(i+1)
            if (i % (width*height)) >= width:
                neighs.append(i-width)
            if (i % (width*height)) < (height-1)*width:
                neighs.append(i+width)
        elif neigh_type == 1:
            for j in range(2*deep+1):
                for l in range(2*deep+1):
                    if (i % width) - (deep-l)  < 0 and l < deep:
                        continue
                    if (i % width) + (l-deep) >= width and l > deep:
                        continue
                    if (i % (width*height)) - (deep-j)*width < 0 and j < deep:
                        continue
                    if (i % (width*height)) + (j-deep)*width >= height*width
```

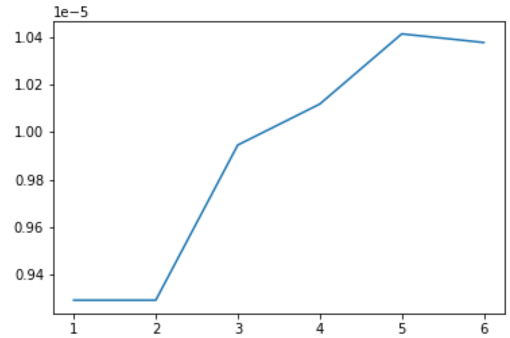
```
and j > deep:
                        continue
                    n = i - deep - width*deep + l + j*width
                    if n != i:
                        neighs.append(n)
        elif neigh_type == 2:
            for j in range(2*deep+1):
                for l in range(2*deep+1):
                    if (i % width) - (deep-l)  < 0 and l < deep:
                        continue
                    if (i % width) + (l-deep) >= width and l > deep:
                        continue
                    if (i % (width*height)) - (deep-j)*width < 0 and j < deep:
                        continue
                    if (i % (width*height)) + (j-deep)*width >= height*width and j > deep:
                        continue
                    n = i - deep - width*deep + l + j*width
                    if n != i:
                        neighs.append(n)
            for j in range(2*bias+1):
                if (i % width) - (bias-j)  >= 0 and j < bias:
                        continue
                if (i % width) + (j-bias) >= width and j > bias:
                        continue
                if (i % (width*height)) - bias*width >= 0:
                    neighs.append(i - bias - width*bias + j)
                if (i % (width*height)) +bias*width < height*width:
                    neighs.append(i - bias + width*bias + j)
            if (i % width) - bias >= 0:
                neighs.append(i - bias)
            if (i % width) + bias < width:
                neighs.append(i + bias)
        return neighs

    @staticmethod
    def neighbor_labels(Y, width, height, cl1 = 0, cl2 = 1, neigh_type = 0,
```

```
deep = 1, bias = 0):
     if not (Y is None):
       Y2 = np.zeros([len(Y)])
       for i in range(len(Y)):
           count_0 = 0
           count_1 = 0
           for j in MSVC.neigh_index(i, width, height, neigh_type, deep,
bias):
               if Y[j] == cl2:
                   count_0 += 1
               elif Y[j] == cl1:
                   count_1 += 1
           Y2[i] = count_1 - count_0
       return Y2
     else:
       return []

 @staticmethod
 def div_cl(X, y, fir_cl, sec_cl):
     n = ((y == fir_cl) | (y == sec_cl)).sum()
     X_new = np.zeros([n, X.shape[1]])
     y_new = np.zeros([n])
     index = np.zeros([n], dtype = int)
     j = 0
     for i in range(len(y)):
         if y[i] == fir_cl or y[i] == sec_cl:
             X_new[j] = X[i]
             y_new[j] = y[i]
             index[j] = i
             j += 1
     return X_new, y_new, index

 @staticmethod
 def merge(X, y):
     return np.append(X, y[:,None], axis = 1)
```

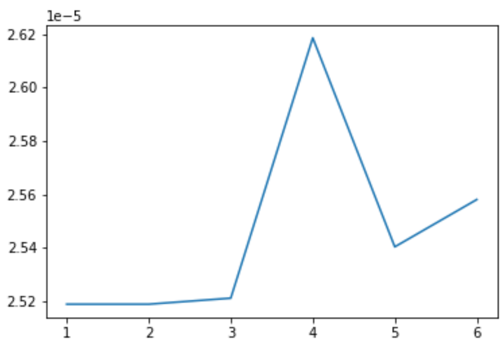
8.2 Поведение константы β



Класс "небо" против класса "грунт"

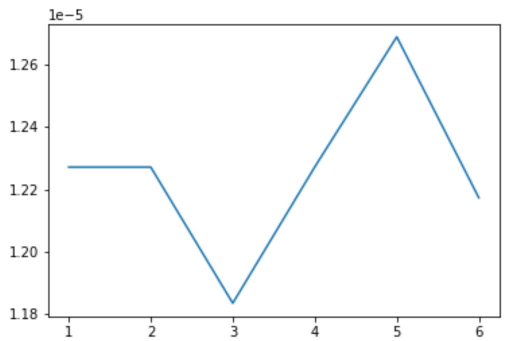


Класс "небо" против класса "камни"

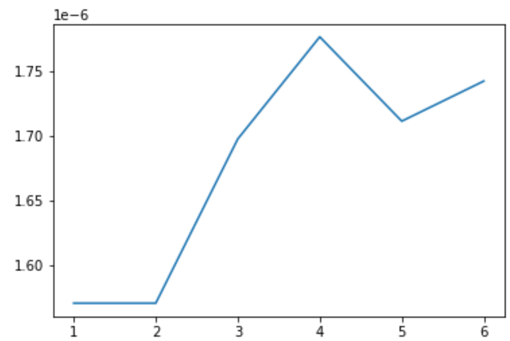


Класс "грунт" против класса "камни"

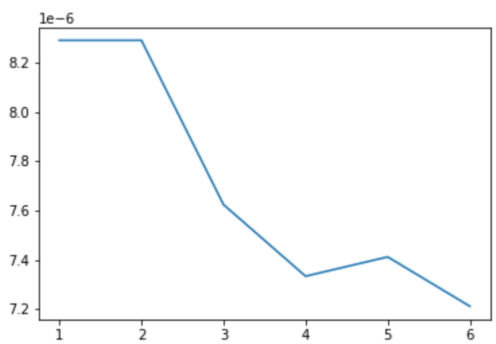
Рис. 3: Поведение костанты β на 6 итерациях MRF-SVM (тип 1)



Класс "небо" против класса "грунт"

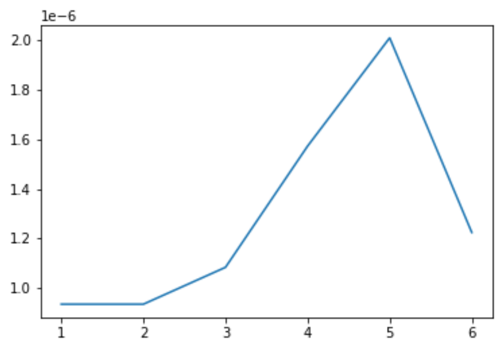


Класс "небо" против класса "камни"

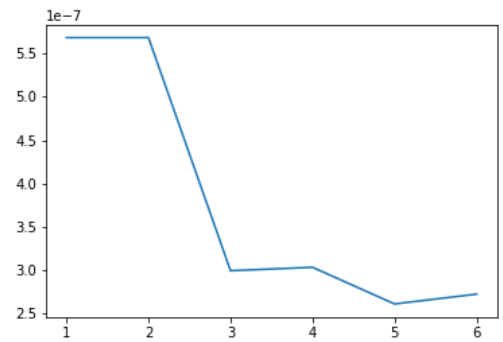


Класс "грунт" против класса "камни"

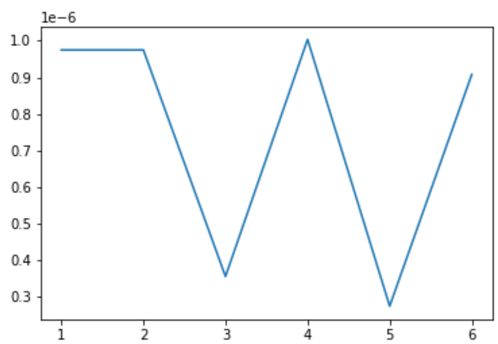
Рис. 4: Поведение костанты β на 6 итерациях MRF-SVM (тип 2, а = 1)



Класс "небо" против класса "грунт"

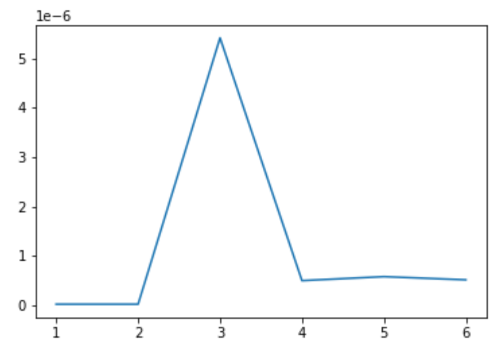


Класс "небо" против класса "камни"

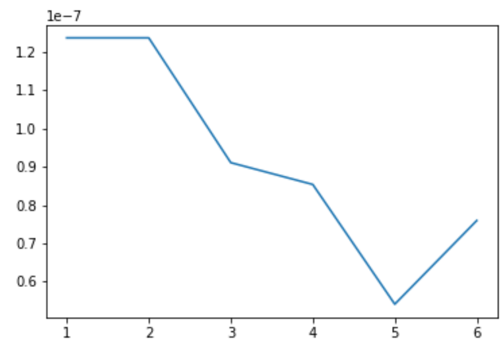


Класс "грунт" против класса "камни"

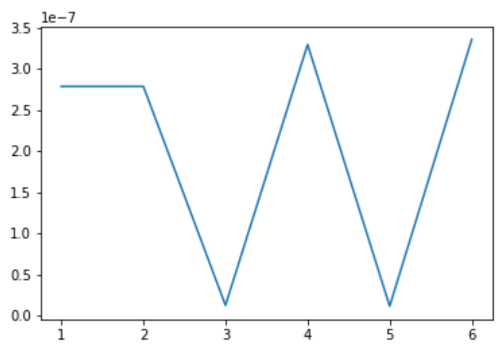
Рис. 5: Поведение костанты β на 6 итерациях MRF-SVM (тип 2, a = 2)



Класс "небо" против класса "грунт"

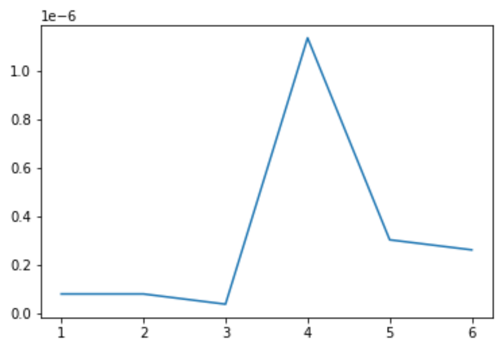


Класс "небо" против класса "камни"

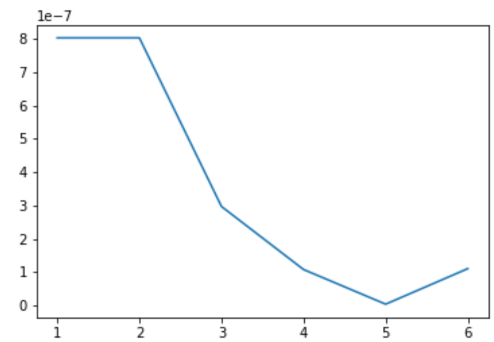


Класс "грунт" против класса "камни"

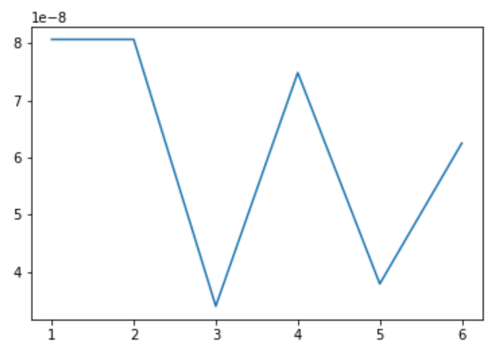
Рис. 6: Поведение костанты β на 6 итерациях MRF-SVM (тип 2, а = 3)



Класс "небо" против класса "грунт"

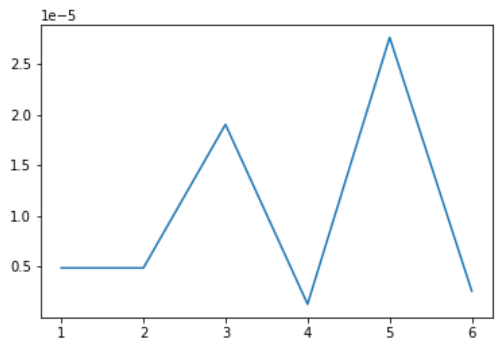


Класс "небо" против класса "камни"

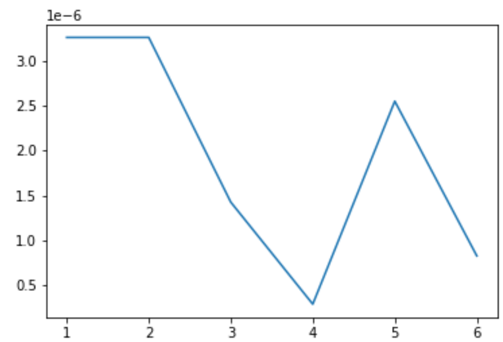


Класс "грунт" против класса "камни"

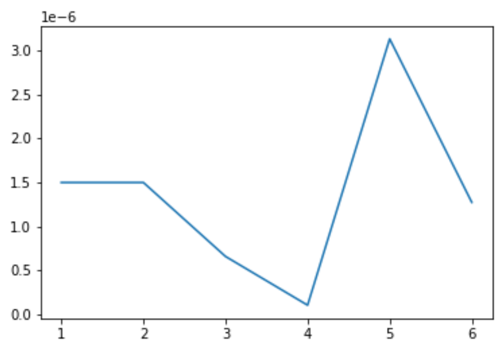
Рис. 7: Поведение костанты β на 6 итерациях MRF-SVM (тип 2, а = 5)



Класс "небо" против класса "грунт"

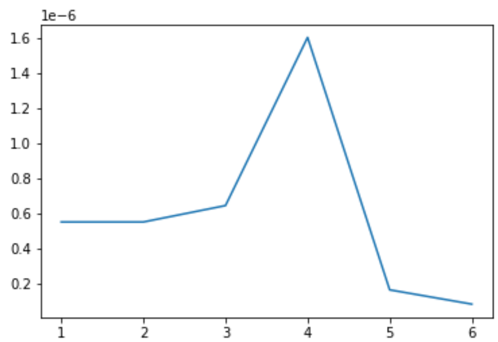


Класс "небо" против класса "камни"

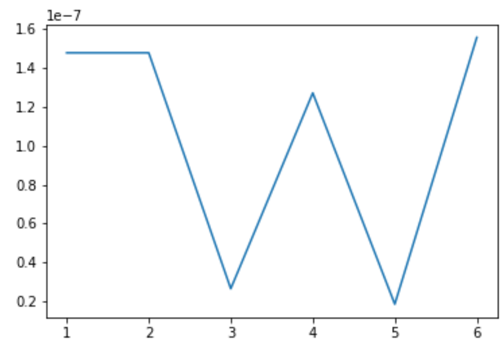


Класс "грунт" против класса "камни"

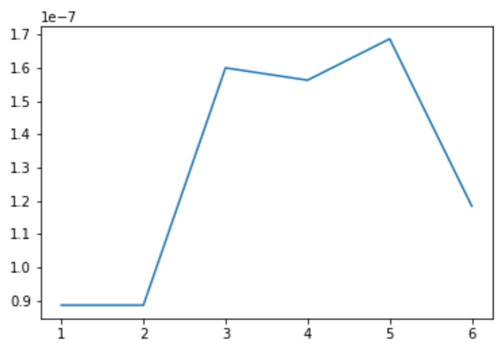
Рис. 8: Поведение костанты β на 6 итерациях MRF-SVM (тип 3, a = 1, bs = 5)



Класс "небо" против класса "грунт"

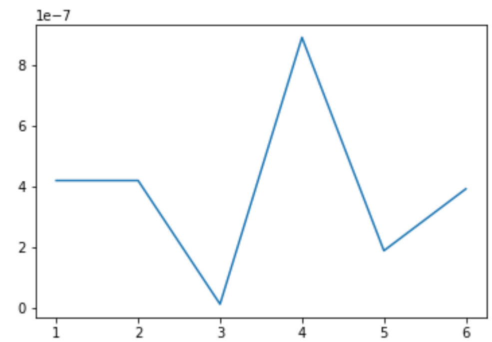


Класс "небо" против класса "камни"

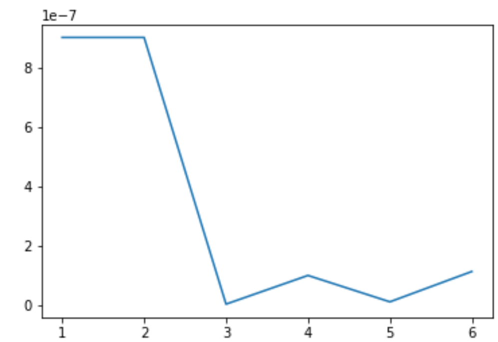


Класс "грунт" против класса "камни"

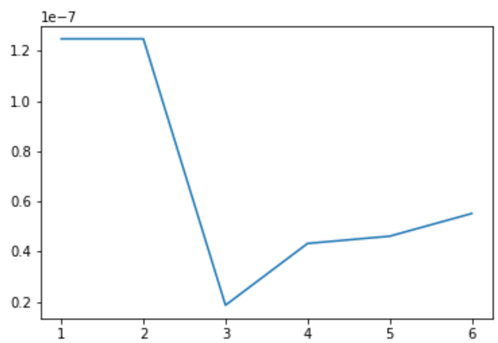
Рис. 9: Поведение костанты β на 6 итерациях MRF-SVM (тип 3, a $=$ 3, bs $=$ 5)



Класс "небо" против класса "грунт"



Класс "небо" против класса "камни"



Класс "грунт" против класса "камни"

Рис. 10: Поведение костанты β на 6 итерациях MRF-SVM (тип 3, a = 5, bs = 5)