МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «ВГУ»)

Факультет прикладной математики, информатики и механики

Кафедра вычислительной математики
и прикладных информационных технологий

Применение персистентных гомологий для задачи классификации изображений

Бакалаврская работа

Направление 01.03.02 Прикладная математика и информатика

Профиль Математическое моделирование и вычислительная математика

Зав. кафедрой _____ д.т.н., проф. Т. М. Леденева __.__.20__

Обучающийся _____ П. М. Снопов

Руководитель _____ д.т.н., проф. Т. М. Леденева

Воронеж 2021

Содержание

Введение

Топологический анализ данных – это новая область анализа данных, которая своими корнями уходит в алгебраическую топологию. Она возникла в результате прорывных работ Герберта Эдельсбруннера [1] и Гуннара Карлссона [2] по устойчивым гомологиям, которые впоследствии стали основными инструментами данной сферы.

В задачах анализа данных большое значение имеет информация о геометрической и топологической структуре исследуемых данных. Зачастую такая структура позволяет выявить некоторые закономерности между различными переменными. Помимо этого, существует также гипотеза о многообразии, которая утверждает, что реальные данные, представленные облаком точек из $\mathbb{R}^n$ лежат на некотором d-многообразии, где $d \leq n$.

Данная работа посвящена обзору современных инструментов топологического анализа данных, а также применению персистентных (устойчивых) гомологий, которые являются основным инструментом данного подхода, к задаче классификации на примере стандартного для классификации датасета MNIST изображений рукописных цифр. Основная цель данной работы – построение алгоритма, который, на основе имеющейся изображений в выборке, будет способен определить, к какому классу относится новое изображение, т.е. изображение, не находящийся в исходной выборке.

Актуальность выбранной темы выражается в современным тенденциях прикладной математики: анализ данных и машинное обучение – одна из самых быстроразвивающихся областей, которая за последние десятилетия изменила окружающий нас мир. Так, на сервисе `arxiv.org` – сервисе по

свободному распространению и открытому доступу научных статей в области физики, математики, компьютерных и других вычислительных наук – ежедневно появляется порядка 150 публикаций каждый день [3].

С другой стороны, данная работа не в последнюю очередь – это работа по прикладной топологии. Эта область сейчас переживает свой расцвет: в последние годы появляется все больше прикладных работ, в которых используются устойчивые гомологии. Помимо этого, развивается и теоретический аппарат [4]. Одним из примеров таких работ служат работы, которые направлены на изучение т.н. латентного пространства некоторых видов нейросетей: VAE и GAN. Эти нейросети в ходе обучения пытаются воссоздать то самое многомерное многообразие, на котором лежат данные, например, изображения. Латентное пространство зачастую скрыто во многих методах анализа данных, и содержит всю необходимую информацию, поэтому изучение его свойств крайне важно. Топологический анализ данных, в силу своего топологического фундамента, может быть крайне полезен для такой задачи. Данная тема является популярной темой современных исследований в этой области[5—8].

В ходе данной работы был изучен теоретический материал по алгебраической и прикладной топологии, машинному обучению и тому, как связать, казалось бы, не связываемые на первый взгляд эти две области математики друг с другом. Была написана программа, которая для каждого изображения из датасета MNIST создает его векторное представление на основе топологических свойств изображения. Была выбрана модель машинного обучения, которая дает наилучший результат в поставленной задаче классификации. Таким образом, был получен алгоритм, который с высокой(более 90%) точностью способен определить рукописную цифру, оцифрованную в виде изображения, который в первую очередь эксплуатирует именно топологические особенности.

В ходе данной работы было также выявлено, что реализованный ал-

горитм является эффективным алгоритмом понижения размерности пространства признаков. Данный алгоритм сравнивался с другими моделями машинного обучения, которые были обучены обычным образом – на вход моделям подавалось плоское представление картинки. В результате сравнения было получено, что алгоритм классификации, который использовал топологические признаки, показывал более высокую точность при меньшем числе признаков, чем стандартные алгоритмы.

1 Необходимые теоретические сведения по алгебраической и прикладной топологии, а также по машинному обучению

1.1 Некоторые сведения из топологии

Здесь будут приведены некоторые теоретические сведения из топологии, используемые в дальнейшем. Более подробно об этом можно прочитать в специальной литературе, например в [9—11].

Основным объектом в топологическом анализе данных можно считать симплициальный комплекс. В некотором смысле, это обобщение графов на более старшие размерности. В частности, по любому графу можно построить (флаговый) комплекс, объявив каждую его клику симплексом.

Симплексом размерности n называют Выпуклую оболочку набора $\{x_0, ..., x_n\} \subset \mathbb{R}^d$, причем такого, что векторы $x_1 - x_0, ..., x_n - x_0$ линейно независимы. Гранью n-симплекса называют k-симплекс, который получается как выпуклая оболочка некоторого подмножества вершин этого симплекса.

Так, n-симплекс можно рассматривать как n-мерное обобщение треугольника и тетраэдра. На рисунке 1.1 приведены примеры симплексов.

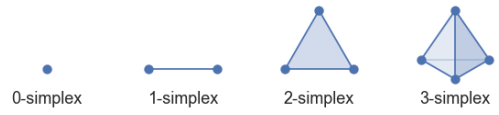


Рис. 1.1: Различные симплексы

Определение 1. *Топологический симплициальный комплекс K – это множество симплексов такое, что:*

1. *Для каждого симплекса из K его грани тоже лежат в K.*

2. *Пересечение любых двух симплексов $\sigma, \tau \in K$ либо пусто, либо является гранью и σ, и τ.*

На рисунке 1.2 представлен пример симплициального комплекса. Так как среди всех симплексов, входящих в представленный на рисунке комплекс, размерность не выше 3, то и размерность симплициального комплекса равна 3. По топологическому симплициальному комплексу можно

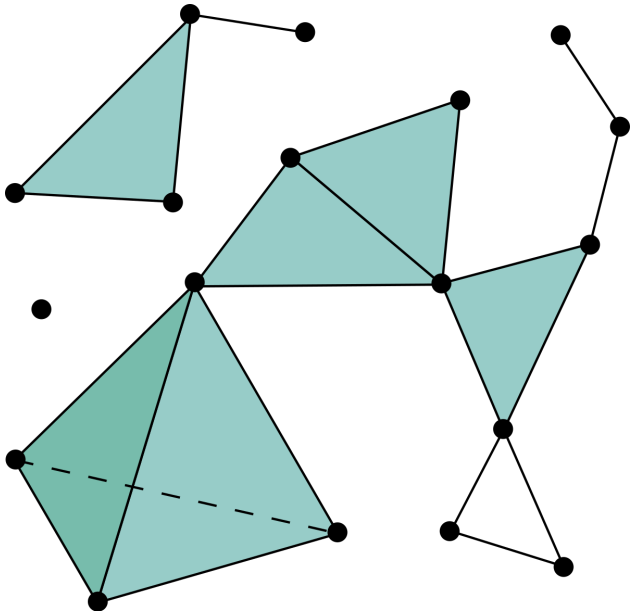


Рис. 1.2: Симплициальный комплекс

построить топологическое пространство – объединение его симплексов. Такие пространства тоже называют симплициальными комплексами. В таком пространстве топология индуцируется топологией на $\mathbb{R}^d$.

Имея топологическое пространство X, говорят, что набор его подмножеств $\{U_i : i \in J\}$ является *покрытием*, если $\bigcup\limits_{i \in J} U_i = X$. Покрытие называется *открытым*, если оно состоит из открытых множеств.

По покрытию пространства можно построить нерв покрытия (рисунке. 1.3) – симплициальный комплекс, соответствующий топологическому

пространству и имеющий различные интересные топологические свойства. Пусть $[m] \coloneqq \{1, ..., m\}$ – m-элементное множество.

Определение 2. *Нерв покрытия $N(U)$, соответствующий топологическому пространству X, – это абстрактный симплициальный комплекс на $[m]$, такой что $\{k_1, ..., k_s\} \in N(U)$ если $U_{k_1} \cap ... \cap U_{k_s} \neq \emptyset$.*

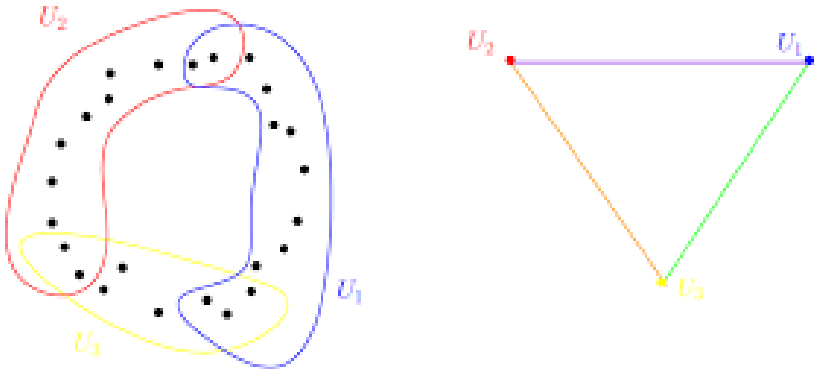


Рис. 1.3: Покрытие пространства и нерв покрытия

Если имеется 2 непрерывных отображения $f, g : X \to Y$, то говорят, что f *гомотопно* g, если существует такое отображение $H : X \times I \to Y$, что $H(x,0) = f$ и $H(x,1) = g$. В таких случаях пишут, что $f \sim g$. Гомотопия является отношением эквивалентности на множестве непрерывных отображений $C(X,Y)$.

Два топологических пространства X,Y гомотопически эквивалентны $(X \sim Y)$, если существует такая пара $(f,g) \in C(X,Y)^2$, такая, что $f \circ g \sim id_Y$ и $g \circ f \sim id_X$.

С понятием нерва покрытия связана очень важная лемма:

Лемма (о нерве)**.** *Если U – открытое покрытие, такое, что любое пересечение его подмножеств либо пусто, либо гомотопически эквивалентно точке, то нерв покрытия пространства X гомотопически эквивалентен*

самому пространству

$$\forall I \in N(U) \left(\bigcap_{i \in I} U_i = \emptyset \vee \bigcap_{i \in I} U_i \sim pt \to N(U) \sim X \right).$$

Как всякая хорошая математическая структура, симплициальные комплексы образуют *категорию* **SCpx**. Морфизмами в такой категории являются симплициальные отображения $\varphi : M \to N$, т.е. такие отображения, что для каждого симплекса из M φ является линейным отображением на симплекс из N.

Имея симплициальный комплекс, можно найти его симплициальные гомологии. Такие гомологии можно рассматривать как функтор из категории симплициальных комплексов в категорию абелевых групп: каждому симплициальному комплексу сопоставляется некоторая абелева группа. Замечательный момент заключается в том, что гомеоморфные симплициальные комплексы имеют одинаковые группы гомологий, а потому теория гомологий является очень удобным инструментом для изучения симплициальных комплексов. На самом деле данные утверждения можно легко обобщить на общий случай топологических пространств.

Дадим формальное определение симплициальным гомологиям. Итак, пусть K – симплициальный комплекс и $k \geq 0$.

Группой k-цепей $C_k(K)$ симплициального комплекса K называют (свободную абелеву) группу, элементами которого являются формальные суммы k-симплексов K, то есть

$$c = \sum_i \varepsilon_i \sigma_i, \;\; \varepsilon_i \in \mathbb{Z},$$

где конечное число $\varepsilon_i \neq 0$, σ_i – k-симплекс. *Границей* k-цепи $\sum_i \varepsilon_i \sigma_i$ назы-

вают $(k-1)$-цепь

$$\partial_k(\sigma) = \sum_{j=0}^{k}(-1)^j \sum_i \varepsilon_i \partial_j \sigma_i,$$

где $\partial_j \sigma_i = \partial_j [v_0, ..., v_k] = [v_0, ..., \hat{v}_j, ..., v_k]$ – $(k-1)$-симплекс, порожденный всеми вершинами, кроме вершины v_j. Гомоморфизм $\partial_k : C_k(K) \to C_{k-1}(K)$ называют *граничным оператором*. Он удовлетворяет следующему свойству:

$$\partial_{k-1} \circ \partial_k = 0.$$

То есть $\operatorname{im} \partial_{k+1} \leq \ker \partial_k \leq C_k(K)$. Последовательность $C_k(K)$ и ∂_k называется *цепным комплексом*

$$... \xrightarrow{\partial_{k+2}} C_{k+1} \xrightarrow{\partial_{k+1}} C_k \xrightarrow{\partial_k} C_{k-1} \xrightarrow{\partial_{k-1}} ... \xrightarrow{\partial_1} C_0.$$

Цепной комплекс можно визуализировать следующим образом (рисунок 1.4).

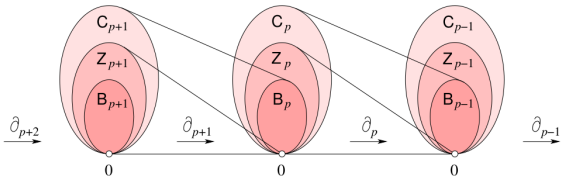


Рис. 1.4: Цепной комплекс

Определение 3. *k-ой группой гомологий симплициального комплекса K называют следующее фактор-пространство:*

$$H_k(K) = \ker \partial_k \,/ \operatorname{im} \partial_{k+1}.$$

Тогда k-ое число Бетти – размерность k-ой группы гомологий:

$$\beta_k(K) = \dim H_k(K).$$

При $k = 0$ число Бетти описывает количество компонент связности

данного пространства. При $k = 1$ – количество циклов. При $k = 2$ число Бетти описывает количество «полостей». В таблице 1.1 представлены первые числа Бетти для некоторых пространств.

Таблица 1.1: Первые числа Бетти для некоторых пространств

Пространство	β_0	β_1	β_2
Pt	1	0	0
D^2	1	0	0
Треугольник	1	0	0
Граница треугольника	1	1	0
S^1	1	1	0
S^2	1	0	1
$\mathbb{T}^2 = S^1 \times S^1$	1	2	1

1.2 Персистентные гомологии и их векторные представления

1.2.1 Персистентные гомологии

Далее будет приведен теоретический материал по устойчивым гомологиям. Подробнее можно прочитать в [1, 12, 13]

О персистентных (устойчивых) гомологиях можно думать как об адаптации понятия гомологии к облаку точек.

Имея облако точек X, существует много способов построения симплициальных комплексов по нему. Например, задав некоторый $\tau > 0$, можно рассматривать не просто точки, но и их окрестности с радиусом τ. Тогда можно рассматривать следующую структуру: подмножество $\sigma \subset X$ будет симплексом, если пересечение окрестностей точек из σ будет непусто. Тогда симплициальным комплексом, построенным по облаку точек, будет совокупность таких симплексов. Такой симплициальный комплекс называется *комплексом Чеха* $\mathrm{Cech}_\tau(X)$.

На комплекс Чеха можно взглянуть с другой стороны: если данные действительно хорошо описывают некоторый топологический объект, который как бы стоит за этими данными, то, подобрав хороший радиус τ, объединение полученных окрестностей будет гомотопически эквивалентно этому объекту. А так как все шары выпуклы, то объединение окрестностей будет гомотопически эквивалентно нерву данного покрытия. А значит, по лемме о нерве, сам нерв будет гомотопически эквивалентен топологическому объекту, который описывается данными. И комплекс Чеха как раз и является нервом покрытия.

Но на практике оказывается, что комплекс Чеха очень сложно посчитать. Поэтому существует другой способ построения симплициального комплекса по облаку точек, более эффективный с точки зрения вычислений: зафиксировав τ, подмножество $\sigma \subset X$ будет симплексом, если $\forall i,j \in \sigma : \rho(i,j) < \tau$. Симплициальным комплексом будет совокупность таких симплексов. Полученный симплициальный комплекс называется *комплексом Вьеториса—Рипса* (или просто *комплексом Рипса*) $\mathrm{Rips}_\tau(X)$.

Алгоритм 1: Алгоритм построения комплекса Вьеториса—Рипса

Исходные параметры: Облако точек X, вещественное число $\alpha > 0$.

Результат: Симплициальный комплекс Вьеториса—Рипса

Для каждой точки x строим её α-окрестность $B_\alpha(x)$;

$i = 1$;

до тех пор, пока $i+1$ *окрестностей попарно имеют непустое пересечение*

выполнять

строим i-ый симплекс на соответствующих вершинах;

$i \leftarrow i + 1$;

конец

То есть, вместо рассмотрения пересечений шаров, как это было в случае комплекса Чеха, рассматривают попарные расстояния между точками. Очевидно, такой подход более эффективен с вычислительной точки зрения, более того, его можно обобщить на общий случай конечного метрического пространства, и даже на случай произвольного конечного множества с

симметрической функцией.

Комплексы Чеха и Рипса связаны между собой: для облака точек $X \subset \mathbb{R}^d$ они имеют одинаковый 1-остов. Более того, справедливо следующее соотношение:

$$\text{Rips}_\tau(X) \subseteq \text{Cech}_\tau(X) \subseteq \text{Rips}_{2\tau}(X).$$

На рисунке. 1.5 изображены комплексы Чеха и Рипса.

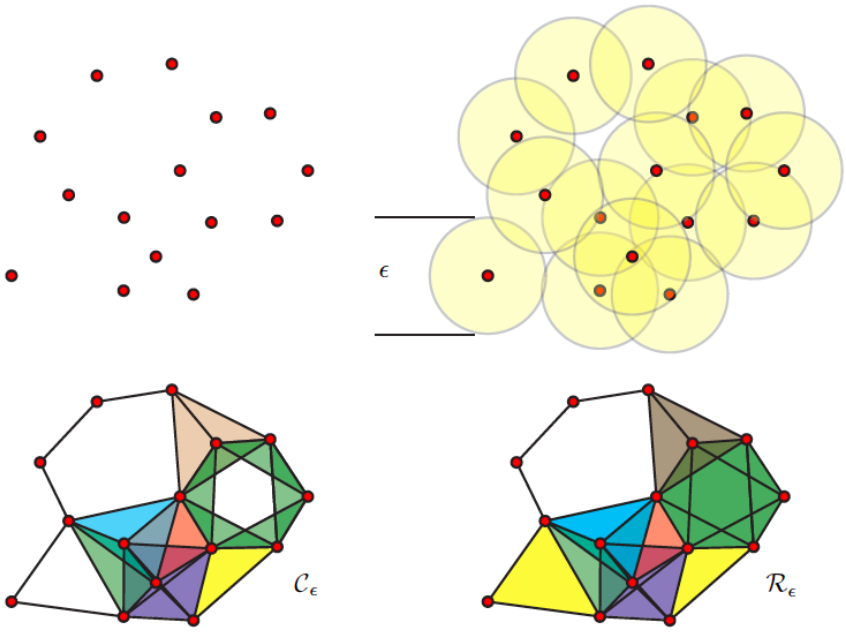


Рис. 1.5: Комплексы Чеха и Рипса

Везде выше параметр τ был зафиксирован, но его можно изменять. Вслед за этим будут меняться и симплициальные комплексы, построенные с учетом τ. *Фильтрацией симплициального комплекса* K называют вложенное семейство подкомплексов $(K_\tau)_{\tau \in T}$, где $T \subseteq \mathbb{R}$, такое, что если $\tau < \tau'$, то $K_\tau \subseteq K_{\tau'}$. Пример такой фильтрации изображен на рисунке. 1.6, где изображения фильтрация симплициальных комплексов Рипса $\text{Rips}_\tau(X)$.

Имея фильтрацию $(K_\tau)_{\tau \in T}$, можно отслеживать изменение $H_k(K_\tau)$ с ростом τ: могут появляться новые компоненты связности, уже существу-

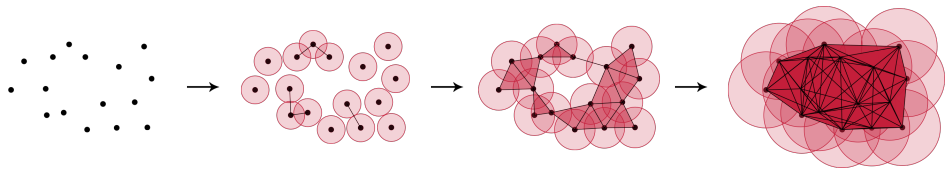


Рис. 1.6: Фильтрация Рипса

ющие могут объединяться в одну компоненту, могут появляться топологические особенности, соответствующие 1 и 2 группе гомологий.

Определение 4. *k-ыми устойчивыми гомологиями фильтрованного комплекса $(K_\tau)_{\tau\in T}$ называют проиндексированное семейство абелевых групп $H_n(T) = \{(H_n(K_\tau))_{\tau\in T}$ вместе с семейством гомоморфизмов*

$$\{(H_n(K_\tau) \to H_n(K_\tau^{'}))_{\tau\leq\tau'}\}.$$

Такое семейство абелевых групп вместе с морфизмами, на самом деле, является примером персистентного $\mathbb{Z}$-модуля($\mathbb{Z}$-модуля устойчивости): *персистентным R-модулем* (A_*, x) называют последовательность R-модулей и гомоморфизмов между ними (над целочисленной временной шкалой $\mathbb{Z}_{\geq 0}$).

$$A_0 \xrightarrow{x} A_1 \xrightarrow{x} A_2 \xrightarrow{x} A_3 \xrightarrow{x} ...$$

Основная теорема про персистентные модули – это структурная теорема. По аналогии с обычными модулями, она формулируется в терминах *конечнопорожденного* модуля устойчивости, т.е. такого модуля (A_*, x), у которого существует конечный набор $a_1, ..., a_k \in A_*$ элементов, что любой элемент $a \in A_*$ можно выразить в виде линейной комбинации элементов вида $x^s a_r = x \circ ... \circ x(a_r)$. Для ее формулировки понадобится еще пару определений: если $0 \leq j < s$, то *интервальным модулем устойчивости* $I_{[j,s)}$ называют модуль устойчивости вида

$$0 \longrightarrow ... \longrightarrow 0 \longrightarrow \underset{j}{R} \xrightarrow{id_R} \underset{j+1}{R} \xrightarrow{id_R} ... \xrightarrow{id_R} \underset{s-1}{R} \xrightarrow{0} \underset{s}{0} \longrightarrow ...$$

Аналогично определяется бесконечный интервальный модуль $I_{[j,\infty)}$

$$0 \longrightarrow ... \longrightarrow 0 \longrightarrow \underset{j}{R} \xrightarrow{id_R} \underset{j+1}{R} \xrightarrow{id_R} ... \xrightarrow{id_R} R \xrightarrow{id_R} ...$$

Можно определить прямую сумму персистентных модулей:

$$(A_*, x) \oplus (B_*, x) = ((A_* \oplus B_*), x),$$

где $(A_* \oplus B_*)_j = A_j \oplus B_j$, а x действует на каждом из слагаемых по отдельности.

Теорема (об интервальном разложении)**.** *Пусть R – произвольное поле, а (A_*, x) – конечнопорожденный персистентный R-модуль. Тогда (A_*, x) имеет единственное с точностью до перестановки слагаемых представление в виде прямой суммы конечного числа интервальный модулей:*

$$(A_*, x) \simeq \big(\bigoplus_k I_{[j_k, s_k]}\big) \oplus \big(\bigoplus_k I_{[r_k, \infty)}\big)$$

Так как персистентные гомологии – это типичный пример конечнопорожденного персистентного модуля, то к нему можно применить структурную теорему. Из теоремы следует, что любой конечнопорожденный персистентный модуль с коэффициентами в поле, а значит и персистентные гомологии, можно закодировать в виде баркода (рисунке. 1.7) – диаграммы, которая содержит интервалы, которые отвечают за время жизни свойств, которые как раз и характеризуются персистентными гомологиями.

Эту же информацию можно закодировать в виде диаграммы персистентности(диаграммы устойчивости) (рисунке. 1.8). Она строится следующим образом: каждый интервал баркода имеет начало t_{birth} и конец t_{death}. На персистентной диаграмме каждый интервал баркода изображается в

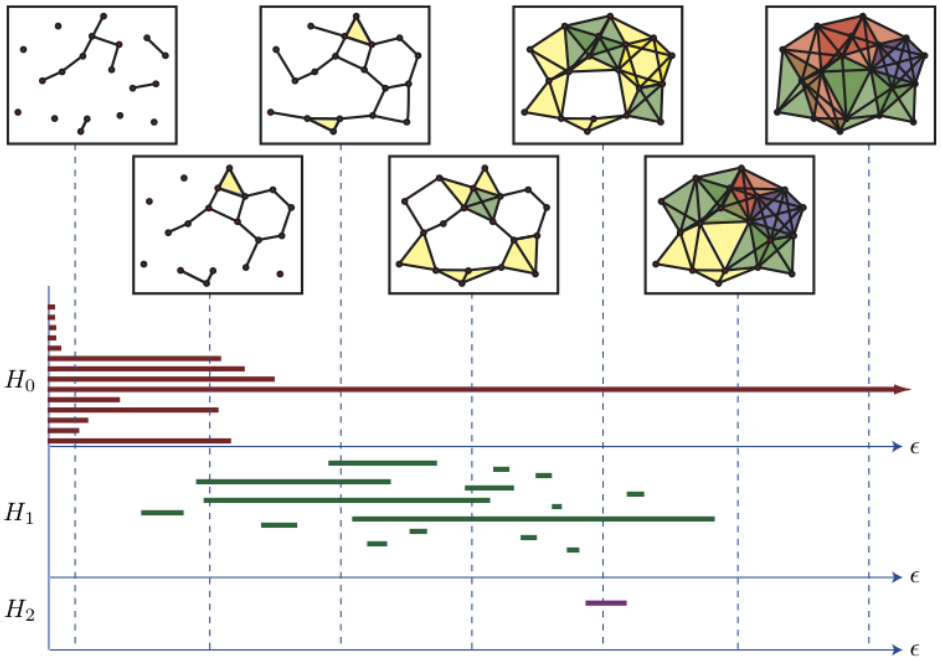


Рис. 1.7: Баркод

виде точки с координатами (t_{birth}, t_{death}), и каждая такая точка соответствует одному из слагаемых $I_{[j_k,s_k]}$ и $I_{[r_k,\infty)}$ из разложения. Таким образом, диаграмма устойчивости B – это множество $\{(x,y) \in \mathbb{R}\times\mathbb{R}\cup\infty | x \leq y\}$. Чем дальше точка на персистентной диаграмме от диагонали, тем она важнее – эта точка сигнализирует о наличии «n-мерного цикла».

1.2.2 Векторные представления

Как было показано выше, диаграмма устойчивости кодирует топологическую информацию. Кажется естественным, что такую топологическую информацию можно использовать во многих задачах анализа данных и машинного обучения. Однако для этого необходимо что-то знать про множество всех устойчивых диаграмм.

Пусть $\mathcal{D}$ – множество устойчивых диаграмм. На нем можно ввести (естественную) метрику:

$$W_p(B, B^{'}) = \inf_{\gamma: B \to B'} \Big(\sum_{u \in B} \|u - \gamma(u)\|_\infty^p \Big)^{\frac{1}{p}},$$

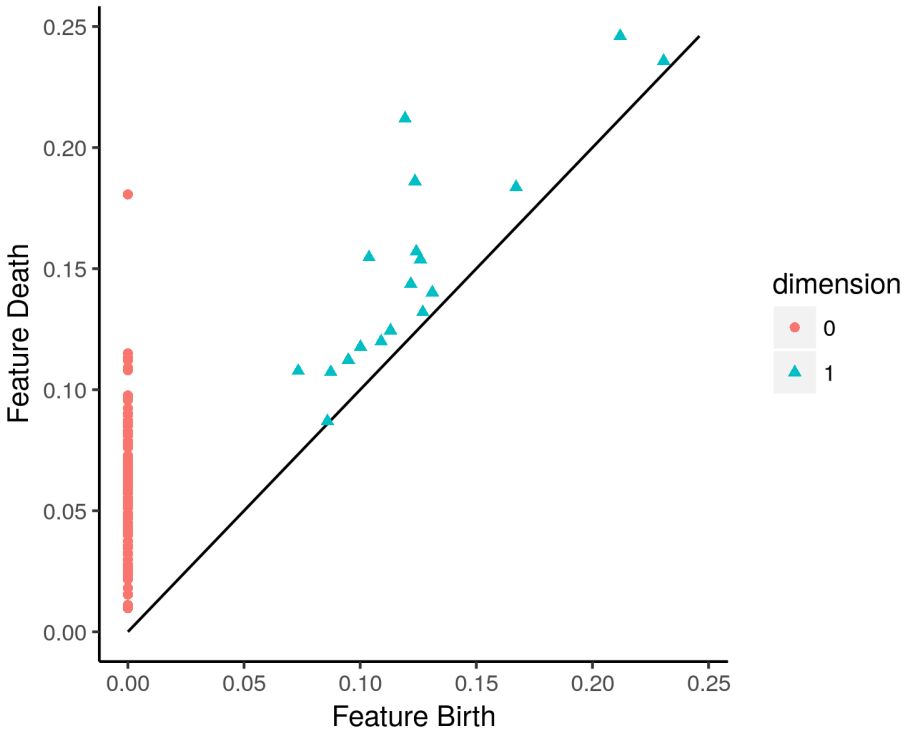


Рис. 1.8: Персистентная диаграмма

где $1 \leq p < \infty$, $B, B^{'}$ – диаграммы персистентности. Такую метрику называют *$p-$метрику Васерштейна*. Другой естественной метрикой является т.н. *bottleneck distance* W_∞:

$$W_\infty(B, B^{'}) = \inf_{\gamma: B \to B^{'}} \sup_{u \in B} \|u - \gamma(u)\|_\infty.$$

Таким образом, $\mathcal{D}$ с любой из указанных выше метрик образует метрическое пространство.

Несложно увидеть, что, к сожалению, оно не является полным: пусть $x_n = (0, 2^{-n}) \in \mathbb{R}^2$ и $B_n = \{x_i\}_{i=1}^n$ – диаграмма устойчивости. Тогда

$$W_p(B_n, B_{n+k}) \leq \frac{1}{2^{n+k}},$$

а значит $\{B_n\}$ фундаментальна. Однако при $n \to \infty$ число недиагональных элементов в B_n уходит в бесконечность, поэтому у последовательности диаграмм нет предела.

Можно немного изменить определение устойчивых диаграмм – пусть теперь у нас всегда она содержит диагональ $\Delta = \{(x,y) \in \mathbb{R}^2 | x = y\}$. Для таких диаграмм p-метрика Васерштейна обобщается естественным способом. Пустую диаграмму, содержащую только диагональ, будем обозначать через $B_\emptyset$. Тогда пространством устойчивых диаграмм можно считать следующее пространство:

$$\mathcal{D} = \{B | W_p(B, B_\emptyset) < \infty\}.$$

Тогда, как показано в [14], такое пространство уже обладает хорошими свойствами, например оно полно и сепарабельно.

Однако для большинства алгоритмов машинного обучения и анализа данных этого недостаточно. Например, для метода опорных векторов необходимо, чтобы пространство признаков было гильбертовым, т.к. такой метод строит разделяющую гиперплоскость; деревья решений строят иерархическую модель «решений», которые, в свою очередь, также определяют гиперплоскость.

Более того, в пространстве $\mathcal{D}$ устойчивых диаграмм, как также было показано в [14], даже такая стандартная статистика, как среднее арифметическое, для диаграмм устойчивости можно по-разному считать и интерпретировать. Например, если одна диаграмма, B_1, содержит 2 точки $B_1 = \{(10, 22), (12, 20)\}$, а вторая диаграмма, B_2, содержит другие две точки $B_2 = \{(10, 22), (12, 20)\}$, то что считать средним? Какую из диаграмм $\{(11, 20), (11, 22)\}$ или $\{(10,21), (12,21)\}$? Аналогичная ситуация обстоит и с дисперсией.

Таким образом, геометрия пространства $\mathcal{D}$ персистентных диаграмм не позволяет комфортно с ними работать. Значит, для того, чтобы использовать диаграммы устойчивости в анализе данных и машинном обучении, нужно каким-то образом преобразовывать исходное пространство $\mathcal{D}$ диа-

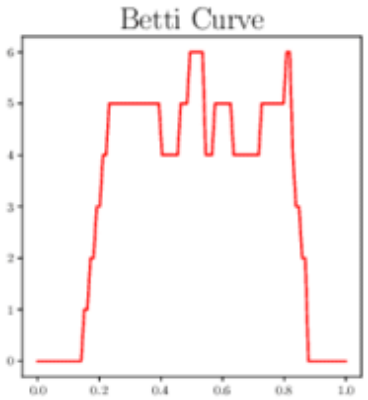


Рис. 1.9: График Betti curve

грамм устойчивости в подходящее, например, гильбертово. Для этого существует несколько подходов. Далее опишем только те, что будут использоваться в данной работе – это betti curves, persistent landscapes и heat kernel.

Итак, пусть $B = \{(b_i, d_i)\}_{i=1}^n$ – диаграмма персистентности. Тогда ее *Betti curve* назовем функцию $\beta_B : \mathbb{R} \to \mathbb{N} : t \mapsto |\{(b_i, d_i) | b_i \leq t < d_i\}|$. Иногда ее называют как persistence indicator function(PIF). Такие функции являются на самом деле ступенчатыми функциями, а потому образуют векторное пространство: суммой таких двух функций $\beta_B + \beta_C := \beta_{B \cup C}$. На рисунке. 1.9 показан график такой функции.

Как было показано выше, в пространстве персистентных диаграмм среднее не единственно. Но у Betti curve такой проблемы нет: для набора $\beta_1, ..., \beta_N$ Betti curves для устойчивых диаграмм $B_1, ..., B_N$ среднее $\bar{\beta}$ можно определить покомпонентно:

$$\bar{\beta}(t) = \frac{1}{N} \sum_{i=1}^{N} \beta_i(t).$$

В [15] показано, что Betti curve устойчива относительно нормы

$$\|\beta\|_1 = \int_{\mathbb{R}} |\beta(t)| dt.$$

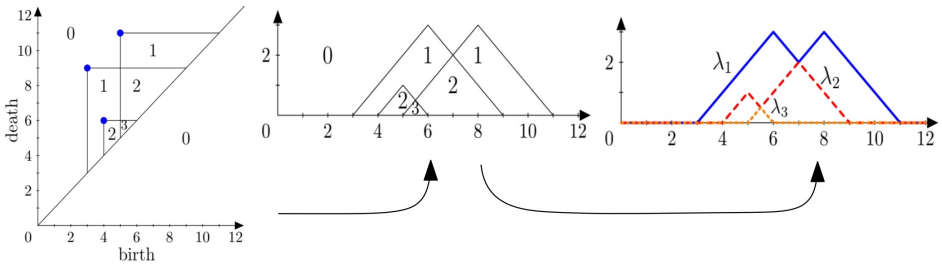


Рис. 1.10: График persistent landscape

Это означает, что небольшие изменения диаграммы приводят к небольшим изменениям Betti curve.

Пусть, все так же, $B = \{(b_i, d_i)\}_{i=1}^n$ – диаграмма персистентности. Тогда для каждой пары (b_i, d_i) можно определить кусочно-линейную функцию $f_{(b_i,d_i)} : \mathbb{R} \to [0, \infty]$:

$$f_{(b_i,d_i)} : x \mapsto \begin{cases} 0, & \text{если } x \notin (b_i, d_i), \\ x - b_i, & \text{если } x \in (b_i, \frac{b_i+d_i}{2}), \\ -x + d, & \text{если } x \in (\frac{b+d}{2}, d). \end{cases}$$

Тогда *persistent landscapes* диаграммы устойчивости B – это последовательность функций $\lambda_k : \mathbb{R} \to [0, \infty]$, где $k = 1, 2, 3, ...$, и где $\lambda_k(x)$ – это k-ое наибольшее значение последовательности $\{f_{(b_i,d_i)}(x)\}_{i=1}^n$, а если такого значения нет, то $\lambda_k = 0$. Эквивалентно, можно определить persistent landscape как функцию $\lambda : \mathbb{N} \times \mathbb{R} \to [0, \infty] : (k,t) \mapsto \lambda_k(t)$. На рисунке. 1.10 представлен пример такого представления.

Также как и Betti curves, для persistent landscapes можно посчитать среднее: пусть $\lambda^1, ..., \lambda^N$ – persistent landscapes для $B_1, ..., B_N$ диаграмм. Тогда их среднее, $\bar{\lambda}$ можно определить покомпонентно:

$$\bar{\lambda}_k(t) \coloneqq \frac{1}{N} \sum_{i=1}^{N} \lambda_k^i(t).$$

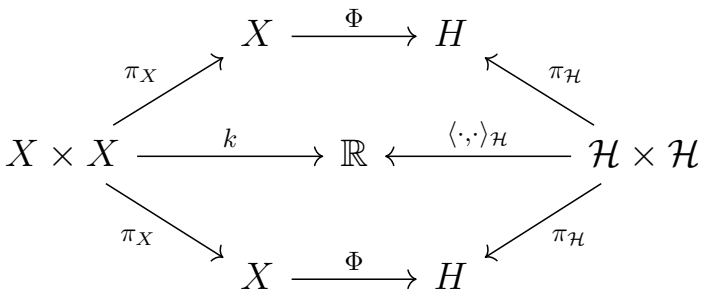
Между persistent landscapes можно посчитать расстояние, например,

используя L^p норму:

$$\left\|\lambda - \lambda^{'}\right\|_p = \left(\sum_{k=1}^{\infty} \int \left|\lambda_k(t) - \lambda_k^{'}(t)\right|^p dt\right)^{\frac{1}{p}}.$$

В [16] показано, что persistent landscape устойчива относительно метрики L^p для $1 \leq p \leq \infty$. Это означает, что небольшие изменения диаграммы приводят к небольшим изменениям persistent landscape.

Также рассматриваем $B = \{(b_i, d_i)\}_{i=1}^n$ – диаграмму персистентности. Определим для нее heat kernel. Напомним, что если X – некоторое множество, то функция $k : X \times X \to \mathbb{R}$ является *ядром(kernel)*, если существует Гильбертово пространство $\mathcal{H}$ и отображение(т.н. feature map) $\Phi : X \to \mathcal{H}$, такое, что $\forall x, y$ имеет место $k(x,y) = \langle \Phi(x), \Phi(y) \rangle_{\mathcal{H}}$, т.е. следующая диаграмма коммутативна:



Заметим, что диаграмма устойчивости B можно единственным образом сопоставить сумму распределений дельта-функций Дирака в каждой точке диаграммы. Такой взгляд на диаграмму устойчивости позволяет каноническим образом получить структуру Гильбертова пространства. Тогда для данной диаграммы устойчивости B можно рассмотреть следующее уравнение теплопроводности с граничным условием Дирихле:

$$\begin{aligned}
&\Delta_x u = \partial_t u, && (x,t) \in \mathrm{Int}(\Omega \times \mathbb{R}_{\geq 0}),\\
&u = 0, && (x,t) \in \partial\Omega \times \mathbb{R}_{\geq 0}),\\
&u = \sum_{x \in B} \delta_x, && (x,t) \in \Omega \times \{0\},
\end{aligned}$$

где δ_x – дельта-функция Дирака в точке x, а $\Omega = \{(x_1, x_2) | x_1 \leq x_2\} \subset \mathbb{R}^2$. Тогда решение данного уравнения – функция в $L_2(\Omega)$, а значит можно рассмотреть отображение $F_\sigma : \mathcal{D} \to L_2(\Omega) : B \mapsto u\big|_{t=\sigma}$. Заметим, что F_σ инъективно. Таким образом, можно определить ядро

$$k_\sigma : \mathcal{D} \times \mathcal{D} \to \mathbb{R} : (B, C) \mapsto \langle \Phi_\sigma(B), \Phi_\sigma(C) \rangle_{L_2(\Omega)}.$$

В [17] приведена мотивация построения именно такого ядра. также в [17] показано, что k_σ устойчива относительно $1-$ метрики Васерштейна.

Все описанные выше методы сопоставляют диаграмме устойчивости некоторую функцию в Банаховом пространстве. Эти методы являются примерами векторизации. *Векторизацией* на множестве X называют отображение $\varphi : X \to V$, где V – векторное пространство. Имея такую векторизацию, можно определить *амплитуду*: отображение $A : X \to \mathbb{R} : x \mapsto \|\varphi(x)\|$, где $\varphi : X \to V$ – векторизация множества X на нормированное векторное пространство. Таким образом, для каждой диаграммы, выбрав определенную векторизацию, можно получить ее амплитуду, т.е. каждой диаграмме сопоставить некоторое вещественное число. Другой способ это сделать – это посчитать т.н. *персистентную энтропию* – это мера энтропии по Шеннону точек на диаграмме персистентности:

$$E(B) = -\sum_{i \in I} p_i \log(p_i), \text{ где } p_i = \frac{d_i - b_i}{\sum\limits_{i \in I} d_i - b_i}.$$

Более подробно с персистентной энтропией можно ознакомиться в [18]. Тогда, получив такие значения, из них можно сделать вектор в $\mathbb{R}^d$, который будет соответствовать диаграмме устойчивости. Такой вектор уже можно будет использовать в задачах анализа данных и машинного обучения.

1.3 Элементы машинного обучения

Здесь будут приведены некоторые теоретические сведения из анализа данных и машинного обучения. За более подробным материалом можно обратиться к [19—21].

1.3.1 Введение в машинное обучение

Машинное обучение – раздел, стоящий на пересечении математики и компьютерных наук, который, главным образом, решает задачу восстановления некоторой зависимости, используя некоторые данные, которые представляют такую зависимость.

Машинное обучение можно разделить на категории: например, очень популярное разделение – это разделение по типу обучения:

- обучение с учителем(Supervised Learning);
- обучение без учителя(Unsupervised Learning);
- обучение с подкреплением(Reinforcement Learning).

Остановимся более подробно на обучении с учителем. Данный вид машинного обучения предполагает, что имеется некоторая выборка X – *множество объектов*, некоторая выборка Y – *множество ответов*, и необходимо восстановить *скрытую зависимость* $y : X \to Y$. Как правило, саму зависимость восстановить порой очень трудно, поэтому достаточно найти такой алгоритм $a : X \to Y$, который приближает y на X. При этом сами объекты задаются через *признаки* $f : X \to D$, где D – *пространство признаков*. Признаки – это какие-либо характеристики, которые описывают данный объект.

Основные задачи обучения с учителем:

- регрессия;

- классификация.

Фундаментальная разница между этими двумя типами задач следующая: задача *регрессии* – восстановление непрерывной функции, которая характеризует скрытую зависимость между входными переменными и выходными. Регрессия таким имеет возможность предсказывать численное значение этой скрытой непрерывной функции по новым значениям входных переменных. Задача *классификации* же в том, чтобы восстановить зависимость между входными параметрами, описывающие объект, и тем, к какому классу данный объект принадлежит. То есть в случае задачи классификации скрытая зависимость описывается некоторой пороговой функцией, где каждое значение области значения соответствует некоторому классу.

Важным шагом в решении задачи как регрессии, так и классификации, является выбор модели и настройка ее параметров. *Модель* – это параметрическое семейство функций

$$A \coloneqq \{a(x) = g(x, \theta) | \theta \in \Theta\},$$

где Θ – множество значений параметра θ, а $g : X \times \Theta \to Y$ – некоторая фиксированная функция.

Решение задачи машинного обучения разделяется на два этапа:

1. этап обучения;

2. этап применения.

Этап обучения представляет из себя построение алгоритма выбора подходящей модели, т.е. имеется отображение $\mu : X \times Y \to A$, которое по каждой выборке (X,Y), которую обычно называют *обучающей выборкой*, ставит в соответствие алгоритм $a \coloneqq \mu(X, Y)$, который наилучшим образом описывает скрытую зависимость $y : X \to Y$. Для выбора модели необходимо сравнивать точность, которую они демонстрируют. Для этого обычно

вводят т.н. *функцию потерь (loss function)* $L(a,x)$, которая измеряет величину ошибки алгоритма $a \in A$ на объекте $x \in X$. Например, для задачи классификации рассматривают *индикатор ошибки (Accuracy)*:

$$L(a,y,x) = [a(x) \neq y(x)].$$

Для задач регрессии в качестве функции потерь рассматривают *квадратичную ошибку*

$$L(a,y,x) = (a(x) - y(x))^2$$

или *абсолютное значение ошибки*

$$L(a,y,x) = |a(x) - y(x)|.$$

Чтобы оценить качество работы алгоритма a на всей выборке X обычно считают среднее арифметическое по значениям функции потерь для каждого объекта $x \in X$(иногда его называют *эмпирическим риском*):

$$Q(a,X,Y) = \frac{1}{|X|} \sum_{i=1}^{|X|} L(a,y_i,x_i).$$

И тогда для выбора модели решают задачу минимизации эмпирического риска:

$$\mu(X,Y) = \arg\min_{a \in A} Q(a,X,Y).$$

Этап применения – это следующий этап, который представляет собой т.н. задачу вывода. На этом этапе выбранный алгоритм a для новых объектов X^*, которые не были задействованы в этапе обучения, выдает ответы $a(X^*)$. Обычно такую выборку X^* называют *тестовой выборкой*. Одна из основных проблем, которая возникает на данном этапе – это проблема *переобучения*. Переобучение – это ситуация, которая возникает в следствие

допущенной ошибки в ходе этапа обучения, и характеризуется тем, что величина ошибки на тестовой выборке сильно превышает величину ошибки на обучающей:

$$Q(\mu(X,Y),X^*,Y^*) >> Q(\mu(X,Y),X,Y).$$

Зачастую такую проблему можно исправить, если правильно подобрать параметры модели, которая использовалась для обучения.

Так как данная работа представляет собой решение задачи классификации, то остановимся поподробнее на соответствующих моделях машинного обучения.

1.3.2 Логистическая регрессия

Логистическая регрессия – один из самых простых и популярных алгоритмов для задачи классификации. Смысл данного метода заключается в том, чтобы построить разделяющую гиперплоскость в пространстве признаков, позволяющую отделить классы друг от друга.

В логистической регрессии строится линейный алгоритм классификации $a : X \to Y$, где X – пространство признаков, а Y – конечное множество номеров классов. Алгоритм имеет вид:

$$a(x,w) = sign(\sum_{j=1}^{n} w_j f_j(x) - w_0) = sign\langle x,w\rangle,$$

где w – вектор весов, w_0 – порог принятия решения, а $\langle\cdot,\cdot\rangle$ – скалярное произведение.

Задача обучения линейного классификатора заключается в том, чтобы по выборке настроить вектор весов. Для этого в логистической регрессии решается задача минимизации эмпирического риска с специальной функцией

потерь вида:

$$Q(w) = \sum_{i=1}^{m} ln(1 + e^{-y_i\langle x_i,w\rangle}) \to \min_w$$

После нахождения решения w, становится возможным не только вычислять классификацию для произвольного объекта, но и оценивать апостериорные вероятности его принадлежности классам:

$$\mathbb{P}(y|x) = \sigma(y\langle x,w\rangle)$$

где $\sigma(z) = \frac{1}{1+e^{-z}}$ – сигмоидная функция.

1.3.3 Метод опорных векторов

Метод опорных векторов – также очень популярный метод машинного обучения, достаточно мощный и многогранный, применяемый в задачах классификации и регрессии.

Фундаментальная идея метода опорных векторов заключается в поиске гиперплоскости с наилучшим отступом – расстоянием между гиперплоскостью и опорными векторами – векторами, которые ближе всего находятся к разделяющей гиперплоскости.

Ищется решение задачи регрессии в линейном случае:

$$f(x) = \langle w,x\rangle - w_0.$$

Функция потерь принимает вид:

$$a(x_i) = |\langle w,x_i\rangle - w_0 - y_i|_\varepsilon$$

для каждого вектора (x_i,y_i).

В таком случае функционал потерь принимает вид:

$$Q_\varepsilon(a,X) = \sum_{i=1}^{l} |\langle w,x_i\rangle - w_0 - y_i|_\varepsilon + \tau\langle w,w\rangle^2 \to \min_{w,w_0}.$$

Последнее слагаемое удерживает коэффициенты w от бесконечного возрастания. Аналогично задаче классификации, решение зависит от скалярного произведения объектов, а не от самих объектов. Минимизация в данном случае эквивалентна задаче квадратичного программирования с ограничениями типа неравенств.

1.3.4 Случайный лес

Случайный лес – это алгоритм машинного обучения, заключающийся в использовании нескольких решающих деревьев. На самом деле, случайный лес – это частный случай бустинга.

Беггинг (bootstrap aggregation) – это метод композиции базовых классификаторов b_t, которые обучаются независимо по случайной выборке длины l с повторениями. Обычно предполагается, что базовые классификаторы хотя бы несколько лучше случайного угадывания и различны между собой. Тогда из данных базовых классификаторов получают один агрегированный, который работает лучше. В случае случайного леса базовые классификаторы – это решающие деревья без усечения(прунинга). Помимо генерирования подвыборки для обучения, генерируют случайное подмножество $F_t \subseteq F$ признаков, на которых обучают базовые классификаторы. Т.е., таким образом, если $\mu(F_t, U_t)$ – метод обучения алгоритма по подвыборке $U_t \subseteq X$ на $F_t \subseteq F$ признаках, то $b_t := \mu(F_t, U_t)$.

Для подбора числа деревьев T применяют критерий *out-of-bag*:

$$\text{out-of-bag}(a) = \sum_{i=1}^{l} \big[\text{sgn}\big(\sum_{t=1}^{T}[x_i \notin U_t]b_t(x_i)\big) \neq y_i\big] \to \min\,.$$

Т.е. при таком критерии подсчитывается число ошибок полученного классификатора на объектах обучающей выборки x_i, при этом не учитываются голоса тех деревьев, которые непосредственно обучались на объекте x_i. Такой критерий является несмещенной оценкой обобщающей способности.

1.3.5 Градиентный бустинг

Градиентный бустинг — это метод машинного обучения, который, как и случайный лес, использует несколько базовых классификаторов. В случае градиентного бустинга используется *линейная комбинация* b базовых алгоритмов b_t:

$$b(x) = C\big(\sum_{t=1}^{T}\alpha_t b_t(x)\big),$$

где $C : \mathbb{R} \to Y$ – решающее правило, $a_t \geq 0$. Например, в задачах регрессии $C = id$, а в задачах бинарной классификации $C = sgn$.

Далее происходит оптимизация функционала качества $Q(\alpha, b)$ с некоторой функцией потерь $L(b,y)$

$$Q(\alpha, b) = \sum_{i=1}^{l} L\big(\sum_{t=1}^{T-1}\alpha_t b_t(x_i) + \alpha b(x_i), y_i\big) \to \min_{\alpha,b}\,.$$

Если $u_{T-1} \coloneqq \sum\limits_{t=1}^{T-1}\alpha_t b_t(x) = (u_{T-1,i})$, а $u_T \coloneqq \sum\limits_{t=1}^{T-1}\alpha_t b_t(x) + \alpha b(x_i) = (u_{T,i})$. Тогда функционал имеет вид:

$$Q(\alpha, b) = \sum_{i=1}^{l} L(u_{T,i}, y_i) \to \min_{\alpha,b}\,.$$

Такую задачу оптимизации можно решить градиентным спуском:

$$u_T \coloneqq u_{T-1} - \alpha g,$$

где $g = L^{'}(u_{T-1}, y)$. Это можно истолковать как добавление нового базового алгоритма b_t. Так как в общем случае $b_t \neq g$, то ищется такой b_t, чтобы вектор $b_t(x)$ приближал антиградиент $-g$:

$$b_t \coloneqq \arg\min_b \sum_{i=1}^{l} (b(x_i) - g_i)^2.$$

Это происходит до тех пор, пока значения функционала не сойдутся, либо пока не выполнится одно из правил ранней остановки.

2 Классификация изображений датасета MNIST

2.1 Сравнительный анализ пакетов для вычисления устойчивых гомологий

В настоящее время существует ряд пакетов для вычисления устойчивых гомологий. Поэтому прежде чем приступать к задаче классификации, необходимо провести анализ и выбрать какой-то определенный пакет. Сравнительный анализ будет проводиться для пакетов с интерфейсом на Python, а конкретно, Dionysus [22], Giotto-TDA [23], GUDHI [24], Ripser [25], которая является частью более обширной библиотеки Scikit-TDA [26] для топологического анализа данных.

Пакеты Dionysus и GUDHI существуют уже длительное время и поэтому более надежны в работе, когда как Scikit-TDA и Giotto-TDA являются более новыми, но быстро развивающимися библиотеками. Сравнительный анализ этих пакетов представлен в табл. 2.1. Следующие алгоритмы используются для вычисления устойчивых гомологий:

- стандартный алгоритм [2], заключающийся в приведении граничной матрицы фильтрации к ступенчатому виду;
- twist algorithm [27], заключающийся в оптимизации стандартного алгоритма путем уменьшения размерности граничной матрицы фильтрации;
- dual algorithm, заключающийся в вычислении устойчивых когомологий, что вычислительно более эффективно;

- multifield algorithm [28], заключающийся в использовании других полей коэффециентов для более эффективного вычисления;
- zigzag algorithm [29], также основанный на персистентных когомологиях, которые вычисляются более эффективно.

Таблица 2.1: Сравнительный анализ возможностей пакетов

	Giotto-TDA	GUDHI	Ripser	Dionysus
Алгоритмы для вычисления устойчивых гомологий	standard, twist, dual	standard, dual, multifield	standard, twist, dual	standard, dual, zigzag
Коэффициенты	$\mathbb{F}_p$	$\mathbb{F}_p$	$\mathbb{F}_p$	$\mathbb{F}_p$ (dual); $\mathbb{F}_2$ (standard, zigzag)
Гомологии	Симплициальные, кубические	Симплициальные, кубические	Симплициальные	Симплициальные
Фильтрации	Виеторис-Рипс, Чех, α	α, Виеторис-Рипс, Чех, кубические, нерв, Witness, Делоне	Виеторис-Рипс	Виеторис-Рипс, α, Чех
Визуализация	Диаграммы устойчивости,	Диаграммы устойчивости, баркоды	Диаграмма устойчивости	Диаграммы устойчивости, баркоды

Из таблицы видно, что библиотека GUDHI представляет максимально различные способы построения фильтраций, а также различные алгоритмы для вычисления устойчивых гомологий. Эта библиотека предоставляет также возможность вычислять кубические гомологии.

Проведем анализ пакетов на синтетических тестах. Для этого воспользуемся пакетов Tadasets [30], который, как и Ripser, является частью библиотеки Scikit-TDA [26] и предоставляет набор синтетических датасетов, полезных для топологического анализа данных. Рассматриваемые синтетические датасеты – это не зашумленные облака точек в $\mathbb{R}^{26}$; количество точек $n = 2000$. Данные описывают следующие многообразия:

- тор (расстояние от центра тора до центра ручки $c = 2$, радиус внутренней окружности $a = 1$);

- «швейцарский рулет» (длина рулета $r = 4$);
- 2-Сфера (радиус $r = 3.14$);
- 3-Сфера (радиус $r = 3.14$);
- букет 2 окружностей («знак бесконечности»).

Во всех пакетах использовался стандартный алгоритм вычисления персистентных гомологий. Целью данного сравнения было нахождение пакета, который быстрее других вычисляет устойчивые гомологии, и для которого потребуются минимальные настройки. Все вычисления производились в среде Google Colab [31].

Таблица 2.2: Тест скорости работы пакетов

	Wall-time sec.				
	Тор	«Швейцарский рулет»	2-Сфера	3-Сфера	«Знак бесконечности»
Ripser	26.3	392	6.84	13.5	50.4
Gudhi	2.2	2.21	2.27	2.27	1.51
Giotto-TDA	14	264	4.14	7.04	29.1
	CPU time sec.				
	Тор	«Швейцарский рулет»	2-Сфера	3-Сфера	«Знак бесконечности»
Ripser	22.4	380	6.2	11.7	50.4
Gudhi	2.12	2.15	2.21	2.23	1.45
Giotto-TDA	13.8	263	4.07	6.72	28.9

На основе данных сравнений можно сделать вывод, что библиотека GUDHI предоставляет наиболее эффективный способ вычисления устойчивых гомологий. В свою очередь, библиотека Dionysus тратила очень много времени на построение фильтрации, поэтому в вычислительном эксперименте ее результатов нет.

Таким образом, библиотека GUDHI представляет наибольшее разнообразие методов построения фильтраций, алгоритмов вычисления устойчивых гомологий, а также является эффективной с вычислительной точки

зрения. Однако, в дальнейшем будет использоваться Giotto-TDA, так как, несмотря на то, что в ряде тестов эта библиотека оказалась не самой эффективной с вычислительной точки зрения (например, на датасете «Швейцарский рулет» Giotto-TDA показала себя достаточно неэффективно, хотя все еще лучше, чем Ripser), эта библиотека предоставляет наиболее удобный интерфейс для использования, так как сам пакет сделан на основе пакета Scikit-Learn, одного из основных пакетов для машинного обучения на Python, а потому все методы имеют похожую структуру, какую имеют методы из пакета Scikit-Learn и других пакетов для научных вычислений на Python (например, NumPy).

2.2 Классификация изображений датасета MNIST

Теперь решим задачу классификации датасета MNIST. MNIST – это датасет, который состоит из 60000 изображений для обучения, а также 10000 изображений для тестирования, где каждое изображение является черно-белым изображением рукописной цифры от 0 до 9. Типичное изображение из данного датасета представлено на рисунке. 2.1. Данный датасет считается одним из стандартных датасетов для обучения различных методов распознавания изображений с помощью машинного обучения, причем в первую очередь для методов, основанных на нейронных сетях. Однако, в следствие нейросетевого бума, который случился в последнее десятилетие, а в частности с появлением и развитием сверточных нейронных сетей, данный датасет слегка утратил свою актуальность – современные нейросети выдают почти 100-процентную точность, и для обучения нейросетей сейчас используют другие датасеты.

Однако при поиске новых подходов к классификации изображений все еще используют датасет MNIST. Поэтому в данной работе будет решена задачи классификации изображений именно с использованием данного

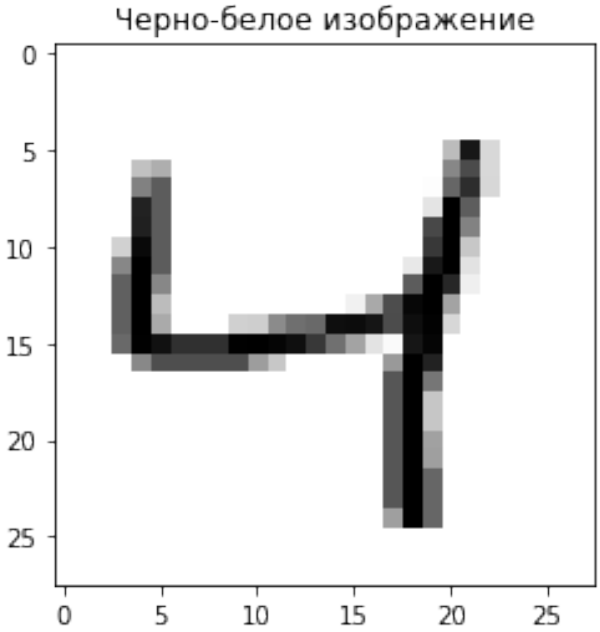


Рис. 2.1: Пример изображения из датасета MNIST

датасета: топологический анализ данных, и в частности устойчивые гомологии – это новые инструменты для решения задачи классификации, в этой области все больше выходят современных исследований.

Опишем общий алгоритм 2, который лежит в основе решения задачи.

Алгоритм 2: Общий алгоритм решения задачи классификации

Исходные параметры: Набор изображений рукописных цифр

Результат: Алгоритм, определяющий рукописную цифру

для каждого *изображения из выборки* **выполнять**

Построить фильтрации;

Найти персистентные гомологии, построить диаграмму персистентости;

Получить топологические признаки из диаграммы персистентности;

конец

На наборе полученных признаков обучить модель машинного обучения;

Как видно из алгоритма, основным этапом является получение топологических признаков. На их основе и будет обучаться модель машинного обучения. Признаки получаются из диаграммы устойчивости различными методами, поэтому для одного изображения можно получить сразу несколько признаков по одной диаграмме. С другой стороны, персистентные гомологии, а значит и диаграмму устойчивости, можно считать для

различных фильтраций одного и того же изображения. Таким образом на основе одного изображения можно сразу получить большое количество признаков, строя по ней различные фильтрации и различными способами векторизуя диаграмму. В данной работе использовался подход, изображенный на рисунке 2.2.

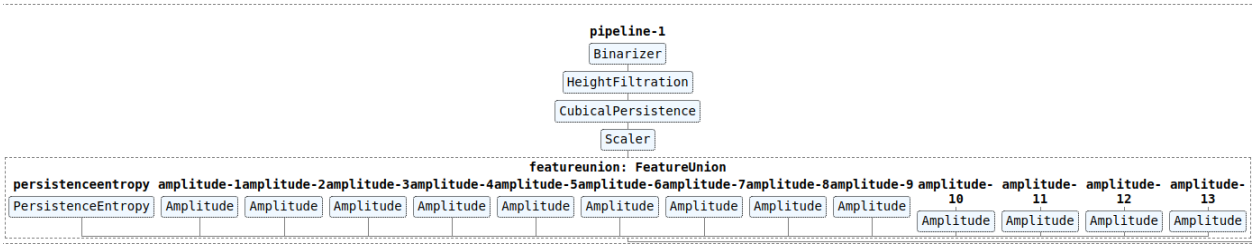


Рис. 2.2: Алгоритм получения признаков для Height Filtration

Опишем подробнее алгоритм получения признаков. Первым шагом является бинаризация изображения с заранее выбранным пороговым значением (в данной работе значение порога равнялось 0.4). Это нужно для того, чтобы далее воспользоваться специальными методами фильтрации, которые работают именно с бинарным изображением. Бинарным изображением будет называть отображение $B : \mathbb{R}^d \to \{0, 1\}$.

Далее по бинарному изображению B строятся специальные фильтрации. По большому счету фильтрации можно воспринимать как трансформации бинарного изображения обратно в черно-белое, пропущенное через специальный фильтр. Так как в результате получается черно-белое изображение, то можно считать устойчивые гомологии сразу для самой картинки, но обычно построение черно-белого изображения через бинаризацию и фильтрацию наиболее сильно подчеркивает различные топологические особенности. В данном алгоритме использовалась т.н. фильтрация по высоте, радиальная фильтрация, а также фильтрация по плотности. Соответствующие фильтрации изображены на рисунке. 2.3.

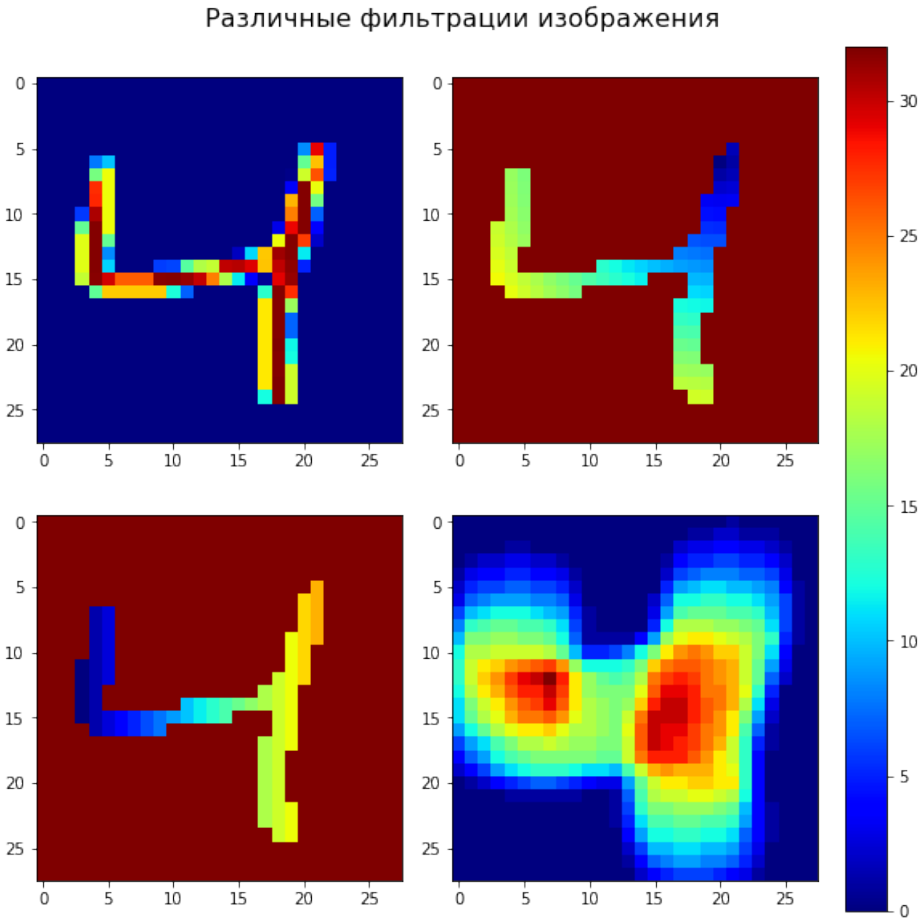


Рис. 2.3: Черно-белые изображения, полученные с помощью различных фильтраций. Для наглядности была использована цветная гамма для представления черно-белых значений. В верхнем левом углу – оригинальное изображение; в верхней правой – радиальная фильтрация. В нижнем левой углу – фильтрация по высоте; в нижнем правом углу – фильтрация по плотности.

Фильтрация по высоте (Height filtration) – это отображение

$$H : I \to \mathbb{R},$$

где I – это бинарное изображение в общем случае размерности d (в нашем случае размерность равна 2), которое каждому пикселю изображения p сопоставляет расстояние до гиперплоскости, которая определена вектором v длины 1, заданным заранее:

$$H : p \mapsto \begin{cases} \langle p,v \rangle, & \text{если } B(p) = 1, \\ H_\infty, & \text{иначе,} \end{cases}$$

где H_∞ – это значение фильтрации H самого дальнего пикселя изображения до гиперплоскости.

Радиальная фильтрация (Radial filtration) – это отображение

$$R : I \to \mathbb{R},$$

которое каждому пикселю изображения сопоставляет расстояние до выбранного заранее центра c:

$$R : p \mapsto \begin{cases} \|c - p\|_2, & \text{если } B(p) = 1, \\ R_\infty, & \text{иначе,} \end{cases}$$

где R_∞ – это расстояние самого дальнего пикселя до центра. То есть фильтрация строится исходя из значения некоторой радиальной функции от пикселя p, у которого $B(p) = 1$.

Фильтрация по плотности (Density filtration) – это отображение $D : I \to \mathbb{R}$, которое каждому пикселю изображения сопоставляет число пикселей p, таких, что $B(p) = 1$, находящихся на растоянии не больше

задаваемого r:

$$D_r(p) := |\{v \in I | B(v) = 1 \wedge \|p - v\| \leq r\}|,$$

где $\|\cdot\|$ – любая норма в $\mathbb{R}^2$. В данном алгоритме использовалась стандартная евклидова метрика.

Так как вектор направления для фильтрации по высоте и центр для радиальной фильтрации задается заранее, то можно формировать различные фильтрации при различных значениях вектора направления и центра. В данном алгоритме фильтрация по высоте строилась с 8 различными значениями для вектора направления, радиальная фильтрация строилась с 9 различными значениями для центра, а фильтрация по плотности строилась с 3 различными значениями для радиуса. Таким образом, для одного изображения получалось 20 фильтрации.

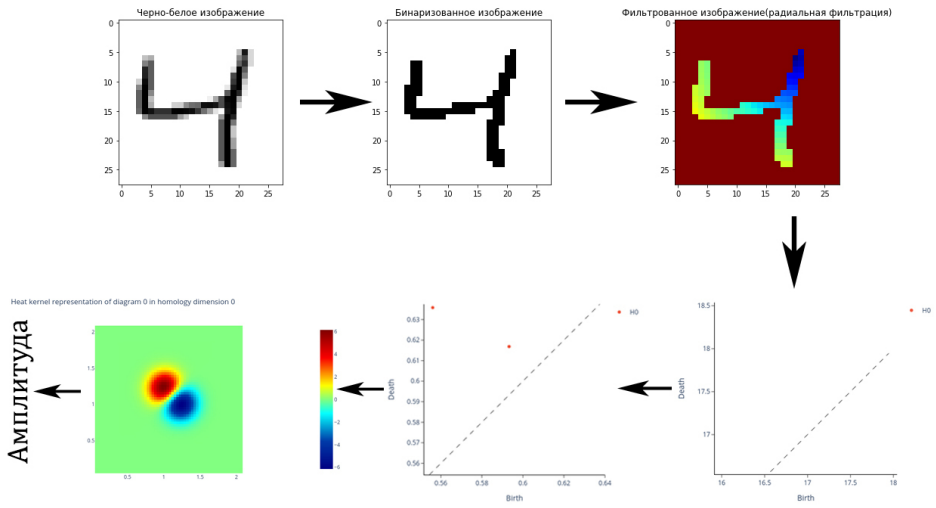
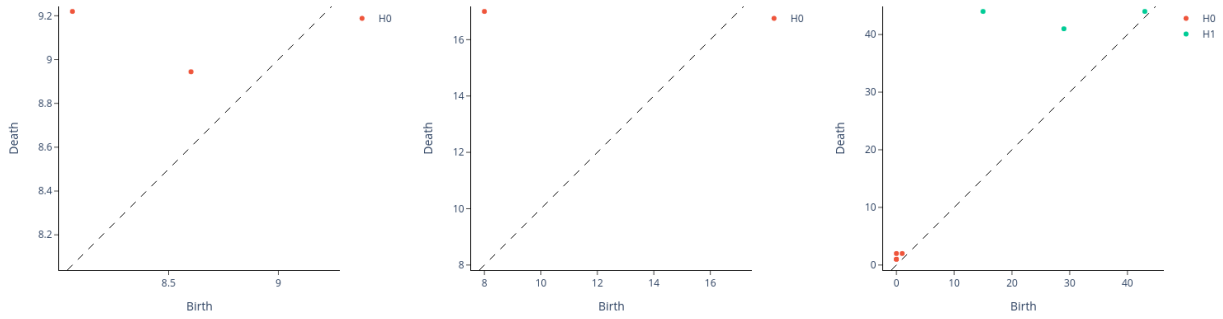


Рис. 2.4: Пример работы алгоритма

Далее для каждой из полученных фильтраций вычислялись устойчивые гомологии 0 и 1 размерности, а по ним строились диаграммы персистентности, которые далее были приведены к одному и тому же масштабу. Посчитанные гомологии являются кубическими, а не симплициальны-

ми, так как такие фильтрации задают семейство кубических комплексов – полных аналогов симплициальных. Разница между ними в том, что симплициальный комплекс – это набор правильно склеенных симплексов, а кубический комплекс – это набор правильно склееных кубов, т.е. подмножеств $I_{a_1} \times ... \times I_{a_N} \subset \mathbb{R}^N$, где $I_a \subset \mathbb{R}$ – это либо интервал $[a, a+1]$, либо $\{a\}$. Кубические гомологии определяются точно так же, как и симплициальные. Стоит заметить, что при работе с картинками именно такой подход являются главным способом вычисления устойчивых гомологий.

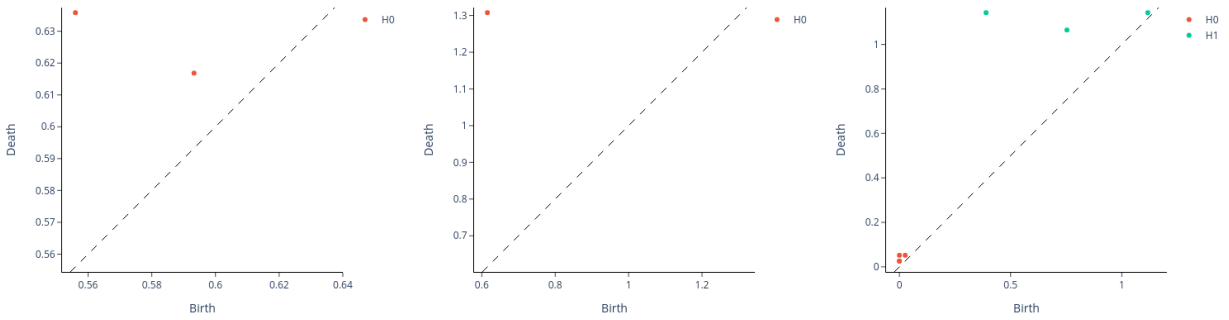


(a) Radial Filtration (b) Height Filtration (c) Density Filtration

Рис. 2.5: Диаграммы устойчивости для различных фильтраций

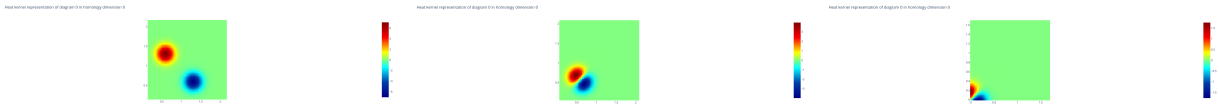
На рисунке. 2.4 представлен пример работы алгоритма для одного бинарного изображения из датасета. На рисунке. 2.5 представлены диаграммы устойчивости для различных фильтраций. На рисунке. 2.6 представлены диаграммы устойчивости, приведенные к одному и тому же масштабу. Видно, что изменились значения шкал рождения/смерти у диаграмм.

В свою же очередь, для каждой диаграммы вычислялись 14 признаков: персистентная энтропия и 13 амплитуд для различных векторных представлений. На рисунке. 2.7 представлен пример таких представлений для различных фильтраций. Таким образом, для одной картинки получалось $20 \times 2 \times 14 = 560$ признаков. Однако вероятно, что некоторые из таких признаков будут коррелировать, а поэтому необходимо будет произвести отбор признаков.



(a) Radial Filtration (b) Height Filtration (c) Density Filtration

Рис. 2.6: Диаграммы устойчивости для различных фильтраций, приведенные к одному масштабу



(a) Height filtration (b) Radial filtration (c) Density filtration

Рис. 2.7: Heat Kernel для различных фильтраций для одного и того же изображения

После получения векторного представления для изображений из выборки, дальнейшим этапом является обучение модели машинного обучения. В данной работе использовались следующие модели:

- метод опорных векторов;
- случайный лес;
- логистическая регрессия;
- LightGBM;
- CatBoost;
- XGBoost.

Обучение моделей проводилось не на всем датасете, а лишь на очень малой его части: в тренировочной выборке находилось 700 изображений, а в тестовой – 300.

Первым шагом было запустить эти модели без настраивания параметров и отбора признаков, и посмотреть, как хорошо они справляются с задачей, т.е. таким образом получить начальное значение для каждой из модели, чтобы в дальнейшем значения качества моделей увеличивать как за счет настраивания параметров модели, так и за счет отбора признаков. Получившиеся результаты отображены в табл. 2.3.

Таблица 2.3: Значения базовых моделей на тренировочной и тестовой выборках

Название модели	Значение на тренировочной выборке	Значение на тестовой выборке
Логистическая регрессия	0.989	0.903
Метод опорных векторов	1.0	0.893
Случайный лес	1.0	0.89
LightGBM	1.0	0.9
XGBoost	1.0	0.883
CatBoost	1.0	0.893

Несмотря на очевидное переобучение, значения на тренировочной выборке куда лучше, чем на тестовой – все модели проявили себя очень хорошо и показали неплохие результаты (тут стоит отметить, что современные сверточные нейронные сети достигают 99.95% точности). Попробуем для каждой модели сделать поиск по сетке, чтобы улучшить результаты моделей. Результат поиска по сетке представлен в табл. 2.4. Можно заметить, что для некоторых моделей проблема переобучения исчезла в результате подбора параметров.

Таблица 2.4: Значения наилучших моделей, полученных в результате подбора параметров поиском по сетке, на тренировочной и тестовой выборках

Название модели	Значение на тренировочной выборке	Значение на тестовой выборке
Логистическая регрессия	0.911	0.917
Метод опорных векторов	0.903	0.893
Случайный лес	0.88	0.87
LightGBM	0.908	0.917
XGBoost	0.907	0.897
CatBoost	1.0	0.927

Как было сказано ранее, скорее всего многие признаки коррелируют между собой. Поэтому дальнейший шаг – это отбор признаков. Сперва убедимся, что уменьшение признаков может привести к росту качества модели. Для этого запустим случайный лес на выборке. Случайный лес может быть использован в качестве отбора признаков – есть возможность подсчитать важность признаков для этой модели. Те признаки, которые оказались наиболее важны для случайного леса, вероятно будут и наиболее важными и для других моделей. Поэтому с помощью случайного леса упорядочим все признаки по убыванию их важности. Далее в цикле будем запускать модели на выборке, но с разным числом признаков – так как они упорядочены, то будем брать первые $50n$ признаков, где $1 \leq n \leq 11$. Запускать модели будем с поиском по сетке, чтобы использовать наилучшие параметры для данной выборке. График зависимости качества построенных моделей от числа признаков представлен на рисунке 2.8.

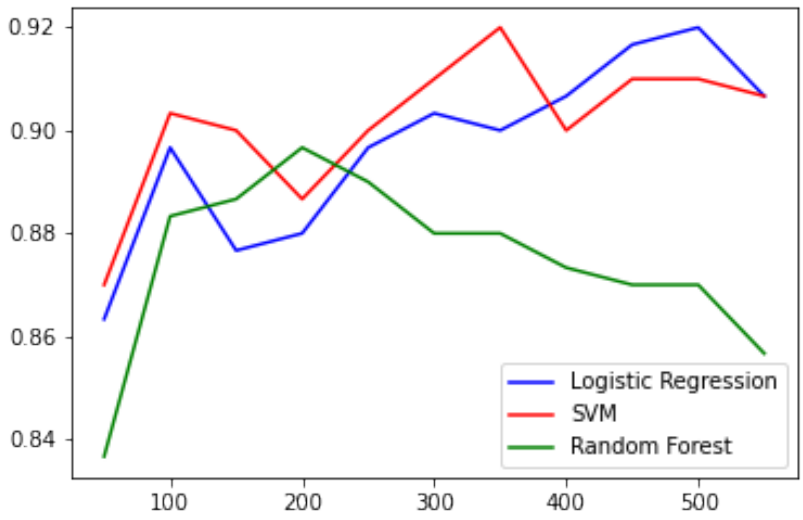
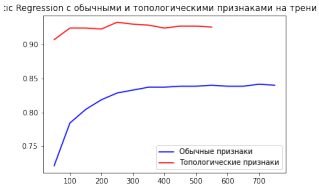


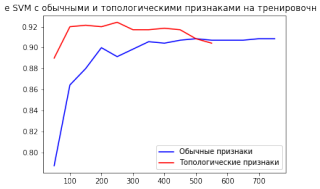
Рис. 2.8: График зависимости качества построенных моделей от числа признаков

На графике видно, что, уменьшив количество признаков, все модели улучшают свои показатели качества.

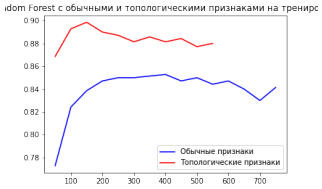
Далее можно сравнить качество моделей, которые используют топологические признаки, с качеством моделей, которые в качестве признаков будут получать просто векторное представление изображения. То есть для таких моделей признаками будут сами значения пикселей изображения. Так как модели с топологическими признаками показали наилучший результат именно при уменьшении количества признаков, то также отсортируем признаки для новых моделей по убыванию важности для случайного леса, и будем брать первые $50n$ признаков, каждый раз обучая модель с поиском по сетке, чтобы подобрать наилучшие параметры. Так как теперь признаков будет $28 \times 28 = 784$, то $1 \leq n \leq 15$. На графике (рисунок 2.9) представлены сравнения различных моделей, но с разными признаками.



(a) Логистическая регрессия на тренировочной выборке

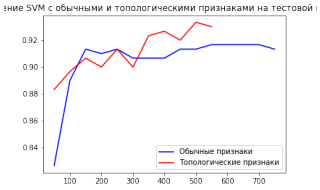


(b) Метод опорных векторов на тренировочной выборке



(c) Случайный лес на тренировочной выборке

(d) Логистическая регрессия на тестовой выборке



(e) Метод опорных векторов на тестовой выборке

(f) Случайный лес на тестовой выборке

Рис. 2.9: Сравнение различных моделей, которые используют различные признаки

Видно, что модели с топологическими признаками при малом количестве признаков достигает лучшей точности. Отсюда можно сделать вывод, что топологические признаки содержат в себе наиболее важную, релевантную информацию о структуре данных в меньшем числе признаков. Поэтому такой подход можно рассматривать как подход для уменьшения

размерности данных.

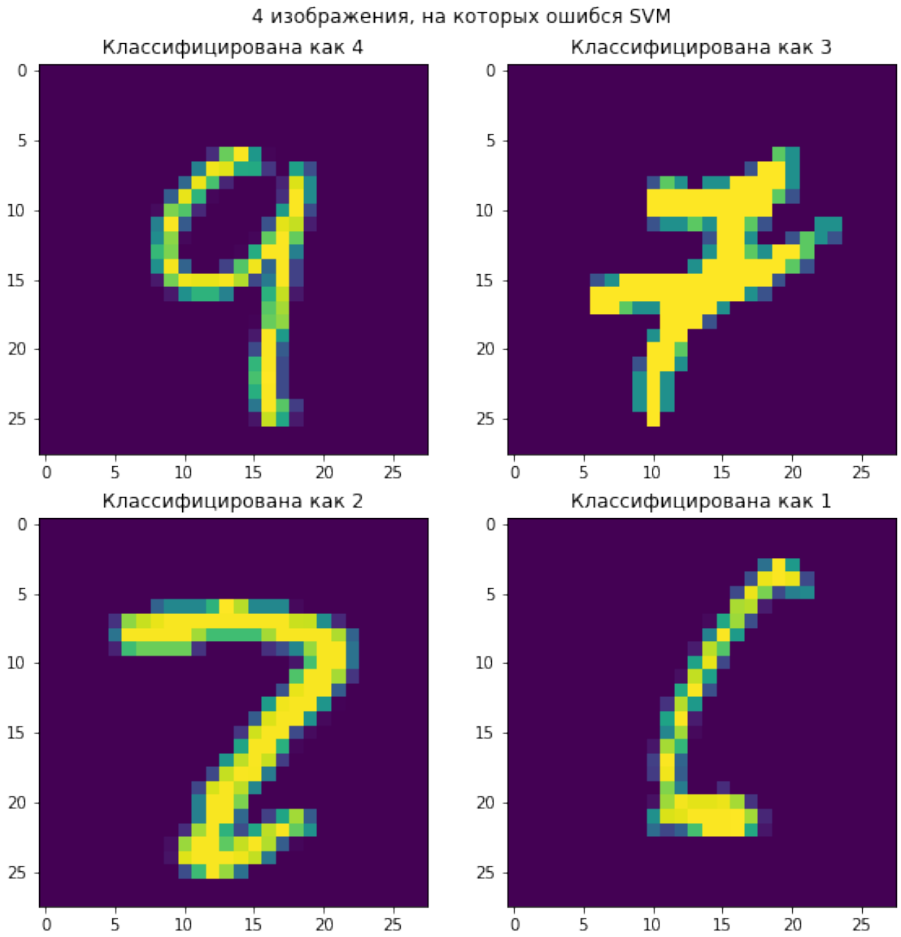


Рис. 2.10: Первые несколько изображений, на которых ошибся классификатор

Видно, что метод опорных векторов с подобранными параметрами на 350 лучших признаках, отобранных с помощью случайного леса, показывает чуть ли не самый высокий результат. Интересно посмотреть, на каких изображений эта модель ошиблась. На рисунке 2.10 представлены как раз такие 4 изображения. Видно, что цифры, на которых ошибся классификатор, действительно похожи на те, которые классификатор предписал данным изображениях.

Заключение

В ходе данной работы был изучен теоретический материал по алгебраической и прикладной топологии и машинному обучению? реализован алгоритм классификации датасета MNIST. Данный алгоритм использовал только топологические свойства изображенных рукописных цифр. На основе таких данных данный алгоритм показывал высокий уровень точности.

В ходе данной работы было выявлено, что реализованный алгоритм является эффективным алгоритмом понижения размерности пространства признаков. Было проведено сравнение с другими моделями машинного обучения, которые были обучены на обычных признаках – плоских представлениях изображений. В результате данного сравнения было обнаружено, что реализованный алгоритм показывает более высокую точность классификации при меньшем числе признаков.

Список использованных источников

1. Edelsbrunner H., Harer J. Computational Topology An Introduction. — Providence : AMS, 2009. — 241 с.
2. Zomorodian A., Carlsson G. Computing Persistent Homology // Discrete Comput. Geom. — 2005. — т. 33, № 2. — с. 249—274.
3. Список недавних публикаций по машинному обучению на arxiv – сервису по свободному распространению и открытому доступу научных статей в области физики, математики, компьютерных и других вычислительных наук. — URL: `https://arxiv.org/list/cs.LG/pastweek` (дата обр. 22.06.2021).
4. Manin Y. I., Marcolli M. Nori diagrams and persistent homology. — 2019.
5. Charlier J., State R., Hilger J. PHom-GeM: Persistent Homology for Generative Models. — 2019.
6. TopoGAN: A Topology-Aware Generative Adversarial Network / F. Wang [и др.] // Computer Vision – ECCV 2020 / под ред. A. Vedaldi [и др.]. — Cham : Springer International Publishing, 2020. — с. 118—136.
7. Characterizing the Latent Space of Molecular Deep Generative Models with Persistent Homology Metrics / Y. Schiff [и др.]. — 2021.
8. Topological Autoencoders / M. Moor [и др.]. — 2021.
9. Элементарная топология / О. Я. Виро [и др.]. — Москва : МЦНМО, 2018. — 358 с.

10. Вик Д. У. Теория гомологий. Введение в алгебраическую топологию. — Москва : МЦНМО, 2005. — 288 с.

11. Хатчер А. Алгебраическая топология. — Москва : МЦНМО, 2011. — 688 с.

12. A roadmap for the computation of persistent homology / N. Otter [и др.] // EPJ Data Sci. — 2017. — т. 6, № 17.

13. Chazal F., Michel B. An introduction to Topological Data Analysis: fundamental and practical aspects for data scientists. — 2021.

14. Mileyko Y., Mukherjee S., Harer J. Probability measures on the space of persistence diagrams // Inverse Problems. — 2011. — т. 27, № 12. — с. 124007. — URL: https://doi.org/10.1088/0266-5611/27/12/124007.

15. Rieck B., Sadlo F., Leitte H. Topological Machine Learning with Persistence Indicator Functions // Topological Methods in Data Analysis and Visualization V. — 2020. — с. 87—101. — URL: http://dx.doi.org/10.1007/978-3-030-43036-8_6.

16. Bubenik P., Dłotko P. A persistence landscapes toolbox for topological statistics // Journal of Symbolic Computation. — 2017. — т. 78. — с. 91—114. — URL: http://dx.doi.org/10.1016/j.jsc.2016.03.009.

17. A Stable Multi-Scale Kernel for Topological Machine Learning / J. Reininghaus [и др.]. — 2014.

18. Atienza N., Gonzalez-Diaz R., Soriano-Trigueros M. On the stability of persistent entropy and new summary functions for TDA // CoRR. — 2018. — URL: http://arxiv.org/abs/1803.08304.

19. Deisenroth M. P., Faisal A. A., Ong. C. S. O. Mathematics for Machine Learning. — 2020. — 417 с.

20. Bishop C. Pattern Recognition and Machine Learning. — New York : Springer-Verlag, 2006. — 738 с.

21. Введение в машинное обучение – курс на образовательной платформе Coursera по основам машинного обучения. — URL: https://www.coursera.org/learn/vvedenie-mashinnoe-obuchenie (дата обр. 22.06.2021).

22. Dionysus 2 – библиотека для вычислений устойчивых гомологий, написанная на C++ и имеющая интерфейс на Python. — URL: https://www.mrzv.org/software/dionysus/ (дата обр. 22.06.2021).

23. giotto-tda: A Topological Data Analysis Toolkit for Machine Learning and Data Exploration / G. Tauzin [и др.]. — 2020.

24. The GUDHI Project. GUDHI User and Reference Manual. — 3.4.1. — GUDHI Editorial Board, 2021. — URL: https://gudhi.inria.fr/doc/3.4.1/.

25. Tralie C., Saul N., Bar-On R. Ripser.py: A Lean Persistent Homology Library for Python // The Journal of Open Source Software. — 2018. — т. 3, № 29. — с. 925. — URL: https://doi.org/10.21105/joss.00925.

26. Saul N., Tralie C. Scikit-TDA: Topological Data Analysis for Python. — 2019. — URL: https://doi.org/10.5281/zenodo.2533369.

27. Chen C., Kerber M. Persistent homology computation with a twist. — 2011.

28. Boissonnat J.-D., Maria C. Computing Persistent Homology with Various Coefficient Fields in a Single Pass // CoRR. — 2020. — URL: http://arxiv.org/abs/2001.02960.

29. Maria C., Oudot S. Computing Zigzag Persistent Cohomology // CoRR. — 2016. — URL: http://arxiv.org/abs/1608.06039.

30. Tadasets – библиотека, содержащая синтетические датасеты для топологического анализа данных. — URL: `https://github.com/scikit-tda/tadasets` (дата обр. 22.06.2021).

31. Google Colab – Сервис Google, предоставляющий возможность запускать код, написанный на Python, в браузере, обладающий интерфейсом Jupyter Notebook, специально созданный для задач машинного обучения, анализа данных, а также образования. — URL: `https://colab.research.google.com/` (дата обр. 22.06.2021).

32. Carlsson G. Topology and data // Bull. Amer. Math. Soc. — 2009. — т. 46.

33. Munkres J. R. Topology. — 2000. — 537 с.

34. Сосинский А. Б. Введение в топологию: Лекционный курс. — Москва : МЦНМО, 2020. — 224 с.

35. Garin A., Tauzin G. A Topological "Reading" Lesson: Classification of MNIST using TDA. — 2019.

36. Herbert E., Dadvid L., Afra Z. Topological persistence and simplification // Discrete Comput. Geom. — 2002.

37. Bubenik P. Statistical Topological Data Analysis using Persistence Landscapes // Journal of Machine Learning Research. — 2015. — т. 16, № 3. — с. 77—102. — URL: `http://jmlr.org/papers/v16/bubenik15a.html`.

38. The structure and stability of persistence modules / F. Chazal [и др.]. — 2013.

Приложение

.1 Код сравнения методов на топологических и обычных признаках

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from gtda.homology import CubicalPersistence
from gtda.images import Binarizer, RadialFiltration, DensityFiltration
from gtda.diagrams import Scaler

from sklearn.pipeline import make_pipeline, make_union
from gtda.diagrams import PersistenceEntropy, Amplitude
from gtda.images import HeightFiltration

from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

from sklearn.model_selection import GridSearchCV
from sklearn.datasets import fetch_openml

X, y = fetch_openml("mnist_784", version=1, return_X_y=True)
y = y.to_numpy()
X = X.to_numpy().reshape((-1, 28, 28))
train_size, test_size = 700, 300


X_train, X_test, y_train, y_test = train_test_split(
    X, y, train_size=train_size, test_size=test_size, random_state=21,
       shuffle=True)
direction_list = [[1, 0], [1, 1], [0, 1],
                  [-1, 1], [-1, 0], [-1, -1], [0, -1], [1, -1]]

center_list = [
    [13, 6],
    [6, 13],
    [13, 13],
    [20, 13],
    [13, 20],
    [6, 6],
    [6, 20],
    [20, 6],
    [20, 20],
]

radius_list = [2, 4, 6]

filtration_list = (
    [
        HeightFiltration(direction=np.array(direction), n_jobs=-1)
        for direction in direction_list
    ]
    + [RadialFiltration(center=np.array(center), n_jobs=-1)
       for center in center_list]
    + [DensityFiltration(radius=r) for r in radius_list]
)

diagram_steps = [
    [
        Binarizer(threshold=0.4, n_jobs=-1),
        filtration,
        CubicalPersistence(n_jobs=-1),
        Scaler(n_jobs=-1),
    ]
    for filtration in filtration_list
```

```
]

metric_list = [
    {"metric": "bottleneck", "metric_params": {}},
    {"metric": "wasserstein", "metric_params": {"p": 1}},
    {"metric": "wasserstein", "metric_params": {"p": 2}},
    {"metric": "landscape", "metric_params": {
        "p": 1, "n_layers": 1, "n_bins": 100}},
    {"metric": "landscape", "metric_params": {
        "p": 1, "n_layers": 2, "n_bins": 100}},
    {"metric": "landscape", "metric_params": {
        "p": 2, "n_layers": 1, "n_bins": 100}},
    {"metric": "landscape", "metric_params": {
        "p": 2, "n_layers": 2, "n_bins": 100}},
    {"metric": "betti", "metric_params": {"p": 1, "n_bins": 100}},
    {"metric": "betti", "metric_params": {"p": 2, "n_bins": 100}},
    {"metric": "heat", "metric_params": {"p": 1, "sigma": 1.6, "n_bins":
       100}},
    {"metric": "heat", "metric_params": {"p": 1, "sigma": 3.2, "n_bins":
       100}},
    {"metric": "heat", "metric_params": {"p": 2, "sigma": 1.6, "n_bins":
       100}},
    {"metric": "heat", "metric_params": {"p": 2, "sigma": 3.2, "n_bins":
       100}},
]


feature_union = make_union(
    *[PersistenceEntropy(nan_fill_value=-1)]
    + [Amplitude(**metric, n_jobs=-1) for metric in metric_list]
)

tda_union = make_union(
    *[make_pipeline(*diagram_step, feature_union)
      for diagram_step in diagram_steps],
    n_jobs=-1
)
X_train_tda = tda_union.fit_transform(X_train)
X_test_tda = tda_union.transform(X_test)


rf_features = RandomForestClassifier()
rf_features.fit(X_train_tda, y_train)
sortedFeatures = rf_features.feature_importances_.argsort()
numOfFeatures = np.arange(50, len(sortedFeatures), 50)

params_rf = []
scores_rf = []
accuracies_rf = []
rf_params = {
    "n_estimators": [500, 1000],
    'max_depth': [80, 90, 100],
    'max_features': [2, 3, 4],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 200, 300]
}
for num in numOfFeatures:
    print(num)
    bestFeatures = sortedFeatures[-num:][::-1]
    X_train_best = X_train_tda[:, bestFeatures]

    rf = RandomForestClassifier(random_state=42)

    rf_grid = GridSearchCV(
        estimator=rf, param_grid=rf_params, cv=3, n_jobs=-1, verbose=4)

    rf_grid.fit(X_train_best, y_train)
```

```
128
129     params_rf.append(rf_grid.best_params_)
130     scores_rf.append(rf_grid.best_score_)
131
132     X_test_num = X_test_tda[:, bestFeatures]
133     accuracies_rf.append(rf_grid.best_estimator_.score(X_test_num, y_test)
            )
134
135 params_svm = []
136 scores_svm = []
137 accuracies_svm = []
138 svm_params = {
139     'kernel': ('linear', 'sigmoid', 'rbf', 'poly'),
140     'C': [10 ** i for i in range(-2, 3)],
141     'degree': [3, 5, 7],
142     'coef0': [j**i for i in np.arange(-4, 1, 1) for j in np.arange(1., 4.,
            1.)],
143     'gamma': ('scale', 'auto'),
144     'decision_function_shape': ('ovr', 'ovo')
145 }
146 for num in numOfFeatures:
147     print(num)
148     bestFeatures = sortedFeatures[-num:][::-1]
149     X_train_best = X_train_tda[:, bestFeatures]
150
151     svm = SVC(random_state=42)
152
153     svm_grid = GridSearchCV(
154         estimator=svm, param_grid=svm_params, cv=3, n_jobs=-1, verbose=4
155     )
156
157     svm_grid.fit(X_train_best, y_train)
158
159     params_svm.append(svm_grid.best_params_)
160     scores_svm.append(svm_grid.best_score_)
161
162     X_test_num = X_test_tda[:, bestFeatures]
163     accuracies_svm.append(svm_grid.best_estimator_.score(X_test_num,
            y_test))
164
165 params_log = []
166 scores_log = []
167 accuracies_log = []
168 log_params = {
169     "C": np.logspace(-3, 3, 7),
170     "penalty": ["l1", "l2", "lasso"],
171     "solver": ["newton-cg", "lbfgs", "liblinear", "sag", "saga"]
172 }
173 for num in numOfFeatures:
174     print(num)
175     bestFeatures = sortedFeatures[-num:][::-1]
176     X_train_best = X_train_tda[:, bestFeatures]
177
178     log = LogisticRegression()
179
180     log_grid = GridSearchCV(
181         estimator=log, param_grid=log_params, cv=3, n_jobs=-1, verbose=4
182     )
183
184     log_grid.fit(X_train_best, y_train)
185
186     params_log.append(log_grid.best_params_)
187     scores_log.append(log_grid.best_score_)
188
189     X_test_num = X_test_tda[:, bestFeatures]
190     accuracies_log.append(log_grid.best_estimator_.score(X_test_num,
            y_test))
191
192 X_train_notop = X_train.reshape(X_train.shape[0], -1)
```

```
X_test_notop = X_test.reshape(X_test.shape[0], -1)
rf_feat_notop = RandomForestClassifier()
rf_feat_notop.fit(X_train_notop, y_train)
sortedFeatures = rf_feat_notop.feature_importances_.argsort()
numOfFeatures_notop = np.arange(50, len(sortedFeatures), 50)

params_notop_svm = []
scores_notop_svm = []
accuracies_notop_svm = []
svm_params = {
    'kernel': ('linear', 'sigmoid', 'rbf', 'poly'),
    'C': [10, 100],
    'degree': [3, 5, 7],
    'coef0': [j**i for i in np.arange(-4, 1, 1) for j in np.arange(1., 4.,
        1.)],
    'gamma': ('scale', 'auto'),
    'decision_function_shape': ('ovr', 'ovo')
}
for num in numOfFeatures_notop:
    print(num)
    bestFeatures = sortedFeatures[-num:][::-1]
    X_train_best = X_train_notop[:, bestFeatures]

    svm = SVC(random_state=42)

    svm_grid = GridSearchCV(
        estimator=svm, param_grid=svm_params, cv=3, n_jobs=-1, verbose=4
    )

    svm_grid.fit(X_train_best, y_train)

    params_notop_svm.append(svm_grid.best_params_)
    scores_notop_svm.append(svm_grid.best_score_)

    X_test_num = X_test_notop[:, bestFeatures]
    accuracies_notop_svm.append(
        svm_grid.best_estimator_.score(X_test_num, y_test))

params_notop_rf = []
scores_notop_rf = []
accuracies_notop_rf = []
rf_params = {
    "n_estimators": [500, 1000],
    'max_depth': [80, 90, 100],
    'max_features': [2, 3, 4],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 200, 300]
}
for num in numOfFeatures_notop:
    print(num)
    bestFeatures = sortedFeatures[-num:][::-1]
    X_train_best = X_train_notop[:, bestFeatures]

    rf = RandomForestClassifier(random_state=42)

    rf_grid = GridSearchCV(
        estimator=rf, param_grid=rf_params, cv=3, n_jobs=-1, verbose=4
    )

    rf_grid.fit(X_train_best, y_train)

    params_notop_rf.append(rf_grid.best_params_)
    scores_notop_rf.append(rf_grid.best_score_)

    X_test_num = X_test_notop[:, bestFeatures]
    accuracies_notop_rf.append(
        rf_grid.best_estimator_.score(X_test_num, y_test))

```

```
261 params_notop_log = []
262 scores_notop_log = []
263 accuracies_notop_log = []
264 log_params = {
265     "C": np.logspace(-3, 3, 7),
266     "penalty": ["l1", "l2", "lasso"],
267     "solver": ["newton-cg", "lbfgs", "liblinear", "sag", "saga"]
268 }
269 for num in numOfFeatures_notop:
270     print(num)
271     bestFeatures = sortedFeatures[-num:][::-1]
272     X_train_best = X_train_notop[:, bestFeatures]
273 
274     log = LogisticRegression()
275 
276     log_grid = GridSearchCV(
277         estimator=log, param_grid=log_params, cv=3, n_jobs=-1, verbose=4
278     )
279 
280     log_grid.fit(X_train_best, y_train)
281 
282     params_notop_log.append(log_grid.best_params_)
283     scores_notop_log.append(log_grid.best_score_)
284 
285     X_test_num = X_test_notop[:, bestFeatures]
286     accuracies_notop_log.append(
287         log_grid.best_estimator_.score(X_test_num, y_test))
```