

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ ГУМАНИТАРНЫЙ

УНИВЕРСИТЕТ»

(РГГУ)

ОТДЕЛЕНИЕ ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ В ГУМАНИТАРНОЙ СФЕРЕ

Кафедра математики, логики и

интеллектуальных систем

в гуманитарной сфере

Палёнов Кирилл Игоревич

**Разработка программной оболочки для сравнения различных методов интеллектуального анализа данных**

Выпускная квалификационная работа студента 4-го курса очной формы обучения

Направление 45.03.04 «Интеллектуальные системы в гуманитарной сфере»

Направленность (профиль) «Разработка и программирование интеллектуальных систем в гуманитарной сфере»

|  |  |
| --- | --- |
| Допущена к защите на ГЭК  Заведующий кафедрой  д.ф.-м.н., профессор  *\_\_\_\_\_\_\_\_\_\_\_\_\_* (Бениаминов Е.М.)  «\_\_\_\_» \_\_\_\_\_\_\_ 2018 г. | Научный руководитель  д.ф.-м.н., профессор  *\_\_\_\_\_\_\_\_\_\_\_\_\_\_*\_(Аншаков О. М.)  «\_\_\_\_» \_\_\_\_\_\_ 2018 г. |

Москва 2018

Abstract

Since the beginning of the 21st century, data mining has become a common technology in data analysis. A variety of machine learning algorithms requires a reasonable approach to its selection. The algorithms can be separated by the problems they solve. The only problem considered in this paper is the problem of classification. The algorithms that solve the classification problem can be scored by some metrics like precision, recall and F-score. The estimation can be obtained by testing algorithms on the real datasets. However, some of the algorithms, such as JSM-method, are not adapted for the scoring classifier standard. The aim of the research is to create a tool, which makes it possible to evaluate the quality of the JSM-method and compare it to the other machine learning algorithms.

Оглавление

[Введение 4](#_Toc516494573)

[Глава 1. Теоретические основы оценки качества работы алгоритма машинного обучения для задач классификации. 6](#_Toc516494574)

[1.1. Кросс-валидация. Формирование обучающих и тестовых выборок. 6](#_Toc516494575)

[1.2. Параметры оценки качества работы классификатора. [4] 8](#_Toc516494576)

[Глава 2. Имплементированные в приложении классификаторы. 10](#_Toc516494577)

[2.1. Логистическая регрессия. 10](#_Toc516494578)

[2.2. Наивный байесовский классификатор. 10](#_Toc516494579)

[2.3. Метод -ближайших соседей. [5] 11](#_Toc516494580)

[2.4. Машина опорных векторов. [6] 11](#_Toc516494581)

[2.5. Деревья решений. [5] 12](#_Toc516494582)

[2.6. Случайный лес. [7] [8] 13](#_Toc516494583)

[2.7. Многослойный персептрон. 13](#_Toc516494584)

[2.8. ДСМ-метод. 14](#_Toc516494585)

[2.8.1. Адаптация ДСМ-метода для имплементации в приложении. 15](#_Toc516494586)

[Глава 3. Программная реализация. 17](#_Toc516494587)

[3.1. Представление данных 17](#_Toc516494588)

[3.2. Реализация ядра. 18](#_Toc516494589)

[3.3. Реализация интерфейса. 22](#_Toc516494590)

[3.4. Интеграция ядра и интерфейса. 23](#_Toc516494591)

[3.5. Системные требования 25](#_Toc516494592)

[Глава 4. Руководство пользователя 26](#_Toc516494593)

[Глава 5. Результаты компьютерных экспериментов. Таблицы точности, полноты и F1-меры. 32](#_Toc516494594)

[Заключение 34](#_Toc516494595)

[Список использованных источников 35](#_Toc516494596)

[Приложение. Фрагменты кода. 36](#_Toc516494597)

Введение

По мере развития информационных технологий и роста объёма информации всё более актуальным становится вопрос эффективного анализа данных, вычленение существенных фрагментов из общего информационного потока. Особое место в анализе информации занимает интеллектуальный анализ данных.

Методология интеллектуального анализа данных базируется на алгоритмах машинного обучения. Широкое разнообразие таких алгоритмов требует мотивированного подхода к отбору последних для решения конкретных задач. В данной работе будут рассматриваться только алгоритмы, применяемые при решении задач классификации. С целью адекватного выбора методов машинного обучения были предложены критерии оценки качества работы алгоритмов. Для задач классификации суть оценок сводится к вычислению количества правильно предсказанных примеров (объектов) по отношению к общей выборке.

Новизна данной работы состоит в создании современного инструмента, позволяющий оценить качество работы ДСМ-метода в самых современных методов интеллектуального анализа данных. Следует отметить, что предшественником данной системы можно рассматривать среду QuDa [1], разработанную П. А. Григорьевым и С. А. Евтушенко в начале 2000-х годов. В QuDa имелась возможность проводить анализ данных с помощью ассоциативных правил, деревьев решений, ДСМ-метода, наивным байесовским классификатором и некоторых других. Среда QuDa предоставляла возможности для сравнения эффективности реализованных в ней методов анализа данных. Однако эта среда с середины 2000-х годов поддерживаться перестала. С тех пор методы машинного обучения интенсивно развивались и в настоящее время имеется необходимость создания новых инструментов, позволяющих сравнивать различные методы машинного обучения.

Целью проекта является создание приложения, вычисляющего оценки эффективности работы алгоритмов машинного обучения.

Для достижения этой цели были решены следующие задачи:

1. Разработан модуль вычисления метрик (ядро).
2. Разработан вспомогательный язык, который описывает наборы данных.
3. Проведена адаптация ДСМ-метода к стандарту, который был имплементирован в ядре.
4. Разработан графический интерфейс для работы с ядром.

Основной областью применимости приложения являются научные и коммерческие исследования, направленные на выбор оптимального алгоритма машинного обучения для решения задач классификации.

Данная работа имеет 5 глав. В первой главе рассмотрены теоретические основы оценки качества работы алгоритмов машинного обучения и методики формирования обучающих и тестовых выборок. Во второй главе кратко описаны механизмы работы алгоритмов машинного обучения, которые имплементированы в программе. Третья глава посвящена описанию приложения, его реализации, также в ней идёт речь о представлении исходных данных в программе. В четвёртой главе приводится руководство пользователя. В пятой главе представлены результаты тестирования алгоритмов машинного обучения на реальных данных.

Глава 1. Теоретические основы оценки качества работы алгоритма машинного обучения для задач классификации.

1.1. Кросс-валидация. Формирование обучающих и тестовых выборок.

Кросс-валидация – процедура обучения и тестирования алгоритма машинного обучения с целью вычисления оценок. Суть процедуры кросс-валидации состоит в следующим: набор данных делится на k блоков (разбиений), в каждом из которых в заданной пропорции содержатся непересекающиеся обучающая и тестовая выборки. Для каждого блока проводится, соответственно, обучение и тестирование и вычисляются метрики.

Стратегия кросс-валидации – метод, по которому формируются блоки набора данных. Здесь и далее – общее число объектов.

Рассмотрим следующие стратегии кросс-валидации:

1. K-Fold [2, pp. 127, 376-377] [3].

Набор данных делится на k блоков. Тогда в каждый блок входит тестовых примеров, при этом тестовые выборки каждого из блоков не пересекаются. Остальные же примеров в каждом блоке входят в обучающую выборку.

1. Leave One Out (LOO) [2, pp. 128, 377-378] [3].

Каждая тестовая выборка состоит из одного примера, в обучающую выборку попадают все оставшиеся примеров. Генерируется блоков, то есть для каждого примера в наборе данных создаётся собственный блок. Стратегия LOO является частным случаем стратегии K-Fold при . Достоинством стратегии LOO по сравнению с K-Fold является то, что в тестовой выборке каждый пример встречается только один раз. Таким образом, удаётся избежать повторений при тестировании. Недостатком же является вычислительная сложность: обучение проводится раз.

1. Leave P Out (LPO) [2, pp. 128, 378]

Сходна со стратегией LOO, однако вместо одного примера в тестовую выборку добавляется примеров. Для N примеров будет сгенерировано блоков. В отличии от K-Fold и LOO, тестовые выборки могут пересекаться при .

1. Shuffle & Split [2, pp. 127, 379] [3]

Из множества всевозможных разбиений выбирается случайным образом выбирается k блоков с заранее определённой пропорцией обучающих и тестовых примеров.

Также на некоторые из этих стратегий могут быть наложены дополнительные условия формирования выборок, а именно:

1. Стратификация [2, pp. 127, 379-380] [3]

Если стратегия разбиения данных является стратификационной, то каждый блок формируется так, что пропорции объектов каждого класса в обучающей и тестовой выборке примерно одинаковы. Целесообразно использовать при сильно неравномерном распределении объектов разных классов в общем наборе данных. Применимо для стратегий K-Fold и Shuffle & Split.

1. Групповые выборки [2, pp. 127-128, 380-382]

Некоторые объекты зависят друг от друга. Тогда их можно объединить в группы. В этом случае разбиение проводится таким образом, чтобы объекты, которые принадлежат группам, входящим в обучающую выборку, не попали в тестовую выборку.

1.2. Параметры оценки качества работы классификатора. [4]

Для оценки работы алгоритма машинного обучения используются различные метрики. Самая простая из них – точность (англ. *Accuracy*) – метрика, вычисляющаяся как , где *p* – количество правильно предсказанных примеров, а *N* – размер обучающей выборки. Этот показатель может быть неинформативен в случае, когда распределение в обучающей выборке смещено в сторону какого-либо из классов. В связи с этим чаще используют взвешенные метрики.

Для вычисления взвешенных оценок применяются таблицы контингентности, которые составляются для каждого класса (табл. 1)

|  |  |  |  |
| --- | --- | --- | --- |
| Класс C | | Решение эксперта | |
| Положительное | Отрицательное |
| Решение алгоритма | Положительное | True Positive (TP) | False Positive (FP) |
| Отрицательное | False Negative (FN) | True Negative (TN) |

Таблица 1

Введём некоторые пояснения для решений:

* TP – количество примеров, классифицированных как принадлежащие к классу С и де-факто ему принадлежащих.
* FP – количество примеров, классифицированных как принадлежащие к классу С, но де-факто ему не принадлежащих.
* FN – количество примеров, не классифицированных как принадлежащие к классу С, но де-факто ему принадлежащих.
* TN – количество примеров, не классифицированных как принадлежащие к классу С и де-факто ему не принадлежащих.

Точность (англ. *Precision*) для класса С – метрика, которая вычисляется как отношение количества де-факто принадлежащих классу С объектов и количества объектов, квалифицированных алгоритмом как принадлежащие классу С.

Полнота (англ. *Recall*) для класса С – метрика, которая вычисляется как отношение количества классифицированных алгоритмом объектов как принадлежащие классу С и количества всех объектов класса C в выборке.

F-мера (англ. *F-measure* или *F-score*) для класса С – метрика, сочетающая в себе точность и полноту. Она представляет собой гармоническое среднее между точностью и полнотой.

Параметр *β* позволяет сместить значение F-меры к точности в при и к полноте при . При имеем

F1-мера есть сбалансированная F-мера, в которой вес точности и полноты одинаков. Здесь и далее в работе речь будет идти только о F1-мере.

Глава 2. Имплементированные в приложении классификаторы.

2.1. Логистическая регрессия.

Вводится множество независимых вещественных переменных – значений входных атрибутов , которые представляют собой признаки (регрессоры). Вводится также фиктивный признак . Целевой признак принимает значения из множества . Вероятность того, что , есть функция логистическая функция , где

,

и – векторы-столбцы вещественных чисел и переменных .

В ходе обучения алгоритма происходит подбор параметров . Как правило, это делается при помощи метода максимального правдоподобия.

2.2. Наивный байесовский классификатор.

В основе наивного байесовского классификатора лежит вероятностная модель

где – переменная класса (целевого признака), а – набор значений атрибутов. После разного рода преобразований получаем вероятность принадлежности примера к классу:

где – константа, зависящая только от значений входных атрибутов. Таким образом, решение о принадлежности примера к классу принимается на основании максимального значения вероятности для данного примера.

2.3. Метод -ближайших соседей. [5]

На основании входных векторов в ходе обучения строится *n*-мерное пространство, где *n* – количество атрибутов. Каждый пример – точка в *n*-мерном пространстве. Каждой точке присваивается класс – значение целевого признака. В ходе тестирования тестовому примеру X присваивается тот класс, к которому принадлежат наибольшее количество примеров из *k* ближайших к X точек (т.е. точек, расстояние от которых до X наименьшее). При этом понятие расстояния вариативное: это может быть манхэттенское расстояние, расстояние Евклида, его квадрат или другое.

2.4. Машина опорных векторов. [6]

Аналогично методу *k*-ближайших соседей имеем *n*-мерное пространство, в котором каждый объект – точка, описанная *n*-мерным вектором. В ходе тестирования строится разделяющая гиперплоскость, имеющая уравнение

где – перпендикуляр к разделяющей гиперплоскости, а параметр – модуль расстояния от гиперплоскости до начала координат. Для нахождения оптимального разделения следует рассмотреть опорные вектора и гиперплоскости, параллельные оптимальной и ближайшие к опорным векторам двух классов. Можно показать, что эти параллельные гиперплоскости могут быть описаны следующими уравнениям (с точностью до нормировки):

Если обучающая выборка линейно разделима, то мы можем выбрать гиперплоскости таким образом, чтобы между ними не лежала ни одна точка обучающей выборки и затем максимизировать расстояние между гиперплоскостями. Ширину полосы между ними легко найти из соображений геометрии, она равна , таким образом наша задача минимизировать . Чтобы исключить все точки из полосы, мы должны убедиться для всех i, что

2.5. Деревья решений. [5]

Дерево решений представляет собой структуру данных, элементами которой являются ветки и листья. На ветках указываются входные атрибуты и их всевозможные значения, в листьях – значение целевой функции – класс, к которому принадлежит объект. Каждый узел дерева соответствует одной из входной переменных. Комбинация листьев и веток образует своего рода правила. При тестировании, следуя по этим правилам, алгоритм определяет принадлежность объекта классу. Существуют различные алгоритмы построения дерева: ID3, C4.5, CART, CHAID, MARS и другие.

2.6. Случайный лес. [7] [8]

Метод основан на использовании ансамблей деревьев решений. Обучение каждого дерева проводится на непересекающихся подмножествах (бэггинг), что решает проблему построения одинаковых деревьев. Обучение проходит следующим образом: пусть обучающая выборка состоит из образцов, количество атрибутов – . Также задаётся параметр . Генерируется случайная подвыборка с повторениями размером из обучающей выборки. Затем на основе этой подвыборки строится решающее дерево, причём при создании каждого узла дерева формируется подмножество атрибутов, на основе которых проводится разбиение. Дерево строится, пока множество подвыборки не пусто. Отсечение ветвей не производится. Окончательное же решение о принадлежности объекта классу принимается путём голосования: примеру относится к тому классу, решение о принадлежности к которому приняло наибольшее количество деревьев.

2.7. Многослойный персептрон.

Многослойный персептрон — нейронная сеть, имеющая более чем один слой. Нейрон — это элемент сети, который имеет несколько входов, каждый из которых имеет вес. Нейроны бывают сенсорными (входные, S), ассоциативными (обучаемые «скрытые» слои, A) и реагирующими (выходные, R). Нейронная сеть обучается методом обратного распространения ошибки. Сенсорные нейроны получают входной вектор. Далее нейрон, получая сигнал, помножает сигналы на веса и суммирует получившиеся величины, после чего передает результат к другому нейрону или на выход сети. Сравнение с корректным образцом даёт ошибку, которая обратно распространяется по сети ко входам, при этом веса корректируются с поправкой на эту ошибку. Процедура продолжается до получения минимально заданного уровня ошибки.

2.8. ДСМ-метод.

В алгоритме работы ДСМ-методе можно выделить несколько функций, соответствующих когнитивным процедурам: индукция, аналогия и абдукция [9]. Индукция является частью процесса обучения, а аналогию можно рассматривать как процесс тестирования. Прямого соответствия абдукции применительно к машинному обучению нет, поэтому в рамках настоящей работы абдукция не рассматривалась.

* Фаза обучения

В ходе обучения формируются три множества – множества положительных, отрицательных и противоречивых примеров. Для каждого положительного примера имеются набор аргументов о принадлежности объекта к классу и не имеется набора аргументов о непринадлежности объекта к классу. Для каждого примера имеются набор аргументов о непринадлежности объекта к классу и не имеется набора аргументов о принадлежности объекта к классу. Для каждого противоречивого примера имеются набор аргументов как и о принадлежности объекта к классу, так и о непринадлежности объекта к классу.

С помощью правил индукции формируются гипотезы о возможных причинах набора целевых свойств. Построенные фрагменты образуют множества положительных и отрицательных гипотез (возможных причин наличия и возможных причин отсутствия набора целевых свойств, соответственно). Положительная гипотеза находится как сходство (пересечение) положительных примеров, а отрицательная гипотеза находится как сходство (пересечение) отрицательных примеров.

* Фаза тестирования

С помощью правил аналогии формируются гипотезы о принадлежности объекта к классу. Множество неопределенных примеров сужается. Множества положительных и отрицательных примеров расширяются.

2.8.1. Адаптация ДСМ-метода для имплементации в приложении.

Особенность ДСМ-метода заключается в том, что он работает только с бинарными данными. Это означает, что все небинарные атрибуты должны быть бинаризованы. Определим соответствующие функции:

Пусть – значение атрибута в конкретном примере, – элемент домена атрибута. Определим функцию-заполнитель бинарной строки (*binarize value*). Для номинального атрибута функция выглядит следующим образом

Для числового атрибута функция выглядит почти также, однако, по причине того, что домен атрибута состоит из числовых промежутков, то вследствие это вместо равенства мы проверяем принадлежность фактического значения атрибуту:

Пусть – небинарный атрибут. Тогда определим функцию бинаризации небинарного атрибута , которая возвращает преобразованный в бинарный кортеж небинарный атрибут A:

Другой особенностью ДСМ-метода является то, что в ходе тестирования некоторые примеры попадают в множество противоречивых, а также могут остаться примеры, класс которых определить не удалось. Таким образом, получается подмножества объектов, которое нельзя однозначно классифицировать как принадлежащие или не принадлежащие классу. Это приводит к формированию рваного выходного вектора и, как следствие, невозможности корректно проводить тестирование и вычислять метрики. Решение заключается в аппроксимации противоречивых объектов к положительным иди отрицательным. Здесь возможны различные интерпретации противоречивого и неопределённого множеств, которые могут быть имплементированы в последующих версиях данного приложения. В настоящей же версии все противоречивые и неопределённые примеры интерпретируются как отрицательные и, соответственно, не принадлежащие классу.

Глава 3. Программная реализация.

Приложение разработано в среде Visual Studio 2017 Community Edition. Решение включает в себя два независимых проекта: приложение Windows Forms, которое реализует интерфейс, и проект на языке Python, реализующее ядро.

3.1. Представление данных

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| age | Job | marital | education | Default | balance | housing | loan | Contact | pdays | y |
| 30 | unemployed | married | primary | no | 1787 | no | no | cellular | -1 | no |
| 33 | services | married | secondary | no | 4789 | yes | yes | cellular | 339 | no |
| 35 | management | single | tertiary | no | 1350 | yes | no | cellular | 330 | no |
| 30 | management | married | tertiary | no | 1476 | yes | yes | unknown | -1 | no |
| 59 | blue-collar | married | secondary | no | 0 | yes | no | unknown | -1 | no |

Программа работает с датасетами, которые представлены в виде csv-файла. Датасеты должны соответствовать стандарту UCI репозитория [10] и иметь вид таблицы с заголовком. Пример набора данных представлен в табл. 2.

Таблица 2

Как видно из примера, данные могут иметь разные типы данных (числовые, бинарные, номинальные). Такая ситуация неприемлема с точки зрения работы ДСМ-метода (см. гл. 2.8.1). Для разрешения этой проблемы был разработан специальный формат, который описывает атрибуты датасета в контексте их типизации. Этот формат имеет вид таблицы, в которой, как следует из табл. 3, для каждого атрибута указывается его имя, тип, домен, а также указывается, является ли атрибут входным или выходным. Для корректной работы программы такая таблица, представленная в файле с расширением .csv должна быть загружена для каждого набора данных.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Values | attribute\_type |
| age | Numeric | 18->35|36->45|46->inf | input |
| job | categorical | unemployed|management|blue-collar|self-employed | input |
| marital | categorical | married|divorced|single | input |
| education | categorical | unknown|secondary|primary|tertiary | input |
| default | binary | yes|no | input |
| balance | numeric | -inf->-1|0->24|25->299|300->699|700-1999|2000->inf | input |
| housing | binary | yes|no | input |
| loan | binary | yes|no | input |
| contact | categorical | unknown|telephone|cellular | input |
| pdays | numeric | -1|1->inf | input |
| y | binary | yes|no | output |

Таблица 3

В табл. 3 в столбце *name* указывается имя атрибута, в столбце *type* – один из трёх типов данных: *binary, numeric, categorical*, в столбце *values* – всевозможные значения, которые может принимать атрибут (если значение числовое и принадлежит промежутку, то указывается нижняя граница промежутка, затем символ ->, и, наконец, верхнее значение промежутка; *(-)inf* означает (минус-)бесконечность; все значения домена разделяются символом |), в столбце же *attribute\_type* указывается *input*, если атрибут является входным, и *output*, если атрибут выходной.

3.2. Реализация ядра.

Ядро реализовано на языке Python 3.6. Проект включает в себя два модуля и два пакета.

Модуль Main.py управляет ядром. Он содержит 3 функции, которые вызываются из интерфейса программы.

Функция GetAttributes(filename) вызывает одноимённую функцию из пакета BitSetConvertor, которая загружает данные о датасете в формате словаря, который содержит ключи inputattributes и outputattributes, и значения, соответствующим входным и выходным атрибутам соответственно. Эти значения также представляют собой словари, в которых ключом выступает имя атрибута, а значением – ещё один словарь, где хранится информация о типе, домене и о том, является ли атрибут входным или выходным. Пример полученного на выходе словаря представлен в Приложении (фрагмент кода 1).

Функция GetData(filename, attributes) вызывает одноимённую функцию из пакета BitSetConvertor, которая загружает датасет из csv-файла. Для хранения данных используется структура DataFrame из библиотеки pandas. Она позволяет эффективно обрабатывать массив данных, обращаться к элементам по меткам столбцов и индексам строк, быстро преобразовывать и выполнять другие манипуляции с данными.

Функция CrossValidation(estimator, data, rowstart, rowfinish, attributes, CV, CVparams) вызывает функцию перекрёстной проверки cross\_validation из модуля CrossValidation.py. На вход она принимает название алгоритма машинного обучения (реализация алгоритмов (кроме ДСМ-метода) импортирована из библиотеки sklearn), датасет в виде *pandas.DataFrame*, индексы, которые определяют срез набора данных для участия в кросс-валидации, атрибуты в формате словаря, а также название стратегии разбиения данных и её параметры. В функции cross\_validation происходит предварительная обработка данных, затем запускается функция cross\_validate из библиотеки sklearn.model\_selection. Каждая метрика (точность, полнота, F1-мера) вычисляется как среднее арифметическое метрик для каждого атрибута, которое, в свою очередь, вычисляется как среднее арифметическое метрик для каждого разбиения (см. гл. 1.1). За основу оценки работы алгоритма на конкретном разбиение берётся взвешенная метрика: вычисляется оценка для каждого класса, делится на поддержку (количество объектов класса в выборке) и вычисляется среднее по взвешенным метрикам для каждого класса.

Полученные метрики упаковываются в словарь и возвращаются.

На UML-диаграмме (пакетов) 1 представлена общая схема модулей и пакетов, которые используются в ядре.



UML-диаграмма 1

В рамках данной работы была адаптирована реализация ДСМ-метода Аншакова О. М. к стандарту библиотеки sklearn. В частности, был создан пакет JSM, в котором содержится реализация алгоритма Норриса в отдельном модуле и класс JSMClassifier, который наследуется от условно абстрактного класса BaseEstimator из библиотеки sklearn.



*UML-диаграмма 2*

Для приведения к стандарту были реализованы условно абстрактные методы класса BaseEstimator get\_params(), set\_params(\*\*parametrs), соответственно, возвращающий и задающий параметры для работы ДСМ-метода, а также методы fit(X,y) и predict(X), обеспечивающие, соответственно, обучение и тестирование ДСМ-метода (в двух последних методах аргумент X принимает входные данные, а y – выходные).

На UML-диаграмме (классов) 2 представлена иерархия наследования класса JSMClassifier от BaseEstimator. В классе JSMClassifier в качестве приватных методов реализованы все процедуры ДСМ-метода (за исключением абдукции, см. гл. 2.8).

3.3. Реализация интерфейса.

Интерфейс реализован в качестве приложения Windows Forms на языке C# (платформа .NET Framework 4.6.1). Оно включает в себя четыре формы, на которых пользователь может изменить данные об атрибутах, выбрать стратегию кросс-валидации, запустить перекрёстную проверку, а также получить информацию об оценках работы алгоритмов машинного обучения. Для хранения данных атрибутов, датасета и данных, введённым пользователем в поля формы, используются классы AttributeDataSets, DataSet, CVObject. Класс CrossValidationSet аккумулирует в себе все необходимые данные для запуска кросс-валидации и подготавливает их для передачи в ядро. В нём также содержится метод CrossValidate(), который запускает процесс перекрёстной проверки.

В проекте используются структура ValueTuple, позволяющие явным образом объединять переменные в кортеж. Это позволяет одновременно возвращать из функции несколько переменных без введения дополнительных структур. Так как платформа .NET Framework 4.6.1 не поддерживает кортежи, был установлен пакет NuGet, поставляющий DLL-библиотеку System.ValueTuple, реализующую кортежи. С примером можно ознакомиться в Приложении (фрагмент кода 2).

3.4. Интеграция ядра и интерфейса.

Для связи ядра, использующего код на языке Python, и интерфейса, написанного на C#, был использован Python-пакет pythonnet 2.3.0 [11]. Это очень мощный инструмент, позволяющий использовать объекты Python в среде .NET и наоборот. В составе пакета поставляется DLL-библиотека Python.Runtime.dll, позволяющая инициализировать модули Python и вызывать функции из этих модулей с передачей аргументов, а также возвращать значения в среду .NET.

В состав библиотеки Python.Runtime.dll входят .NET-классы, представляющие собой полные эквиваленты классам в Python, например, PyInt (эквивалент целого числа), PyFloat (эквивалент числа с плавающей точкой), PyString (эквивалент строки), PyList (эквивалент списка), PyDict (эквивалент словаря) и множество других. Все они наследуются от класса PyObject, через который можно представить Python-объект любого класса Python. У объекта класса PyObject (и, следовательно, у объектов всех остальных классов) имеется метод InvokeMethod(string name, params PyObject[] args), позволяющий вызвать любой метод из модуля, пакета или библиотеки и передать ему аргументы и возвращающий объект типа PyObject как результат работы метода в коде Python. Также в составе библиотеки имеется класс Py, который имеет методы для импорта модулей или пакетов, создания отдельных потоков для выполнения Python-кода и другие. Кроме того, Python.Runtime.dll добавляет ко всем .NET-объектам метод ToPython(), который преобразует любой .NET-объект к Python-объекту. Пример использования объектов библиотеки Python.Runtime.dll расположен в Приложении, фрагмент кода 3.

Схема интеграции интерфейса и ядра представлена на UML-диаграмме (компонентов) 3.



UML-диаграмма 3

3.5. Системные требования

Для корректной работы программы к компьютеру выдвигаются следующие системные требования:

* ОС Windows 7 и выше
* 64-разрядная архитектура
* ОЗУ не менее 2 Гб
* .NET Framework 4.6.1
* Python 3.6
* Пакеты pandas и sklearn для Python 3.6

В переменной среды Path должен быть прописан путь к директории Python 3.6.

Модули Python Main.py, CrossValidation.py, а также пакеты (папки с модулями) JSM и BitSetConvertor должны находиться в одной директории с исполняемым файлом MachineLearningScroesSystem.exe. Чтобы исполняемый файл имел доступ к вышеуказанным модулям и папкам из других директорий, необходимо прописать пути к папкам, в которых хранятся модули Main.py и CrossValidation.py, а также к папкам JSM и BitSetConvertor в переменной среды PYTHONPATH.

Глава 4. Руководство пользователя

Для начала работы с приложением необходимо запустить исполняемый файл MachineLearningScoresSystem.exe. Откроется следующее окно (рис. 1).

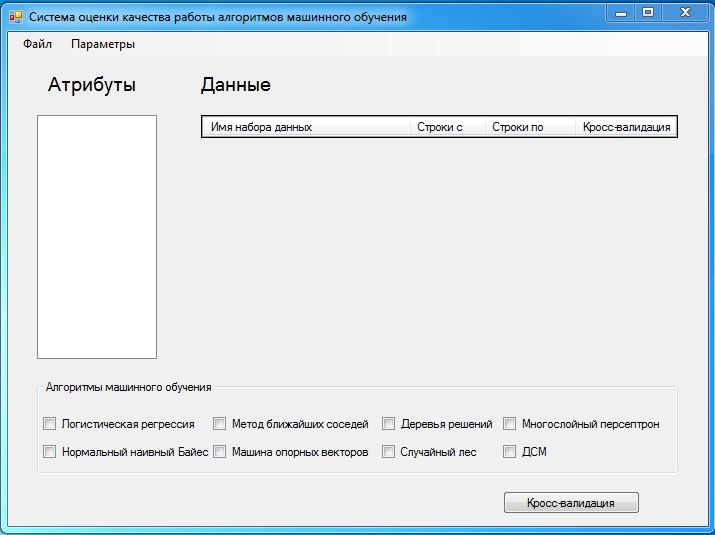


Рис. 1

Для загрузки информации об атрибутах нужно выбрать «Файл» → «Загрузить csv-файл с атрибутами». Если загрузка выполнена успешно, то название файла атрибутов без расширения отобразится в списке «Атрибуты» (рис. 2).

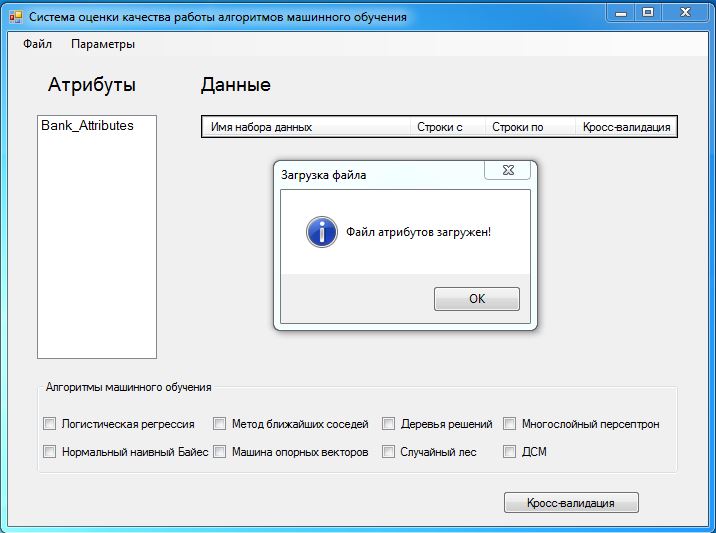


Рис. 2

Для загрузки csv-файла с данными необходимо выбрать набор атрибутов из списка загруженных, затем нажать «Файл» → «Загрузить csv-файл с данными» (рис.3).

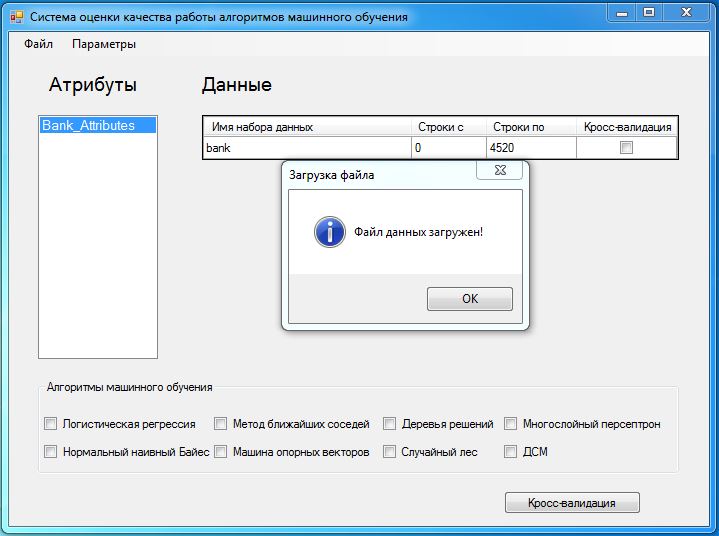


Рис. 3

Чтобы редактировать атрибуты, необходимо выбрать «Параметры» → «Препроцессинг». Откроется редактор атрибутов (рис. 4). В редакторе можно добавлять или удалять атрибуты, а также менять имена, типы, домены уже имеющихся атрибутов. Кроме того, имеется возможность сделать входной атрибут выходным и наоборот. Следует отметить, что всё содержимое поля «Значения» для каждого атрибута должно быть заключено в квадратные скобки, при этом каждое атрибута заключено в одинарные кавычки. Значения разделяются запятой. Если значение представляет собой числовой промежуток, то он должен быть заключён в квадратные скобки, внутри которых в одинарных кавычках через запятую указывается нижняя и верхняя граница промежутка.

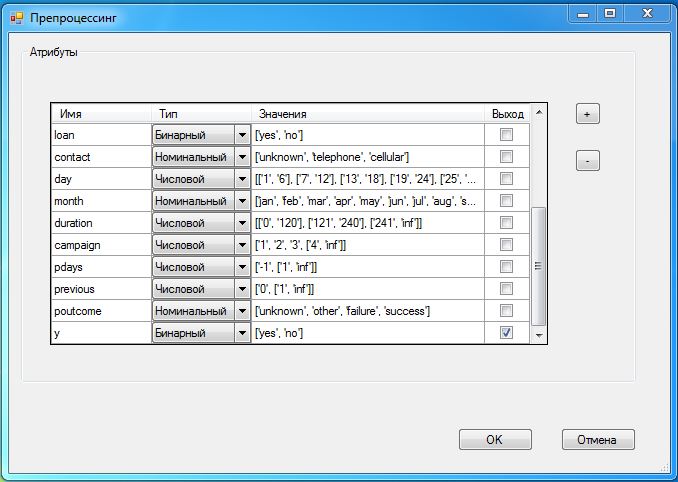


Рис. 4

Чтобы выбрать стратегию перекрёстной проверки, следует выбрать «Параметры» → «Стратегия разбиения данных» (рис. 5) и задать соответствующие параметры.

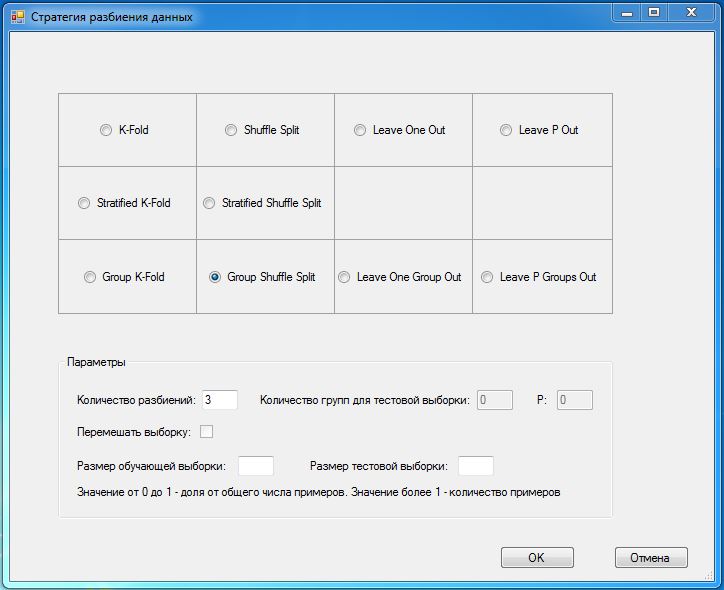


Рис. 5

На главной форме можно задать размеры среза данных для каждого датасета, который будет принимать участие в кросс-валидации. Для этого надо воспользоваться полями «Строки с» и «Строки по» в разделе «Данные». В разделе «Алгоритмы машинного обучения» следует выбрать методы машинного обучения для кросс-валидации. Для получения оценок работы алгоритмов следует нажать на кнопку «Кросс-валидация».

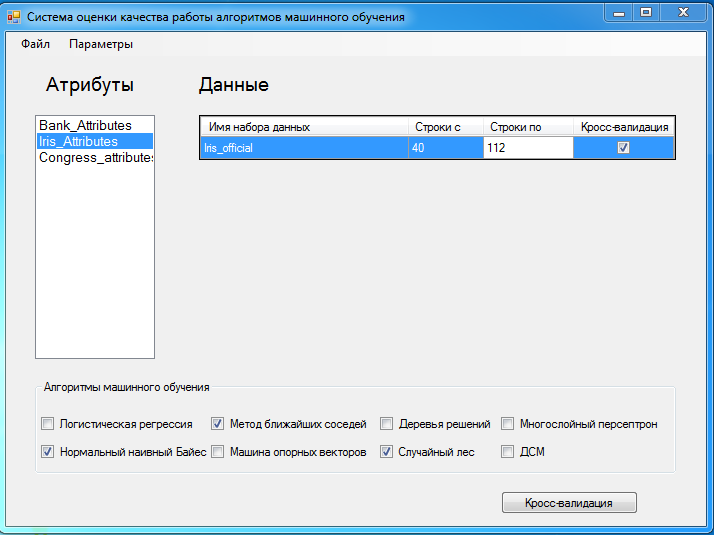


Рис. 6

Метрики алгоритмов появятся в отдельной форме (Рис. 7).

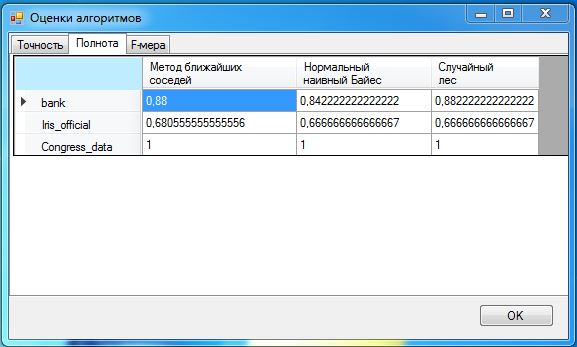


Рис. 7

Глава 5. Результаты компьютерных экспериментов. Таблицы точности, полноты и F1-меры.

Тестирования проводились на датасетах bank (16 входных атрибутов, 1 выходной атрибут, 40 примеров), Iris (4 входных атрибутов, 1 выходной атрибут, 150 примеров) и Congress (16 входных атрибутов, 1 выходной, 434 примера). Стратегия кросс-валидации K-Fold, 3 разбиения с перемешиванием.

Получились следующие результаты:

Точность:

|  |  |  |  |
| --- | --- | --- | --- |
|  | bank | iris | congress |
| ДСМ | 0,679925 | 0,603445 | 0,470573 |
| Машина опорных векторов | 0,682609 | 0,980144 | 0,960834 |
| Многослойный персептрон | 0,420737 | 0,959952 | 0,965352 |
| Метод ближайших соседей | 0,658802 | 0,945886 | 0,924366 |
| Наивный Байес | 0,755423 | 0,953005 | 0,942294 |
| Случайный лес | 0,649351 | 0,939452 | 0,969913 |
| Логистическая регрессия | 0,700921 | 0,939298 | 0,960724 |
| Деревья решений | 0,723199 | 0,952985 | 0,95591 |

Полнота:

|  |  |  |  |
| --- | --- | --- | --- |
|  | bank | iris | congress |
| ДСМ | 0,679925 | 0,603445 | 0,470573 |
| Машина опорных векторов | 0,682609 | 0,980144 | 0,960834 |
| Многослойный персептрон | 0,420737 | 0,959952 | 0,965352 |
| Метод ближайших соседей | 0,658802 | 0,945886 | 0,924366 |
| Наивный Байес | 0,755423 | 0,953005 | 0,942294 |
| Случайный лес | 0,649351 | 0,939452 | 0,969913 |
| Логистическая регрессия | 0,700921 | 0,939298 | 0,960724 |
| Деревья решений | 0,723199 | 0,952985 | 0,95591 |

F1-мера:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Bank | iris | Congress |
| ДСМ | 0,679925 | 0,603445 | 0,470573 |
| Машина опорных векторов | 0,682609 | 0,980144 | 0,960834 |
| Многослойный персептрон | 0,420737 | 0,959952 | 0,965352 |
| Метод ближайших соседей | 0,658802 | 0,945886 | 0,924366 |
| Наивный Байес | 0,755423 | 0,953005 | 0,942294 |
| Случайный лес | 0,649351 | 0,939452 | 0,969913 |
| Логистическая регрессия | 0,700921 | 0,939298 | 0,960724 |
| Деревья решений | 0,723199 | 0,952985 | 0,95591 |

Заключение

В рамках выпускной квалификационной работы были получены следующие результаты:

* Освоена работа с библиотеками машинного обучения на языке Python
* Найден эффективный способ интеграции Python и среды .NET
* Разработано оконное приложение для оценки качества работы алгоритма машинного обучения
* Реализация ДСМ-метода на языке Python 3.6 приведена к стандарту библиотеки Sckit-learn
* Разработано представление данных, которое позволяет автоматически производить бинаризацию датасета.
* Разработано ядро, вычисляющее метрики алгоритмов машинного обучения

При работе над совершенствованием данного приложения можно предложить решение следующих задач:

* Создать удобную визуализацию для сравнения эффективности различных методов анализа данных
* Имплементировать различные варианты ДСМ-метода и подобных ему методов анализа данных, в частности, обобщённый ДСМ-метод, ВКФ-метод [12], ИРОМ-метод Н. Д. Лыфенко [13] и другие
* Имплементировать другие методы машинного обучения

Список использованных источников

|  |  |
| --- | --- |
| [1] | P. A. Grigoriev and S. A. Yevtushenko, "Elements of an Agile Discovery Environment," *Lecture Notes in Computer Science,* vol. 2843, 2003. |
| [2] | scikit-learn developers, «scikit-learn user guide Release 0.19.1,» 21 Ноябрь 2017. [В Интернете]. Available: http://scikit-learn.org/stable/\_downloads/scikit-learn-docs.pdf. [Дата обращения: 9 Июнь 2018]. |
| [3] | «Скользящий контроль,» [В Интернете]. Available: http://www.machinelearning.ru/wiki/index.php?title=Скользящий контроль. [Дата обращения: 9 Июнь 2018]. |
| [4] | Д. Баженов, «Оценка классификатора (точность, полнота, F-мера),» 21 Июль 2012. [В Интернете]. Available: http://bazhenov.me/blog/2012/07/21/classification-performance-evaluation.html. [Дата обращения: 09 Июнь 2018]. |
| [5] | Ю. Кашницкий, «Открытый курс машинного обучения. Тема 3. Классификация, деревья решений и метод ближайших соседей.,» 13 Март 2017. [В Интернете]. Available: https://habr.com/company/ods/blog/322534/. [Дата обращения: 9 Июнь 2018]. |
| [6] | «Классификация данных методом опорных векторов,» 29 Сентябрь 2010. [В Интернете]. Available: https://habr.com/post/105220/. [Дата обращения: 9 Июнь 2018]. |
| [7] | В. Радченко, «Открытый курс машинного обучения. Тема 5. Композиции: бэггинг, случайный лес,» 27 Март 2017. [В Интернете]. Available: https://habr.com/company/ods/blog/324402/. [Дата обращения: 9 Июнь 2018]. |
| [8] | А. Калинин, «Random Forest: прогулки по зимнему лесу,» 29 Январь 2017. [В Интернете]. Available: https://habr.com/post/320726/. |
| [9] | О. М. Аншаков, «ДСМ-метод: теоретико-множественное объяснение,» *НТИ,* pp. 12-13, 2012. |
| [10] | Center for Machine Learning and Intelligent Systems, «UC Irvine Machine Learning Repository,» [В Интернете]. Available: https://archive.ics.uci.edu/ml/index.php. [Дата обращения: 9 Июнь 2018]. |
| [11] | «Python for .NET,» [В Интернете]. Available: https://pythonnet.github.io/. [Дата обращения: 09 Июнь 2018]. |
| [12] | Д. В. Виноградов, «ВКФ-метод интеллектуального анализа данных: обзор результатов и открытых проблем,» *Искусственный интеллект и принятие решений,* № 2, pp. 9-16, 2017. |
| [13] | N. D. Lyfenko, «Automatic Document Classification Based on J.S. Mill’s Ideas,» в *Proceedings of the First International Scientific Conference “Intelligent Information Technologies for Industry” (IITI’16)*, 2016. |

Приложение. Фрагменты кода.

В данном приложении представлены различные фрагменты исходного кода:

* Пример представления атрибутов в ядре

{

'inputattributes':

{'age': {'type': 'numeric', 'values': [['18', '35'], ['36', '45'], ['46', 'inf']]},

'marital': {'type': 'categorical', 'values': ['married', 'divorced', 'single']},

'loan': {'type': 'binary', 'values': ['yes', 'no']}},

'outputattributes':

{'y': {'type': 'binary', 'values': ['yes', 'no']}}

}

Фрагмент кода 1

* Пример использования кортежа в C#

public (double Precision, double Recall, double F1score) CrossValidate()

{

PyObject results;

using (Py.GIL())

{

results = CrossValidation() //код, вызывающий Python-модуль (упрощено в демонстрационных целях)

}

double Precision = results["Precision"];

double Recall = results["Recall"];

double F1score = results["F1Score"];

return (Precision: Precision, Recall: Recall, F1score: F1score); //возвращаем кортеж

}

public void foo()

{

(double Precision, double Recall, double F1score) = CrossValidate(); //присваиваем возвращаемое значение функции кортежу кортеж

Console.WriteLine("Precision = " + Precision);

Console.WriteLine("Recall = " + Recall);

Console.WriteLine("F1score = " + F1score);

}

Фрагмент кода 2

* Пример использования объектов Python.Runtime.dll

public PyString Estimator { get => \_estimator; set => \_estimator = value; }

public string EstimatorName { get => \_estimatorname; set => \_estimatorname = value; }

public string DatasetName { get => \_datasetname; set => \_datasetname = value; }

public PyObject Dataset { get => \_dataset; set => \_dataset = value; }

public PyInt Rowstart { get => \_rowstart; set => \_rowstart = value; }

public PyInt Rowfinish { get => \_rowfinish; set => \_rowfinish = value; }

public PyObject Attributes { get => \_attributes; set => \_attributes = value; }

public PyString CV { get => \_cv; set => \_cv = value; }

public PyDict CVparams { get => \_cvparams; set => \_cvparams = value; }

public void CrossValidate()

{

PyObject results;

using (Py.GIL()) //создаём новый поток

{

results = Py.Import("Main").InvokeMethod("CrossValidation",

new PyObject[] { Estimator, Dataset, Rowstart, Rowfinish, Attributes, CV, CVparams });

//импортируем модуль Main.py, вызываем метод CrossValidation из модуля Main.py и передаём ему аргументы из массива объектов PyObject

}

Фрагмент кода 3