Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра ЭВМ

Дисциплина: СПОВМ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

Программный модуль оценки качества алгоритмов

машинного обучения

БГУИР КП 1-40 02 01 2\* ПЗ

Студент:

Гр. 150501 Белов А.В.

Гр. 150504 Емчик А.Р.

Руководитель: Лукашевич М.М

Минск 2014

Учреждение образования

«Белорусский государственный университет информатики   
и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой

––––––––––––––––––––––––

(подпись)

\_\_\_\_––––––––––––––––––– 20 г.

ЗАДАНИЕ

по курсовому проектированию

Студентам    *Белову Александру Владимировичу, Емчику Арсению Руслановичу*

1. Тема проекта: Программный модуль оценки качества алгоритмов машинного обучения–––––––––––––––––––––    \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Срок сдачи студентом законченного проекта–––––*15 мая 2014 г.–––*

3. Исходные данные к проекту *Среда разработки –PyCharm Community Edition 3.1.3, язык программирования – Python 2.7.6. Приложение работает в семействе Debian-based операционных систем. Позволяет протестировать алгоритмы машинного обучения на конкретном наборе данных с использованием различных метрик оценки качества.*

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

*Титульный лист, Лист задания, Введение, Обзор аналогов и проблемной области Системное проектирования, Функциональное проектирование, Разработка программного модуля, Руководство пользователя, Заключение, Список литературы*

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

*1. Схема структурная––––––––––––––––––––––––––––––*

*2. Диаграмма классов*

*3. Блок-схема алгоритма.–––––*

6. Консультант по проекту *Лукашевич М.М. –*

7. Дата выдачи задания –––––*19 февраля 2014 г*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

*разделы 1,2 к 01.03 – 15 %;––––––––––––––––––––––––––––––––––––––    –\_\_*

*раздел 3 к 20.03 – 10 %;–––––––––––––––––––––––––––––––––––––       –––*

*разделы 4,5 к 01.04 – 20 %;––––––––––––––––––––––––––––––––––––––––*

*разделы 6,7 к 09.04 – 35 %;–––––––––––––––––––––––––––––––––––––––*

*раздел 8 к 16.04 – 5 %;––––––––––––––––––––––––––––––––––––––––       \_*

*оформление пояснительной записки и графического материала к 16 04 – 15 %*

*Защита курсовой работы с 16.04 по 15.05––––––––––––––––––––––––––––\_\_*

РУКОВОДИТЕЛЬ

*Лукашевич М.М.*

(подпись)

Задание принял к исполнению *Белов Александр Владимирович*

(дата и подпись студента)

Задание принял к исполнению *Емчик Арсений Русланович*

(дата и подпись студента)

Ведомость курсовой работы

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Обозначение | | | | Наименование | | | | Дополнительные сведения | | | |
|  | | | | Текстовые документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| БГУИР КР 1–40 02 01 2 ПЗ | | | | Пояснительная записка | | | | 33 с. | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | | Графические документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| ГУИР 400201.1 СС | | | | Схема структурная | | | | Формат А3 | | | |
|  | | | |  | | | |  | | | |
| ГУИР 400201.2 БС | | | | Схема алгоритма работы системы | | | | Формат А3 | | | |
|  | | | |  | | | |  | | | |
| ГУИР 400201.3 ДК | | | | Диаграмма классов | | | | Формат А3 | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  |  |  |  |  | БГУИР КП 1-40 02 01 2\* ПЗ | | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм. | Л. | № докум. | Подп. | Дата | Программный модуль оценки качества алгоритмов машинного обучения  Ведомость курсовой  работы |  | | | | Лист | Листов |
| Разраб. | | Белов |  | 15.05.14 |  | Т | |  | 1 | 1 |
| Пров. | | Лукашевич |  | 15.05.14 | Кафедра ЭВМ  гр. 150501 | | | | | |
|  | |  |  |  |
|  | |  |  |  |
|  | |  |  |  |

СОДЕРЖАНИЕ

[Введение 6](#_Toc387774967)

[1. Обзор аналогов и проблемной области 10](#_Toc387774968)

[1.1 Аналитический обзор литературы 10](#_Toc387774969)

[1.2 Обзор схожих программных средств 12](#_Toc387774970)

[1.3 Постановка задачи 15](#_Toc387774971)

[1.4 Краткий обзор используемых алгоритмов 17](#_Toc387774972)

[1.6 Краткий обзор используемых наборов данных 18](#_Toc387774973)

[2. Системное проектирование 20](#_Toc387774974)

[3. Функциональное проектирование 22](#_Toc387774975)

[4. Разработка программного модуля 26](#_Toc387774976)

[5. Руководство пользователя 28](#_Toc387774977)

[Заключение 32](#_Toc387774978)

[Список литературы 34](#_Toc387774979)

[Приложение 36](#_Toc387774980)

# 

# Введение

Машинное обучение - один из наиболее обширных и значимых разделов искусственного интеллекта, основным назначением которого является построение обучаемых моделей и оценка их качества. Среди алгоритмов машинного обучения обычно различают обучение с учителем (supervised learning) и обучение без учителя  (unsupervised learning). Классический пример обучения с учителем - обучение многослойного перцептрона методами Back propagation и Resilient propagation. В этом же ключе можно упомянуть и метод опорных векторов (support vector machine), а также деревья принятия решений (decision trees). Среди алгоритмов обучения без учителя наиболее известным и, зачастую, упоминаемым в первую очередь является алгоритм K-средних (K-means).

В обучении с учителем алгоритм получает несколько примеров пар «входные значения – выходные значения» и обучает функцию, которя ставит выходы в соответствие входам. С другой стороны, в обучении без учителя испытуемая система должна обнаружить скрытые взаимосвязи между объектами без вмешательства со стороны экспериментатора [1].

Наиболее базовыми задачами машинного обучения являются задачи классификации и кластеризации данных. Задача классификации состоит в том, чтобы определить класс объектов из некоторой тестовой выборки на основе имеющихся данных о классах объектов из обучающей выборки. Задача же кластеризации состоит в организации объектов в однородные группы на основе совокупности признаков. При этом задача классификации относится к классу задач обучения с учителем, в то время как задача кластеризации относится к классу обучения без учителя.

Одним из наиболее интересных инструментов машинного обучения являются нейронные сети, которые могут решать широкий круг задач обработки и анализа данных: распознавание и классификация образов, прогнозирование, управление, автоматическая торговля, анализ естественного языка, медицинских данных и т.д. Конкурентами нейронных сетей являются классические методы анализа данных, однако нейронные сети имеют над ними ряд преимуществ. Используя способность обучаться на множестве примеров, нейронная сеть способна решать задачи, в которых неизвестны закономерности развития ситуации и зависимости между входными и выходными данными. Нейронные сети устойчивы к зашумленности входных данных и способны адаптироваться к изменениям окружающей среды. Также нейронные сети обладают потенциальным сверхвысоким быстродействием и значительной отказоустойчивостью за счет массового параллелизма обработки информации.

Практически в каждой предметной области при ближайшем рассмотрении можно найти постановки нейросетевых задач [2]:

* Экономика и бизнес: предсказание рынков, автоматический дилинг, оценка риска невозврата кредитов, предсказание банкротств, оценка стоимости недвижимости, выявление пере- и недооцененных компаний, автоматическое рейтингование, оптимизация портфелей, оптимизация товарных и денежных потоков, автоматическое считывание чеков и форм, безовасность транзакций по пластиковым карточкам.
* Медицина: обработка медицинских изображений, мониторинг состояния пациентов, диагностика, факторный анализ эффективности лечения, очистка показаний приборов от шумов.
* Авионика: обучаемые автопилоты, распознавание сигналов радаров, адаптивное пилотирование сильно поврежденного самолета.
* Связь: сжатие видео-информации, быстрое кодирование-декодирование, оптимизация сотовых сетей и схем маршрутизации пакетов.
* Интернет: ассоциативный поиск информации, электронные секретари и агенты пользователя в сети, адресная реклама, коллаборативная фильтрация.
* Политические технологии: анализ и обобщение социологических опросов, предсказание динамики рейтингов, выявление значимых факторов, объективная кластеризация электората, визуализация социальной динамики населения.
* Безопасность и охранные системы: системы идентификации личности, распознавание голоса, распознавание лиц, автомобильных номеров, мониторинг информационных потоков.

Однако есть у нейронных сетей и недостатки. Как следствие отсутствия формализованных алгоритмов настройки сети и ее высокой сложности, возникает необходимость в привлечении высококлассных специалистов, обладающих необходимыми знаниями. Отсутствует строгая теория по выбору архитектуры нейронной сети под специфические задачи, хотя некоторая работа в этом направлении ведется. Из обученной сети практически невозможно извлечь приобретенные знания, то есть, нейронная сеть является своего рода черным ящиком.

Для реализации нейросетевых алгоритмов используются различные языки программирования, в том числе и специализированные диалекты LISP. Для написания реально используемых систем часто используют С++ благодаря его чрезвычайно мощным возможностям параллельного программирования, в том числе и на GPU, а также высокой скорости выполнения. В то же время, для прототипирования приложений, связанных с машинным обучением, обычно используются более специализированные языки: Matlab, R, Python.

В то время, как для задачи классификации используются многослойный перцептрон, метод опорных векторов и деревья принятия решений, для задачи кластеризации используют нейронные сети Кохонена, алгоритмы K-means, Affinity propagation, Spectral clustering, Hierarchical clustering, DBScan и Gaussian mixtures.

Наиболее серьезным препятствием перед использованием алгоритмов машинного обучения в реальных условиях является выбор оптимального алгоритма для определенной задачи. Помочь в этом могут алгоритмы оценки качества обучения. Однако и они тоже подвержены проблеме оптимального выбора. Разные метрики, предназначенные для разных алгоритмов машинного обучения, оценивают алгоритмы с разных позиций. Нельзя с уверенностью сказать, какая метрика будет более точно отображать реальную эффективность алгоритма на определенном наборе данных. Поэтому целью данной работы будет являться создание инструмента, позволяющего исследовать поведение различных алгоритмов классификации и кластеризации на различных наборах данных с помощью различных метрик.

Основным языком для приложения является Python – популярный язык программирования, используемый для разработки самостоятельных программ и различного рода прикладных сценариев в самых разных областях применения [3]. Связано это, прежде всего, с возможностью использования библиотек машинного обучения PyBrain и Scikit-learn о которых подробнее будет рассказано в постановке задачи.

# Обзор аналогов и проблемной области

## 1.1 Аналитический обзор литературы

В качестве литературных источников при ознакомлении с предметной областью можно использовать следующие материалы:

1. **Stuart Russel, Peter Norvig. Artifical Intelligence. A Modern Approach. 2010 – 1154c.**

Книга известных исследователей в области искусственного интеллекта Стюарта Рассела и Питера Норвига раскрывает последние наиболее важные применения искусственного интеллекта: распознавание речи, машинный перевод, роботизированные транспортные средства и домашних роботов. Также рассмотрено применение систем искусственного интеллекта в области поиска и обработки информации, техники использования больших объемов данных.

Что характерно, Питер Норвиг является директором по исследованиям в корпорации Google. Так что можно надеяться на описание наиболее интригующих исследований непосредственно от активного участника событий. Также он вместе со своим харизматичным коллегой по работе Себастианом Траном является организатором первых изветных онлайн-курсов по искусственному интеллекту и сооснователем образовательной организации Udacity.

Книга снабжена большим количеством псевдокода и доступных для понимания математических выкладок. Поэтому неудивительно то, что «Искусственный интеллект. Современный подход» используется во многих профильных учебных заведениях в качестве готового учебного пособия.

1. **А. А. Ежов, С. А. Шумский. Нейрокомпьютинг и его применения в экономике и бизнесе. 1998 – 216с.**

В то время, как книга Саймона Хайкина является полным математическим экскурсом в область нейронных сетей, в книге Ежова и Шумского рассмотрены, скорее, вопросы практического применения нейронных сетей в более популярной форме. Прочитать данную книгу можно рекомендовать, прежде всего, начинающему исследователю, чтобы получить первичное представление о возможностях, проблемах и методах выбранной области практической науки.

Интересно и то, что материал написан именно для лучшего понимания основ нейросетевой обработки данных преимущественно в области финансов. То есть все выкладки объясняются максимально простым языком, понятным неподготовленным читателям. Для повышения интереса к области книга снабжена наиболее интересными примерами применения нейронных сетей из реальной жизни, которые все еще не потеряли своей актуальности, несмотря на достаточно долгий срок, прошедший с момента издания.

1. **Марк Лутц. Изучаем Python. 2010 – 1280с.**

В данной книги изложены основополагающие принципы программирования на языке Python. Данная книга является весьма основательным учебником, написанным, что для студентов немаловажно, сравнительно доступным языком и основанным на материалах курсов, которые ведет автор Марк Лутц вот уже более десяти лет. Это позволяет быстро овладеть основными принципами программирования на Python за сравнительно небольшой промежуток времени независимо от уровня начальной подготовки читающего. Хотя базовые знания программирования точно помогут процессу усвоения читателем философии языка.

Особенность четвертого издания, выпущенного в 2010 году, состоит в особом акцентировании отличий версии языка 3.0 от версии 2.6. Но при этом и та, и другая версия языка рассматриваются с одинаковым вниманием, что немаловажно для нас, так как в работе над приложением использовалась, преимущественно, версия 2.7, причиной чего явились весьма специфические требования библиотек.

В целом, книга является достаточным для большинства задач настольным руководством для языка Python.

1. **Саймон Хайкин. Нейронные сети. Полный курс. 2006 – 1103с.**

«Нейронные сети. Полный курс» Саймона Хайкина – известная всем исследователям в области искусственного интеллекта книга об основных парадигмах нейронных сетей. Интересно в ней в первую очередь то, что весь материал снабжен не только строгим математическим обоснованием, но и иллюстрируется наглядными примерами и содержит множество решений встречающихся в реальной жизни задач. В частности в книге на должном уровне рассмотрены задачи распознавания образов, управления и обработки сигналов, столь типичные для этой области.

В нейронных сетях, да и в машинном обучении в целом, невозможно никуда деться от необходимости понимания математических предпосылок используемых методов. Для нашей работы материала, изложенного в книге, хватает с огромным запасом. Рассмотрены там даже методы, которые не будут использоваться по причине их излишней сложности в реализации и понимании их математического аппарата студентом типичной инженерной специальности с курсом высшей математики в три семестра.

Так же, как и книга Марка Лутца является настольным руководством для программистов на языке Python, труд Саймона Хайкина является библией исследователя программных нейронных сетей. Книга обязательна к прочтению.

## 1.2 Обзор схожих программных средств

Зачастую при изучении и оценке эффективности нейронных сетей и других алгоритмов машинного обучения используются программные пакеты, обладающие огромным избыточным функционалом, что можно рассматривать и как преимущество, в случае достаточно хорошей теоретической подготовки и необходимости в экспериментах, и как недостаток, в случае обладания лишь базовым представлением о работе нейронных сетей и используемых алгоритмов. Зачастую, чтобы приступить к углубленному изучению темы, необходимо сначала получить начальное представление о ней, обойдясь сравнительно малыми затратами. Тем не менее, существует ряд аналогов для нашей разработки, которые используются уже достаточно широко и предоставляют широкий спектр возможностей для работы с алгоритмами машинного обучения.

Одним из наиболее мощных пакетов для разработки и редактирования нейронных сетей обладает продукт компании американской The MathWorks – пакет прикладных программ MATLAB. Графический интерфейс для построения нейронных сетей Neural Network Toolbox впервые появился в MATLAB версии 6.0. Преимущество использования именно продукта MATLAB заключается в возможности комбинировать нейронные сети с многочисленными алгоритмами и функциями для инженерных расчетов, а также наличие продвинутых возможностей по оценке полученных алгоритмов. Также преимуществом MATLAB является наличие высокоуровневого интерпретируемого языка программирования, включающего основанные на матрицах структуры данных и широкий спектр функций, а также объектно-ориентированные возможности и интерфейсы к программам, написанным на других языках. И, безусловно, немаловажным положительным фактором является наличие очень большого сообщества пользователей, которое позволяет очень быстро получить ответы на возникающие вопросы и квалифицированную помощь.

Недостатками пакета программ MATLAB является, все же, относительно небольшое число реализованных специфичных структур нейронных сетей, хотя надо заметить, что все основные архитектуры в нем присутствуют. Также реализовано крайне небольшое число функций активации нейронов – целых три штуки. Рассматривать ли это как недостаток данного программного пакета – зависит от поставленной задачи, однако в графическом интерфейсе отсутствует возможность задания произвольной функции активации нейрона, что могло бы быть полезным для исследований конкретных нейросетевых алгоритмов.

Единственным в мире крупным нейросетевым программным продуктом, полностью переведенным на русский язык (по заявлению компании-разработчика), является программный пакет Statistica, разработанный, как это ни странно, компанией Statsoft, тоже американской, а точнее, его модуль Statistica Automated Neural Networks. Как и MATLAB, Statistica представляет большой пакет программ для инженерных расчетов, а значит, нейронные сети можно успешно комбинировать с другими видами алгоритмов в сложных законченных системах. В системе реализовано большое количество нейросетевых архитектур, достаточное для решения многих практических задач и, что немаловажно, мастер решений, который автоматически анализирует задачу и выбирает несколько подходящих для реализации архитектур. Также в пакете программ представлено большое число различных алгоритмов обучения сети, что позволяет иметь большую свободу в выборе оптимальных алгоритмов для решения определенного типа задач. Рекламируются также возможность создания гибридных нейронных сетей, состоящих из нескольких различных архитектур и база готовых решений [4].

Несмотря на все богатство функциональных возможностей и хорошую русификацию, критике можно подвергнуть и продукт Statistica Automated Neural Networks. Так, благодаря впечатляющим функциональным возможностям, мы имеем достаточно сложный интерфейс самой программы. В итоге задача изучения нейронных сетей и экспериментирования с их различными структурами и типами плавно превращается в задачу изучения интерфейса одной отдельно взятой программы. В целом, программа рассчитана на практическое применение в реальных областях специально обученными людьми, что не соответствует задаче академического обучения. Но продукт компании Statsoft можно смело рекомендовать пользователям, достаточно глубоко изучившим нейронные сети и желающим применить полученные знания в какой-либо прикладной задаче [5].

Существует достаточно большое количество мелких продуктов для конструирования нейронных сетей. Многие программисты реализуют их в процессе обучения, так как это позволяет лучше отточить необходимые в работе навыки.

Одним из примеров может служить немецкая программа MemBrain. Отличается от вышеперечисленных крупных продуктов она тем, что предлагает бесплатную лицензию для некоммерческого и академического использования. Однако данный продукт, все же, не совсем подходит для начальных этапов работы с нейросетями по причине отсутствия поддержки какого-либо языка кроме английского, что, впрочем, не должно стать для студентов проблемой, но дополнительным раздражающим элементом стать вполне может [6].

В целом, во всех приведенных программных средствах присутствует возможность оценки качества используемых алгоритмов, но ни один из этих программных пакетов не является достаточно простым, чтобы обеспечить первичное ознакомление с алгоритмами машинного обучения и быстрый выбор наиболее эффективного способа обработки конкретного набора данных. Эту задачу мы и попытаемся решить в своей работе.

## 1.3 Постановка задачи

Основной задачей при проектировании приложения для оценки качества алгоритмов машинного обучения является создание достаточно простого и удобного инструмента для быстрой проверки первичных гипотез относительно выбора оптимального алгоритма для конкретного набора данных.

По причине исключительно большой трудоемкости построения алгоритмов машинного обучения с нуля на языках системного программирования, подобных С++, в данной работе будет широко использоваться высокоуровневый язык программирования Python, который, к тому же, многие программисты предпочитают использовать в задачах машинного обучения наряду с языками Matlab и R.

Особо стоит отметить использование в приложении библиотек PyBrain и Scikit-learn, реализующих многие алгоритмы машинного обучения и содержащие большое количество доступных для использования метрик оценки данных алгоритмов.

PyBrain – одна из лучших библиотек в мире Python для реализации и изучения разнообразных алгоритмов, связанных с нейронными сетями. Scikit-learn реализует часть алгоритмов машинного обучения, которая лежит за пределами нейросетевых алгоритмов: бинарные деревья, метод опорных векторов, алгоритмы кластеризации. Кроме того, данная библиотека снабжена набором метрик оценки качества алгоритмов машинного обучения буквально на все случаи жизни.

Для построения графического интерфейса используется библиотека Glade, генерирующая описание графического интерфейса на основе XML. Потенциально есть возможность получить кроссплатформенную реализацию графического интерфейса, так как существует порт Glade для семейства ОС Windows. В пользу Glade также говорит наличие удобного инструмента для проектирования интерфейса – Glade Interface Designer.

Основными целевыми операционными системами будут являться Debian-based дистрибутивы Linux, что соответствует сложившейся ситуации в области систем машинного обучения и позволяет следовать в полной степени лучшим традициям проектирования свободного программного обеспечения. Принципиально возможно портирование приложения для использования в сочетании с семейством операционных систем Windows по причине кроссплатформенности абсолютного большинства используемых библиотек.

## 1.4 Краткий обзор используемых алгоритмов

Алгоритм обратного распространения ошибки – метод обучения многослойного перцептрона, основная идея которого состоит в распространении сигналов ошибки от выходов сети к ее входам, в противоположность прямому распространению сигналов в обычном режиме работы [4]. Алгоритм предполагает два прохода по всем слоям сети: прямого и обратного. При прямом проходе входной вектор подается на сенсорные узлы сети, после чего распространяется от слоя к слою. В результате генерируется набор выходных сигналов, который является фактической реакцией сети на данный входной образ. Во время обратного прохода все синаптические веса нейронов настраиваются в соответствии с правилом коррекции ошибок, в результате чего формируется сигнал ошибки. Этот сигнал распространяется по сети в направлении, обратном направлению синаптических связей [5].

Метод эластичного обратного распространения ошибки во многих случаях является более оптимальным выбором, чем обычный алгоритм обратного распространения ошибки. Плюс эластичного распространения в отсутствии необходимости подбирать параметры обучения, которые в случае с обратным распространением ошибки часто приводили к параличу сети.

Деревья принятия решений, также называемые деревьями классификации, используется для построения прогнозных моделей и классификации данных на основе разбиения исходных данных на подмножества, основанные на значениях атрибутов.

Основной идеей метода опорных векторов является перевод исходных данных в пространство большей размерности и поиска оптимальной разделяющей гиперплоскости в этом пространстве.

Метод К-средних – наиболее популярный из всех, используемых для кластеризации данных. Принцип его работы заключается в подборе положения центров кластеров таким образом, чтобы суммарное квадратичное отклонение точек от центров оказалось минимальным.

Метод Mean-shift основан на поиске максимумов функции плотности для наборов данных. Алгоритм автоматически устанавливает число кластеров и обычно используется в задачах компьютерного зрения и обработки изображений.

Плотностный алгоритм кластеризации, он же DBScan, решает проблемы разбиения данных на кластеры произвольной формы, причем было доказано, что в отличие от других алгоритмов кластеризации, которые обычно создают кластеры по форме близкие к сферическим, DBScan способен правильно распознать более большой спектр пространственных форм размещения данных.

## 1.6 Краткий обзор используемых наборов данных

В ходе подготовки к выполнению курсовой работы был проведен поиск подходящих для обработки наборов данных:

* Iris [9]

Классификация цветов по 4 признакам. Классический набор данных для задачи классификации, используется в приложении в качестве одного из наборов данных, предлагаемых для загрузки по умолчанию. Содержит примерно 150 значений.

* HIGGS [10]

Классификация микрочастиц и очистки данных от шума. Данные имеют 28 признаков и сгенерированы с помощью метода Монте-Карло. Число значений составляет довольно внушительную цифру 11 миллионов из которых последние 500 тысяч используются в качестве тестового набора.

* Computer Hardware Data Set [11]

Набор данных по сравнительной производительности процессоров, выраженной в 9 атрибутах. Применяется для задач регрессионного анализа, содержит около двухсот значений.

* Heart Disease Data Set [12]

База данных сердечных заболеваний, предполагается использование для задач классификации. Используется только 14 атрибутов из 75, содержащихся в официальной статистике. Примеров всего около трехсот.

* Auto MPG Data Set [13]

Набор данных, собранный из библиотеки Statlib, разработка которой ведется в университете Карнеги-Меллона. Содержит технический данные реальных автомобилей, вроде количества цилиндров и мощности. Всего 8 атрибутов и 400 примеров. Применяется в задачах регрессионного анализа, однако может использоваться и в задачах классификации и кластеризации.

* Million Song Dataset [14]

На основе данных о прослушиваниях, тексте и метках миллиона песен предлагается дать пользователю рекомендацию на основе его прослушиваний. Полный датасет занимает объем 280 гигабайт. Проводится конкурс на лучший алгоритм рекомендаций.

* Asteroids [15]

Данные по классификации 20 тысяч астероидов внутренней части солнечной системы. Предполагается определить, к какой виду и к какой группе относится конкретный астероид.

# Системное проектирование

Схема структурная *ГУИР.400201.1 СС*.

Рассмотрим программные модули приложения:

* Модуль «Пользовательский интерфейс»

Основной задачей данного модуля является обеспечение взаимодействия пользователя и ядра приложения. Включает в себя графический интерфейс, реализованный в Glade Interface Designer.

* Модуль «Загрузка данных»

Обеспечивает загрузку данных в приложение для их дальнейшего использования в алгоритмах машинного обучения. Содержит несколько тестовых наборов данных. Поддерживается формат csv. Обеспечивает подсчет возможных классов данных, определение бинарных наборов данных, получение срезов набора данных в различном процентном соотношении классов.

* Модуль «Построение графиков по классификаторам»

Модуль предназначен для построения графиков ROC кривых и матриц несоответствия для визуальной оценки результатов обучения классификатора.

* Модуль «Алгоритмы классификации»

Модуль алгоритмов классификации содержит в себе алгоритмы для выполнения классификации данных: алгоритм обратного распространения ошибки (Back propagation), алгоритм эластичного обратного распространения ошибки (Resilient propagation), алгоритм построения деревьев принятия решений, метод опорных векторов.

* Модуль «Алгоритмы кластеризации»

Модуль алгоритмов классификации содержит в себе алгоритмы для выполнения кластеризации данных: алгоритм К-средних (K-means), алгоритм среднего сдвига (Mean shift), плотностный алгоритм кластеризации (DB scan).

# Функциональное проектирование

Диаграмма классов *ГУИР.400201.3 ДК*.

Переречисислим основные функции приложения и дадим их краткую характеристику.

* Название функции: load\_CSV

Аргументы функции:

1. file\_path - путь к CSV файлу.

Возвращаемое значение: указатель на CSV файл.

Описание функции: данная функция открывает указанный файл для чтения, считывает все данные из CSV файла в массив и возвращает указатель на открытый файл.

* Название функции: get\_full\_data\_set

Аргументы функции:

1. num\_inputs - количество входов нейронной сети
2. num\_outputs - количество выходнов значений нейронной цепи (по умолчанию 1)

Возвращаемое значение: data\_set - 100% набор данных

Описание функции: данная функция получает на вход количество входных и выходных значений, загружает весь набор данных из CSV файла и возвращает этот набор как массив.

* Название функции: get\_levels

Аргументы функции: нет

Возвращаемое значение: возвращает массив уникальных классов для обучающей выборки.

Описание функции: данная функция совершает циклический проход по загруженному ранее набору данных (обучающей выборке), формирует и возвращает массив уникальных классов.

* Название функции: get\_data\_set\_array\_with\_specific\_level

Аргументы функции:

1. level - класс (уровень) из обучающей выборки

Возвращаемое значение: массив значений из обучающей выборки, имеющих класс равный level.

Описание функции: функция служит для выборки из набора данных тех значений, у которых класс равен level (аргумент функции).

* Название функции: count\_inputs

Аргументы функции:

1. number\_inputs - количество входов нейронной сети

Возвращаемое значение: описание функции: по загруженному файлу CSV определяет количество входов нейронной сети, если оно явно не установлено через аргумент функции number\_inputs. Иначе возвращает установленное пользователем значение.

* Название функции: buildNet

Аргументы функции:

1. hidden\_layers -  количество скрытых слоев нейронной сети.
2. num\_inputs - количество входов нейронной сети.
3. num\_outputs - количество выходов нейронной сети.
4. hiddenclass - функция (класс) активации.

Возвращаемое значение: класс - нейронная сеть

Описание функции: данная функция создает класс нейронной сети, используя стандартную библиотеку Pybrain.

* Название функции: show\_CSV

Аргументы функции: нет

Возвращаемое значение: нет

Описание функции: функция выводит в виде таблицы загруженный ранее файл CSV. Вывод осуществляется в консоль. Метод предназначен для проверки содержимого фала.

* Название функции: is\_binary.

Аргументы функции: нет

Возвращаемое значение: true или false

Описание функции: функция возвращает true, если загруженный ранее файл CSV является бинарной выборкой (т.е. содержит только 2 класса), false - иначе.

* Название функции: get\_data\_set

Аргументы функции:

1. percent - процент обучающей выборки от всего набора данных.
2. num\_inputs -  количество входов.
3. num\_outputs - количество выходов (по умолчанию 1).

Возвращаемое значение: выборка из набора данных или весь набор данных (если указанно 100%).

Описание функции: возвращает выборку из набора данных (в зависимости от указанных процентов). Выбирает равное число данных из всех доступных классов, формирует и возвращает массив данных.

* Название функции: getResult

Аргументы функции:

1. predict – функция-классификатор
2. data\_set - набор данных (массив)

Возвращаемое значение: возвращает текст с результатами оценочных тестов.

Описание функции: формирует и возвращает текст, содержащий различные показатели для алгоритма, а также результаты, погрешности и ошибки проведенных тестов.

* Название функции: train

Аргументы функции:

1. cycles - число циклов обучения.
2. percent - процент обучающей выборки от всего набора данных.
3. hidden\_layers - число скрытых слоев.
4. hiddenclass - функция (класс) активации.
5. num\_outputs - количество входов нейронной сети.
6. num\_inputs - количество выходов нейронной сети.

Возвращаемое значение: класс тренированной нейронной цепи.

Описание функции: функция осуществляет тренировку нейронной цепи заданное количество циклов и на заданной обучающей выборке. Для обучения использует выбранную функцию активации. Есть возможность изменить число скрытых слоев нейронной сети.

# Разработка программного модуля

Блок-схема алгоритма получения набора данных с равномерным распределением значений по классам и заданным процентным соотношением относительно исходного набора данных *ГУИР.400201.2 БС*.

Данный алгоритм позволяет гарантировать равномерное обучение классификатора благодаря получению сбалансированного набора данных.

Алгоритм получает на вход значение типа integer, обозначающее желаемый объем результирующего набора данных относительно исходного (от 1% до 100%) .

Результатом работы алгоритма является искомый набор данных в виде объекта класса SupervisedDataSet из библиотеки PyBrain, который содержит ассоциативный массив, состоящий из массива входных данных «input» и массива выходных данных «target».

Словесное описание алгоритма:

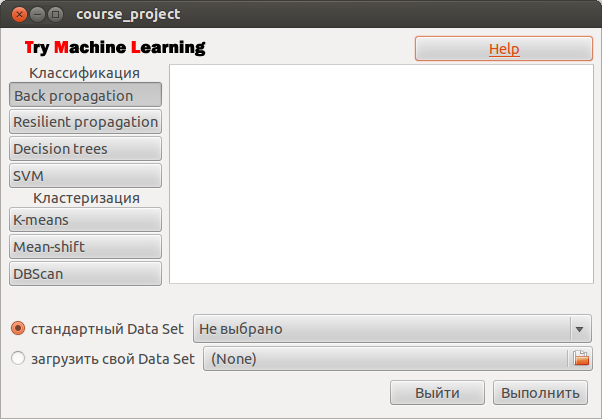
1. Проверить переданное в функцию значение процентов на корректность (оно не должно быть меньше 0 и больше 100)
2. Если передан параметр 100%
   1. Вызвать функцию получения полного набора данных и вернуть полученное значение
3. Получить возможные классы выходных значений
4. Рассчитать общее число значений и число значений на каждый класс в результирующем наборе данных
5. Для каждого класса выходных данных из полученных ранее
   1. Получить частичный набор данных, состоящий только из значений текущего класса
   2. Пока не достигнуто необходимое число значений в классе
      1. Сгенерировать случайное число – номер случайного набора значений в наборе данных текущего класса
      2. Если полученное значение ранее не добавлялось в результирующий набор данных
         1. Добавить номер значение в список уже сгенерированных номеров
         2. Получить значение под текущим номером из набора данных текущего класса
         3. Добавить полученное значение в результирующий набор данных
6. Вернуть результирующий набор данных

Данный алгоритм получения набора данных с равномерным распределением значений по классам используется для всех алгоритмов классификации, присутствующих в приложении. Именно его использование позволяет избежать известной ошибки слишком узкого обучения классификаторов, в результате которой классификатор не имеет возможности точно определить класс данных из тестовой выборки.

# **Руководство пользователя**

Для начала использования приложения необходимо установить в систему Python-библиотеки Scikit-learn, Pybrain и их зависимости: NumPy, Matplotlib, SciPy. Предпочтительно использование Python версии 2.7.6.

Для запуска графического интерфейса приложения необходимо использовать файл main\_GUI.py. При запуске появляется GTK окно, показанное на рисунке 5.1.

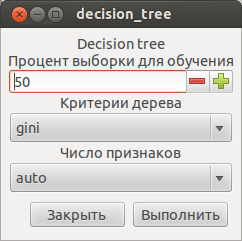


*Рис. 5.1. Главное окно приложения*

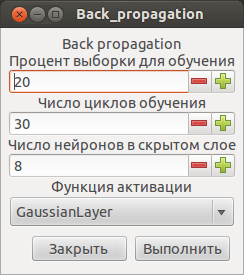
Основной интерфейс приложения доступен из главного окна: слева от панели вывода результатов анализа можно заметить 7 кнопок выбора текущего алгоритма обучения, разделенных на группы «Классификация» и «Кластеризация». В нижней части окна сосредоточены средства выбора набора данных и кнопки управления приложением «Выйти» и «Выполнить».

При выборе набора данных возможна загрузка одного из идущих в комплекте с приложением датасетов, либо выбор своего набора данных из имеющихся на диске файлов формата csv.

При выборе одного из алгоритмов и загрузке набора данных одним из предложенных вариантов следует запустить выполнение целевого алгоритма кнопкой «Выполнить». При этом будет открыто диалоговое окно установки параметров алгоритма машинного обучения. На рисунке 5.2 изображено диалоговое окно метода деревьев принятия решений, на рисунке 5.3 – алгоритма обратного распространения ошибки.

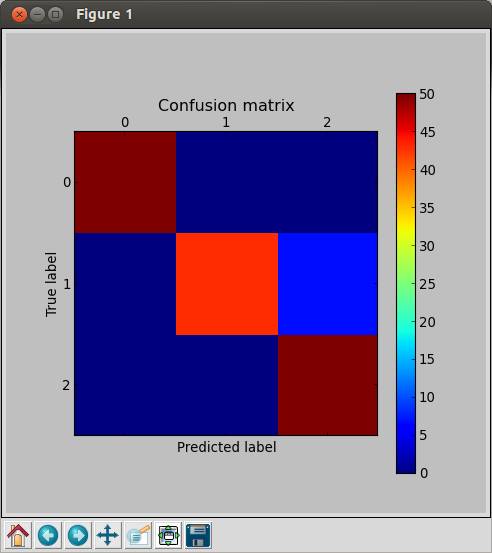


*Рис. 5.2 – Окно установки параметров алгоритма деревьев принятия решений*



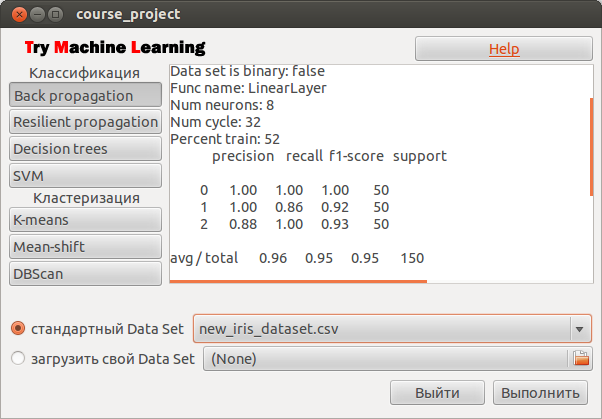
*Рис. 5.3 – Окно установки параметров алгоритма обратного распространения ошибки*

После установки параметров алгоритма в диалоговом окне следует запустить его на выполнение нажатием кнопки «Выполнить». При этом будет запущено построение графических метрик оценки алгоритма. На рисунке 5.4 изображена матрица несоответствий, представляющая визуально отклонения классификатора от правильных результатов.

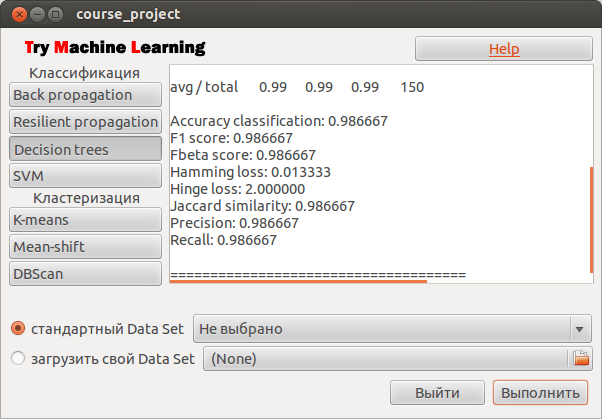


*Рис. 4 – Матрица несоответствий*

После закрытия окон с визуальными метриками алгоритмов машинного обучения в панель вывода результатов анализа главного окна приложения будут выведены все представленные в текущем методе классификации либо кластеризации числовые метрики оценки качества. На рисунке 5.5 изображено главное окно приложения после завершения работы алгоритма обратного распространения ошибки, на рисунке 5.6 – после завершения алгоритма построения деревьев принятия решений.



*Рис. 5.5 – Главное окно приложения после завершения алгоритма обратного распространения ошибки*



*Рис. 5.6 – Главное окно приложения после завершения алгоритма построения деревьев принятия решений*

# Заключение

В ходе выполнения данного курсовой работы было спроектировано и разработано приложение для тестирования алгоритмов машинного обучения на произвольных наборах данных и оценки их качества. В ходе тестирования приложения была подтверждена возможность его использования для заявленных в работе целей.

В работе над проектом использовалась система контроля версий git и удаленный репозиторий Github. По данным системы Github, во время работы над проектом было сделано около 90 коммитов кода. Системы управления проектами задействованы не были ввиду их низкой эффективности в условиях команд с малой численностью. В качестве основной среды разработки использовалась IDE PyCharm Community Edition версии 3.1.3 от компании JetBrains.

В качестве основного языка разработки использовался Python версии 2.7.6. Для вспомогательных задач, например, для написания скриптов для подготовки наборов данных к работе с программой, использовался язык Ruby версии 2.1.1 – последней стабильной на момент написания работы.

Одним из наиболее весомых преимуществ приложения является наличие строгого и эргономичного графического интерфейса для взаимодействия с пользователями, обладающими начальным уровнем компьютерной грамотности. Однако предполагается, что использоваться приложение будет, в большей степени, людьми подготовленными. Обеспечивается это, в первую очередь, выбором в качестве целевой среды выполнения семейства Debian-based дистрибутивов Linux. Тем не менее, интуитивная понятность графического интерфейса программы позволит всем категориям потенциальных пользователей взаимодействовать с приложением с максимальным уровнем комфорта и добиться быстрого выполнения поставленных задач.

Разработанный программный продукт соответствует всем основным требованиям, предъявляемым к курсовой работе, позволяет быстро и с затратой минимума усилий подобрать наиболее оптимальный в работе с конкретным набором данных алгоритм машинного обучения, обеспечивает пользователя графическим интерфейсом, полностью покрывающим все основные варианты использования программного продукта. Все это позволяет считать цель работы с успехом достигнутой, программный продукт следует считать готовым к применению в реальных задачах.

# Список литературы

[1] S. Russell, P. Norvig. Artifical Intelligence. A Modern Approach. – Prentice Hall, 2010. – 1153 с.

[2] А. А. Ежов, С. А. Шумский «Нейрокомпьютинг и его применения в экономике и бизнесе» – Москва, 1998. – 216 с.

[3] М. Лутц. Изучаем Python. – СПб Символ-Плюс, 2011 – 1280 с.

[4] Описание продукта MATLAB [Электронный ресурс]. – Электронные данные. – Режим доступа : http://mathworks.com/products/matlab/

[5] Описание продукта Statistica Automated Neural Networks [Электронный ресурс]. – Электронные данные. – Режим доступа : http://statsoft.com/products/statistica-automated-neural-networks/

[6] Описание продукта Membrain [Электронный ресурс]. – Электронные данные. – Режим доступа : http://membrain-nn.de /english/details\_en.htm

[7] Метод обратного распространения ошибки [Электронный ресурс]. – Электронные данные. – Режим доступа : http://ru.wikipedia.org/wiki/ Метод обратного\_ распространения\_ошибки

[8] С. Хайкин «Нейронные сети. Полный курс» – Москва Вильямс, 2006. – 1105 с.

[9] Iris [Электронный ресурс]. – Электронные данные. – Режим доступа : http://archive.ics.uci.edu/ml/datasets/Iris

[10] HIGGS [Электронный ресурс]. – Электронные данные. – Режим доступа : http://archive.ics.uci.edu/ml/datasets/HIGGS

[11] Computer Hardware Data Set [Электронный ресурс]. – Электронные данные. – Режим доступа : http://archive.ics.uci.edu/ml/datasets/Computer+Hardware

[12] Heart Disease Data Set [Электронный ресурс]. – Электронные данные. – Режим доступа : http://archive.ics.uci.edu/ml/datasets/Heart+Disease

[13] Auto MPG Data Set [Электронный ресурс]. – Электронные данные. – Режим доступа : http://archive.ics.uci.edu/ml/datasets/Auto+MPG

[14] Million Song Dataset [Электронный ресурс]. – Электронные данные. – Режим доступа : http://labrosa.ee.columbia.edu/millionsong/

[15] Asteroids [Электронный ресурс]. – Электронные данные. – Режим доступа : http://www.freebase.com/astronomy/asteroid

# Приложение

***\_interface.py***

# coding=utf-8

# module \_interface

# =======================================================

# IMPORTS

# =======================================================

from abc import ABCMeta, abstractmethod

import csv

from random import randint

from pybrain.tools.shortcuts import buildNetwork

from pybrain.datasets import SupervisedDataSet

import control

import sklearn.metrics as metrics

# =======================================================

# Classifier Interface

# =======================================================

class InterfaceML:

\_\_metaclass\_\_ = ABCMeta

# private class variables

\_\_row\_count = None

\_\_binary = False

\_data\_set = None

\_csv\_file = None

\_hFile = None

# private class methods

def \_\_get\_full\_data\_set(self, num\_inputs=-1, num\_outputs=1):

num\_inputs = self.\_count\_inputs() if num\_inputs == -1 else num\_inputs

if num\_inputs <= 0 or num\_outputs <= 0:

return

self.\_\_data\_set = SupervisedDataSet(num\_inputs, num\_outputs)

self.\_hFile.seek(0)

for row in self.\_csv\_file:

indata = row[:num\_inputs]

outdata = row[num\_inputs:]

self.\_\_data\_set.addSample(indata, outdata)

return self.\_\_data\_set

def \_\_get\_levels(self):

levels = []

for i in self.\_\_data\_set['target']:

level = int(i)

if level not in levels:

levels.append(level)

return levels

def \_\_get\_data\_set\_array\_with\_specific\_level(self, level):

result = []

for i in self.\_\_data\_set:

current\_level = int(i[1])

if current\_level == level:

result.append(i)

return result

# protected class methods

def \_count\_inputs(self, number\_inputs=-1):

self.\_hFile.seek(0)

for row in self.\_csv\_file:

return len(row) - 1 if number\_inputs == -1 else number\_inputs

return 0

def \_buildNet(self, hidden\_layers, num\_outputs, num\_inputs, hiddenclass):

return buildNetwork(num\_inputs, hidden\_layers, num\_outputs)

# public class methods

def load\_CSV(self, file\_name):

self.\_hFile = open(file\_name, 'rb')

self.\_csv\_file = csv.reader(self.\_hFile)

self.row\_count = sum(1 for row in self.\_csv\_file)

self.\_data\_set = self.\_\_get\_full\_data\_set()

if len(self.\_\_get\_levels()) == 2:

self.\_\_binary = True

return self.\_csv\_file

# test method

def show\_CSV(self):

self.\_hFile.seek(0)

for row in self.\_csv\_file:

print row

def is\_binary(self):

return self.\_\_binary

def get\_data\_set(self, percent, num\_inputs=-1, num\_outputs=1):

num\_inputs = self.\_count\_inputs() if num\_inputs == -1 else num\_inputs

if num\_inputs <= 0 or num\_outputs <= 0 or percent <= 0 or percent > 100:

return

if percent == 100:

return self.\_\_data\_set

levels = self.\_\_get\_levels()

ds = SupervisedDataSet(num\_inputs, num\_outputs)

num\_rows = int(self.row\_count \* (float(percent) / 100.0))

num\_rows\_in\_class = int(num\_rows / len(levels))

for level in levels:

data\_set = self.\_\_get\_data\_set\_array\_with\_specific\_level(level)

used\_samples = []

for i in range(num\_rows\_in\_class):

added = False

while (not added):

row = randint(0, num\_rows\_in\_class)

if row not in used\_samples:

used\_samples.append(row)

sample = data\_set[row]

ds.addSample(sample[0], sample[1])

added = True

return ds

def getResult(self, predict, data\_set):

y\_true, y\_predict = control.calculate\_entire\_ds(predict, data\_set)

result = metrics.classification\_report(y\_true, y\_predict)

result += "\nAccuracy classification: %f\n" % metrics.accuracy\_score(y\_true, y\_predict)

result += "F1 score: %f\n" % metrics.f1\_score(y\_true, y\_predict)

result += "Fbeta score: %f\n" % metrics.fbeta\_score(y\_true, y\_predict, beta=0.5)

result += "Hamming loss: %f\n" % metrics.hamming\_loss(y\_true, y\_predict)

result += "Hinge loss: %f\n" % metrics.hinge\_loss(y\_true, y\_predict)

result += "Jaccard similarity: %f\n" % metrics.jaccard\_similarity\_score(y\_true, y\_predict)

result += "Precision: %f\n" % metrics.precision\_score(y\_true, y\_predict)

result += "Recall: %f\n" % metrics.recall\_score(y\_true, y\_predict)

if self.is\_binary():

result += "Average precision: %f\n" % metrics.average\_precision\_score(y\_true, y\_predict)

result += "Matthews correlation coefficient: %f\n" % metrics.matthews\_corrcoef(y\_true, y\_predict)

result += "Area Under the Curve: %f\n" % metrics.roc\_auc\_score(y\_true, y\_predict)

return result

# abstract methods

@abstractmethod

def train(self,

cycles,

percent,

hidden\_layers=3,

hiddenclass=None,

num\_outputs=1,

num\_inputs=-1):

"""train net

:param cycles:

:param percent:

:param hidden\_layers:

:param num\_outputs:

:param num\_inputs:

:param hiddenclass:

"""

***back\_propagation.py***

# coding=utf-8

# =======================================================

# IMPORTS

# =======================================================

from pybrain.supervised.trainers import BackpropTrainer

import \_interface

# =======================================================

# Back propagation

# =======================================================

class Backprop(\_interface.InterfaceML):

def train(self,

cycles,

percent,

hidden\_layers=3,

hiddenclass=None,

num\_outputs=1,

num\_inputs=-1):

num\_inputs = self.\_count\_inputs() if num\_inputs == -1 else num\_inputs

if num\_inputs <= 0 or num\_outputs <= 0 or cycles <= 0 or (percent > 100 or percent <= 0):

return

network = self.\_buildNet(hidden\_layers, num\_outputs, num\_inputs, hiddenclass)

data\_set = self.get\_data\_set(percent, num\_inputs, num\_outputs)

trainer = BackpropTrainer(network, data\_set)

for i in range(cycles):

trainer.train()

return network

\_b = Backprop()

train = \_b.train

getResult = \_b.getResult

load\_CSV = \_b.load\_CSV

get\_data\_set = \_b.get\_data\_set

show\_CSV = \_b.show\_CSV

is\_binary = \_b.is\_binary

***control.py***

# coding=utf-8

# module control

# =======================================================

# IMPORTS

# =======================================================

import pylab as pl

from sklearn.metrics import confusion\_matrix, roc\_curve, auc

# =======================================================

# CONTROL

# =======================================================

class Control:

# public methods

def calculate\_entire\_ds(self, function, data\_set):

y\_true = []

y\_predict = []

for i in data\_set:

predict = int(round(function(i[0])))

true = int(i[1][0])

y\_predict.append(predict)

y\_true.append(true)

# print "{} {}".format(predict,true)

# print y\_true

# print y\_predict

return y\_true, y\_predict

def draw\_confusion\_matrix(self, function, data\_set):

y\_true, y\_predict = self.calculate\_entire\_ds(function, data\_set)

cm = confusion\_matrix(y\_true, y\_predict)

pl.matshow(cm)

pl.title('Confusion matrix')

pl.colorbar()

pl.ylabel('True label')

pl.xlabel('Predicted label')

pl.show()

def draw\_roc(self, function, data\_set):

y\_true, y\_predict = self.calculate\_entire\_ds(function, data\_set)

# print y\_true

# print y\_predict

fpr, tpr, thresholds = roc\_curve(y\_true, y\_predict)

roc\_auc = auc(fpr, tpr)

print("Area under the ROC curve : %f" % roc\_auc)

# Plot ROC curve

pl.clf()

pl.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc\_auc)

pl.plot([0, 1], [0, 1], 'k--')

pl.xlim([0.0, 1.0])

pl.ylim([0.0, 1.0])

pl.xlabel('False Positive Rate')

pl.ylabel('True Positive Rate')

pl.title('Receiver operating characteristic example')

pl.legend(loc="lower right")

pl.show()

\_control = Control()

draw\_confusion\_matrix = \_control.draw\_confusion\_matrix

draw\_roc = \_control.draw\_roc

calculate\_entire\_ds = \_control.calculate\_entire\_ds

***DBScan.py***

# coding=utf-8

# =======================================================

# IMPORTS

# =======================================================

import numpy as np

import pylab as pl

import mpl\_toolkits.mplot3d.axes3d as p3

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import DBSCAN

from sklearn import metrics

# import files

import \_interface

# =======================================================

# DBSCAN clustering algorithm

# =======================================================

class DBScanC(\_interface.InterfaceML):

\_\_n\_clusters = None

\_\_cluster\_centers = None

\_\_data = None

\_\_labels = None

\_\_labels\_true = None

\_\_core\_samples = None

def \_\_set\_data(self):

if len(self.\_data\_set['target'][0]) != 1:

return

self.\_\_data = np.array(self.\_data\_set['input'])

self.\_\_data = StandardScaler().fit\_transform(self.\_\_data)

self.\_\_labels\_true = np.ravel(self.\_data\_set['target'])

def train(self, m\_eps=0.4, m\_min\_sales=10):

self.\_\_set\_data()

db = DBSCAN(eps=m\_eps, min\_samples=m\_min\_sales).fit(self.\_\_data)

self.\_\_core\_samples = db.core\_sample\_indices\_

self.\_\_labels = db.labels\_

self.\_\_n\_clusters = len(set(self.\_\_labels)) - (1 if -1 in self.\_\_labels else 0)

def getResult(self):

text = ('Estimated number of clusters: %d\n' % self.\_\_n\_clusters)

text += ("Homogeneity: %0.3f\n" % metrics.homogeneity\_score(self.\_\_labels\_true, self.\_\_labels))

text += ("Completeness: %0.3f\n" % metrics.completeness\_score(self.\_\_labels\_true, self.\_\_labels))

text += ("V-measure: %0.3f\n" % metrics.v\_measure\_score(self.\_\_labels\_true, self.\_\_labels))

text += ("Adjusted Rand Index: %0.3f\n"

% metrics.adjusted\_rand\_score(self.\_\_labels\_true, self.\_\_labels))

text += ("Adjusted Mutual Information: %0.3f\n"

% metrics.adjusted\_mutual\_info\_score(self.\_\_labels\_true, self.\_\_labels))

text += ("Silhouette Coefficient: %0.3f"

% metrics.silhouette\_score(self.\_\_data, self.\_\_labels))

return text

# print('Estimated number of clusters: %d' % self.\_\_n\_clusters)

# print("Homogeneity: %0.3f" % metrics.homogeneity\_score(self.\_\_labels\_true, self.\_\_labels))

# print("Completeness: %0.3f" % metrics.completeness\_score(self.\_\_labels\_true, self.\_\_labels))

# print("V-measure: %0.3f" % metrics.v\_measure\_score(self.\_\_labels\_true, self.\_\_labels))

# print("Adjusted Rand Index: %0.3f"

# % metrics.adjusted\_rand\_score(self.\_\_labels\_true, self.\_\_labels))

# print("Adjusted Mutual Information: %0.3f"

# % metrics.adjusted\_mutual\_info\_score(self.\_\_labels\_true, self.\_\_labels))

# print("Silhouette Coefficient: %0.3f"

# % metrics.silhouette\_score(self.\_\_data, self.\_\_labels))

# labels\_true = np.ravel(data\_set['target'].astype(np.int))

# labels\_predict = ms.labels\_.astype(np.int)

# print metrics.completeness\_score(labels\_true, labels\_predict)

# print metrics.homogeneity\_score(labels\_true, labels\_predict)

# print metrics.mutual\_info\_score(labels\_true, labels\_predict)

def showPlot2D(self):

unique\_labels = set(self.\_\_labels)

colors = pl.cm.Spectral(np.linspace(0, 1, len(unique\_labels)))

for k, col in zip(unique\_labels, colors):

if k == -1:

# Black used for noise.

col = 'k'

markersize = 6

class\_members = [index[0] for index in np.argwhere(self.\_\_labels == k)]

cluster\_core\_samples = [index for index in self.\_\_core\_samples

if self.\_\_labels[index] == k]

for index in class\_members:

x = self.\_\_data[index]

if index in self.\_\_core\_samples and k != -1:

markersize = 14

else:

markersize = 6

pl.plot(x[0], x[1], 'o', markerfacecolor=col,

markeredgecolor='k', markersize=markersize)

pl.title('Estimated number of clusters: %d' % self.\_\_n\_clusters)

pl.show()

def showPlot3D(self):

fig = pl.figure()

ax = p3.Axes3D(fig)

ax.view\_init(7, -80)

for l in np.unique(self.\_\_labels):

ax.plot3D(self.\_\_data[self.\_\_labels == l, 0], self.\_\_data[self.\_\_labels == l, 1], self.\_\_data[self.\_\_labels == l, 2],

'o', color=pl.cm.jet(float(l) / np.max(self.\_\_labels + 1)))

pl.title("Number of estimated clusters : %d" % self.\_\_n\_clusters)

pl.show()

\_d = DBScanC()

train = \_d.train

getResult = \_d.getResult

showPlot2D = \_d.showPlot2D

showPlot3D = \_d.showPlot3D

load\_CSV = \_d.load\_CSV

get\_data\_set = \_d.get\_data\_set

show\_CSV = \_d.show\_CSV

***decision\_tree.py***

# coding=utf-8

# =======================================================

# IMPORTS

# =======================================================

from sklearn.tree import DecisionTreeClassifier

import \_interface

# =======================================================

# Decision Tree

# =======================================================

class DTree(\_interface.InterfaceML):

def train(self,

percent,

criterion="gini",

max\_features=None,

num\_outputs=1,

num\_inputs=-1):

num\_inputs = self.\_count\_inputs() if num\_inputs == -1 else num\_inputs

data\_set = self.get\_data\_set(percent, num\_inputs, num\_outputs)

clf = DecisionTreeClassifier(criterion=criterion, max\_features=max\_features).fit(data\_set['input'], data\_set['target'])

return clf

\_d = DTree()

train = \_d.train

getResult = \_d.getResult

load\_CSV = \_d.load\_CSV

get\_data\_set = \_d.get\_data\_set

show\_CSV = \_d.show\_CSV

is\_binary = \_d.is\_binary

***kmeans.py***

\_\_author\_\_ = 'belov'

from time import time

import numpy as np

import pylab as pl

from sklearn import metrics

from sklearn.cluster import KMeans

from sklearn.decomposition import PCA

from sklearn.preprocessing import scale

# import files

import \_interface

# =======================================================

# K-Means clustering algorithm

# =======================================================

class Kmeans(\_interface.InterfaceML):

\_\_data = None

\_\_labels = None

\_\_n\_samples = None

\_\_n\_features = None

\_\_n\_digits = None

def \_\_set\_data(self):

if len(self.\_data\_set['target'][0]) != 1:

return

self.\_\_data = self.\_data\_set['input']

self.\_\_n\_samples, \_\_n\_features = self.\_\_data.shape

self.\_\_n\_digits = len(np.unique(self.\_data\_set['target']))

self.\_\_labels = np.ravel(self.\_data\_set['target'])

def train(self, init = "k-means++", n\_init = 10):

self.\_\_set\_data()

km = KMeans(init=init, n\_clusters=self.\_\_n\_digits, n\_init=n\_init)

return km

def getResult(self, estimator):

t0 = time()

data = self.\_\_data

estimator.fit(data)

text = ('Time: %.2fs\n Inertia: %i\n Homogeneity: %.3f\n Completness: %.3f\n '

'V-measure: %.3f\n Adjusted Rand Index: %.3f\n Adjusted Mutual Information: %.3f\n '

'Silhouette Coefficient: %.3f\n'

% ((time() - t0), estimator.inertia\_,

metrics.homogeneity\_score(self.\_\_labels, estimator.labels\_),

metrics.completeness\_score(self.\_\_labels, estimator.labels\_),

metrics.v\_measure\_score(self.\_\_labels, estimator.labels\_),

metrics.adjusted\_rand\_score(self.\_\_labels, estimator.labels\_),

metrics.adjusted\_mutual\_info\_score(self.\_\_labels, estimator.labels\_),

metrics.silhouette\_score(data, estimator.labels\_, metric='euclidean', sample\_size=300)))

return text

def showPlot(self, km):

reduced\_data = PCA(n\_components=2).fit\_transform(self.\_\_data)

kmeans = km

kmeans.fit(reduced\_data)

# Step size of the mesh. Decrease to increase the quality of the VQ.

h = .02 # point in the mesh [x\_min, m\_max]x[y\_min, y\_max].

# Plot the decision boundary. For that, we will assign a color to each

x\_min, x\_max = reduced\_data[:, 0].min() - 2, reduced\_data[:, 0].max() + 2

y\_min, y\_max = reduced\_data[:, 1].min() - 2, reduced\_data[:, 1].max() + 2

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h), np.arange(y\_min, y\_max, h))

# Obtain labels for each point in mesh. Use last trained model.

Z = kmeans.predict(np.c\_[xx.ravel(), yy.ravel()])

# Put the result into a color plot

Z = Z.reshape(xx.shape)

pl.figure(1)

pl.clf()

pl.imshow(Z, interpolation='nearest',

extent=(xx.min(), xx.max(), yy.min(), yy.max()),

cmap=pl.cm.Paired,

aspect='auto', origin='lower')

pl.plot(reduced\_data[:, 0], reduced\_data[:, 1], 'k.', markersize=2)

# Plot the centroids as a white X

centroids = kmeans.cluster\_centers\_

pl.scatter(centroids[:, 0], centroids[:, 1],

marker='x', s=169, linewidths=3,

color='w', zorder=10)

pl.title('K-means clustering on the digits dataset (PCA-reduced data)\n'

'Centroids are marked with white cross')

pl.xlim(x\_min, x\_max)

pl.ylim(y\_min, y\_max)

pl.xticks(())

pl.yticks(())

pl.show()

\_m = Kmeans()

train = \_m.train

getResult = \_m.getResult

showPlot = \_m.showPlot

load\_CSV = \_m.load\_CSV

get\_data\_set = \_m.get\_data\_set

show\_CSV = \_m.show\_CSV

***mshift.py***

# coding=utf-8

# =======================================================

# IMPORTS

# =======================================================

import numpy as np

import pylab as pl

from sklearn.cluster import MeanShift, estimate\_bandwidth

import mpl\_toolkits.mplot3d.axes3d as p3

from sklearn import metrics

# import files

import \_interface

# =======================================================

# Mean-shift clustering algorithm

# =======================================================

class MSC(\_interface.InterfaceML):

\_\_n\_clusters = None

\_\_cluster\_centers = None

\_\_data = None

\_\_labels = None

\_\_labels\_true = None

def \_\_set\_data(self):

if len(self.\_data\_set['target'][0]) != 1:

return

self.\_\_data = np.array(self.\_data\_set['input'])

self.\_\_labels\_true = np.ravel(self.\_data\_set['target'])

# have to change the name of func train

def train(self, m\_quantile=0.15, m\_cluster\_all=True):

self.\_\_set\_data()

bandwidth = estimate\_bandwidth(self.\_\_data, quantile=m\_quantile) #, n\_samples=500

ms = MeanShift(bandwidth=bandwidth, bin\_seeding=True, cluster\_all=m\_cluster\_all)

ms.fit(self.\_\_data)

self.\_\_labels = ms.labels\_

self.\_\_cluster\_centers = ms.cluster\_centers\_

labels\_unique = np.unique(self.\_\_labels)

self.\_\_n\_clusters = len(labels\_unique)

return ms

def getResult(self):

text = ('Estimated number of clusters: %d\n' % self.\_\_n\_clusters)

text += ("Homogeneity: %0.3f\n" % metrics.homogeneity\_score(self.\_\_labels\_true, self.\_\_labels))

text += ("Completeness: %0.3f\n" % metrics.completeness\_score(self.\_\_labels\_true, self.\_\_labels))

text += ("V-measure: %0.3f\n" % metrics.v\_measure\_score(self.\_\_labels\_true, self.\_\_labels))

text += ("Adjusted Rand Index: %0.3f\n"

% metrics.adjusted\_rand\_score(self.\_\_labels\_true, self.\_\_labels))

text += ("Adjusted Mutual Information: %0.3f\n"

% metrics.adjusted\_mutual\_info\_score(self.\_\_labels\_true, self.\_\_labels))

text += ("Silhouette Coefficient: %0.3f"

% metrics.silhouette\_score(self.\_\_data, self.\_\_labels))

return text

def showPlot3D(self):

fig = pl.figure()

ax = p3.Axes3D(fig)

ax.view\_init(7, -80)

for l in np.unique(self.\_\_labels):

ax.plot3D(self.\_\_data[self.\_\_labels == l, 0], self.\_\_data[self.\_\_labels == l, 1], self.\_\_data[self.\_\_labels == l, 2],

'o', color=pl.cm.jet(float(l) / np.max(self.\_\_labels + 1)))

pl.title("Number of estimated clusters : %d" % self.\_\_n\_clusters)

pl.show()

\_m = MSC()

train = \_m.train

getResult = \_m.getResult

showPlot3D = \_m.showPlot3D

load\_CSV = \_m.load\_CSV

get\_data\_set = \_m.get\_data\_set

show\_CSV = \_m.show\_CSV

***resilient\_propagation.py***

# coding=utf-8

# =======================================================

# IMPORTS

# =======================================================

from pybrain.supervised.trainers import RPropMinusTrainer

import \_interface

# =======================================================

# Resilient propagation

# =======================================================

class Rprop(\_interface.InterfaceML):

def train(self,

cycles,

percent,

hidden\_layers=3,

hiddenclass=None,

num\_outputs=1,

num\_inputs=-1):

num\_inputs = self.\_count\_inputs() if num\_inputs == -1 else num\_inputs

if num\_inputs <= 0 or num\_outputs <= 0 or cycles <= 0 or (percent > 100 or percent <= 0):

return

network = self.\_buildNet(hidden\_layers, num\_outputs, num\_inputs, hiddenclass)

data\_set = self.get\_data\_set(percent, num\_inputs, num\_outputs)

trainer = RPropMinusTrainer(network, dataset=data\_set)

for i in range(cycles):

trainer.train()

return network

\_rp = Rprop()

train = \_rp.train

getResult = \_rp.getResult

load\_CSV = \_rp.load\_CSV

get\_data\_set = \_rp.get\_data\_set

show\_CSV = \_rp.show\_CSV

is\_binary = \_rp.is\_binary

***support\_vector.py***

# coding=utf-8

# =======================================================

# IMPORTS

# =======================================================

import numpy as np

from sklearn import svm

import \_interface

kernels = ["rbf", "linear", "poly", "sigmoid", "precomputed"]

# =======================================================

# Support Vector Machines

# =======================================================

class SVM(\_interface.InterfaceML):

def train(self,

percent,

num\_outputs=1,

num\_inputs=-1,

kernel = "rbf",

hiddenclass=None):

num\_inputs = self.\_count\_inputs() if num\_inputs == -1 else num\_inputs

data\_set = self.get\_data\_set(percent, num\_inputs, num\_outputs)

clf = svm.SVC().fit(data\_set['input'], np.ravel(data\_set['target']))

return clf

\_s = SVM()

train = \_s.train

getResult = \_s.getResult

load\_CSV = \_s.load\_CSV

get\_data\_set = \_s.get\_data\_set

show\_CSV = \_s.show\_CSV

is\_binary = \_s.is\_binary

