

ТЕХНИЧЕСКИ УНИВЕРСИТЕТ-СОФИЯ

Факултет по компютърни системи и технологии

Специалност: "Компютърно и софтуерно инженерство"

ДИПЛОМНА РАБОТА НА ТЕМА

**Класификация и кълстерилизация на звукови файлове с
използване на методи от дълбоко самообучение и
генетични алгоритми.**

Изготвил: Мария Влахова

Специалност: Компютърно и софтуерно инженерство

Курс: магистър

Фак. № 121316084

Съдържание

1. Резюме
2. Постановка на проблема, цели и задачи
3. Преглед на областта
 - Data Mining
 - *Data Mining и процесът на вземане на решение*
 - Източници и тенденции в растежа на обема на данните
 - Данни, информация, знание
 - Характеристики на данните, използвани в Data Mining
 - Задачи на Data Mining
 - Методи на Data Mining
 - Стратегии на обучение на модела
 - Класификация
 - Кълстерилизация
4. Методи и алгоритми за разпознаване и клъстеризиране на звукови сигнали
 - Данни и характеристики
 - Алгоритъм на програмата
 - Извличане и разпределение на данните
 - Извличане на характеристики
 - Модели за класификация
 - Модели за кълстерилизация
5. Програмна реализация
6. Анализ на експерименталните резултати
7. Заключение
8. Приложения

Резюме

Целта на този проект е създаване на класификатор и клъстериизатор за звукови файлове(wav формат)с използването на питон и либроса библиотека.За момента звуците се класифицират в 6 категории:Акустична китара,аплодисменти,лай,барабани ,звук от автобус и други.Най-добрая постигнат резултат за класификация е чрез използване на невронна мрежа-accuracy: 90.17%,а за клъстериизация е в 6 категории.

Постановка на проблема,цели и задачи

Звуковите файлове са една от областите ,в която тепърва ще навлиза изпозването на изкуствения интелект.Това е причината да бъде израната задача свързана с обработката на звукови формати.Поради необятността на звуците, няма надеждни автоматични системи за аудио маркиране с общо предназначение.По настоящем са необходими много ръчни усилия за задачи като анотиране на звукови колекции и предоставяне на надписи за нередуциращи събития в аудиовизуално съдържание.За да се справи с този проблем, ¹Freesound (инициатива на MTG-UPF, поддържаща база данни с над 370 000 Creative Commons Licensed Sounds) и екипа за изследване на машините на Google (създателите на AudioSet - мащабен набор от ръчно анотирани аудио събития с над 500 класове) се обединяват, за да създават подходящ датасет.

Предизвикателството е да се създаде система за автоматична аудио маркиране с общо предназначение и система за автоматично групиране на база на вътрешно сходство, като се използват набор от данни от аудио

¹ Уеб-страница:Freesound General-Purpose Audio Tagging Challenge извлечено на 28/12/2018 г.
<https://www.kaggle.com/c/freesound-audio-tagging/data>

файлове, обхващащи широк спектър от реални среди. Звуците включват неща като музикални инструменти, човешки звуци, домашни звуци и животни от библиотеката на Freesound, аnotирани с помощта на речник от над 40 етикета от онтологията на AudioSet на Google.

За се реши този проблем са създадени модели ,които предсказват етикета на съответния файл или групират файловете .Входът на моделите ще бъдат аудио файловете(wav формат),като ще се използват 600 от тях като тренировъчни данни и 100 като тестови.Тези колекции от данни ще са генериирани на базата на данни от 10 различни класа с цел да се обучи класификатора да разпознава не само търсените 5 групи,но и че даден файл не попада в тях.Задачата на програмата е да бъдат създадени обучителни модели(Логистична регресия,Невронни мрежи и Рандом дървета,Kmeans,Mean Shift и други) ,които да се използват за класификация и кълстерилизация на музикалните файлове,а целта е да се доближим максимално да най-добрите резултати постигнати върху тези данни(accuracy: 90.17%).

Преглед на областта

²Data Mining

Едно общоприето определение на понятието Data Mining (Usama, Piatetsky-Shapiro, 1996) е следното: Data Mining е процесът на разкриване в „сировите” необработени данни на неизвестни преди нетривиални, не очевидни, обективни, практически полезни и достъпни интерпретации на знания, необходими за вземането на решения в различни области

² Уеб-страница: СЪВРЕМЕННИ МЕТОДИ ЗА ИНТЕЛИГЕНТЕН АНАЛИЗ НА ДАННИ
Д-р инж.Мартин Пъшев Иванов, главен асистент, НБУ –департамент „Информатика“
<http://eprints.nbu.bg/3082/1/%D0%98%D0%BD%D1%82%D0%B5%D0%BB%D0%B8%D0%B3%D0%B5%D0%BD%D1%82%D0%B5%D0%BD%D0%20%D0%BD%D0%BD%D0%BB%D0%B8%D0%BD%D0%20%D0%BD%D0%BD%D0%BB%D0%BD%D0%20-%D0%BF%D1%83%D0%B1%D0%BB%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F.pdf>

човешката дейност. Това определение акцентира върху следните особености на получените от този процес резултати:

- Не очевидни – закономерностите не могат да бъдат установени със стандартни методи за обработка на информацията или по експертен път.
- Обективни – закономерностите съответстват максимално на действителността (за разлика, напр. от експертното мнение, което винаги съдържа субективен елемент).
- Практически полезни – закономерностите (изводите) имат конкретно значение, на което може да се намери практическо приложение.

Data Mining представлява интердисциплинарна област, възникнала и развиваща се а на базата на такива съседни области като приложна статистика, машинно обучение, изкуствен интелект , теорията на базите от данни и др. Data Mining ползва теоретични резултати и приложни процедури, които се включват както в областите, показани на фигура 1, така и в много други направления на науката, съдържащи потенциал за постигане на целите на интелигентния анализ на данните (Graham, 2011), (Han, 2006).



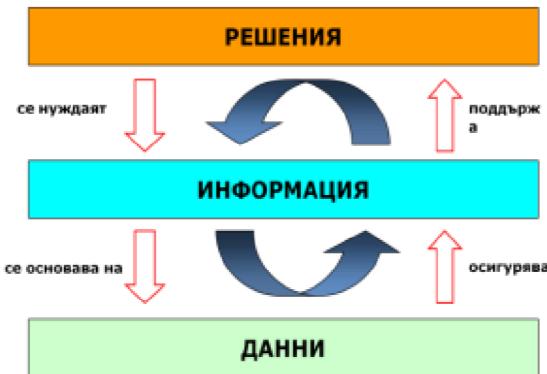
фиг. 1 -Интердисциплинарата област Data Mining и контактните ѝ научни области.

Понятието Data Mining може да бъде определено в няколко различни аспекти, в зависимост от целите на изследването и от контекста, в който то се употребява:

- подход и методология – определя насоките за изследване и развитие на ефективни модели и методи за анализ на данните и разкриване на съществените за практиката шаблони и връзки.
- технология – ефективни средства и техники на изпълнение на задачите по обработка на данните и извличане на данните, включително използването на специализиран софтуер.
- процес - това е динамичен процес на поддържането на задачите за вземане на решения, чрез няколко свързани помежду си фази.

Data Mining и процесът на вземане на решение

³Data Mining като процес на анализ на данните и извличането на практически полезни връзки и зависимости е фаза, свързваща средствата и технологиите за набиране, съхраняване и достъп до данните от една страна, и процесите и моделите, и алгоритмите за вземане на решение от друга. Всеки акт на решение се нуждае от информация, изградена въз основа на данни и от установени знания, относно връзките и зависимостите между параметрите на управлявания процес. Самия процес може да се види илюстриран на фигура 2.



фиг. 2 - Цикъл на информационното осигуряване на задачите за вземане на решение.

Фигурата илюстрира двата основни потока, които са включени в информационния цикъл на Data Mining, чийто резултати са придобиване на знание и вземане на решение:

- Данни – информация – знания и решения: вземането на решения изисква информация, която се основава на данни.

³ Data Mining: Practical Machine Learning Tools and Techniques [Ян Х. Уитен](#), Айб Франк 1999 г.

- Задачи – действия и методи за решаване – приложения: данните осигуряват информация, която поддържа решенията.

Големият обем информация, свързан с различните дейности в обществото от една страна позволява да се получат точни разчети, а от друга – превръща процеса на търсене на решение в сложна задача. В резултат на това се появява цял клас програмни системи, предназначени да облекчат работата на специалистите-аналитици – системи за поддържане на вземането на решения (Decision Support System - DSS). Могат да бъдат отбелязани три основни задачи, решавани в DSS:

- въвеждане на данните;
- съхраняване на данните;
- анализ на данните.

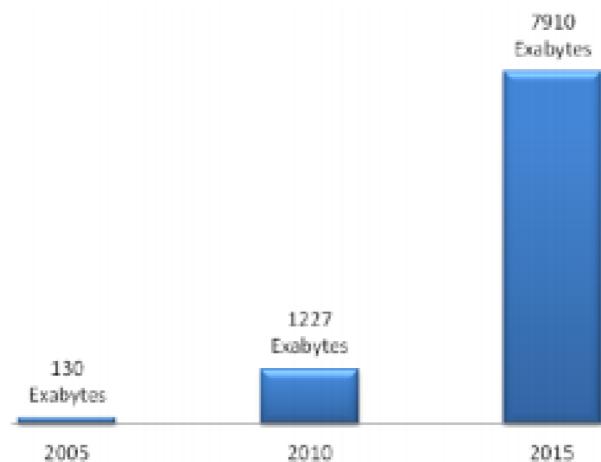
Източници и тенденции в растежа на обема на данните

Източниците на данни в съвременния свят представляват необозримо множество. Те са повсеместни и са свързани с всички икономически, управленски, социални, научни, образователни страни на човешката дейност. Значителна част от данните са машинно генериирани, а обемът им нараства експоненциално във времето. Най-общо източниците на данните могат да бъдат систематизирани в следните групи:

- Релационни бази от данни;
- Хранилища на данни;

- Транзакционни бази от данни;
- Разширени информационни системи и приложения;
- Обектно-релационни бази данни;
- Текстови източници и приложения;
- Web.

Този списък е условен и подлежи на актуализация с разширението на областите, в които намират приложение информационните технологии. Експоненциалният ръст на данните, според изследване на IDC Digital Universe Study за последните години е илюстриран на фиг. 3 .



**Фиг. 3: Тенденция в ръста на обема данни.
Оценка на IDC Digital Universe Study за общия обем
генерирани и съхранени данни по години.**

Данни, информация, знание

Следва да бъде взето под внимание, че въпреки честото им смесване в популярния език, категориите „данни”, „информация” и „знания” са свързани, но все пак съществено различни понятия и отразяват различни равнища на абстракция на представата за света и протичащите явления и процеси в него. За правилното разбиране и формулиране на проблемите на Data Mining е от значение тези различия да бъдат ясно дефинирани. В най-общия случай данните представляват формална регистрация на някакви факти. Те могат да се представят чрез числови стойности, текст, графика, изображения, звук, аналогово или цифрово видео и др. Данните могат да бъдат получени в резултат на измерване, експерименти, аритметични и логически операции и др. Те трябва да бъдат представени във форма, позволяваща съхранението им, предаването им и обработката им. Иначе казано – данните са необработения („сиров“) материал, предоставен от някакъв източник и използван от потребителя за създаване на информация въз основа на тези данни. Информацията е по-висша форма на абстракция на представата за обектите и явленията и се представя чрез образци, схеми, асоцииции, свързващи известните данни. Знанието е най-висша форма на абстракция, която се получава въз основа на информацията като резултат от разсъдъчната дейност и от прилагането на някои, често интуитивни процедури. Знанията представляват съвкупност от сведения, които образуват описание, съответстващо на дадено равнище на осведоменост по определен въпрос, предмет, проблем и т.н. Знанията се използват за получаване на конкретни предимства и резултати.

Характеристики на данните, използвани в Data Mining

Познаването на начините на получаване и представяне на различните типове данни е от съществено значение за избор на методи за тяхната обработка. Значителна част от данните се получават чрез измерване (респ.остойностяване) на параметри и атрибути на наблюдаваните явления и процеси. Под “остойностяване” се разбира процесът на присвояване на някакви символни (вкл.логически) или числови стойности на наблюдаваните характеристики в съответствие с някакво правило (т.е.изобразяването им върху някакво символно или числово множество). Множествата, носители на стойностите, в които се изобразяват характеристиките, заедно с правилата на изобразяването се наричат “скали”. Според свойствата си скалите могат да бъдат различни категории. Преди всичко те могат да бъдат разделени на дискретни и непрекъснати:

- Дискретни скали – в случаите, когато остойностяването се извършва върху дискретно или изброимо множество от числови или символни стойности. В тази категория влизат и скалите, представлящи логически типове данни (true/false, 0/1 и т.н.). Данните, представени върху дискретни скали се наричат дискретни данни. Възможните стойности на тези данни могат да бъдат обозначени (номерирани) с естествените числа.
- Непрекъснати скали – множествата, носители на стойностите са непрекъснати и обикновено са числови, зададени в краен или безкраен интервал. Данните, представени върху непрекъснати скали се наричат непрекъснати данни. Освен това общо разделение е прието да се работи с четири основни типа скали – номинална, рангова (ординална), интервална и относителна
- . - Номинална скала (nominal scale) – съдържа само категории – данните се представят символно, като не могат да бъдат подредени в някакъв ред и

върху тях не могат да бъдат прилагани аритметични операции.

Номиналната скала може да се състои от наименования, категории, класификационни признаци на обекти. Примери за такива скали са имена на географски обекти, цветове, професии, символи в някаква азбука, пол, семейно положение и др. В тази скала са дефинирани единствено операциите „равно” (=) и „не равно” (\neq). Данните, представени в такава скала понякога се наричат „категорийни”, но коректното им наименование носи името на скалата – „номинални”.

- Рангова (ординална) скала (ordinal scale) – в която представените обекти имат някаква относителна позиция един спрямо друг. Ако позициите се представят чрез числа, то тези стойности по никакъв начин не отразяват степента на различие на обектите. Ранговата скала дава възможност върху множеството на обектите да бъде дефинирана релация на наредба.

Класически пример за такава скала са всички бални системи за оценяване на знания и умения на учащи се, системите за класиране в съревнования, някои метеорологични и сейзмични скали и др. За тази скала освен релациите „равно” (=) и „неравно” (\neq) е дефинирана и релация на наредба (т.е. валидни са операциите за сравнение “по-голямо” $>$ и „по-малко” $<$).

При някои изследвания представените в тази скала данни също се определят като категорийни, доколкото върху тях не могат да бъдат прилагани основните математически операции.

- Интервална скала (interval scale) – скала, при която разликите във стойностите на представените величини могат да бъдат изчислени (като дължина на интервал), но определянето на количествено отношение между тях няма смисъл. Тази скала притежава свойствата на номиналната и ранговата скала, като позволява при това да се намери разликата между две величини и да се определи количествено стойността на признака. Типичен пример за такава скала са температурните скали на Целзий или на

Фаренхайт, при които могат да бъдат измерени съответните стойности, те могат да бъдат съпоставяни по величина („ $<$ ”, „ $>$ ”), могат да бъдат пресметнати и съпоставяни интервалите на изменението им (като температурни разлики), но самите температурни стойности не могат да бъдат съпоставяни в никакво количествено отношение (не можем да твърдим, че температурата 20°C е три пъти по-ниска от температура 60°C). За номиналната скала са приложими операциите: равно ($=$), не равно (\neq), по-голямо ($>$), по-малко ($<$), операциите събиране (+) и изваждане (-). Номиналната скала представя по принцип непрекъснати числови данни.

- Относителна скала (ratio scale) – скала, в която има определена точка на отчитане (репер) и има смисъл определянето на количествено отношение за стойностите ѝ. Пример за такава скала са повечето от скалите за измерване на физически величини: маса, разстояние, скорост, енергия, остойносяванията в икономиката и финансите, физиологични характеристики в медицината и др. Това е числовска скала, предоставяща най-голямо съвършенство при представяне на данните, защото освен операциите, дефинирани и допустими за горните три скали, тя позволява и прилагането на аритметичните операции „умножение” (\cdot) и „деление” ($/$).

В някои случаи се употребява и т. нар. „дихотомна скала” (dichotomous scale), съдържаща само две възможни стойности. Пример за такава скала е представянето на логически стойности, на пол (мъж, жена), ден/нощ и т.н. По същество тази скала представлява разновидност на номиналната, но самостоятелната ѝ употреба се мотивира от множеството изследвания, в които тя е удобна форма за представяне на даните.

Възможни са и други класификации на данните според други техни особености или целите на изследването, за което се използват. Така

например според представянето им в базите от данни могат да бъдат разделени на следните групи данни:

- Релационни данни – това са данните от релационните таблици.
- Многомерни данни – данни, представени в OLAP-кубове.
- Измерение (dimension) или “ос” – в многомерни данни – обединение на данни от един и същи тип, което позволява да се структурира многомерната БД.

По критерий “постоянство на стойностите в процеса на решаване на задачата” данните могат да бъдат:

- Променливи данни – които изменят своите стойности в процеса на решаване на задачите.
- Постоянни данни – данни, които съхраняват своите стойности в процеса на решаване на задачите и не зависят от външни фактори.
- Условно-постоянни данни – данни, които винаги могат да променят своите стойности, но тези изменения не зависят от процеса на решаване на задачата, а от външни фактори.

Според времевите си характеристики данните могат да бъдат:

- Данни за период – характеризират някакъв период от време (доход за месец, средна температура за месец, потребление на енергия, курс на валута за периода и т.н.).
- Точкови данни – представлят стойността на някаква променлива в конкретен момент от време.

Според структурираното им представяне данните се разделят на следните категории:

- Структурирани данни – това са данни, които са групирани в релационни схеми (редове и колони в стандартна релационна база от данни).

Конфигурацията на данните и съгласуваността им позволяват да се получат отговори на прости или съставни заявки със средствата за манипулиране на релационни бази от данни (SQL заявки).

- Полу-структурни данни – това е форма на структурираните данни, която не следва изрично релационната схема. За тези данни е присъщо, че те са „само-описващи“ се и съдържат етикети или други маркери (метаданни), които уточняват фактическото и, съдържание и позволяват то да бъде представено в организирана форма (например като йерархии от записи и полета). -Неструктурирани данни – това са данни във формат, който не може да бъде представен чрез релационна схема или в самоописващи се структури. Такива са например свободния текст, файловете с графично и видео съдържание и др.

Задачи на Data Mining

Задачите на Data Mining могат да се класифицират в две най-общи категории, в зависимост от поставената цел и от резултата, който се получава от анализа:

- Дескриптивни задачи – свързани са основно с установяването на някакви присъщи характеристики и връзки, описващи състоянието на наличните до момента данни и поведението на процесите, които те описват. Такива са

задачите на първоначалния статистически анализ на данните, задачите за кълстеризация и др.

- Предиктивни задачи – отнасят се до съставянето на прогнози или предвиждане за бъдещото изменение на данните и параметрите на процесите въз основа на наблюдаваните им стойности и динамика до момента. Такива са задачите за класификация, регресионните статистически модели, анализът на динамични редове (time series) и т.н.

Методи на Data Mining

Въщност Data Mining не разполага със свои собствени методи, а използва методите на съседните на научно-приложни области . Както неколкократно бе отбелоязано, Data Mining е по-скоро конструктивна, прагматично ориентирана, а не теоретична област (Zaki,2014), (Olson,2008). Всеки метод, потвърдил на практика ефективността си, може да бъде използван за целите на Data Mining, дори и в случаите, когато неговата ефективност не е обоснована теоретично по формален път. Всички такива случаи, обаче, трябва да се вземат предвид от изследователите и практикуващите DM специалисти и да не се прилагат сляпо и немотивирано, като им се възлагат свръх-големи очаквания. Data Mining не е вълшебно решение на проблема за изследване на данните, а път и технология, които непрекъснато се усъвършенстват, разширяват и които откриват нови възможности и хоризонти за приложение. Сред методите на Data Mining следва по-специално да бъде обърнато внимание на следните групи:

- Методи на математическата и приложна статистика – това са добре познати традиционни методи за обработка и анализ на масови съвкупности от данни, които отговарят на целите на подхода Data Mining. Такива

методи са дескриптивен (първичен) анализ и описание на изходните данни.,анализ на връзките (корелационен анализ и регресионен анализ, факторен анализ, дисперсионен анализ). ,многомерен статистически анализ (компонентен анализ, дискриминантен анализ, многомерен регресионен и корелационен анализ) и анализ на динамични редове и прогнозиране.

- Методи, основаващи се на теорията на изкуствения интелект, машинното обучение и евристични методи – това са методи, чрез които може да бъде изследвана връзката между състоянието на съвкупност от входни данни или предпоставки и изходния резултат. Обикновено те позволяват да се моделира поведението на реални технически, технологични или икономически процеси, без да се анализира в детайли тяхната структура. Голяма част от методите от тази група нямат строго формална обосновка и се основават по-скоро на общи рационални принципи, потвърдени от многократния изследователски и практически опит. Тези методи привличат вниманието на разработчиците с разширени си възможности, със способността си да решават множество нетрадиционни задачи и с факта, че могат лесно и ефективно да бъдат реализирани в програмни инструменти и да бъдат вградени в специализирани софтуерни системи. Без да бъде изчерпателен, списъкът на тези методи включва: Изкуствени невронни мрежи – подходящи за решаване на апроксимационни и оптимизационни задачи. ,еволюционно програмиране(включващо алгоритми за групово отчитане на аргументите),генетични алгоритми (за решаване на оптимизационни задачи, които се представят с по-сложни или неявни формални модели),асоциативна памет(за търсене на аналоги и прототипи на поведение в съвкупност от данни.), размита логика и изчисления(приложими когато моделът съдържа параметри, чиито стойности не могат да бъдат точно определени или тези стойности съдържат твърде високо ниво на „шум”), дървета на решенията (

използвани в класификационните задачи за съставяне на класификационни правила), системи за обработка на експертни знания(чрез правила, създадени от експерти) и други .

Стратегии на обучение на модела

В случаите, когато е известен типът и структурата на задачата, за окончателната ѝ формулировка е необходимо да бъдат конкретизирани редица нейни определящи параметри. Така например в задачите за класификация се изисква да бъдат определени онези правила, атрибути и стойности, които ще доведат до коректното относяне на текущо разглеждания обект към съответния клас. В оптимационните задачи, например, трябва да бъдат пресметнати онези стойности на управляващите параметри на модела, които ще доведат до желаното целево състояние и т.н. Определянето на целесъобразните стойности на параметрите на модела в много от задачите става чрез итеративен процес, наречен „обучение” на модела и може да извършен по пътя на две основни стратегии:

- „Обучение с учител” (контролирано обучение, supervised learning) – при които стойностите се определят въз основа на предварително зададени образци, описващи наблюдаваното явление или обект. Стойностите на параметрите на модела се определят чрез изчислителен алгоритъм (процес), като същите периодично се съпоставят с регистрираното в известните образци състояние на обекта и се коригират така, че грешката да бъде минимална. Процесът на обучение на модела обикновено включва два етапа. На първият етап се използва съвкупността от предварително зададените образи, наречена „обучаващо множество” (training set) или „обучаваща извадка”. Вторият етап включва оценка на надеждността и

точността на модела. В него също се използва съвкупност от известни наблюдения на явлението, които не участват в процеса на обучение, наречена „тестово множество“ (test set). Процедурата на втория етап проиграва съставения модел с даните от тестовото множество и сравнява моделираните и фактическите стойности. Пример за задача от този тип е класификацията, при която изследователят разполага с множество примерни образци за принадлежността на обектите към даден клас, заедно с текущите стойности на атриутите им. Класически пример за тази стратегия са и методите за обучение на изкуствени невронни мрежи.

- „Обучение без учител“ (unsupervised learning) – в този случай отсъства предварителна информация, представена в известни, наблюдавани образци на връзка между параметрите на процеса. Обучаващият алгоритъм съдържа механизъм за адаптиране на тези параметри към най-подходящите им стойности чрез използването на съответни критерии за целта. Пример за такава задача е къстерилизацията, при която отсъства каквато и да е информация за принадлежността на наблюдаваните обекти към някаква група.

⁴Класификация

Целта на класификацията е да създаде кратък обобщен модел на изменението на зависимия атриут (принадлежност към клас), свързано с поведението на атриутите-предиктори. Популярни определения на задачата за класификация са: – системно разпределение на изучаваните предмети, явления, процеси по родове, видове, типове според някакви съществени признаки с цел удобство на изследването им, групировка на изходните данни и разполагането им в определен ред, отразяващ степента на сходството им. – подредени по някакъв принцип множество обекти, които имат сходни класификационни признаки (едно или няколко

⁴ Записки към курса по „Извличане на информация“ четен от Иван Койчев

свойства), избрани за определяне на сходството или различието между тези обекти. Задачите за класификация могат да бъдат систематизирани в няколко основни категории: Двоична класификация – с два възможни изхода, определени от принадлежността на обектът към дадена група („да“ или „не“). Класификация с множество възможни изходи – обектът се класифицира в една от няколко възможни групи (класове) в зависимост от стойността на атриутите си. Едномерна (по един признак) и многомерна класификация (по множество признания) – в зависимост от това, колко атрибути (класификатора), описващи обекта се вземат под внимание в задачата за класификация. Всеки класификационен подход използва съвкупност от признания, за да класифицира отделен обект. Нека тези признания се представят от случайните променливи X_1, \dots, X_k (предикторни променливи), а Y е зависимата променлива, съдържаща стойностите (етикетите) на възможните класове. Всяка променлива X_i притежава домейн (област на изменение) $\text{dom}(X_i)$ ($i=1, \dots, k$). Зависимата променлива Y притежава домейн $\text{dom}(Y)$. Р е съвместното вероятностно разпределение на $\text{dom}(X_1) \times \dots \times \text{dom}(X_k) \times \text{dom}(Y)$. Обучаващата база от данни D_t е случайна извадка от P . Предиктор d е функцията: $d: \text{dom}(X_1) \times \dots \times \text{dom}(X_k) \rightarrow \text{dom}(Y)$. - Ако Y е категорийна (номинална) променлива, задачата е класификационна и се използва етикета на класа C вместо Y . $|\text{dom}(C)| = J$. - C – етикет на класа, d - класификатор. - Нека r е запис, случайно подбран от P . Дефинира се мярка на погрешната класификация (misclassification rate) за d : $RT(d, P) = P(d(r.X_1, \dots, r.X_k) \neq r.C)$. Дефиниране на задачата: Нека съвкупността от данни D е случайна извадка от вероятностното разпределение P , да се намери класификатор d , който да минимизира $RT(d, P)$. Основни въпроси при съставянето на задачата за класификация - Какво е естеството на съвкупността от данни, която трябва да се класифицира? Трябва да бъде определено предназначението на класификацията и то трябва да бъде свързано с интерпретацията на очакваните резултати. - Каква трябва да бъде точността на

класификацията? Постигането на висока точност в повечето случаи изискава продължителна работа на алгоритмите, което може да се окаже неподходящо за целите на бизнеса. - Доколко разбираема трябва да бъде класификацията? Някои от моделите (дървета на решенията) дават резултати, които обясняват по-добре връзката между предикторната и зависимата променлива, докато при други (невронни мрежи) тази връзка не е така изяснена. Моделите на задачата за класификация (според формата на представянето и метода за решаването им) са твърде разнообразни: - Дървета на решенията (класификационни дървета); CHAID - Chi-square Automatic Interaction Detector; -Случайни “гори”(Random forest) -Логистична регресия; Невронни мрежи; Метод на “най-близкия съсед”; - Наивна Бейсова класификация и др. Методите за решаване на класификационните задачи като цяло се основават на стратегията „обучение с учител”. Както беше отбелязано, тази стратегия изиска наличието на две съвкупности от исторически данни-наблюдения на поведението на класифицираните обекти – с известни класове. Първата съвкупност образува обучаващото множество. С нейна помощ се настройват параметрите на класификационния модел. Втората съвкупност е тестовото множество, което се използва за оценка на точността на решението. Процесът на класификация има за цел да се построи модел, който еднозначно съпоставя стойностите на атриутите-предиктори с класа, към който принадлежи обекта. Основните фази на процеса “класификация” са две – конструиране на модела и неговото използване:

1. Съставяне на модела (фаза на обучение) – описание на множеството от предефинирани класове.

1.1. Всеки обект от извадката се свързва с един от предефинираните класове чрез атрибута “етикет на клас” (ръководено обучение);

1.2. Избира се множество от обекти, използвани за съставяне на модела – т.нар. “обучаващо множество”;

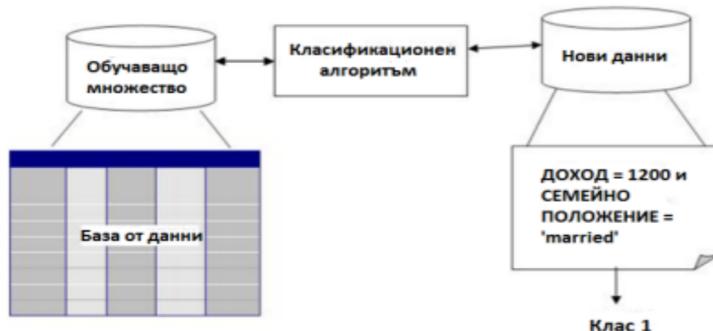
1.3. Моделът се представя като система от класификационни правила, дървета на решенията или в някакъв формален вид.

1.4. Съставяне на модела

2. Прилагане на модела за класифициране на нови обекти

Разделят се данните на тренировъчно и тестово множество или на тренировъчно ,верификационно и тестово множество и се обучава модела с тренировъчното и после се тества/оценява модела на база тестовото .

Тестовото множество не бива да съвпада с обучаващото множество. Точността на модела се определя от процента правилно класифицирани обекти в тестовото множество и ако е приемлива, съставеният модел се използва за практическото решаване на задачата. Самия процес на обучение като последователност от стъпки можете да видите на фигура 4.



фиг. 4-Процес на класификация – прилагане на модела.

Клъстеризация

⁵⁶Групирането/ Клъстеризацията е задачата да се разделят данните в групи, така че точките от данни в същите групи да са по-близки до другите точки от данни в същата група от тези в други групи. С прости думи, целта е да се отделят групи със сходни черти и да се присвоят на клъстери.

Клъстеризирането може да бъде разделено на две подгрупи:

- Твърд клъстериинг: при твърд клъстериинг, всяка точка от данните или принадлежи към клъстера изцяло, или не.
- Меко клъстериране: При меко клъстеризиране, вместо да се поставя всяка точка от данни в отделен клъстери, се определя вероятност или вероятност то да бъде в тези клъстери.

Има повече от 100 алгоритми за клъстериране, но малко от тях се използват широко. Като цяло могат да бъдат обединени в следните групи:

*Моделите за свързване: Както подсказва името, тези модели се основават на идеята, че данните, разположени по-близо до пространството за данни, проявяват по-голяма прилика помежду си, отколкото пунктовете за данни, разположени по-далеч. Тези модели могат да следват два подхода. При първия подход те започват с класифицирането на всички точки от данни в отделни клъстери и след това ги обобщават, тъй като разстоянието намалява. Във втория подход всички точки за данни се класифицират като един клъстери и след това се разделят, когато разстоянието се увеличава. Също така, изборът на функция за разстояние е субективен. Тези модели са много лесни за интерпретиране, но нямат машабируемост за обработка на

⁵ Уеб-страница: What is Clustering in Data Mining? 01/04/2015

<https://bigdata-madesimple.com/what-is-clustering-in-data-mining/>

⁶ Записки към курса по „Извличане на информация“ четен от Иван Койчев

големи масиви от данни. Примери за тези модели са йерархичният алгоритъм за кълстеризация и неговите варианти.

*Центроидни модели: Това са итеративни алгоритми за кълстериране, при които понятието за сходство се получава от близостта на дадена точка до центъра на кълстерите. К-означава алгоритъм за кълстеризиране е популярен алгоритъм, който попада в тази категория. При тези модели не от необходимите в крайна сметка кълстери трябва да бъдат споменати предварително, което налага да има предварително познаване на масива от данни. Тези модели работят итеративно, за да намерят местната оптима.

*Модели на разпределение: Тези модели за кълстериране се основават на представата за това доколко всички точки на данни в кълстера принадлежат към една и съща дистрибуция (Например: Нормално, Гаусово). Тези модели често страдат от свръхнатоварване. Популярен пример за тези модели е алгоритъмът за очакване-максимизация, който използва многовариантни нормални разпределения.

*Модели на плътност: Тези модели търсят пространството за данни за области с различна плътност на точките от данни в пространството за данни. Той изолира различни области с различна плътност и присвоява точките за данни в тези региони в един и същ кълстер. Популярни примери за модели с плътност са DBSCAN и OPTICS.

Класификацията принадлежи към групата модели, наречени “обучение с учител” (supervised learning), а кълстеризацията към обучение без учител (unsupervised learning).

Методи и алгоритми за разпознаване и кълстеризиране на звукови сигнали

Данни и характеристики

Файловете , които са на разположение за обработка са :

*train.csv - това е csv файл, в който се намират етикетите за всеки аудио файл. Има следните полета: fname(името на файла), label(етикета, който му е даден за класификация) и manually_verified(Boolean (1 or 0) flag , който показва дали този аудио файл е ръчно или автоматично анотиран).

*audio_train.zip - това е архив с аудио файловете. Всички аудио мостри в този набор от данни се събират от Freesound [2] и са предоставени тук като некомпресирани PCM 16 бита, 44,1 kHz, моно аудио файлове. Тъй като съдържанието на Freesound е съдействано съвместно, качеството на записването и техниките могат да се различават значително. Всички звуци в Freesound се разпространяват под лицензи за Creative Commons (CC). По-специално, всички звуци от Freesound, включени в FSDKaggle2018, се освобождават под CC-BY или CC0. Основните данни предоставени в този набор от данни, са получени след процеса на маркиране на данни, който е описан в раздела за процеса на етикетиране на данни по-долу.

FSDKaggle2018 звуците са неравномерно разпределени в следните 41 категории на онтологията на AudioSet:

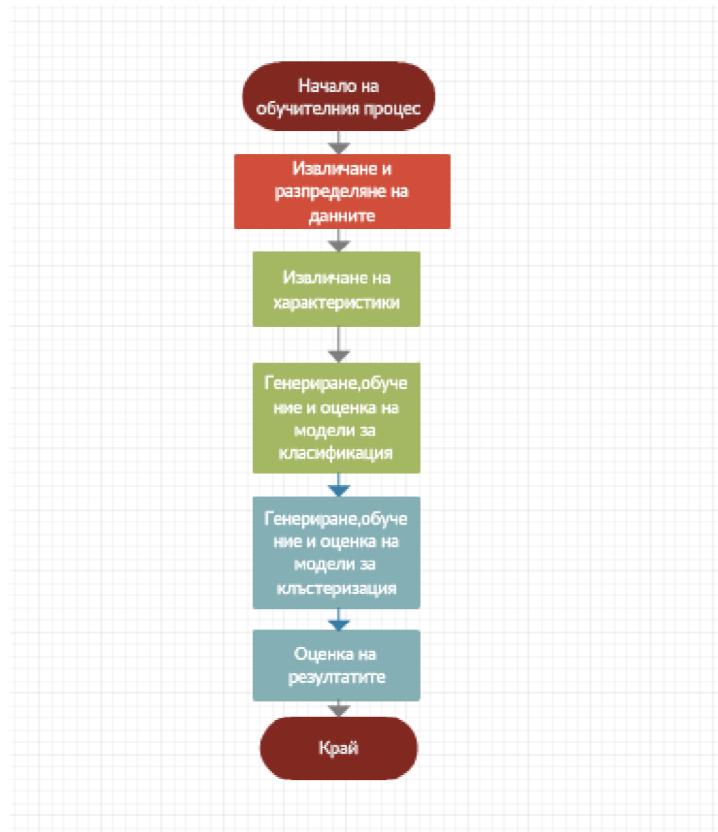
"Acoustic_guitar", "Applause", "Bark", "Bass_drum", "Burping_or_eructation",
"Bus", "Cello", "Chime", "Clarinet", "Computer_keyboard", "Cough",
"Cowbell", "Double_bass", "Drawer_open_or_close", "Electric_piano", "Fart",
"Finger_snapping", "Fireworks", "Flute", "Glockenspiel", "Gong",
"Gunshot_or_gunfire", "Harmonica", "Hi-hat", "Keys_jangling", "Knock",
"Laughter", "Meow", "Microwave_oven", "Oboe", "Saxophone", "Scissors",

"Shatter", "Snare_drum", "Squeak", "Tambourine", "Tearing", "Telephone",
"Trumpet", "Violin_or_fiddle", "Writing"

Всички аудио мостри в този набор от данни имат само един етикет (т.е. са само анотирани с един етикет). Процесът на етикетиране на данни започна от ръчно картографиране между маркерите на Freesound и категориите (или етикетите) на AudioSet Ontology, които бяха проведени от изследователи в групата за музикални технологии (Universitat Pompeu Fabra, Барселона). Използвайки това картографиране, редица аудио образци от Freesound автоматично са анотирани с етикети от онтологията на AudioSet. Тези анотации могат да се разбират като слаби етикети, тъй като те изразяват присъствието на звукова категория в аудио мостра. След това се извършва процес на потвърждаване на данните, в който редица участници слушат етикираните звуци и ръчно оценяват присъствието / отсъствието на автоматично определена категория звук, съгласно описание на категорията AudioSet.

Алгоритъм на програмата

Алгоритъма на програмата можете да видите на фигура 5, като при него са следвани всички правила и добри практики за създаване на машинно самообучение.



фиг. 5-Обобщена блок-схема на алгоритъма

Извличане и разпределяне на данните

За извлечане на данните ще се използва librosa.core модула. Ще бъдат използвани 500 музикални файла като тренировъчни данни и 100 за тестови.

⁷⁸Извличане на характеристики (features)

Това е може би най-важната стъпка от проекта, За извлечането на характеристиките от звуковите данни, ще бъде използвана библиотеката librosa⁹. LibROSA е инструмент за музикален и аудио анализ. Той осигурява градивните елементи, необходими за създаването на системи за извлечане на данни от музика. От него ще се използват следните модули: основния модул (librosa.core), който се използват за зареждане на музикалните данни от твърдия диск и Spectral features¹⁰ модула, който се използват за извлечане на характеристиките от музикалните файлове. Spectral features въщност са разпределението на енергията над набор от честоти - формират основата на много анализи в MIR и цифрова обработка на сигнали като цяло. Повечето спектрални представления в модула се основават на краткосрочното преобразуване на Фурье. Тези които ще се използват са следните:

*Скалата за честота Mel често се използва за представяне на аудио сигналите, тъй като осигурява груб модел на човешкия коефициент на възприятие на честотите. Този инструмент дава много добро темброво

⁷ Уеб-страница: AN EVALUATION OF AUDIO FEATURE EXTRACTION TOOLBOXES David Moffat, David Ronan, Joshua D.Reiss извлечено 28/12/2018 г.

https://www.ntnu.edu/documents/1001201110/1266017954/DAFx-15_submission_43_v2.pdf/06508f48-9272-41c8-9381-7639a0240770

⁸ Уеб-страница: Urban Sound Classification Aaqib Saeed

<http://aqibsaeed.github.io/2016-09-03-urban-sound-classification-part-1/>

⁹ Уеб-страница: документация на библиотеката LibROSA

<https://librosa.github.io/librosa/>

¹⁰ Уеб-страница: librosa: Audio and Music Signal Analysis in Python Brian McFee , Colin Raffel , Dawen Liang , Daniel P.W. Ellis, Matt McVicar, Eric Battenbergk , Oriol Nieto http://conference.scipy.org/proceedings/scipy2015/pdfs/brian_mcfee.pdf

представяне на музиката. От тази скала в проекта са използвани два модела-librosa.feature.melspectrogram,които дават спектрограмата и librosa.feature.mcff,които дават Мел-честота на cepstral коефициентите *Chroma-степен на класа (или насыщеността на цвета на звука),често се използват за кодиране на хармония, и подтискане на вариациите във височината на октавата, croma_stft и tonzenn. В разработката се използва chroma_stft.(Изчислява хроматограма от спектрограмата на вълната или мощността)

* Спектрален контраст -това е друга характеристика (feature),която ще се използва от тази библиотека. Той де-факто дава спектралния контраст или разликата между най-високите и ниски тонове. Тази характеристика (feature) може да бъде намерен в librosa.feature.spectral_contrast.
След извличането си тези данни, те ще бъдат обединени в един общ масив с цел по-лесното им използване от класификационните и клъстеризиационните модели .

Модели за класификация

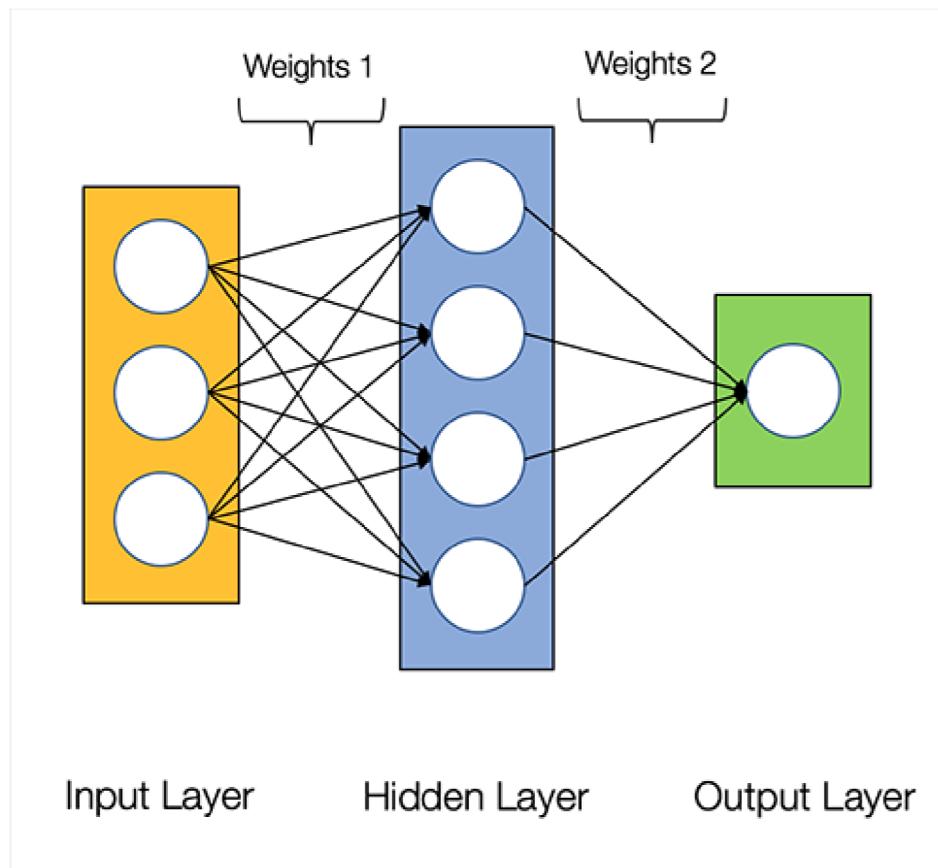
В този проект са използвани няколко различни модела за класификация с цел да се сравнят получените резултати и да се избере най-добрая за решаването на съответната задача. За оценка на моделите е използвана метриката точност(accuracy), както и бързодействие на всеки алгоритъм.

Еднослоен персептрон

Това е алгоритъма ,които се приема за базов. Той е обучение с учител(включва използването на етикетирани набори от данни, които имат входове и очаквани резултати) и изкуствените невронни мрежи ¹¹

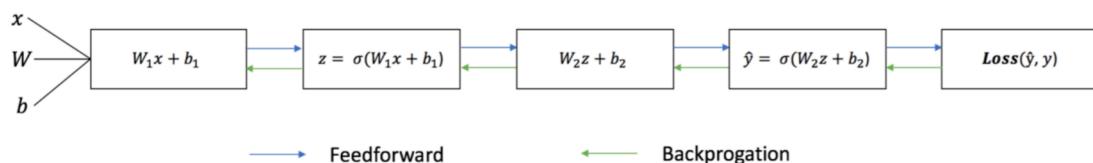
¹¹ Уеб-страница:Невронни мрежи 2010г.
http://techs-mobile.blogspot.com/2010/05/blog-post_14.html

Самообучението с невронни мрежи е един надежден метод за приближаване на целочислени, непрекъснати и векторни целеви функции. За определени типове задачи, като, например, научаването да се интерпретират сложните реални данни, изкуствените невронни мрежи (ИНМ) се нареждат сред най-ефективни известни досега алгоритми. Изучаването на “ИНМ” е било частично вдъхновено от наблюдението, че биологичните самообучаващи се системи са построени от много големи мрежи от взаимосвързани неврони. По аналогия изкуствените невронни мрежи са построени от силно свързано множество прости единици – възли, където всеки възел има няколко непрекъснати входове (които, възможно, са изходи на други възли) и произвежда един единствен непрекъснат изход (който може да бъде вход на множество други възли). Пример за такава можете да видите на фигура 6.



фиг. 6-ИИМ

Отделните възли са взаимосвързани по нива, които формират един насочен ацикличен граф. В общия случай, ИНМ могат да бъдат графи с различен тип структура – ациклични или циклични, насочени или не. Най-често срещани и използвани подходи към ИНМ се базират върху алгоритъма BACKPROPAGATION. В този алгоритъм се подразбира, че мрежата има една фиксирана структура, съответстваща на насочен граф, който може да има и цикли. Обучението съответства на избор на определена стойност на тегло за всяка дъга на графа. Макар, че някои типове цикли са разрешени, преобладаващото мнозинство от практическите приложения използват ациклични мрежи с пряко разпространение на активация(тоест базирани на схемата показана по-долу на фигура 7).



фиг. 7-Алгоритъм за обучение на невронни мрежи

Преди да се опише начина на обучение на мрежата от множество взаимосвързани възли, трябва да се разгледа, как може да се научи тегловният вектор на един единствен персептрон. В този случай задачата за обучение е да се определи такъв тегловен вектор, който кара перцептронът да произвежда правилния изход ± 1 за всеки зададен обучаващ пример.

За решаване на този проблем са известни няколко алгоритъма. Ще бъдат представени само два от тях: *правилото на персептрона* и *делта правилото*. Тези два алгоритъма гарантират сходимостта към малко по-различни приемливи хипотези при малко по-различни условия. Алгоритмите са важни за ИНМ, тъй като осигуряват основата за обучение на мрежа, състояща от много възли.

Един от начините да се научи приемливия тегловен вектор е да се започне със случаен тегла, а след това итеративно да се прибавя грешката към всеки обучаващ пример, модифицирайки теглата всеки път, когато примерът е класифициран неправилно. Този процес се повтаря, итеративно обхождайки всички обучаващи примери толкова пъти, колко е необходимо на персептрона, за да класифицира всички обучаващи примери без грешки. Теглата се модифицират на всяка стъпка в съответствие с *правилото за*

обучение на персептрона, което променя теглото w_i , асоциирано с входа x_i , в съответствие с формулата:

$$w_i \leftarrow w_i + \eta(t - o)x_i, \text{ където } w_i \leftarrow \eta(t - o)x_i$$

Тук t е целевият изход за текущ обуславящ пример, o е изходът, генериран от персептрона, а η - една положителна константа, наречена *скоростта на обучение*. Ролята на скоростта на обучение е да отслабва степента, в която теглата се променят на всяка стъпка. Обикновено, тя се установява равна на някоя малка стойност (например 0.1) и понякога се намалява при нарастването на броя на итерациите по настройка на теглата.

Макар, че правилото на персептрона намира нужния тегловен вектор, когато обуславящите примери са линейно сепарабелни, неговата сходимост може да пропадне, ако те не са линейно сепарабелни. Второто обуславящо правило, наречено *делта правило*, е създадено да преодолее това затруднение. Ако обуславящите примери не са линейно сепарабелни, делта правило осигурява сходимост към апроксимацията, най-добре прилягаща към целевото понятие.

Ключовата идея на делта правило е да се използва *градиентното спускане* за търсене в пространството на хипотези от възможни тегла, за да намери теглата (хипотезата), които най-добре прилягат към обуславящите примери. Това правило е много важно, тъй като градиентното спускане осигурява основа на алгоритъма BACKPROPAGATION, който може да обучава мрежи от множество взаимосвързани възли. То също така е важно, тъй като градиентното спускане може да служи като основа за алгоритми за обучение, които трябва да търсят в пространства на хипотези, съдържащи множество от различни типове непрекъснато параметризираны хипотези. На практиката широко се използват и двета метода.

Многослойен перцепtron¹²

Многослойен перцепtron е клас изкуствена невронна мрежа с право разпространение на сигнала (*feedforward*) и обратно разпространение на грешката (*back propagation*). Състои се от поне три слоя възли на мрежата. С изключение на входните възли (входове), всеки възел представлява неврон, който използва нелинейна активационна функция. Многослойният перцепtron използва техника на обучение с учител (*supervised learning*).

Многослойните перцептрони са полезни в научноизследователската работа, поради способността им да решават задачи стохастично, което често позволява намирането на приблизителни („достатъчно добри“) решения на изключително сложни задачи (например апроксимация на фитнес функция). Те са универсални апроксиматори на функции, както показва теоремата на Цибенко, така че могат да се използват за създаването на математически модели на базата на регресионен анализ, като по-специално са подходящо средство за решаване на задачи за класификация.

Логистична регресия¹³

При регресионен анализ логистичната регресия (или logit regression) оценява параметрите на логистичен модел; това е форма на биномна регресия. Математически бинарният логистичен модел има зависима променлива с две възможни стойности, като пропуск / неуспех, победа / загуба, жив / мъртъв или здрав / болен; те се представят от индикаторна

¹² Статия в Уикипедия: Многослойен перцепtron. В Wikipedia, The Free Encyclopedia. Извлечено на 28.12.2018, от <https://bg.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B3%D0%BE%D1%81%D0%BB%D0%BE%D0%B5%D0%BD%D0%BF%D0%B5%D1%80%D1%86%D0%B5%D0%BF%D1%82%D1%80%D0%BE%D0%BD>

¹³ Статия в Уикипедия: Логистична регресия. В Wikipedia, The Free Encyclopedia. Извлечено на 28.12.2018, от https://en.wikipedia.org/wiki/Logistic_regression

променлива, където двете стойности са означени с "0" и "1". В логистичния модел логаритмичността (логаритъма на коефициента) за стойността, обозначена като "1" е линейна комбинация от една или повече независими променливи ("прогнозни"); независимите променливи могат да бъдат двоични променливи (два класа, кодирани от индикаторна променлива) или непрекъсната променлива (всяка реална стойност).

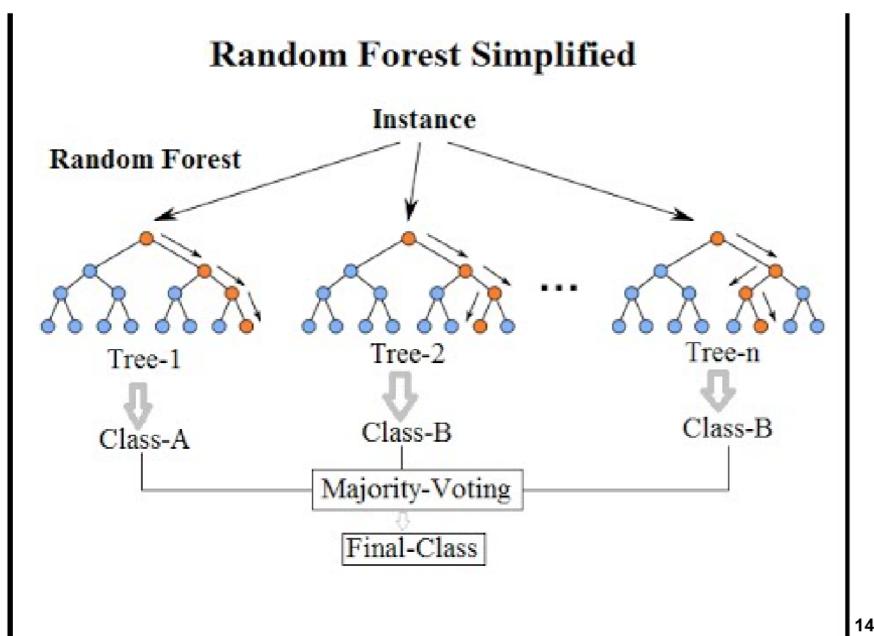
Съответната вероятност за стойността "1" може да варира между 0 (със сигурност стойността "0") и 1 (определеното стойността "1"), оттук и етикетирането; функцията, която преобразува логаритмите в вероятността, е логистичната функция, откъдето идва и името. Единицата за измерване на скалата за логаритмичен коефициент се нарича logit, от логистичната единица, оттам и алтернативните имена. Могат да се използват и аналогични модели с различна сигмоидна функция, вместо логистична функция, като probit модел; дефиниращата характеристика на логистичния модел е, че увеличаването на една от независимите променливи размножава мултипликативно коефициента на дадения резултат при постоянна скорост, като всяка зависима променлива има свой собствен параметър; за двоична зависима променлива това обобщава коефициента на вероятностите.

Логистичната регресия е разработена от статистика на Дейвид Кокс през 1958 г. Двоичният логистичен модел на регресията има разширения на повече от две нива на зависимата променлива: категоричните изходи с повече от две стойности се моделират от много категорийна(Multinomial logistic regression) логистична регресия и ако множеството категории са подредени, чрез ординарна логистична регресия, например пропорционалните коефициенти на логистичен модел. Самият модел просто моделира вероятността за изход от гледна точка на входа и не извършва статистическа класификация, макар че може да се използва за класификатор, например като се избира стойност на прекъсване и се

класифицират входове с вероятност по-голяма от изключването като един клас, под пределната граница като другата.

Random Forest

Random Forest е алгоритъм, който борави с класификационни дървета и опростеното му представяне можете да видите на фигура 8.



фиг. 8-Опростено представяне на алгоритъма Random Forest

Класификационните дървета класифицират примери, описани на езика Атрибут = Стойност. Класификационното дърво представлява една информационна структура, състояща се от възли, съединени със дъги – клонове. Всеки възел определя някой *тест* – проверка на стойността на определен атрибут от примера, а всеки клон, излизаш от този възел, съответства на една от възможни стойности на проверявания атрибут. Листата на класификационното дърво представляват стойностите (дискретни) на целевия атрибут. Класифицирането на новия пример започва от най-горния възел на дървото (коренът) и се осъществява чрез проверка за стойността на атрибута, описан в този възел; след това

¹⁴Уеб-страница:Random Forests(r), Explained By [Ilan Reinstein](#), KDnuggets.
<https://www.kdnuggets.com/2017/10/random-forests-explained.html>

примерът се “пуска” надолу по клона, който съответства на конкретната стойност на проверявания атрибут в дадения пример. Описаният процес се повтаря в текущия възел докато примерът не стигне до някое от листата на дървото.

Научаването на понятия чрез класификационни дървета е най-подходящо за следните случаи:

-*Примерите са представени чрез двойки атрибут-стойност.*

Примерите се описват с помощта на фиксирано множество от атрибути (напр. *Температура*) и техните стойности (напр. *Горещо*). Най-лесната ситуация за научаване на класификационни дървета е когато всеки атрибут може да приема неголям брой номинални (качествени) възможни стойности (напр. *Горещо*, *Топло*, *Студено*). Обаче, ще се разгледа и разширение на базовия алгоритъм, позволяващ работата с атрибути с непрекъснати стойности (напр., когато *Температура* се представя чрез числовата стойност).

-*Целевата функция приема дискретни стойности.* Показаното на рисунка класификационно дърво назначава Булева (двоична) класификация (т.е. *да* или *не*) на всеки пример. Класификационните дървета могат лесно да бъдат разширени за научаване на функции с повече от две възможни стойности. Едно по-съществено разширение на класификационните дървета (така наречени *регресионни дървета*) позволява да научават и функции с непрекъснатия диапазон от реални стойности.

-*При дизюнктивното описание на научаваното понятие.* Както вече беше отбелязано, класификационните дървета по много естествен начин представляват дизюнктивните понятия.

- *Обучаващите данни могат да съдържат грешки.* Методи за научаване на класификационни дървета са устойчиви на наличие на грешки в данни – както на грешки в класификация на обучаващите примери, така и на грешки в стойностите на атрибути на тези примери.

- *Обучаващите данни могат да съдържат неизвестни (липсващи) стойности на атрибути.* Методите за научаване на класификационни дървета могат да се използват и в случаи, когато стойностите на атрибути в някои обучаващи примери са неизвестни (напр. стойността на *Влажност* за някои дни е неизвестна).

Разработени са различни алгоритми за научаване на класификационни дървета, обаче повечето от тях са варианти на базовия алгоритъм, който построява дървото от горе-надолу (от корена – към листата) и реализира евристичното претърсване на пространство от възможни класификационни дървета. Най-типичните представители на този подход са алгоритъм ID3 (Quinlan 1986) и неговият наследник C4.5 (Quinlan 1993). Ще бъде разгледан базовият алгоритъм, който приблизително съответства на ID3. Той описва начина на построяване на класификационното дърво, предназначено за научаване на Булевите функции, т.е. решава задачата за научаване на понятия.¹⁵ Този алгоритъм е представен с псевдокод на фигура 9.

ID3(Примери, Цел атрибут, Атрибути)

Примери са обучаващите примери. *Цел атрибут* е атрибутът, чиято стойност трява да бъде предсказана, а *Атрибути* са останалите атрибути на примери, които се тестват от наученото класификационно дърво. Алгоритът връща класификационното дърво, което коректно класифицира зададените *Примери*.

- Създай най-горен възел - *Корен* на дървото
- Ако всички *Примери* са положителни, *Върни*, като наученото, дърво с един единствен възел – *Корен*, маркиран със знака “+”.
- Ако всички *Примери* са отрицателни, *Върни*, като наученото, дърво с един единствен възел – *Корен*, маркиран със знака “-”.
- Ако *Атрибути* е празното множество, *Върни*, като наученото, дърво с един единствен възел – *Корен*, маркиран със знака, който съвпада с най-често срещано сред *Примери* значение на *Цел атрибут*.
- Иначе

¹⁵ Записки към курса “Машинно самообучение” д-р Геннадий Павлович **Агре** Софийски Университет София

Започни

- $A \leftarrow$ този атрибут от Атрибути, който най-добре класифицира Примери
- Класификационен атрибут на Корен $\leftarrow A$
- За всяка възможна стойност v_i на A направи:
 - Добави в дървото новия клон под Корен, съответстващ на теста $A = v_i$
 - Нека Примери($A=v_i$) са подмножество от Примери, които имат стойността v_i на атрибута A
 - Ако Примери(v_i) е празното множество
 - То добави под този нов клон листо, маркирано със знака, който съвпада с най-често срещано сред Примери значение на Цел_атрибут
 - Иначе добави под този нов клон под-дърво $ID3(\text{Примери}(A=v_i), \text{Цел_атрибут}, \text{Атрибути} - \{A\})$

Край

• Върни Корен

фиг. 9-Псевдокод на алгоритъма ID3

Основната идея на алгоритъма е “разделяй и владей” – цялото множество от примери се разделя на по-малки множества, които се обработват по-лесно. По тази причина подобните алгоритми се наричат “разделящи”. Построяването на дървото се започва от горе на-долу с въпроса “кой атрибут трябва да бъде тестван в корена на дървото?”. За да намери отговора, всеки атрибут се оценява на базата на определен статистически показател (тест), определящ до колко добре този атрибут *самостоятелно* може да класифицира наличните обучаващи примери. Най-добрият атрибут се избира и се ползва като тест в кореновия възел на дървото. За всяка възможна стойност на този атрибут се създава наследник на корена и обучаващите примери се сортират за всеки съответен възел-наследник (т.е. долу по клона, съответстващ на конкретната стойност на този атрибут в примерите). Целият процес се повтаря за всеки възел-наследник, използвайки само обучаващите примери, асоциирани с този възел, за да бъде избран нов най-добра атрибут, който ще се служи като тест в съответния възел. По този начин се осъществява евристичното търсене за приемливо класификационно дърво, като алгоритмът никога не се връща за преразглеждане на вече приети по-рано решения.

Най-важният момент в работата на ID3 е изборът, кой атрибут трява да се тества във дадения възел. Естествено, хубаво е да се избере такъв атрибут, който е най-полезен за класификацията на примери. Каква е добра количествена оценка за “полезност” на един атрибут? Ще бъде дефинирана статистическата мярка, наречена “информационна печалба”, която измерва доколко добре даденият атрибут разделя обучаващите примери в съответствие с тяхната целева класификация. На всяка стъпка от процеса на изграждане на дървото ID3 използва тази мярка за информационната печалба за избор сред атрибути-кандидати.

За точното дефиниране на информационната печалба ще е необходимо да се дефинира една мярка, която в теорията на информация се нарича **ентропия** и характеризира (не)еднородността на произволен набор от примери. При зададеното множество от примери S , съдържащо положителни и отрицателни примери на някое целево понятие, ентропията S относително тази двоична класификация се определя като:

$$Entropy(S) \equiv -p_+ \log_2 p_+ - p_- \log_2 p_-$$

където p_+ е пропорцията на положителните примери в S , а p_- е пропорцията на отрицателните примери в S . Във всички изчисления ще се определи стойността на $0 * \log_2 0$ като равна на 0. За илюстрация да предположим, че S е колекцията от 14 примера на някое двоично понятие, която включва 9 положителни и 5 отрицателни примера (ще се използва то [9+, 5-] за описание на тази колекция). Ентропията на S относно тази двоична класификация ще бъде:

$$Entropy([9+, 5-]) = -(9/14)\log_2(9/14) - (5/14)\log_2(5/14) = 0.940.$$

В общия случай, когато целевият атрибут може да приема c различни стойности, ентропията на S относително c -ичната класификация се определя като:

$$Entropy(S) = -\sum p_i \log_2 p_i$$

където p_i е вероятността, че един случаен избран пример от S принадлежи към клас i . За оценка на тази вероятност може да се използва пропорцията на примери от този клас в S . Обърнете внимание, че логаритамът остава с базата 2, тъй като ентропията е мярката на

очакваната дължина на кодиране, която се мери в битове. Отбележете също така, че в този случай максималната стойност на ентропията е $\log_2 c$.

След въвеждане на ентропията като мярка за еднородността на множество от обучаващите примери, сега можем да въведем и мярка за възможността на всеки атрибут да класифицира самостоятелно обучаващите примери. Тази мярка, наречена *информационна печалба*, е просто очакваното намаляване на ентропията, предизвикано от разделяне на примери в съответствие със стойностите на избрания атрибут. И така, информационната печалба $Gain(S, A)$ на атрибута A относително множеството от примери S се определя като:

$$Gain(S, A) \equiv Entropy(S) - \sum_{v=Стойности(A)}^{} |S_v| \cdot Entropy(S_v)$$

където $Стойности(A)$ е множеството от възможни стойности на атрибута A , а S_v е подмножеството на S , в което всички примери имат стойността на атрибута A равна на v (т.e. $S_v = \{s \in S \mid A(s) = v\}$). Първият член на информационната печалба е ентропията на оригиналното множество S , а вторият – очакваната стойност на ентропията след като S ще бъде разделено чрез използване на атрибута A . Тази очаквана ентропия е просто сума от ентропиите на всяко подмножество S_v претеглена от пропорцията на примери ($|S_v|/|S|$), принадлежащи към S_v . Следователно $Gain(S, A)$ е очакваното намаляване на ентропията, предизвикано от това, че на нас е известна стойността на атрибута A . Или, по друг начин, $Gain(S, A)$ е информацията за стойността на целевата функция при известното значение на някой атрибут A . Стойността на $Gain(S, A)$ е броят на битове, спестени при кодиране на целевото значение на произволен член от S след като стана известна стойността на атрибута A .

Както един метод за самообучение чрез индукция, ID3 може да бъде характеризиран като търсене в пространството на хипотези на такава хипотеза, която съвместима с обучаващите примери. Пространството на хипотези, претърсвано от ID3, е множеството от възможни класификационни дървета. ID3 изпълнява евристичното (от *простото-към-сложното*) търсене в пространството на хипотези като започва с празното дърво, а след това постепенно усложнява хипотези в търсене на класификационното дърво, което коректно класифицира обучаващите данни. Оценъчната функция, която направлява това търсене, е мярката за информационната печалба.

Разглеждайки ID3 в термините на неговото пространство на търсене и използваната стратегия за търсене, можем да добием представа за възможностите и ограничения на този алгоритъм:

- Пространството на хипотези на ID3 се състои от всички класификационни дървета и е *пълното* пространство на функции с дискретни стойности, относително наличните атрибути. Тъй като всяка функция с дискретни стойности може да бъде представена като класификационно дърво, ID3 избягва един от главните рискове на методите, претърсващи непълното пространство от хипотези (като, например, методи, разглеждащи само конюктивните хипотези): това, че пространството на хипотези може и да не съдържа целевата функция.
- При търсене в пространството от възможни класификационни дървета ID3 обработва само една единствена текуща хипотеза. Това контрастира, например, с по-рано разгледания метод за елиминиране на кандидати в пространството на версии, който обработват множеството от *всички* хипотези, съвместими с налични обучаващи данни. Избирайки само една единствена текуща хипотеза, ID3 губи възможности, произтичащи от явното представяне на всички съвместими хипотези. Например, алгоритът не може да определи, колко алтернативни дървета, съвместими с наличните обучаващи данни, могат да бъдат построени или да дава заявки (за нови примери), които позволяват оптималния избор между състезаващи се хипотези. В своята чиста форма ID3 никога не се връща при търсенето. След като веднъж избере *някой* атрибут за тестване на определеното ниво от дървото, алгоритът никога не се връща да преразгледа този свой избор. Това, естествено, води до обичайни рискове на градиентното търсене без възврат – сходимостта към локално *оптимални* решения, които не са оптимални глобално. В случая на ID3 локалното оптимално решение съответства на класификационното дърво, което е било избрано от алгоритъма при претърсване на един единствен път в общото пространство от възможни дървета. Обаче, това дърво може да се окаже по-малко

ефективно от другите дървета, които би могли да бъдат срещнати при преглед на други възможни пътища на търсене.

Едно от предимствата при използване на статистическите свойства на всички примери (напр. информационната печалба) е че полученото търсене е по-малко чувствително към грешки в отделни обучаващи примери. По тази причина ID3 може лесно да бъде разширен, за да работи със зашумени обучаващи данни чрез модификация на критерия за край, като да приема хипотезите, които не покриват абсолютно точно обучаващите данни.

¹⁶*Long Short-Term Memory Neural Networks*

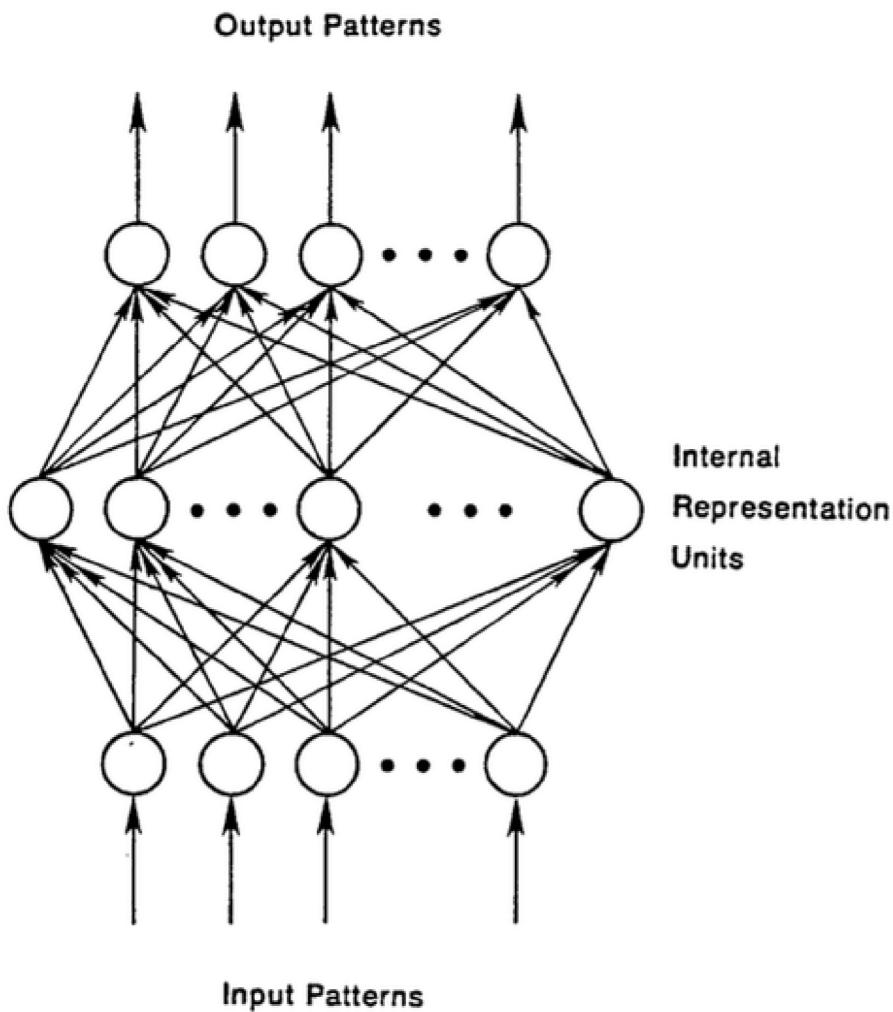
Recurrent neural networks са мрежи ,които имат памет ,за разлика от стандартните невронни мрежи и точно това им предимство им дава възможност да намират контекст .Те са много подходящи за времеви данни (ако се разглежда музикалния файл като битове в секунда това ще е най-подходящата мрежа за нас) и въпреки че не се използва този похват ,нашите данни са взаимозависими и заради това този похват би дал добри резултати.

Нека се запознаем по-подробно как точно работят тези модели.

За да се разберат LSTM и RNN мрежите , първо трябва да се разберат основите на feedforward мрежи(виж фиг 10). И двете мрежи са наименувани по начина, по който те канализират информация чрез поредица от математически операции, извършвани в възлите на мрежата. Човек излъчва информация директно (никога не докосва даден възел два пъти), докато другият го цикли през цикъл, а последните се наричат повтарящи се.

В случая на захранващи мрежи(feedforward), входните примери се подават в мрежата и се трансформират в изход,чрез контролирано обучение.Продукцията ще бъде етикет, име, приложено към входа. Тоест, те картографират необработените данни в категории, като разпознават модели, които могат да сигнализират например, че входното изображение трябва да бъде обозначено като "котка" или "слон".

¹⁶ Уеб-страница:A Beginner's Guide to LSTMs and Recurrent Neural Networks извлечено на 28/12/2018 г.
<https://skymind.ai/wiki/lstm>

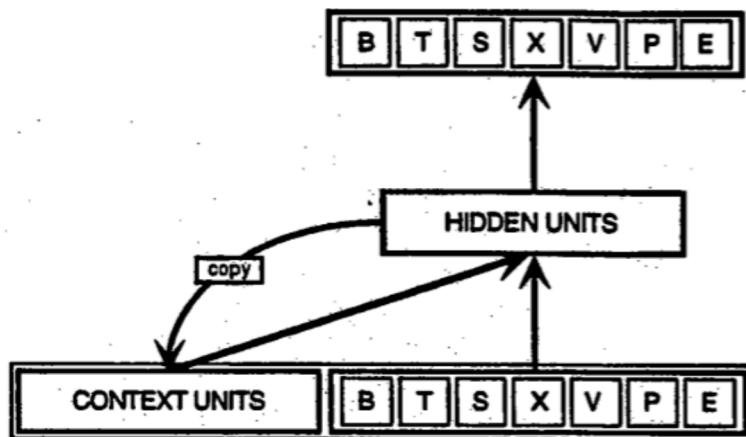


фиг. 10-feedforward мрежи

Една feedforward мрежа се обучава върху етикетирани изображения, докато минимизира грешката, която прави, когато отгатва категориите им. С обучения набор от параметри (или тежести, колективно известни като модел), мрежата се насочва към категоризиране на данните, които никога не е виждала. Обучена мрежа за обратна връзка може да бъде изложена на произволна случайна колекция от снимки и първата снимка, на която е изложена, няма да промени непременно как тя класифицира втората. Виждайки фотография на котка, няма да доведе мрежата да възприеме слон. Тоест, една feedforward мрежа няма представа за реда във времето и единственият вход, който тя счита, е настоящият пример, на

който е била изложена. Feedforward мрежи страдат от амнезия по отношение на неотдавнашното им минало; те запомнят носталгично само формиращите моменти на обучение.

От друга страна, повтарящите се мрежи(Recurrent Neural Network) вземат като вход не само текущия входящ пример, който виждат, но и това, което те възприемат по-рано във времето. Една диаграма на ранна, приста повтаряща се мрежа, предложена от Елман(където BTSXPE в дъното на чертежа представлява входния пример в текущия момент и CONTEXT UNIT представлява изхода от предишния момент) е показана на фигура 11.



фиг. 11-Повтаряща се мрежа на Елман(Recurrent Neural Network)

Решението на повтаряща се мрежа, постигнато в стъпка $t-1$, влияе върху решението, което ще достигне след миг по време на стъпка t . Така че повтарящите се мрежи имат два източника на вход, настоящето и близкото минало, които се съчетават, за да се определи как те реагират на новите данни, както и в живота. Повтарящите се мрежи се отличават от захранващите мрежи чрез тази обратна връзка, свързана с техните предишни решения, като погълъщат свои собствени изходи миг след момент като вход. Често се казва, че повтарящите се мрежи имат памет. Добавянето на памет към невронни мрежи има за цел да има информация в самата последователност. Тази последователна информация се запазва в

повтарящата се мрежа, която успява да обхване много стъпки от време, за да повлияе на обработката на всеки нов пример. Намирането на корелации между събитията, разделени от много моменти, и тези взаимовръзки се наричат "дългосрочни зависимости", защото едно събитие надолу по течението зависи и е функция на едно или повече събития, които са се появили преди това. Един от начините да се мисли за RNNs е следното: те са начин да се споделят тежестите с течение на времето.

Точно както човешката памет циркулира невидимо в тялото, засягайки нашето поведение, без да разкрива пълната му форма, информацията циркулира в скритите състояния на повторящи се мрежи. Английският език е пълен с думи, които описват кръговете за обратна връзка на паметта. Когато кажем, че човек е обитаван от действията си, например, просто говорим за последиците, които миналото производство се проявява в днешно време. Френските наричат това "Le passé qui ne passe pas" или "Миналото, което не изчезва".

Ще опишем процеса на пренасяне на паметта напред математически:

$$\mathbf{h}_t = \phi(W\mathbf{x}_t + U\mathbf{h}_{t-1}),$$

Скритото състояние при стъпка от време t е \mathbf{h}_{t-1} . Това е функция на входа едновременно със стъпка x_t , модифицирана от матрицата на теглото W (подобно на тази, която се използва за захранващи мрежи), добавена към скритото състояние на предишната стъпка от време \mathbf{h}_{t-1} , умножена по своето собствено скрито състояние матрицата U , скрита в състояние, известна още като преходна матрица и подобна на Markov верига.

Матриците за теглото са филтри, които определят колко е важно да се придае както настоящият вход, така и миналото скрито състояние.

Грешката, която генерира, ще се върне чрез backpropagation и ще се използва, за да коригира теглата им, докато грешката не може да намалее. Сумата от вложеното тегло и скритото състояние е смазана от функцията ϕ - или логистична сигмоидна функция, или танх, в зависимост от това - което е стандартно средство за кондензиране на много големи или много малки стойности в логистично пространство, както и за направата на градиенти за обратна пропаганда. Тъй като тази цикъл на обратна връзка възниква във всеки етап от серията, всяко скрито състояние съдържа следи не само за предишното скрито състояние, но и за всички, които предхождат h_{t-1} , докато паметта може да продължи.

При дадена поредица от букви една повтаряща се мрежа ще използва първия знак, за да определи своето възприятие за втория знак, така че първоначалната q може да я доведе до заключението, че следващото писмо ще бъде u , докато първоначалното t може да я доведе до че следващото писмо ще бъде h .

Тъй като повтарящите се мрежи обхващат времето, те вероятно са най-добре илюстрирани с анимация (първата вертикална линия на възлите, които се появяват, може да се смята за предаваша мрежа, която става повтаряща се, докато се разгръща във времето).

Целта на периодичните мрежи(RNN) е точно да се класифицира последователното въвеждане. Разчита се на обратната пропаганда на грешки и наклон на спускане, за да го направим.

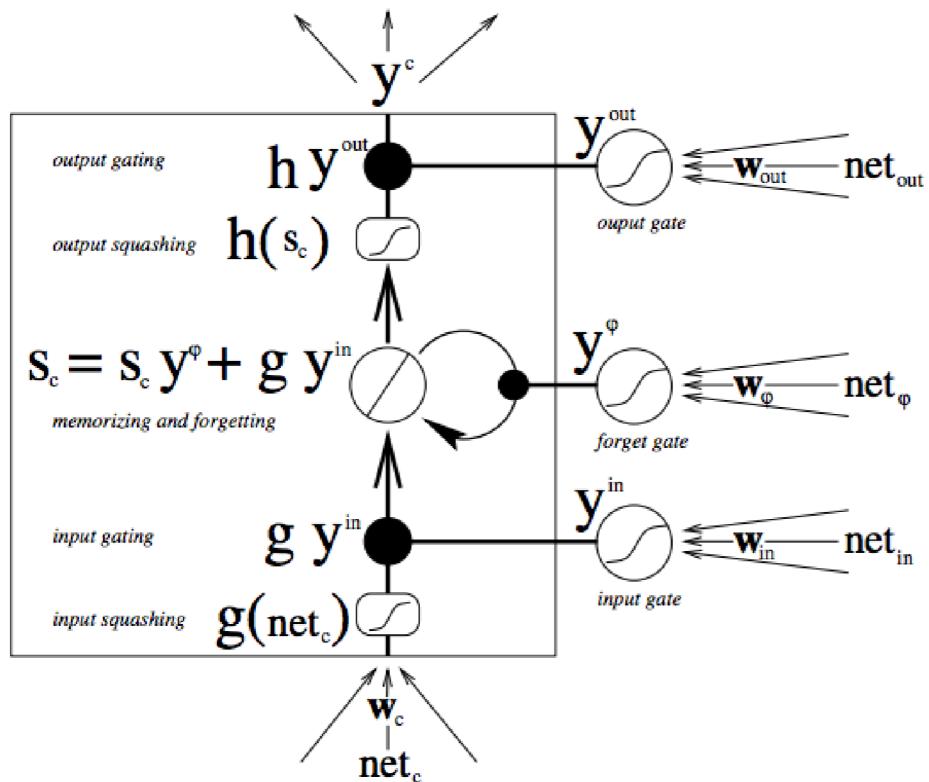
Обратната пропаганда в захранващите мрежи(feedforward) се движи назад от крайната грешка чрез изходите, теглата и входовете на всеки скрит слой, възлагайки тази тежест на отговорността на част от грешката чрез изчисляване на техните частични деривати - $\partial E / \partial w$ или връзката между техните проценти на промяната. Тези деривати се използват от нашето правило за обучение, с наклон на наклон, за да настроите теглата нагоре или надолу, независимо от това, коя посока намалява грешката.

Периодичните мрежи(RNN) разчитат на разширение на обратната пропаганда, наречена backpropagation through time или ВРТТ. Времето в този случай се изразява просто в добре дефинирана, подредена серия от изчисления, свързващи една стъпка към следващата стъпка, която трябва да работи назад. Невронните мрежи, независимо дали са повтарящи се или не, са просто вложени композитни функции като $f(g(h(x)))$. Добавянето на елемент от време само разширява серията от функции, за които се изчисляват дериватите с правилото за веригата.

В средата на 90-те години германските изследователи Sepp Hochreiter и Juergen Schmidhuber предлагат вариант на повтаряща се мрежа с т.нар. дългосрочни единици за краткосрочна памет или LSTM като решение на проблема с изчезващия градиент. LSTMs помагат да се запази грешката, която може да бъде обратно пропагандирана във времето и слоевете. Чрез поддържането на по-постоянна грешка те позволяват на повтарящите се мрежи да продължат да се учат в много стъпки от време (над 1000), като по този начин отварят канал за отдалечено свързване на причините и ефектите. Това е едно от основните предизвикателства за машинното обучение и AI, тъй като алгоритмите често се сблъскват с обкръжения, където сигналите за награда са редки и забавени, като самия живот. (Религиозните мислители са се справили със същия проблем с идеи за карма или божествена награда, теоретизирайки невидими и далечни последици за нашите действия.) LSTM съдържат информация извън нормалния поток на повтарящата се мрежа в затворена клетка. Информацията може да се съхранява в, написана или чете от клетка, подобно на данните в паметта на компютъра. Клетката взема решения за това, какво да съхранява и кога да позволи да се четат, пишат и изтриват, през портите, които се отварят и затварят. За разлика от цифровото съхранение на компютри, обаче, тези врати са аналогови, изпълнени с елементарно умножение от сигмоиди, които са в обхвата от 0-1.

Аналоговият има предимство пред цифровата форма, че е диференцируем и поради това е подходящ за обратно проникване. Тези врати действат върху сигналите, които получават и подобно на възлите на невронната мрежа, блокират или предават информация въз основа на силата и импорта си, които филтрират със собствени набори. Тези тегла, като теглата, които модулират входните и скритите състояния, се коригират чрез процеса на учене на повтарящи се мрежи. Това означава, че клетките научават кога да се позволи на данните да влизат, да излизат или да бъдат изтривани чрез итеративния процес на вземане на предположения, грешка в обратната пропаганда и регулиране на тежестите чрез наклон на наклон.

Диаграмата по-долу илюстрира как потокът на данни минава през клетка на паметта и се контролира от нейните врати(виж фигура 12).

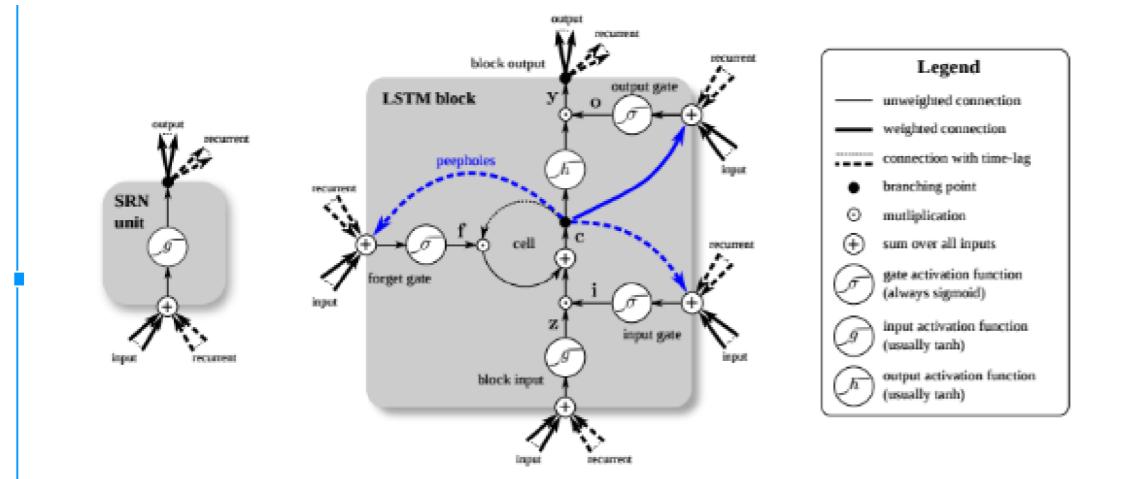


фиг. 12-LSTM workflow

Като се започне от дъното, тройните стрелки показват къде информацията се влива в клетката в множество точки. Тази комбинация от настоящо входно състояние и състоянието на старите клетки се подава не само на самата клетка, но и на всяка от нейните три порти, която ще реши как ще се обработва входа.

Черните точки са самите порти, които определят съответно дали да позволят нов вход, да изтрият състоянието на настоящите клетки и / или да оставят това състояние да повлияе на изхода на мрежата в момента. S_c е текущото състояние на клетката на паметта и $g_y \text{ in}$ е текущият вход към нея. Не трябва да се забравя, че всяка порта може да бъде отворена или затворена и те ще рекомбинират своите отворени и затворени състояния на всяка стъпка. Клетката може да забрави състоянието си или не; да бъде написано или не; и да се чете или не, във всяка стъпка от време и тези потоци са представени тук. Големите букви дават резултата от всяка операция.

Друга диаграма, сравняваща прости повторяща се мрежи (вляво) с LSTM клетка (в дясно) можете да видите на фигура 13.



фиг. 13-Сравнителна характеристика между устройството на скритите слоеве на RNN и LSTM

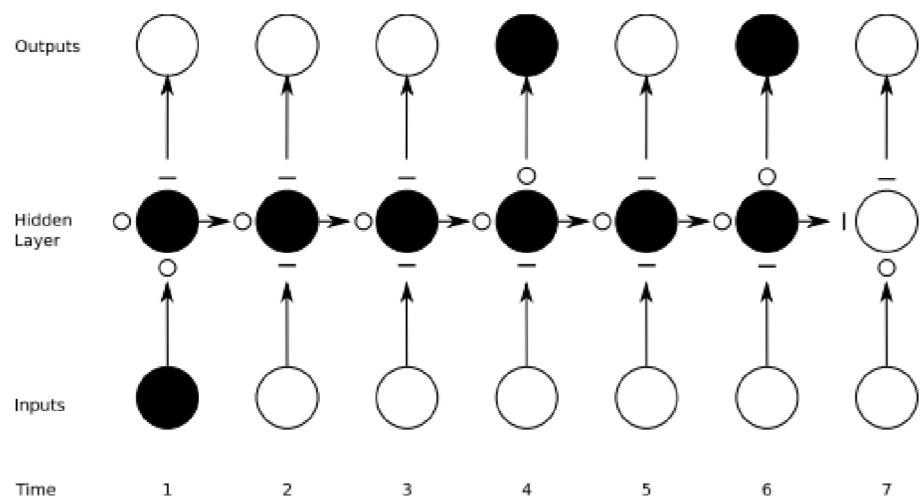
Важно е да се отбележи, че клетките на паметта на LSTM дават различни роли на добавяне и умножаване при трансформацията на входа.

Централният знак плюс в двете диаграми е по същество тайната на LSTMs. Тази основна промяна им помага да запазят постоянна грешка, когато трябва да бъдат обратно пропагандирани в дълбочина. Вместо да се определи последващото състояние на клетката, като се умножи текущото му състояние с нов вход, те се добавят и това буквально прави разликата.

Различни групи тегла филтрират входа за въвеждане, изход и забравяне.

Забравената порта е представена като линейна идентификационна функция, защото ако вратата е отворена, текущото състояние на клетката на паметта просто се умножава по един, за да се разпространи напред още една стъпка от време.

LSTMs имат забравена порта, въпреки че целта им е да свържат отдалечени събития към окончателен изход, защото понякога е добре да се забравя. Примерно ако се анализира текстов корпус и се стигне до края на документ, може да няма причина да се смята, че следващият документ има някаква връзка с него и поради това клетката на паметта трябва да бъде нулирана преди мрежата да погълне първия елемент на следващия документ. В диаграмата по-долу(фиг 14) може да се видят портите по време на работа, с прости линии, представляващи затворени врати, и прости кръгове, представляващи отворени. Линиите и кръговете, които се движат хоризонтално надолу по скрития слой, са забравящите порти.



фиг. 14-LSTM с отворени и затворени порти

Следва да се отбележи, че докато feedforward мрежите намират един вход към един изход, повтарящите се мрежи могат да посочват един към много. LSTM позволява на невронната мрежа да работи на различни машаби от време наведнъж.

Модели за клъстерирана

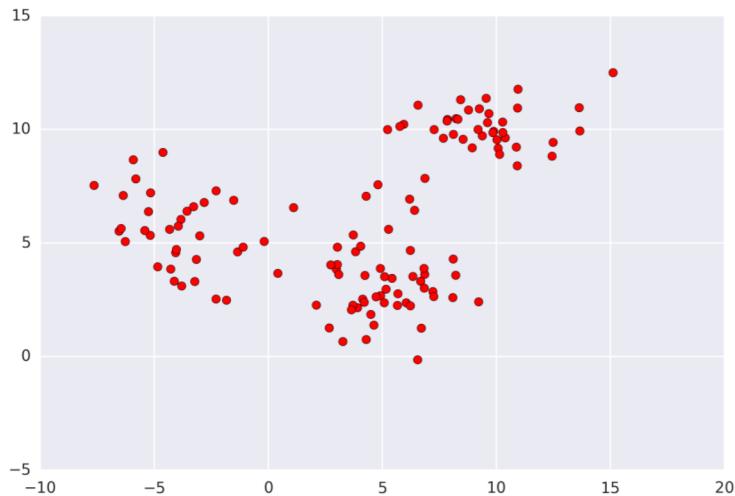
Моделите, които са използвани са K-means, K-means++, като за намирането на оптимално количество клъстери съм използвала генетичен алгоритъм и Mean-shift. За оценка на моделите се използва метриката silhouette_score, както и бързодействие на моделите

*Mean-shift¹⁷(средно отместване)

Първата стъпка при прилагане на средно отместване (и всички алгоритми за клъстериране) представя данните по математически начин. За средно

¹⁷ Уеб-страница: Mean Shift Clustering MATT NEDRICH извлечено на 28/12/2018 г.
<https://spin.atomicobject.com/2015/05/26/mean-shift-clustering/>

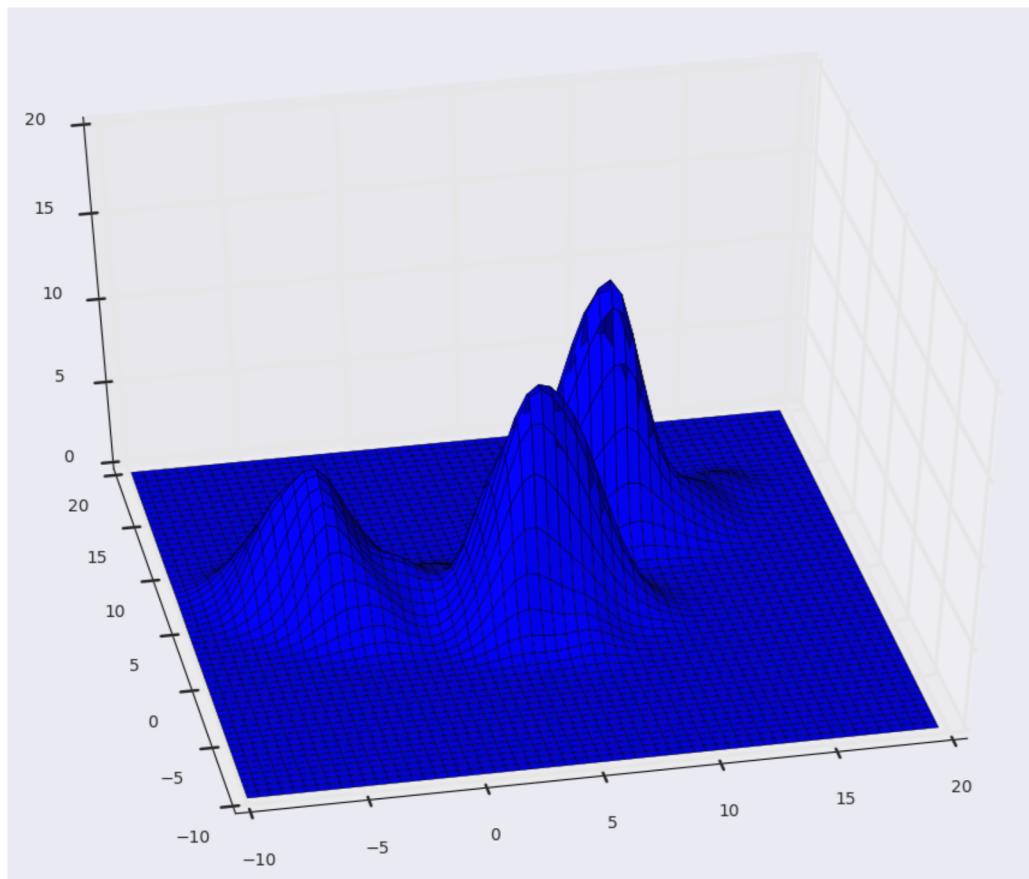
изместване, това означава представяне на вашите данни като точки, като дадените по-долу(виж фиг 15).



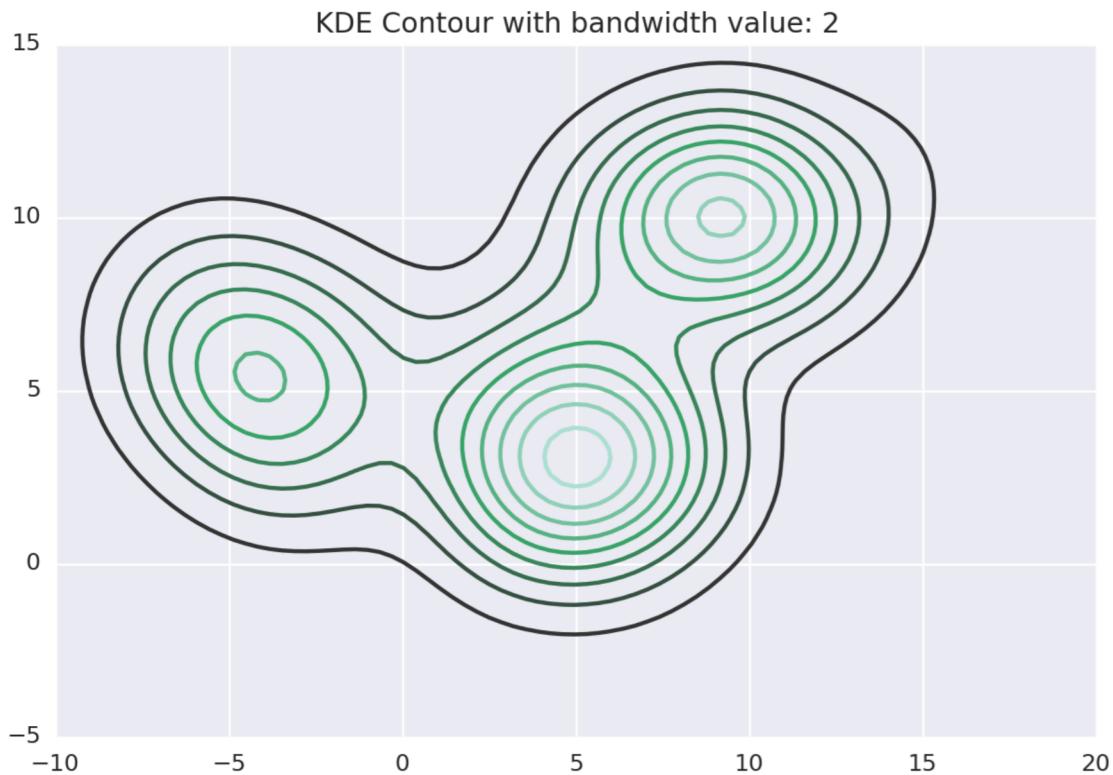
фиг. 15-mean_shift_points

Средната промяна се основава на концепцията за оценка на плътността на ядрото (KDE). KDE е метод за оценка на основната дистрибуция (наричана също функция на вероятностната плътност) за набор от данни.

Той работи чрез поставяне на ядро във всяка точка от набора от данни. Ядрото е математическа дума за функцията за претегляне. Има много различни видове ядра, но най-популярната е ядрото на Гаус. Добавят се всички отделни ядра до генерира вероятностна повърхност (например функция на плътност). В зависимост от използвания параметър на честотната лента на ядрото, резултантната функция на плътността ще варира. По-долу е показана повърхността на KDE за точките по-горе, като се използва ядрото на Гаус с честотна лента на ядрото 2. Първото изображение е повърхностно(виж фиг-16), а второто е контурно изображение на повърхността(виж фиг-17).



фиг. 16-KDE повърхностно изображение



фиг. 17-KDE контурно изображение

И така, как се вписва mean shift(седната смяна) в картината? Средната смяна използва идеята на KDE за постепенно изкачване до най-близкия връх на повърхността на KDE. Това става чрез итеративно преместване на всяка точка нагоре, докато достигне връх. В зависимост от използваната пропускателна способност на ядрото, повърхността на KDE (и крайната кълстерилизация) ще бъде различна. Като крайен случай си представете, че се използват изключително високи и малки ядра (например малка честотна лента на ядрото), тогава резултантната повърхност на KDE ще има пик за всяка точка. Това ще доведе до поставянето на всяка точка в свой собствен кълстер. От друга страна ако се използва ядро с голяма пропускателна способност ,това ще доведе до широка гладка повърхност на KDE с един връх, към който ще се изкачат всички точки, което ще доведе до само еди

клъстер. Ядрата между тези две крайности ще доведат до по-хубави клъстери.

The Mean Shift Algorithm

Както е описано по-горе, алгоритъмът за средна смяна(mean shift) минава итеративно през всяка точка в набора от данни се изкачи до най-близкия пик на KDE. Алгоритъмът започва с копиране на оригиналния набор от данни и замразяване на оригиналните точки. Копираните точки се изместват спрямо оригиналните замразени точки.

Общата схема на алгоритъма е:

```
def shift(p, original_points):
    shift_x = float(0)
    shift_y = float(0)
    scale_factor = float(0)
    for p_temp in original_points:
        # numerator
        dist = euclidean_dist(p, p_temp)
        weight = kernel(dist, kernel_bandwidth)
        shift_x += p_temp[0] * weight
        shift_y += p_temp[1] * weight
        # denominator
        scale_factor += weight
    shift_x = shift_x / scale_factor
    shift_y = shift_y / scale_factor
    return [shift_x, shift_y]
```

```
for p in copied_points:  
    while not at_kde_peak:  
        p = shift(p, original_points)
```

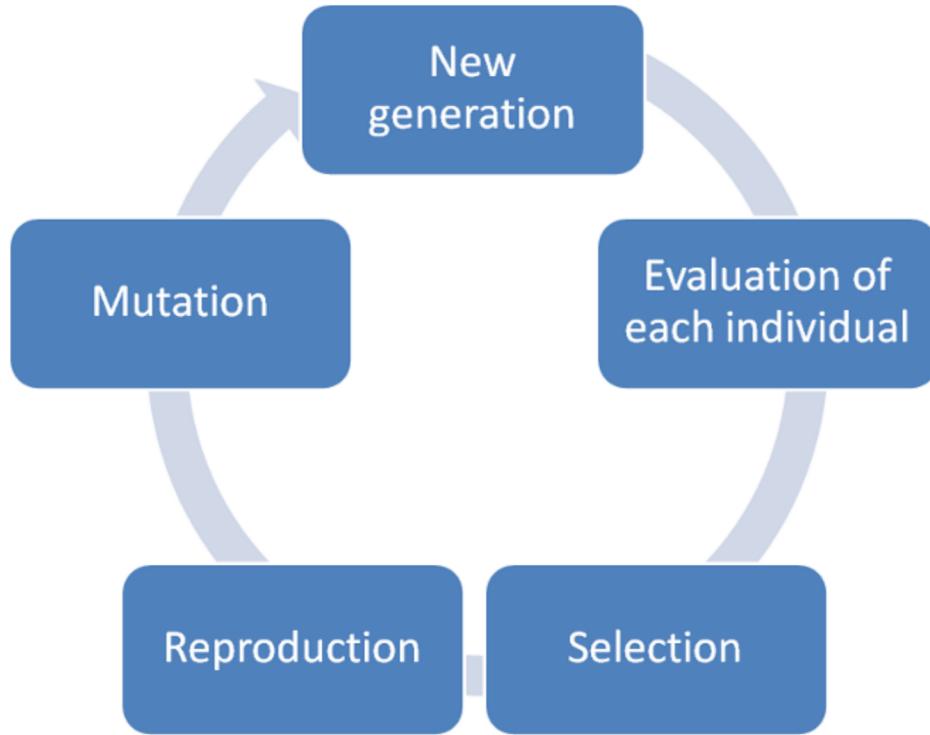
Генетичен алгоритъм с K-means и K-means++

Генетичен алгоритъм

¹⁸ Генетичен алгоритъм е евристика за търсене, която е вдъхновена от теорията на Чарлз Дарвин за естествената еволюция. Този алгоритъм отразява процеса на естествен подбор, при който най-силните индивиди са избрани за възпроизвеждане, за да произведат потомство на следващото поколение.

Процесът на естествен отбор започва с подбора на по-силни индивиди от населението. Те произвеждат потомство, което наследява характеристиките на родителите и ще бъде добавено към следващото поколение. Ако родителите имат по-добра фитнес оценка, потомството им ще бъде по-добро от родителите и ще имат по-голям шанс да оцелеят. Този процес продължава да се повтаря докато накрая се намери поколение с най-силните индивиди. На фиг. 18 можете да се видят основните стъпки, през които се минава за постигане на желания резултат.

¹⁸ Уеб-страница: Introduction to Genetic Algorithms [Vijini Mallawaarachchi](#) извлечено на 28/12/2018 г.
<https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3>



фиг.18-Genetic algorithm workflow

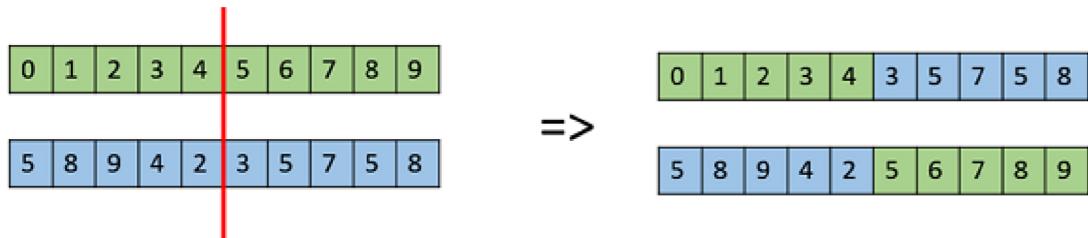
Evaluation of each individual-тази стъпка започва с набор от индивиди, които се наричат Population(население). Всеки индивид е възможно решение на проблема, който искате да решите и се характеризира с набор от параметри (променливи), известни като Гени. След това се оценява годността на всеки индивид да участва в създаването на новото поколение(или казано с други думи колко добро е решението). Това става с помощта на фитнес функция ,която дава фитнес резултат на всеки индивид(вероятността индивидът да бъде избран за възпроизвеждане). Selection-Идеята на фазата на подбор е да се подберат най-приспособените индивиди и да се оставят да предадат гените си на следващото поколение.

Две двойки индивиди (родители) се избират въз основа на техните фитнес резултати. Индивидите с висок фитнес имат повече шанс да бъдат избрани за възпроизвеждане.

Reproduction(Crossover)¹⁹

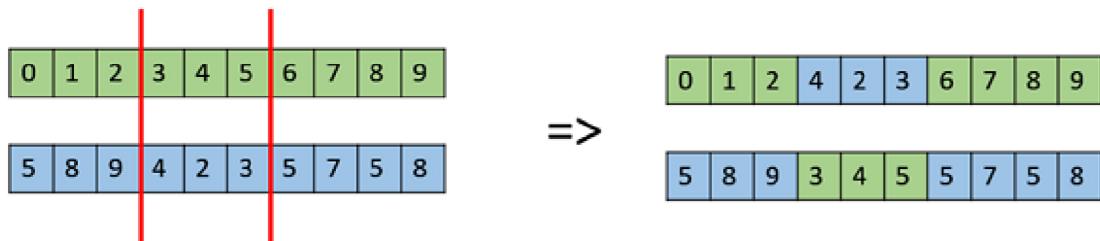
Тази фаза е изключително важна тъй като при нея се генерира новото поколение. За всяка двойка родители, които трябва да бъдат чифтосани, се избира случайна точка на пресичане от гените. Потомството се създава чрез обмен на гените на родителите помежду си, като има няколко възможни подхода:

*One Point Crossover-избира се случайна точка на кръстосване и опашките на двамата ѝ родители се разменят(виж фигура 19)



фиг. 19-One Point Crossover

*Multi Point Crossover-избират се множество случайни точки на кръстосване и опашките на двамата родители се разменят(виж фигура 20).



фиг. 20-Multi Point Crossover

¹⁹ Уеб-страница: Genetic Algorithms - Crossover Tutorialspoint извлечено на 28/12/2018 г.
https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_crossover.htm

*Uniform Crossover-не се разделя хромозомата на сегменти, а се третира всеки ген поотделно. В този случай по същество ще “хвърляме монета” за всяка хромозома, за да се реши дали ще бъде включена в гените на наследника. Също така може да се даде предимство на единия родител(виж фиг. 21).



фиг. 21-Uniform Crossover

*Davis' Order Crossover (OX1)

OX1 се използва за превключвания на базата на пермутация с намерението да се предава информация за относително подреждане на отделните извори. Тя работи както следва :

- 1.Създават се две случаини точки на кръстосване в родителя и се копира сегмента между тях от първия родител до първото потомство.
- 2.Сега, започвайки от втората точка на кръстосване във втория родител, се копират оставащите неизползвани номера от втория родител към първото дете.
- 3.Повтаря се това за второто дете ,като втория родител става първи.

Mutation

²⁰Казано с прости думи, мутацията може да се определи като малка случаина настройка в хромозомата, за да се получи ново решение. Тя се използва за поддържане и въвеждане на разнообразие в генетичната популация. Видове мутация:

²⁰ Уеб-страница: Genetic Algorithms - Mutation TutorialsPoint извлечено на 28/12/2018 г.
https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_mutation.htm

-Bit Flip Mutation(виж фиг. 22)-избира се един или повече случаини битове(гени) и ги се обръщат(от 0 на 1-ца и обратното). Това се използва за двоично кодирани GA.

<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr></table>	0	0	1	1	0	1	0	0	1	0	=>	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr></table>	0	0	1	0	0	1	0	0	1	0
0	0	1	1	0	1	0	0	1	0													
0	0	1	0	0	1	0	0	1	0													

фиг. 22-Bit Flip Mutation

-Random Resetting-Случайното нулиране е разширение на Bit Flip Mutation. При него произволна стойност от множеството допустими стойности се присвоява на случаино избран ген.

-Swap Mutation(виж фиг. 23)- избират се две позиции на хромозомата на случаен принцип и се обменят стойностите. Това е често срещано при кодирането на базата на пермутация.

<table border="1"><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td></tr></table>	1	2	3	4	5	6	7	8	9	0	=>	<table border="1"><tr><td>1</td><td>6</td><td>3</td><td>4</td><td>5</td><td>2</td><td>7</td><td>8</td><td>9</td><td>0</td></tr></table>	1	6	3	4	5	2	7	8	9	0
1	2	3	4	5	6	7	8	9	0													
1	6	3	4	5	2	7	8	9	0													

фиг. 23-Swap Mutation

-Scramble Mutation-Scramble мутация(виж фиг. 24) също е популярна с пермутационни представления. При това, от цялата хромозома, се избира подмножество гени и техните стойности се разбъркват.

<table border="1"><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr></table>	0	1	2	3	4	5	6	7	8	9	=>	<table border="1"><tr><td>0</td><td>1</td><td>3</td><td>6</td><td>4</td><td>2</td><td>5</td><td>7</td><td>8</td><td>9</td></tr></table>	0	1	3	6	4	2	5	7	8	9
0	1	2	3	4	5	6	7	8	9													
0	1	3	6	4	2	5	7	8	9													

фиг. 24-Scramble Mutation

-Inversion Mutation(виж фиг. 25)-избира се подмножество гени като в Scramble мутацията, но вместо да се разбърква подмножеството, просто се инвертира целия низ

<table border="1"><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr></table>	0	1	2	3	4	5	6	7	8	9	=>	<table border="1"><tr><td>0</td><td>1</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>7</td><td>8</td><td>9</td></tr></table>	0	1	6	5	4	3	2	7	8	9
0	1	2	3	4	5	6	7	8	9													
0	1	6	5	4	3	2	7	8	9													

фиг. 25 -Inversion Mutation

New Generation

Това е последната стъпка от алгоритъма и при нея се решава кои индивиди ще останат в популацията. Тук има 2 възможни подхода:

1. заменя се цялата популация с наследниците
2. заменя се някакъв процент от популацията с наследниците (тук най-често се заместват индивидите с най-ниска фитнес оценка с дъщерните или се заместват индивидите на рандом принцип)

K-means

K-means алгоритмите са едни от най-често използваните алгоритми за клъстерирация, защото постигат добър баланс между сложност, бързодействие и резултати. За да се намери най-оптималното решение най-бързо, те в повечето случаи се използват съвместно с генетичен алгоритъм (подхода, който и аз съм избрала да следвам).

Този алгоритъм работи на базата на даден набор от данни, чрез предефиниран брой от клъстери. Изходът от него е K клъстера. Самият алгоритъм можете да видите описан по-долу.

1. *k* начални "среди" (*kmeans*) (в този случай $k = 3$) са произволно генериирани -на случаен принцип

2. k клъстери се създават чрез свързване на всяко наблюдение с най-близката средна стойност. Определя се за всеки индивид към кой клъстер трябва да принадлежи.

3. Центроидът на всеки от кластерите става новата средна стойност.

4. Стъпки 2 и 3 се повтарят до достигане на сходство.

Недостатък на алгоритъма е ,че ако се пусне два пъти един след друг алгоритъма ще даде различни резултати.При $K\text{-means}++$ е променена само стъпка 1-центровете не са генериирани на случаен принцип,благодарение на което ако се пусне два пъти един след друг алгоритъма ще даде еднакви резултати.

Програмна реализация

За целия проект е използвана платформата google colab и езика питон.Той включва:

Извличане на характеристики (feature)

Тъй като са налични само със звукови файлове се налага да се обработят и да бъдат извлечени техните характеристики , с които в последствие да се обучат моделите.За тази цел е използвана библиотеката либроса(Librosa).Самото извлечане се случва във функцията “extract_feature”, а след това тази информация се привежда в нужния за обучението вид във функцията “parse_audio_files”.Както може да се види всички извлечени данни се залепят и се превръщат в масив.

```
def extract_feature(file_name):
    X, sample_rate = librosa.load(file_name)
    stft = np.abs(librosa.stft(X))
    mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate,
                                         n_mfcc=40).T, axis=0)
```

```

chroma = np.mean(librosa.feature.chroma_stft(S=stft,
sr=sample_rate).T,axis=0)

mel = np.mean(librosa.feature.melspectrogram(X,
sr=sample_rate).T,axis=0)

contrast = np.mean(librosa.feature.spectral_contrast(S=stft,
sr=sample_rate).T,axis=0)

return mfccs,chroma,mel,contrast

def parse_audio_files(files_names):
    features, labels = np.empty((0,187)), np.empty(0)
    counter=0
    for fn in files_names:
        try:
            mfccs, chroma, mel, contrast =
extract_feature("drive/app/TUDiplomna/audio/"+fn)
        except Exception as e:
            counter=counter+1
            continue
        ext_features = np.hstack([mfccs,chroma,mel,contrast])
        features = np.vstack([features,ext_features])
        labels = np.append(labels, fn)
    return features,labels

tr_features, tr_files= parse_audio_files(train_files)
ts_features, ts_files = parse_audio_files(test_files)

```

Еднослойен персептрон

За целите на задачата е използвана готовата имплементация на невронна мрежа,която keras²¹ предоставя.Самия алгоритъм се състои от следните стъпки:

*Създава се опашка от слоеве

```
model = Sequential()
```

*Добавя се слой-Dense модел ,на който се подават нужните параметри.

```
model.add(Dense(187, input_dim=187,init='uniform', activation='relu'))
```

*Добавя се втори слой ,който се явява изходен .Той приема всички параметри и ги преобразува до б тези възможни изхода.

```
model.add(Dense(6, activation='softmax')
```

*Компилира се невронната мрежа с оптимайзер 'adam'(подобрен stochastic gradient descent) и се очаква да бъде изчислена метриката точност(accuracy) като резултат

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

*Обучава се мрежата за 150 епохи ,като примерите ще бъдат обучавани на серии по 10

```
model.fit(X_train, y_train_cat, epochs=150, batch_size=10)
```

*Оценява се колко сполучливо работи модела

```
scores = model.evaluate(X_test, y_test_cat, verbose=0)
```

Данните трябва да са във вид на двудименсионален масив(матрица).

Многослоен перцептрон

1. Без отпаднали връзки.

За целите на задачата отново е използвана готовата имплементация на невронна мрежа,която keras предоставя.Самия алгоритъм естествено е доста подобен

*Създава се опашка от слоеве

```
model = Sequential()
```

*Добавят се 1ви слой(Dense модел) и се подават нужните параметри -187 входни параметъра и 18 изходни.

```
model.add(Dense(18, input_dim=187, activation='relu'))
```

*Добавя се 2ри слой(Dense модел) и му се подават нужните параметри -18 изходни,а входните се подразбират-бройката ,която идва от горния слой.

```
model.add(Dense(18, activation='relu'))
```

²¹Уеб-страница:Библиотека Keras документация
<https://keras.io/>

*Добавя се трети слой ,които се явява изходен .Той приема всички параметри и ги преобразува до 6 те възможни изхода.

```
model.add(Dense(6, activation='sigmoid'))
```

*Компилира се невронната мрежа с оптимайзер 'adam'(подобрен stochastic gradient descent) и се изчислява метриката точност(accuracy) като резултат
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

*Обучава се мрежата за 150 епохи ,като примерите ще бъдат обучавани на серии по 10

```
model.fit(X_train, y_train_cat, epochs=150, batch_size=10)
```

*Оценява се колко сполучливо работи модела

```
scores = model.evaluate(X_test, y_test_cat, verbose=0)
```

Данните трябва да са във вид на двудименсионален масив(матрица).

2. С отпаднали връзки

Целта на отпадането на връзки е да не се пре-научи модела("overfitting"), което би довело нашия модел да се справя супер с тренировъчните данни и не добре с тестовите. Отново е използвана keras библиотека и самия алгоритъм се състои от следните стъпки:

*Създава се опашка от слоеве

```
model = Sequential()
```

*Добавя се 1ви слой(Dense модел) и се подават нужните параметри ,които са 18 входни параметъра и 62 изходни.

```
model.add(Dense(18, input_dim=187, activation='relu'))
```

*Прекъсват се връзките с невроните от по-долния слой на 20% от невроните.

```
model.add(Dropout(0.2))
```

*Добавя се 2ви слой(Dense модел) и се подават нужните параметри -18 изходни, а входните се подразбират-бройката ,която идва от горния слой.

```
model.add(Dense(18, activation='relu'))
```

*Прекъсват се връзките с невроните от по-долния слой на 20% от невроните.

```
model.add(Dropout(0.2))
```

*Добавя се трети слой ,които се явява изходен .Той приема всички параметри и ги преобразува до 6 те възможни изхода.

```
model.add(Dense(6, activation='sigmoid'))
```

*Компилира се невронната мрежа с оптимайзер 'adam'(подобрен stochastic gradient descent) и се изчислява метриката точност(accuracy) като резултат
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

*Обучава се мрежата за 150 епохи ,като примерите ще бъдат обучавани на серии по 10

```
model.fit(X_train, y_train_cat, epochs=150, batch_size=10)
```

*Оценява се колко сполучливо работи модела

```
scores = model.evaluate(X_test, y_test_cat, verbose=0)
```

Данните трябва да са във вид на двудименсионален масив(матрица).

Логистична регресия

За имплементацията на логистичната регресия е използвана библиотеката sklearn и най -добри резултати върху данните бяха постигнати с подразбиращите се параметри,които са:

multi_class(problem is fit for each label)=ovr

tol (Tolerance for stopping criteria)=1e-4

C (Inverse of regularization strength)=1.0

class_weight (Weights associated with classes)= None

solver (Algorithm to use in the optimization problem)='liblinear'.

max_iter(Maximum number of iterations for the solvers to converge)=100

n_jobs(Number of CPU cores used when parallelizing over classes)=None

Класовете са обучени на принципа един срещу всички ,а алгоритъма следва стъпките:

*създава се модела с подразбиращите се параметри

```
logisticRegr = LogisticRegression()
```

*Подават се тренировъчни данни и се стартира обучението

```
logisticRegr.fit(X_train, labels_train)
```

*Подават се тестовите данни и на база на тях се изчисляват метриките

```
score = logisticRegr.score(X_test,labels_test)
```

Класификационни дървета(RandomForest)

За имплементацията на този алгоритъм е избрана отново библиотеката sklearn. При него се създава набор от дървета за вземане на решения от произволно подбран набор от тренировъчни данни. След това се агрегират гласовете от различни дървета за решения, за да определи крайния клас на тествания обект. В проекта се работи с 50 дървета с максимална дълбочина 50, като самата имплементация можете да видите на фигура 26.

```
st= time.time()
classifier = RandomForestClassifier(n_estimators=50, max_depth=50, random_state=1)
classifier.fit(X_trai, Y_train)
core=classifier.score(X_test, Y_test)
print("Mean accuracy on the given test data and labels.")
print(score)
print('Metrics for Random Forest classifier')
print ('Total score: '+str(np.mean(scores)))
print ('Total time: '+str(int(time.time()-st)))
```

фиг. 26-RandomForest имплементация

Наивен Бейсов Класификатор

За имплементацията на този алгоритъм е избрана отново библиотеката sklearn. От тази библиотека се използва класификатора MultinomialNB с подразбиращите се параметри (**alpha** : 1.0 (изглажддане), **fit_prior**: True (дали да се използва класовата вероятност), **class_prior** : None (първоначални вероятности за класовете). Задава се с масив или None). Самият код можете да видите на фиг. 27.

```
st= time.time()
classifier = MultinomialNB()
classifier.fit(X_trai, Y_train)
core=classifier.score(X_test, Y_test)
print("Mean accuracy on the given test data and labels.")
print(score)
print('Metrics for Multinomial Naive bayes classificatio ')
print ('Total score: '+str(np.mean(scores)))
print ('Total time: '+str(int(time.time()-st)))
```

фиг. 27-Мултикласов Наивен Бейсов класификатор

LSTM

За рекурентната мрежа отново е използвана имплементацията на keras, като алгоритъма се състои от следните стъпки:

*Създава се опашка от слоеве

```
model = Sequential()
```

*Добавя се 1ви слой(Dense модел) и му се подават нужните параметри в нашия случай са 18 входни параметъра

```
model.add(Dense(18, input_dim=187, activation='relu'))
```

*Прекъсват се връзките с невроните от по-долния слой на 20% от невроните.

```
model.add(Dropout(0.2))
```

*Добавя се LSTM слой с 64 изхода

```
model.add(LSTM(64, return_sequences=True))
```

*Добавя се изходения слой . Той приема всички параметри и ги преобразува до 6 те възможни изхода.

```
model.add(Dense(6, activation='sigmoid'))
```

*Компилира се невронната мрежа с оптимайзер 'rmsprop'(най-подходящ за RNN) и се очаква да бъде изчислена метриката точност(accuracy) като резултат

```
model.compile(loss = 'categorical_crossentropy', optimizer='rmsprop',metrics = ['accuracy'])
```

*Обучава се мрежата за 150 епохи , като примерите ще бъдат обучавани на серии по 10

```
model.fit(X_train, y_train_cat, epochs=150, batch_size=10)
```

*Оценява се колко сполучливо работи модела

```
scores = model.evaluate(X_test, y_test_cat, verbose=0)
```

MeanShift

Отново се използва стандартната имплементация предложена от sklearn библиотеката с подразбиращите се стойности за параметрите. Самата имплементация може да се види на фигура 28.

```

start = time.time()
#https://scikit-learn.org/stable/modules/clustering.html#mean-shift
clustering = MeanShift()
cluster_labels = clustering.fit_predict(X)
silhouette_avg = silhouette_score(X, cluster_labels)
print("The average silhouette_score is :", silhouette_avg)
end = time.time()
print("Time for executing mean shift model")
print(end - start)
print(" in seconds")

```

фиг. 28-MeanShift алгоритъм с изчисляване на silhouette score

Kmeans алгоритми съчетани с генетичен алгоритъм

За имплементация на генетичния е избрана библиотеката pyeasyga²², тъй като тя дава както възможност за пренаписване на функционалностите ,така и дефолтна имплементация на всяка една от стъпките(която е използвана). Единствено от целия алгоритъм задължително тряба да се пренапише фитнес функцията. Конкретното решение на оптимизационния проблем е намерено ,като се търси най-голям скок между оценките на индивидите. Конкретната имплементация е описана във fitness функцията по-долу.

```

def fitness(individual, data):
    temp=0.77;
    cNum=1
    for x in data:
        if temp / float(2.45) > x['score']:
            temp=x['score']
            cNum=x['clusterNum']
    return cNum

```

²² Уеб-страница: Документация за pyeasyga извлечено на 28/12/2018 г.
<https://readthedocs.org/projects/pyeasyga/>

```
start = time.time()
```

За алгоритмите Kmeans и Kmeans++ се използва дефолтната имплементация, коята sklearn предоставя. За съчетаването на кълстеризиационните алгоритми с генетичен алгоритъм се минава през следните стъпки:

*Събира се оценката за всеки възможен брой на кълстери(аналогично е и за Kmeans++)

```
for i in range(2,20):
```

```
    cluster_labels = KMeans(n_clusters=i, init='random',
                           random_state=10).fit_predict(X)
    data.append({'clusterNum': i, 'score': silhouette_score(X, cluster_labels)})
```

*Инициализира се генетичният алгоритъм ,като му се подават събранныте данни

```
ga = pyeasyga.GeneticAlgorithm(data)
```

*Задава се фитнес функцията

```
ga.fitness_function = fitness
```

*Стартира се генетичният алгоритъм

```
ga.run()
```

*Намира се най-добрия резултат

```
print ga.best_individual()
```

Анализ на експерименталните резултати

Класификация-за измерване ефективността на алгоритмите ще бъдат използвани метриките: бързодействие(време за изпълнение) и accuracy, което представлява точността на модела ,която математически можем да представим със следната формула:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

True positive (TP) = случаите ,които коректно са класифицирани, че са x

False positive (FP)=случаите ,които некоректно са класифицирани, че са x

True negative (TN) = случаите ,които коректно са класифицирани, че не са x

False negative (FN)= случаите, които некоректно са класифицирани, че не са x

Accuracy-то ще бъде изчислено върху тестовото множество след като са обучени моделите.

алгоритъм	време за изпълнение	accuracy
<i>One layer perceptron</i>	11.84 сек	73.00%
<i>Multilayer perceptron</i>	8.12 сек	79.00%
<i>Logistic regression</i>	2.83 сек	75.00%
<i>Random Forest</i>	~0 сек	77.00%
<i>Recurrent Neural Network</i>	0.31 сек	79.00%

таблица 1- Сравнителна характеристика на класификационните модели

Както може да се види на таблица 1, по-простите модели се представят доста прилично и с усложняването на модела се постига съвсем леко подобрене на точността. По отношение на бързодействието на моделите очаквано *Logistic regression*, *Random Forest* и *Recurrent Neural Network* се представлят най-добре, тъй като те използват стандартна имплементация, а архитектурата на перцептронните е къстамизирана.

Клъстеризация-за измерване ефективността на алгоритмите ще бъде използвани метриките: бързодействие(време за изпълнение) и silhouette_score.²³ Silhouette score се изчислява, като се използва средното

²³ Уеб-страница: Документация за silhouette score извлечено на 28/12/2018 г.
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html

междуклъстерно разстояние (a) и средното най-близко разстояние (b) за всяка проба. Математически се изислява чрез формулата: $(b - a) / \max(a, b)$. За да се изясни, b е разстоянието между пробата и най-близкия кълстър, от който пробата не е част. Обърнете внимание, че Silhouette Coefficient се дефинира само ако броят на етикетите(кълстърите) е $2 \leq n_labels \leq n_samples - 1$. Тази функция връща средния silhouette score над всички преби. Най-добрата стойност е 1, а най-лошата стойност е -1. Стойностите в близост до 0 показват припокриващи се кълстъри. Отрицателните стойности обикновено показват, че проба е присвоена на грешен кълстър, тъй като различен кълстър е по-сходен.

алгоритъм	време за изпълнение	Silhouette score	Брой на създадените кълстъри
MeanShift	2.53 сек	0.30	43
KMeans	3.27 сек	0.20	7
KMeans++	3.22 сек	0.14	6

таблица 2- Сравнителна характеристика на кълстъризационните модели В таблица 2 можете да видите ,че алгоритмите от семейството на KMeans дават много по-добри резултати,въпреки ,че са по-прости.

Заключение

Като цяло бяха постигнати доста добри резултати съобразно данните с които разполагаме, но за подобряване представянето на моделите трябва да се работи върху подобреие в извлечането на данни. Това може да стане като се добавят допълнителни характеристики(features) като:tempo,

beats,roll-off frequency,spectral flatness, tempogram,spectral bandwidth и други или се заменят някой от досега съществуващите. Трябва много да се внимава като се добавят нови характеристики (features), защото те вместо подобрение могат да донесат шум в системата. Друга възможна посока за бъдеща работа е да се тестват по-голям брой модели с цел подобряване на точността. За класификационната задача може да се добави конволюционна невронна мрежа и наивен бейсов класификатор, а с цел кълстерилизация

-Agglomerative Hierarchical Clustering и Expectation–Maximization

Clustering. Тези модели ще са подходящи за данните ,защото:

-CNN(convolutional neural network)-Има множество изследвания ,че този вид мрежи дават добри резултати за класификация на музикални файлове ,тъй като откриват скрити зависимости. Пример за такова е изследването на Kevin Wilson ”CNN ARCHITECTURES FOR LARGE-SCALE AUDIO CLASSIFICATION”²⁴

²⁵Agglomerative Hierarchical Clustering с архитектура bottom-up-Тъй като при този модел се приема първоначално всяка единица за кълстер и след това се слепват на база на между кълстерно разстояние и затова би трябвало да се получат добри резултати.

-Expectation–Maximization Clustering(EM)-този модел носи плюсовете на Kmeans семейството, но предлага много по-голяма гъвкавост ,тъй като данните са с гаусова дистрибуция .

²⁴ Уеб-страница: Документация за silhouette score извлечено на 28/12/2018 г.
<https://arxiv.org/pdf/1609.09430v2.pdf>

²⁵ Уеб-страница: Документация за silhouette score извлечено на 28/12/2018 г.
<https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68>

Приложения

Copс код:

```
!apt-get install -y -qq software-properties-common python-software-properties  
module-init-tools  
!add-apt-repository -y ppa:alessandro-strada/ppa 2>&1 >/dev/null  
!apt-get update -qq 2>&1 >/dev/null  
!apt-get -y install -qq google-drive-ocamlfuse fuse  
  
# Colab用のAuth token作成  
from google.colab import auth  
auth.authenticate_user()  
  
# Drive FUSE library用のcredential生成  
from oauth2client.client import GoogleCredentials  
creds = GoogleCredentials.get_application_default()  
import getpass  
!google-drive-ocamlfuse -headless -id={creds.client_id}  
-secret={creds.client_secret} </dev/null 2>&1 | grep URL  
vcode = getpass.getpass()  
!echo {vcode} | google-drive-ocamlfuse -headless -id={creds.client_id}  
-secret={creds.client_secret}  
  
# drive/を作り、そこにGoogle Driveをマウントする  
!mkdir -p drive  
!google-drive-ocamlfuse drive
```

```

#colab

# google-drive-ocamlfuseのインストール
# https://github.com/astrada/google-drive-ocamlfuse
!apt-get install -y -qq software-properties-common python-software-properties
module-init-tools
!add-apt-repository -y ppa:alessandro-strada/ppa 2>&1 >/dev/null
!apt-get update -qq 2>&1 >/dev/null
!apt-get -y install -qq google-drive-ocamlfuse fuse

# Colab用のAuth token作成
from!pip install git+https://github.com/librosa/librosa
import librosa
google.colab import auth
auth.authenticate_user()

# Drive FUSE library用のcredential生成
from oauth2client.client import GoogleCredentials
creds = GoogleCredentials.get_application_default()
import getpass
!google-drive-ocamlfuse -headless -id={creds.client_id}
-secret={creds.client_secret} </dev/null 2>&1 | grep URL
vcode = getpass.getpass()
!echo {vcode} | google-drive-ocamlfuse -headless -id={creds.client_id}
-secret={creds.client_secret}

# drive/を作り、そこにGoogle Driveをマウントする
!mkdir -p drive
!google-drive-ocamlfuse drive

```

"""\Install and import Libraries""""

```
!pip install git+https://github.com/librosa/librosa  
!pip install np_utils  
!pip install keras  
!pip install pyeasyga
```

```
import librosa  
import numpy as np  
import pandas as pd  
import random  
import time  
import glob  
import os  
import tensorflow as tf  
import np_utils  
import pyeasyga  
  
from pyeasyga import pyeasyga  
from os import listdir  
from os.path import isfile, join  
  
from keras.layers.recurrent import LSTM  
from keras.models import Sequential  
from keras.layers import Dense  
from keras.models import Sequential  
from keras.layers import Dense  
from keras.layers import Dropout  
from keras.wrappers.scikit_learn import KerasClassifier  
from keras.utils import np_utils  
from keras.utils.np_utils import to_categorical
```

```
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.cluster import MeanShift
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import StratifiedKFold
```

"***Load Data***"

```
def load_sound_files(file_paths):
    raw_sounds = []
    for fp in file_paths:
        fp="drive/app/TUDiplomna/audio/"+fp
        X,sr = librosa.load(fp)
        raw_sounds.append(X)
    return raw_sounds

start = time.time()
mypath="drive/app/TUDiplomna/audio/"
labelsN=["Acoustic_guitar", "Applause", "Bark", "Bass_drum",
"Burping_or_eructation", "Bus", "Cello", "Chime", "Clarinet",
"Computer_keyboard"]
onlyfiles = []
```

```

tr_labels=pd.read_csv("drive/app/TUDiplomna/train.csv" ,
error_bad_lines=False)
for f in range(len(tr_labels)):
    if tr_labels.iloc[f].label in labelsN:
        onlyfiles.append(tr_labels.iloc[f].fname)
end = time.time()
print("Time for data loading:")
print(end - start)
print("seconds")
random.shuffle(onlyfiles,random.random) #mix files
train_files= onlyfiles[:500]
test_files= onlyfiles[500:600]

```

"""\#Feature Extraction"""

```

def extract_feature(file_name):
    X, sample_rate = librosa.load(file_name)
    stft = np.abs(librosa.stft(X))
    mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate,
n_mfcc=40).T,axis=0)
    chroma = np.mean(librosa.feature.chroma_stft(S=stft,
sr=sample_rate).T,axis=0)
    mel = np.mean(librosa.feature.melspectrogram(X,
sr=sample_rate).T,axis=0)
    contrast = np.mean(librosa.feature.spectral_contrast(S=stft,
sr=sample_rate).T,axis=0)
    #tonnetz =
    np.mean(librosa.feature.tonnetz(y=librosa.effects.harmonic(X),sr=sample_rate).
T,axis=0)
    return mfccs,chroma,mel,contrast

```

```

def parse_audio_files(files_names):
    features, labels = np.empty((0,187)), np.empty(0)
    counter=0
    for fn in files_names:
        try:
            mfccs, chroma, mel, contrast =
            extract_feature("drive/app/TUDiplomna/audio/"+fn)
        except Exception as e:
            counter=counter+1
            continue
        ext_features = np.hstack([mfccs,chroma,mel,contrast])
        features = np.vstack([features,ext_features])
        labels = np.append(labels, fn)
        #print(counter)
    return features,labels

start = time.time()
tr_features, tr_files= parse_audio_files(train_files)
ts_features, ts_files = parse_audio_files(test_files)
end = time.time()
print("Time for extracting test and train features:")
print(end - start)
print("seconds")

def lablesExtraction(files,labels):
    labelsSet=[]
    for i in range(0, len(files)):
        for j in range(0, 9472):#check where this num come from
            if files[i]==labels['fname'].values[j]:

```

```

label=labels['label'].values[j]
if label != "Acoustic_guitar" and label != "Applause" and label != "Bark"
and label != "Bass_drum" and label != "Bus":
    labelsSet.append('None')
else:
    labelsSet.append(label)
break
return labelsSet

def oneHotEncoder(labels):
    encoder = LabelEncoder()
    encoder.fit(labels)
    encoded_Y = encoder.transform(labels)

start = time.time()

tr_features, tr_files= parse_audio_files(train_files)
ts_features, ts_files = parse_audio_files(test_files)

X_train=pd.DataFrame(tr_features)
labels_train=lablesExtraction(tr_files,tr_labels)
Y_train=oneHotEncoder(labels_train)
#pd.DataFrame(tr_labels)

X_test=pd.DataFrame(ts_features)
labels_test=lablesExtraction(ts_files,tr_labels)
Y_test=oneHotEncoder(labels_test)

end = time.time()
print("Time for extracting test and train features and labels:")

```

```

print(end - start)
print("seconds")

"""\#\#\#Classification*\*\*\*\*"""

def baseline_model():
    # create model -Baseline: 53.72% (15.21%)
    model = Sequential()
    model.add(Dense(187, input_dim=187, activation='relu'))
    model.add(Dense(6, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam',
    metrics=['accuracy'])
    return model

def nn_dropout():#Baseline: 75.87% (9.01%)
    model = Sequential()
    model.add(Dense(18, input_dim=187, init='uniform', activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(18, init='uniform', activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(6, init='uniform', activation='softmax'))

    model.compile(loss='categorical_crossentropy', optimizer='rmsprop',
    metrics=['accuracy'])
    return model

start = time.time()
model = baseline_model()
# Fit the model

```

```

model.fit(X_train, Y_train, epochs=150, batch_size=10, verbose=0)
# evaluate the model
scores = model.evaluate(X_test, Y_test, verbose=0)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
end = time.time()
print("Time for executing one layered perceptron")
print(end - start)
print("seconds")

start = time.time()
model = Sequential()
model.add(Dense(12, input_dim=187, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(6, activation='sigmoid'))
# Compile model
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
# Fit the model
history=model.fit(X_train, Y_train, epochs=150, batch_size=10,
verbose=0)#history
end= time.time()

print("compile time in sec:")
print(end-start)
# evaluate the model
scores = model.evaluate(X_test, Y_test, verbose=0)
print("%s: %.2f%%" % (model.metrics_names[0], scores[0]*100))
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
end = time.time()
print("Time for executing two layers perceptron")

```

```

print(end - start)
print("seconds")

start = time.time()
model = nn_dropout()
# Fit the model
model.fit(X_train, Y_train, epochs=150, batch_size=10, verbose=0)
# evaluate the model
scores = model.evaluate(X_test, Y_test, verbose=0)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
end = time.time()
print("Time for executing two layers perceptron with drop out")
print(end - start)
print(" in seconds")

start = time.time()
# all parameters not specified are set to their defaults
logisticRegr = LogisticRegression()
logisticRegr.fit(X_train, labels_train)
# Use score method to get accuracy of model
score = logisticRegr.score(X_test, labels_test)
print("Accuracy:")
print(score)
end = time.time()
print("Time for executing linear regression model")
print(end - start)
print(" in seconds")

st= time.time()
classifier = MultinomialNB()

```

```

classifier.fit(X_trai, Y_train)
core=classifier.score(X_test, Y_test)
print("Mean accuracy on the given test data and labels.")
print(score)
print('Metrics for Multinominal Naive bayes classificatio ')
print ('Total score: '+str(np.mean(scores)))
print ('Total time: '+str(int(time.time()-st)))

st= time.time()
classifier = RandomForestClassifier(n_estimators=50, max_depth=50,
random_state=1)
classifier.fit(X_trai, Y_train)
core=classifier.score(X_test, Y_test)
print("Mean accuracy on the given test data and labels.")
print(score)
print('Metrics for Random Forest classifier')
print ('Total score: '+str(np.mean(scores)))
print ('Total time: '+str(int(time.time()-st)))

st= time.time()
classifier = RandomForestClassifier(n_estimators=50, max_depth=50,
random_state=1)
classifier.fit(X_trai, Y_train)
core=classifier.score(X_test, Y_test)
print("Mean accuracy on the given test data and labels.")
print(score)
print('Metrics for Random Forest classifier')
print ('Total score: '+str(np.mean(scores)))
print ('Total time: '+str(int(time.time()-st)))

```

```

#RNN
embed_dim = 128
lstm_out = 200
batch_size = 32

model = Sequential()
model.add(Dense(64, input_dim=187, init='uniform', activation='relu', dropout = 0.2))
model.add(LSTM(64, return_sequences=True))
model.add(Dense(6,activation='softmax'))
model.compile(loss = 'categorical_crossentropy', optimizer='adam',metrics = ['accuracy'])
print(model.summary())

"""#Clusterization"""

start = time.time()
#https://scikit-learn.org/stable/modules/clustering.html#mean-shift
clustering = MeanShift()
cluster_labels = clustering.fit_predict(X)
silhouette_avg = silhouette_score(X, cluster_labels)
print("The average silhouette_score is :", silhouette_avg)
end = time.time()
print("Time for executing mean shift model")
print(end - start)
print(" in seconds")

# define a fitness function -da q napisha po dobre
def fitness(individual, data):
    temp=0.77;

```

```

cNum=1
for x in data:
    if temp / float(2.45) > x['score']:
        temp=x['score']
        cNum=x['clusterNum']
return cNum

start = time.time()
#KMeans(random init)
data=[];
for i in range(2,20):
    cluster_labels = KMeans(n_clusters=i, init='random',
random_state=10).fit_predict(X)
    data.append({'clusterNum': i , 'score': silhouette_score(X, cluster_labels)})

# initialise the GA with data
ga = pyeasyga.GeneticAlgorithm(data)
ga.fitness_function = fitness          # set the GA's fitness function
ga.run()                               # run the GA
print ga.best_individual()
end = time.time()
print("Time for executing Kmeans with GA")
print(end - start)
print(" in seconds")

start = time.time()
#KMeans++
data=[];
for i in range(2,20):
    cluster_labels = KMeans(n_clusters=i, init='random').fit_predict(X)

```

```
data.append({'clusterNum': i , 'score': silhouette_score(X, cluster_labels)})\n\n# initialise the GA with data\n\nga = pyeasyga.GeneticAlgorithm(data)\n\nga.fitness_function = fitness          # set the GA's fitness function\n\nga.run()                            # run the GA\n\nprint(nga.best_individual())\n\nend = time.time()\n\nprint("Time for executing Kmeans++ with GA")\n\nprint(end - start)\n\nprint(" in seconds")
```