Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій

Курсова робота на тему:

Методи класифікації для різних задачах машинного навчання.

Виконала:
студентка 6-го курсу
групи IAB21
Сокол Олена

Київ 2018

Зміст

Вступ

Методи класифікації

Класифікація - це віднесення об'єктів до певного класу по набору ознак. Наприклад, розпізнавання номерів машин, або в медицині діагностика захворювань, або кредитний скоринг в банківській сфері. Це один з розділів машинного навчання, що присвячений вирішенням наступного завдання. Нехай ми маємо безліч об'єктів (ситуацій), розділених деяким чином на класи. Дано скінченну кількість об'єктів, для яких відомо, до яких класів вони належать. Ці об’єкти називається навчальною вибіркою. Класова приналежність інших об'єктів невідома. Потрібно побудувати алгоритм, здатний класифікувати довільний об'єкт.

Класифікувати об'єкт - значить, вказати номер (або найменування класу), до якого належить даний об'єкт.

Розділ1. Типи задач машинного навчання

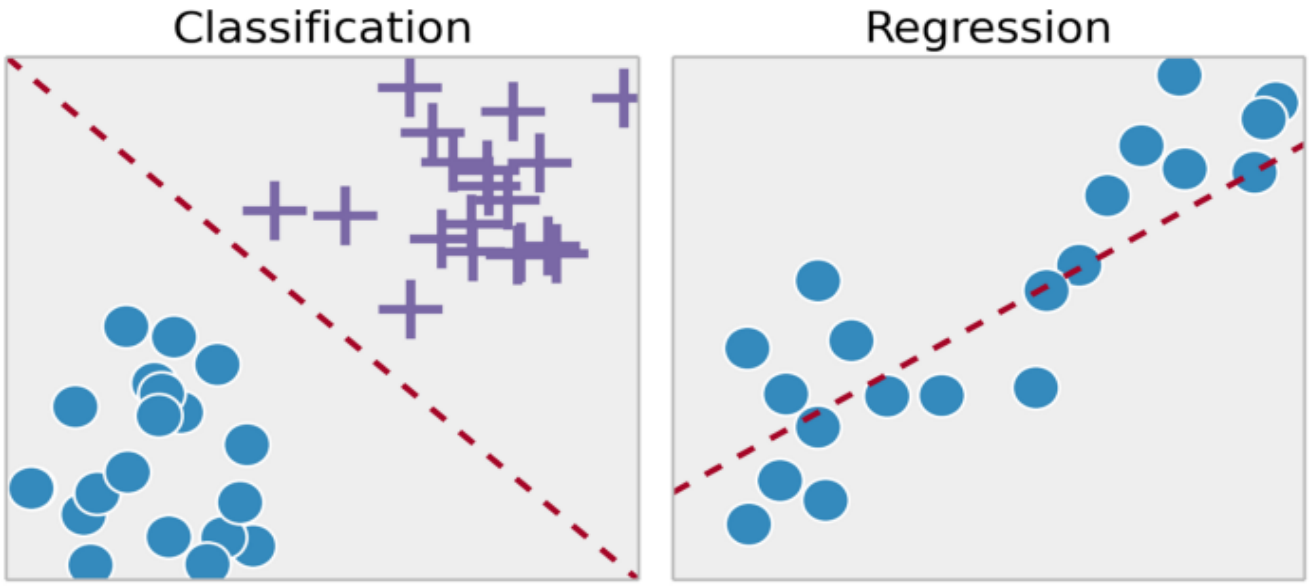
У машинному навчанні є 4 типи задач:

- Навчання з учителем;
- Навчання без вчителя;
- Навчання з частковим залученням вчителя;
- Навчання з підкріпленням.

Навчання з вчителем

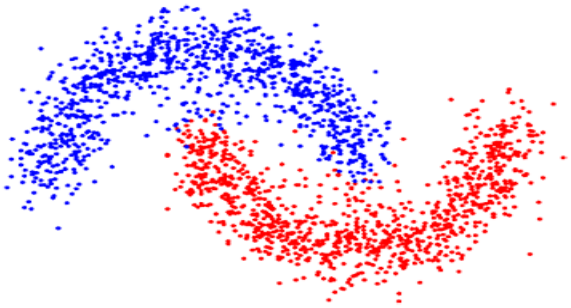
Навчання з учителем - це навчання на тренувальному наборі даних, шляхом підгонки результатів навчання на тренувальний набір даних.

Основне завдання - знайти найбільш оптимальні параметри моделі для прогнозування. Якщо результат прогнозу є дійсним числом, то це завдання регресії. Якщо має обмежену кількість значень, де ці значення є неврегульованими, то це завдання класифікації.



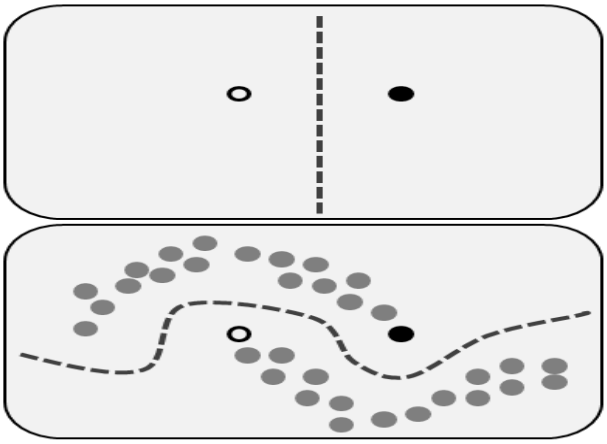
Навчання без вчителя

У навчанні без контролю у нас менше інформації про об'єкти. Зокрема, тренувальний набір не має маркованих даних, що відносяться до певного класу заздалегідь зумовлених даних. Яка наша мета зараз? Спостерігати деяку схожість між групами об'єктів і включати їх до відповідних кластерів.



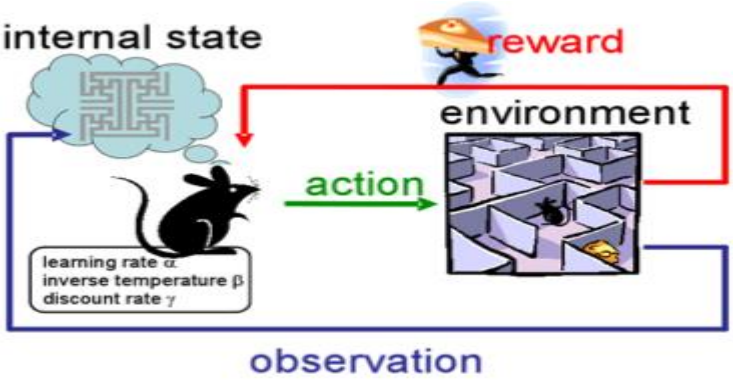
Навчання з частковим залученням вчителя

Навчання з частковим залученням вчителя включає обидві проблеми, які ми описали раніше: вони використовують марковані і немарковані дані. Це відмінна можливість для тих, хто не може розподілити маркувати свої дані. Цей метод дозволяє значно підвищити точність, оскільки ми можемо використовувати немарковані дані в тренувальному наборі даних з невеликою кількістю маркованих даних.



Навчання з підкріпленням

Навчання з підкріпленням не схоже на будь-яку з наших попередніх завдань, тому що тут ми не маємо в своєму розпорядженні ні зумовленими маркованими даними, ні немаркованими наборами даних. Навчання з підкріпленням - область машинного навчання, пов'язана з тим, як агенти програмного забезпечення повинні вживати дії в деякому середовищі, щоб максимізувати деяке поняття кумулятивної нагороди.
Ідея навчання з підкріпленням полягає в тому, що система буде вчитися в середовищі, взаємодіяти з нею і отримувати винагороду за виконання дій.



Уявіть, що ви робот в якомусь дивному місці. Ви можете виконувати дії і отримувати нагороди від навколишнього середовища за них. Після кожної дії поведінка навколишнього середовища стає більш складнішою тому ви тренуєтеся, щоб вести себе найбільш ефективним способом на кожному кроці. У біології це називається адаптацією до природного середовища.

У машинному навчанні завдання класифікації відноситься до розділу навчання з учителем. Навчання без вчителя - кластеризація або таксономія, класи називаються - кластери або таксономи.

Кластеризація & Класифікація

Кластеризація (або кластерний аналіз) - це задача розбиття множини об'єктів на групи, які називаються кластерами. Усередині кожної групи повинні виявитися «схожі» об'єкти, а об'єкти різних групи повинні бути якомога більш відмінні. Головна відмінність кластеризації від класифікації полягає в тому, що перелік груп чітко не заданий і визначається в процесі роботи алгоритму.

SWOT аналіз

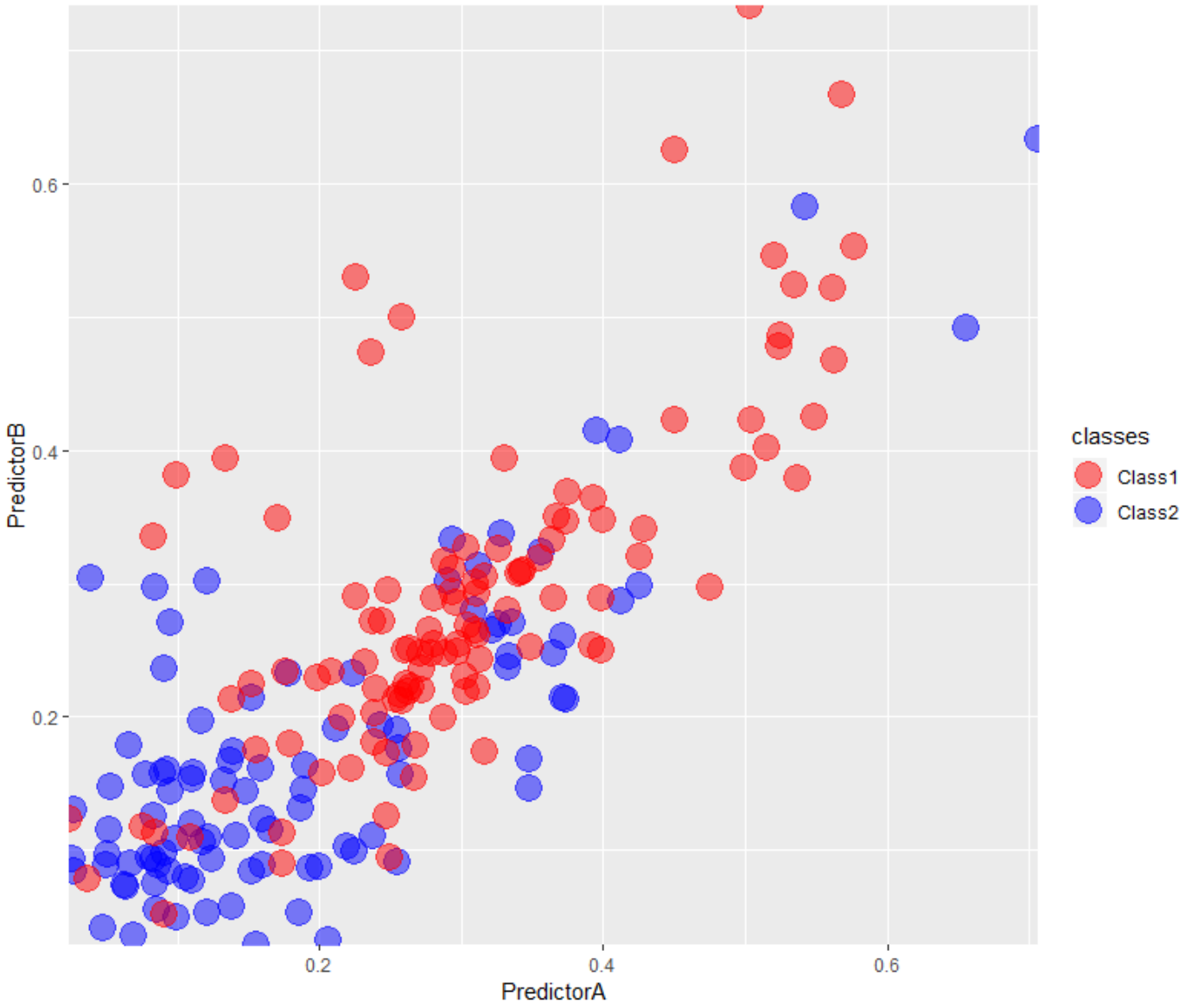
• Робить прогноз більш точним; • Можна вказати кількість груп • Задаємо свої умови на поділ груп	• Складніший в обчисленні; • Не відшукує всі зв’язки у класі, тільки за вказаним класифікатором;
• Швидкість в обрахунках; • Використовувати «сирі» дані	• Вдосконалення методів МН, що може витіснити дані методи

Розділ3. Методи класифікації

Реалізуємо їх на мові програмування R. Надалі будемо використовувати тренувальну вибірку «twoClassData», що складається із

«classes» - («Class1», «Class2») класи, на які будемо розподіляти нашу модель;

«PredictorA», «PredictorB» - змінні, на основі яких спостереження(об’єкти) будуть розподілені на класи.



Naive Bayes classifier

Наївний байєсовий класифікатор - це сімейство алгоритмів класифікації при якому кожен параметр класифікації даних розглядається незалежно від інших параметрів. Два параметра вважаються незалежними, коли значення одного параметра не впливають на інший.

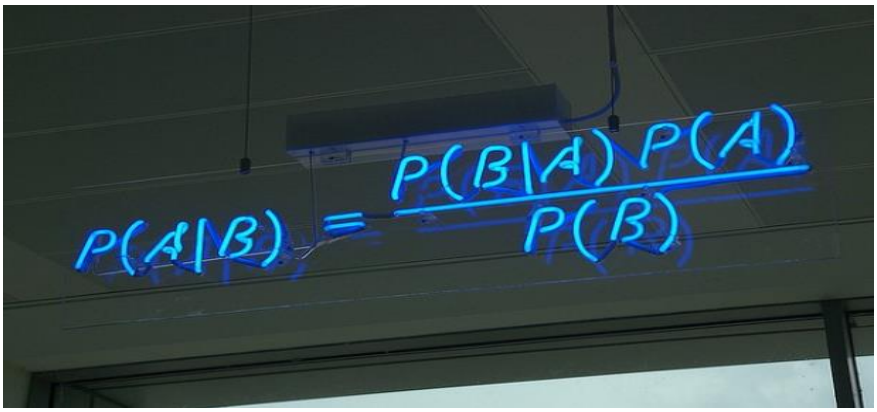
Наприклад:

Скажімо, у нас є набір даних про пацієнта: пульс, рівень холестерину, вага, ріст і поштовий індекс. Всі параметри будуть незалежними, якщо значення всіх параметрів не впливають один на одного. Для цього набору даних розумно припустити, що ріст пацієнта і поштовий індекс незалежні, оскільки ріст людини і його поштовий індекс ніяк не пов'язані. Але давайте подивимося далі, чи всі параметри незалежні? На жаль, ні. Є співвідношення, які залежні:

- якщо ріст збільшився, ймовірно, збільшилася вага;
- якщо збільшився рівень холестерину, ймовірно, збільшилася вага;
- якщо збільшився рівень холестерину, ймовірно, збільшився пульс.

Це підводить нас до іншого питання ... Чому метод називається наївним? Припущення, що всі параметри набору даних незалежні - це досить наївне припущення. Зазвичай так не буває.

Томас Байес був англійським математиком-статистиком, в честь якого була названа теорема Байєса.



По суті, теорема дозволяє нам передбачити клас на підставі набору параметрів, використовуючи ймовірність. На основі наших даних представимо рівняння для класифікації:

P(Class1| PredictorA, PredictorB) =

P(PredictorA|Class1)* P(PredictorB|Class1)*P(Class1) / P(PredictorA) * P(PredictorB).

Рівняння знаходить ймовірність класу 1, на підставі параметрів A і B.

Приведемо приклад:

Дано набір даних - 1000 фруктів.

Фрукт може бути бананом, апельсином або яким-небудь іншим (це класи).

Фрукт може бути довгим, солодким або жовтим (це параметри).

Class	Long	Sweet	Yellow	Total
Banana	400	350	450	500
Orange	0	150	300	300
Other	100	150	50	200
Total	500	650	800	1000

Тобто, якщо ми отримаємо тільки параметри - довжину, вміст цукру і колір фрукта (не знаючи його класу), то зможемо обчислити ймовірність того, що фрукт виявиться бананом, апельсином або чимось іншим.

Припустимо, що невідомий фрукт довгий, солодкий і жовтий. Для обчислення ймовірності потрібно виконати 4 простих кроки:

Крок 1: Щоб обчислити вірогідність того, що невідомий фрукт - це банан, давайте спочатку вирішимо, чи схожий цей фрукт на банан. Ось як обчислюється ймовірність класу «Банан» на підставі параметрів «довгий», «солодкий», «жовтий»:

P(Banana|Long, Sweet, Yellow)

Крок 2: Почнемо з чисельника і підставимо все значення в рівняння:

1. P(Long|Banana) = 400/500 = 0.8
2. P(Sweet|Banana) = 350/500 = 0.7
3. P(Yellow|Banana) = 450/500 = 0.9
4. P(Banana) = 500/1000 = 0.5

0.8 x 0.7 x 0.9 x 0.5 = 0.252

Крок 3: Проігноруємо знаменник, оскільки він буде однаковим для всіх наступних обчислень.

Крок 4: Проробимо ті ж обчислення для інших класів:

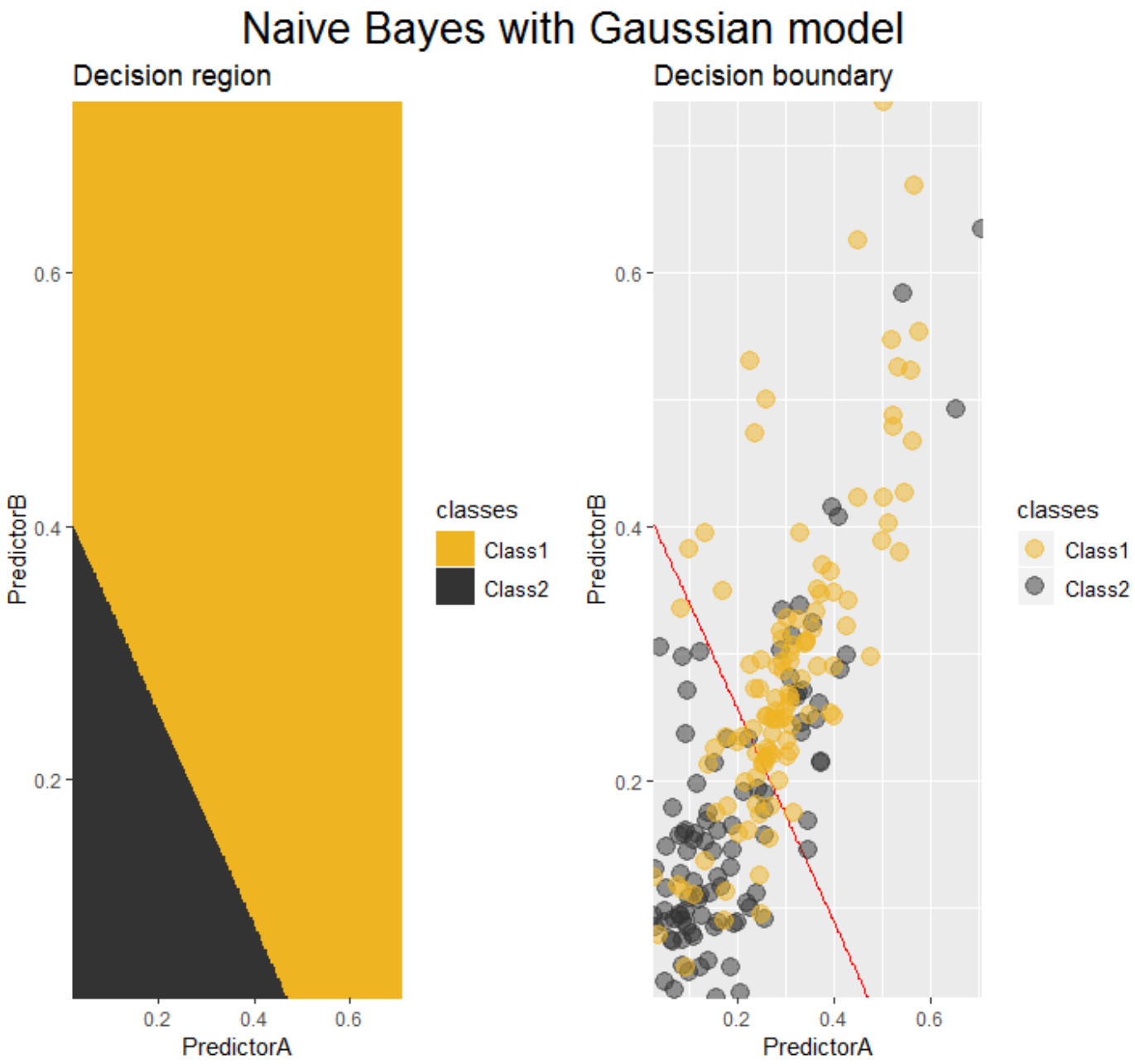
P (Orange | Long, Sweet, Yellow) = 0

P (Other | Long, Sweet, Yellow) = 0.01875

Оскільки 0,252 більше, ніж 0,01875, то наївний байєсовий алгоритм класифікує цей довгий, солодкий і жовтий фрукт як банан.

Отже, алгоритм складається з простої арифметики. Це прості обчислення: множення і ділення. Незважаючи на простоту, наївний байесовский алгоритм може бути дуже точним. Наприклад, було встановлено, що він може бути успішно застосований для фільтрації спаму.

Реалізуємо алгоритм на мові R(використовуючи тренувальну вибірку «twoClassData»):



Тепер ми будемо оцінювати нашу модель за 4 критеріями:

Accuracy - емпірична точність, обчислена на початковому наборі даних.

AccuracyCV - середнє значення точності, отримане схемою перехресної перевірки (cross-validation).

AccuracyInf - найнижча межа точності.

AccuracyPAC - середнє значення точності(AccuracyCV) віднімаємо від стандартного відхилення отримане схемою перехресної перевірки (cross-validation).

Accuracy	AccuracyCV	AccuracyCVInf	AccuracyCVPAC
0.75	0.735	0.706	0.657

Logistic regression

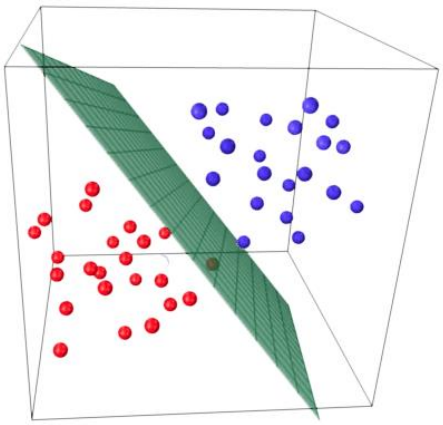
Лінійна регресійна модель не завжди здатна якісно передбачати значення залежної змінної. Вибираючи для побудови моделі лінійне рівняння, ми природним ніяк не накладаємо ніяких обмежень на значення залежної змінної. А такі обмеження можуть бути істотними. Наприклад, при проектуванні оптимальної довжини шахти ліфта в новій будівлі необхідно врахувати, що ця довжина не може перевищувати висоту будівлі взагалі.

Лінійна регресійна модель може дати результати, несумісні з реальністю. З метою вирішення даних проблем корисно змінити вид рівняння регресії і підстроїти його для вирішення конкретного завдання.

Взагалі, логіт регресійна модель призначена для передбачення значень на інтервалі від 0 до 1. Враховуючи таку специфіку, її часто використовують для передбачення ймовірності настання деякої події в залежності від значень деякого числа предикторів.

Можна використовувати логіт регресію і для вирішення завдань з бінарним класифікатором. Такі завдання з'являються, коли залежна змінна може приймати тільки два значення. Наведемо конкретний приклад. Нехай потрібно передбачити ефективність операції з пересадки серця. Такі операції дуже складні і результату від їх проведення може бути тільки два-пацієнт живий чи помер (точніше, пережив він місяць після трансплантації - цей термін є визначальним).

В якості предикторів використовуються дані передопераційного обстеження і клінічні параметри, наприклад, вік, рівень холестерину в крові, тиск, група крові і т.д. Завдання зводиться до класифікації пацієнтів на дві групи. Для першої групи прогноз позитивний, для другої - негативний. Рішення такого завдання може вплинути на прийняття рішення про проведення операції - чи варто взагалі проводити пересадку, якщо ймовірність пережити місяць після трансплантації для пацієнта невелика?



Математична основа логістичної регресії

Отже, як вже було сказано, в логіт регресійній моделі передбачені значення залежної змінної або змінної відгуку не можуть бути менше (або рівними) 0, або більше (або рівними) 1, не залежно від значень незалежних змінних; тому, ця модель часто використовується для аналізу бінарних залежних змінних або змінних відгуку.

При цьому використовується наступне рівняння регрессіі:

y = exp (b0 + b1 * x1 + ... + bn * xn) / [1 + exp (b0 + b1 * x1 + ... + bn * xn)]

Легко побачити, що незалежно від регресійних коефіцієнтів чи величин x, передбачені значення (y) в цій моделі завжди будуть лежати в діапазоні від 0 до 1.

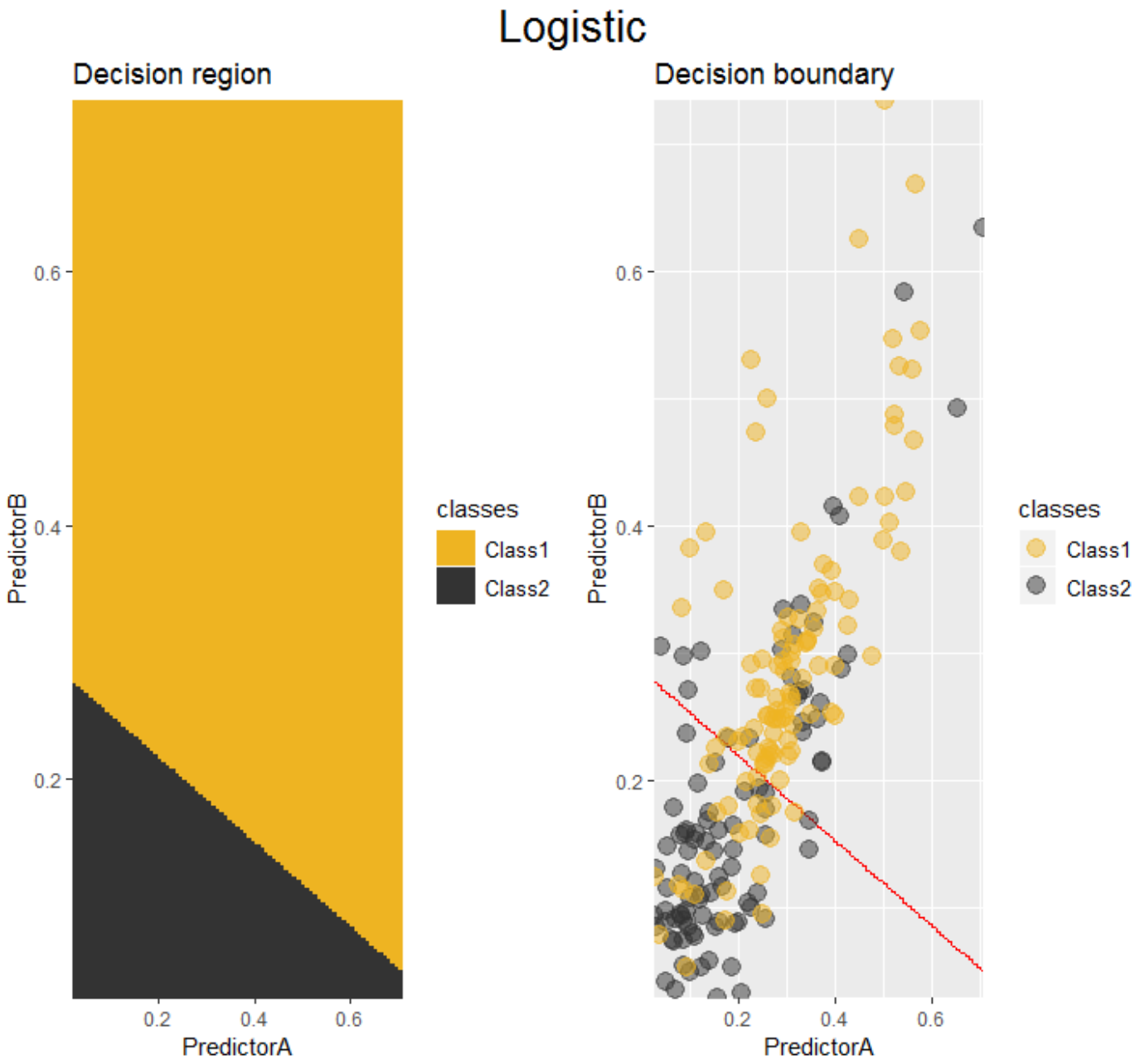
Логістична регресія – один з найпопулярніших методів класифікації, адже цей метод досить точно і швидко працює з великим об’ємом вибірки. Також при вирішенні задач класифікації об'єкти можна розділяти на кілька груп. Наприклад, не тільки - (0 - поганий, 1 - хороший), а й кілька груп (1, 2, 3, 4 групи ризику).

Проте є недоліки при роботі із малою кількістю спостережень у вибірці. Коли розмірність вибірки(кількість спостережень) < 500, то можливе завищення оцінки коефіцієнтів регресії.

Присутні обов’язкові умови до вибірки при побудові моделі. Потрібно мінімально 10 спостережень на кожну незалежну змінну (рекомендоване значення 30-50):

Наприклад, смерть пацієнта. Якщо 50 пацієнтів з 100 вмирають, то максимальне число незалежних змінних в моделі = 50/10 = 5.

Реалізуємо алгоритм на мові R(використовуючи тренувальну вибірку «twoClassData»):



Accuracy	AccuracyCV	AccuracyCVInf	AccuracyCVPAC
0.75	0.754	0.718	0.656

Бачимо покращення у моделі, але вони не суттєві.

Decision Trees(C4.5, CART, Random Forests)

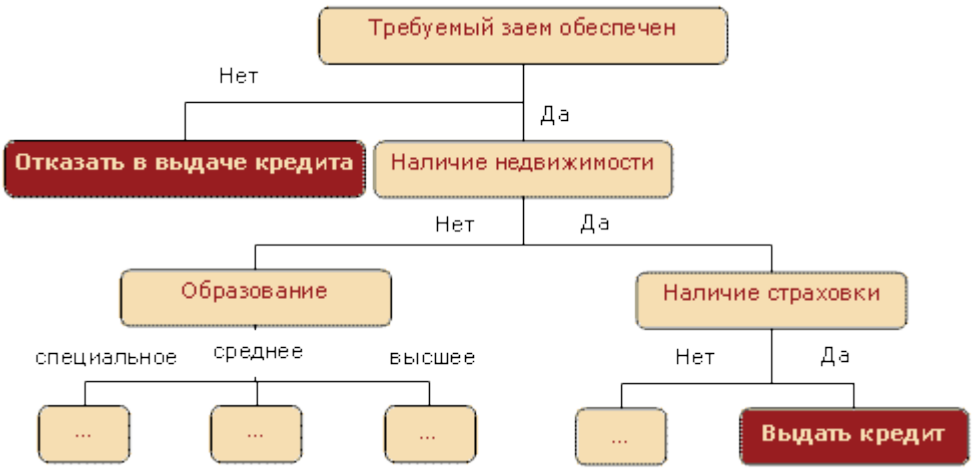
C4.5

Алгоритм C4.5 будує класифікатор в формі дерева рішень.

Припустимо, що у нас є набір даних - це дані про групу потенційних клієнтів банку. Ми знаємо різні параметри кожного пацієнта: вік, освіта, наявність страхування наявність нерухомості, історію сім'ї і так далі (це параметри). На підставі цих параметрів ми хочемо передбачити, чи може пацієнт отримати кредит. Пацієнт може

потрапити в один з 2 класів: отримає та не отримає кредит. Алгоритму C4.5 повідомляє клас кожного клієнта.

Використовуючи набір параметрів пацієнта і відповідний клас, C4.5 будує дерево рішень, здатне передбачити клас для нових пацієнтів на підставі їх параметрів.



Класифікація методом дерева рішень створює своєрідні блок-схеми для розподілу нових даних. У кожній точці блок-схеми задається питання про значимість того чи іншого параметра, і в залежності від цих параметрів він або вона [клієнти] потрапляють в певний клас.

Задля того, щоб визначити найбільш інформативну змінну(змінна, що буде стояти на початковому етапі відбору) використовують принцип припливу інформації(Entropy and Information Gain).

Ентропія(Entropy) – це ступінь невизначеності стану об'єкта (джерела інформації).

Використовується для визначення того, щоб визначити скільки інформації кодується у певному рішенні. Залежить не тільки від числа його можливих станів, а й від ймовірності цих станів.

У загальному випадку, відповідно до теорії ймовірностей, джерело інформації характеризується ансамблем станів K = {k1, k2, ..., kN} з ймовірністю станів - {P(k1), P(k2), ..., P(kN)} за умови, що сума ймовірностей всіх станів дорівнює 1. Міра кількості інформації, як невизначеності вибору дискретним джерелом стану з ансамблю K, запропонована К. Шенноном в 1946 році і отримала назву ентропії дискретного джерела інформації або ентропії кінцевого ансамблю:

$$B(k) = -\sum_{1}^{d} P(k\_d) * \log2(P(k\_d)).$$

d - к-сть змінних,

P(k_d) – ймовірність класу k змінної d.

Розрахуємо ентропію та знайдемо найбільш інформативні змінні.

Нехай клієнти банку розподілені таким чином:

№	Клієнт	Наявність забезпечення	Наявність нерухомості	Наявність страхування
1	Good	yes	no	yes
2	Bad	no	yes	yes
3	Good	yes	yes	no
4	Bad	no	no	no
5	Good	no	no	no
6	Good	yes	no	yes
7	Bad	no	yes	no

Обрахуємо ентропію в загальному випадку, тільки на наданих клієнта – «Good»/«Bad»:

B(Клієнт) = -p('yes')log2(p('yes')) - p('no')log2(p('no'))

B(Клієнт) = -(4/7)log2(4/7) - p(3/7)log2(3/7)

B(Клієнт) = 0.985228136034

Тож значення загальної ентропії - 0.985.

Щоб дізнатися порядок змінних у дереві застосовують критерій - Information Gain.

Gain(variable) = total_entropy - remainder(variable)

$$\text{remainder(variable)} = \sum_{\text{branch}}^{\text{branch\_d}} P(\text{variable\_branch\_d})*B(\text{branch})$$

total_entropy – загальна ентроія;

B(branch) – ентропія змінної;

P(variable_branch_n) – ймовірність кожного класу змінної.

Обрахуємо Information Gain для змінної «Наявність забезпечення».

remainder(Наявність забезпечення) = P('yes, Good')*B('yes') + P('yes, Bad')*B('yes') + P('no, Good')*B('no') + P('no, Bad')*B('no').

remainder(Наявність забезпечення) = (2/7)*B('yes') + (1/7)*B('yes') + (2/7)*B('no') + (2/7)*B('no').

remainder(Наявність забезпечення) = 0.463587.

Gain(Наявність забезпечення) = 0.985228 - 0.463587 = 0.521641.

Для порівняння обраховуємо Information Gain для інших змінних:

Gain('Наявність страхування') = 0.020244;

Gain('Наявність нерухомості') = 0.128085.

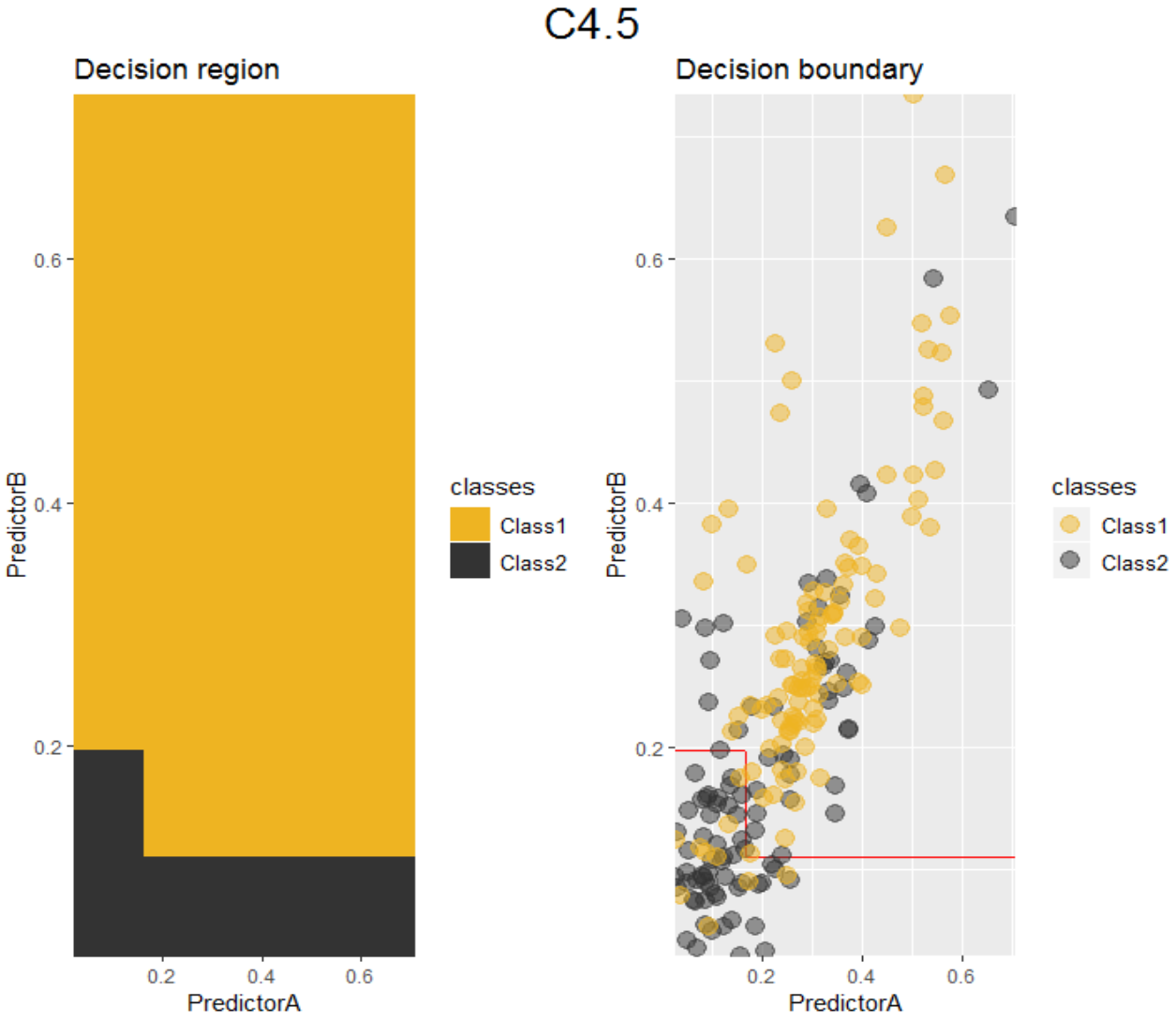
Отже, змінна «Наявність забезпечення» буде розташована на «вершині» дерева.

Відмінності C4.5 від інших систем, що використовують дерева рішень:

- По-перше, C4.5 використовує приплив інформації, при створенні дерева рішень.
- По-друге, щоб уникнути перенавчання(overfiting) вибірки використовують метод 'pruning' - відсікання гілок, які збільшують похибку моделі.
- По-третє, C4.5 може працювати з дискретними і неперервними значеннями. Обмежуючи діапазони і встановлюючи пороги даних, трансформуючи неперервні дані в дискретні.

Ймовірно, найбільшим гідністю дерев рішень є їх проста інтерпретація, також вони мають досить високу швидкість роботи, а отримані дані легкі для розуміння.

Реалізуємо алгоритм на мові R(використовуючи тренувальну вибірку «twoClassData»):



Accuracy	AccuracyCV	AccuracyCVInf	AccuracyCVPAC
0.755	0.726	0.678	0.594

CART

CART (classification and regression trees) - це абревіатура, що позначає методи класифікації і регресії з використанням дерева рішень. Це методика навчання, заснована на деревах рішень, яка повертає класифікаційні або регресивні дерева.

Наприклад, знову візьмемо набір даних про клієнта. Ви можете спробувати передбачити, чи отримає клієнт кредит. Тут можливе використання двох класів: «Good» - отримає, «Bad» - не отримає.

Задля того, щоб визначити найбільш інформативну змінну(змінна, що буде стояти на початковому етапі відбору) у методі CARD використовують Gini Impurity .

$$G(k) = \sum_{i=1}^{d} P(k\_d) * (1 - P(k\_d))$$

d - к-сть змінних,

P(k_d) – ймовірність класу k змінної d.

Застосуємо Gini Impurity на нашому прикладі:

G(Клієнт) = P('yes') * (1 - P('yes')) + P('no') * (1 - P('no'));

G(Клієнт) = 4 / 7 * (1 - 4/7) + 3 / 7 *(1 - 3/7);

G(Клієнт) = 0.489796.

Gini Impurity - це показник того, як часто рандомним чином вибраний елемент з набору буде неправильно позначений, тобто у нашому випадку неправильно класифікованих об`єктів буде ~ 49%.

Визначення найбільш інформативної змінної робиться аналогічним чином, як обчислення інформації для ентропії, за винятко, що замість того, щоб взяти зважену суму ентропій кожної змінної, ми беремо зважену суму Gini Impurity.

Gini_Gain(variable) = total_impurity - impurity_remainder(variable)

$$remainder(variable) = \sum_{branch}^{branch\_d} P(attribute\_branch\_d)*G(branch)$$

total_impurity – загальний показник Gini Impurity;

G(branch) – Gini Impurity змінної;

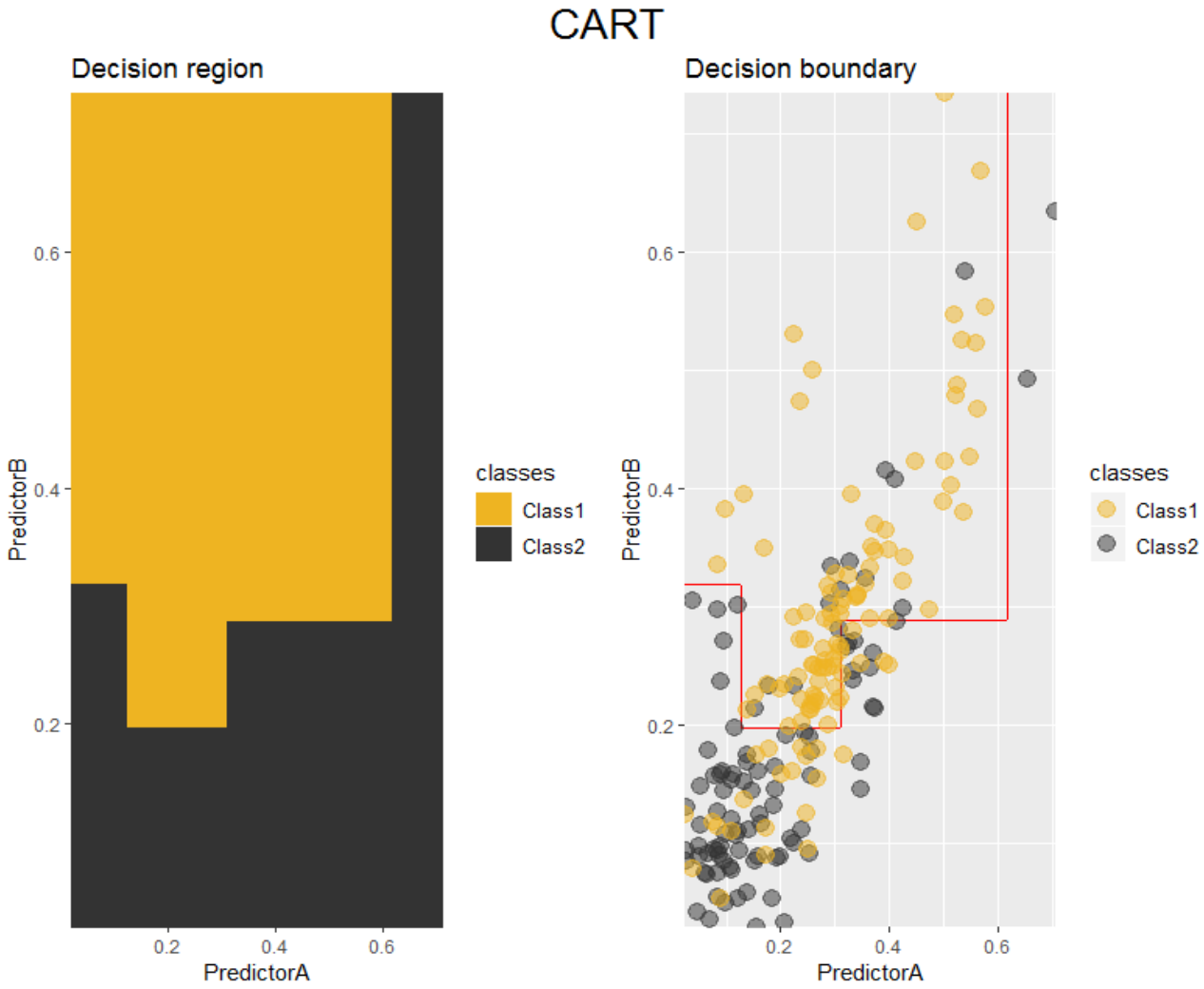
P(variable_branch_d) – ймовірність кожного класу змінної.

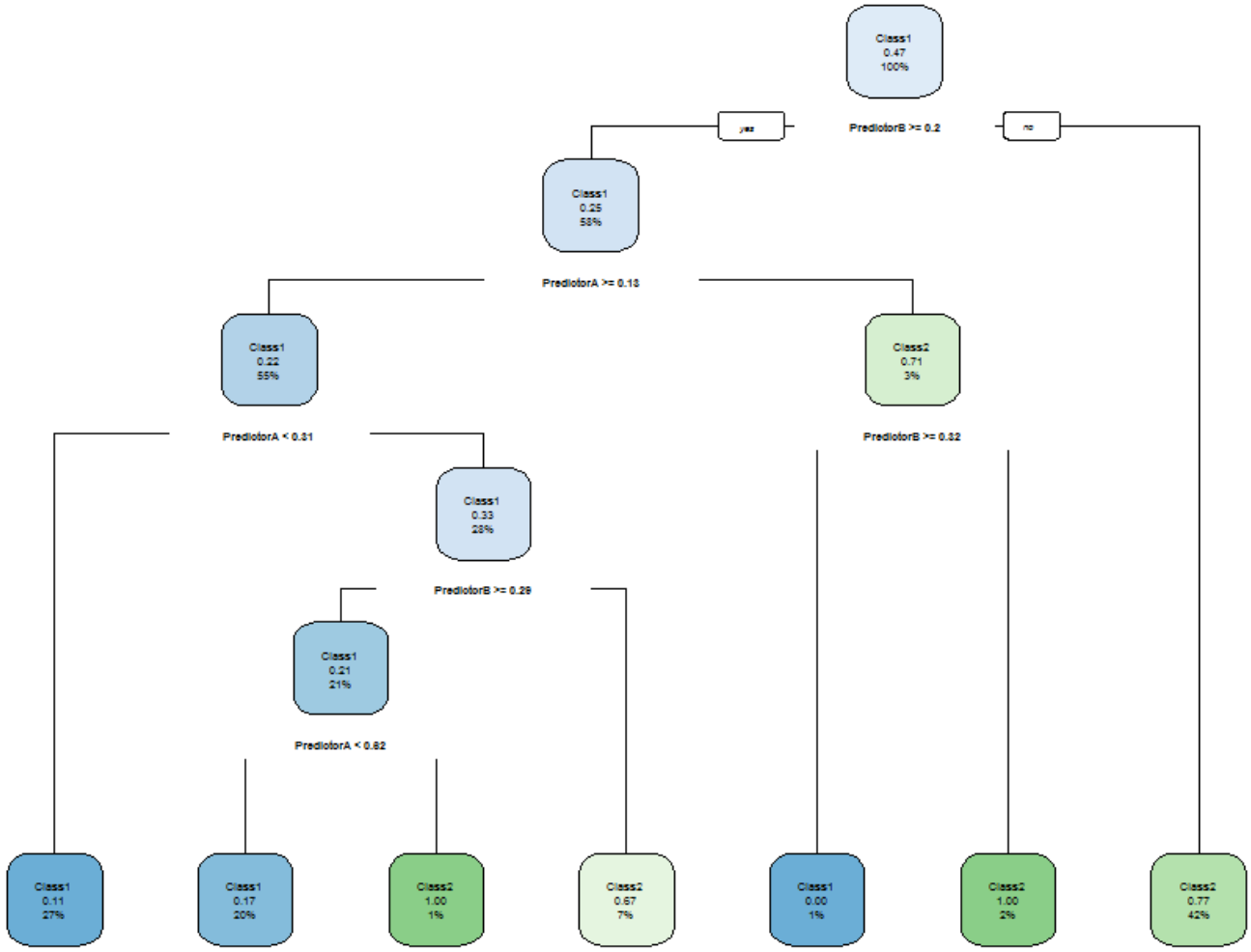
Співвідношення CART з C4.5 :

C4.5	Cart
Для оцінки точності моделі в процесі класифікації використовується приплив інформації(Entropy and Information Gain) до сегменту даних.	Для оцінки точності моделі в процесі класифікації використовується критерій Джині(Gini Impurity)
Використовує однопрохідний метод проріджування, щоб зменшити перенавчання.	Починаючи з низу дерева, CART оцінює помилку класифікації в вузлі і поза вузла. Якщо похибка перевищує граничну, то гілка відкидається
Вузли дерева рішень можуть мати дві або більше гілок	Вузли рішення мають дві гілки

Як і C4.5, CART досить швидкий та отримані дані легкі для розуміння.

Реалізуємо алгоритм на мові R(використовуючи тренувальну вибірку «twoClassData»):





Accuracy	AccuracyCV	AccuracyCVInf	AccuracyCVPAC
0.817	0.727	0.694	0.636

Random Forest

Random forest (англ. Випадковий ліс) - алгоритм машинного навчання, що полягає у використанні комітету (ансамблю) дерев. Алгоритм поєднує в собі дві основні ідеї: метод беггінга, метод випадкових підпросторів. Алгоритм застосовується для задач класифікації, регресії і кластеризації.

Для задання алгоритму спочатку треба задати початкові параметри. Нехай навчальна вибірка складається з N прикладів, загальна кількість змінних у вибірці - M, кількість параметрів у кожному дереві - m (в задачах класифікації зазвичай m = sqrt{M}).

Всі дерева будуються незалежно один від одного на різних частинах(«ділянках») вибірки:

Згенеруємо випадкову підвибірку з повторенням розмірності N з навчальної вибірки. (Таким чином, деякі приклади потраплять в неї кілька разів, а приблизно N / 3 прикладів не ввійдуть в неї взагалі).

Побудуємо дерево, що класифікує приклади даної підвибірки, причому в ході створення чергового вузла дерева будемо вибирати ознаки, на основі яких проводиться розбиття, не з усіх M ознак, а лише з m випадково обраних. Вибір найкращого з цих m ознак можна здійснюватися різними способами. Використовується невизначеність Джині(Gini Impurity), що застосовується також в алгоритмі побудови вирішальних дерев CART або критерій приросту інформації(Entropy and Information Gain).

Дерево будується до повного вичерпання підвибірки і не піддається процедурі прунінга (на відміну від вирішальних дерев, побудованих за таким алгоритмам, як CART або C4.5).

Класифікація об'єктів проводиться шляхом голосування: кожне дерево комітету відносить об'єкт до одного з класів, і перемагає той клас, за який проголосувала найбільша кількість дерев.

Оптимальне число дерев підбирається таким чином, щоб мінімізувати помилку класифікатора на тестовій вибірці.

Оскільки приблизно N / 3 прикладів не ввійдуть в тренувальну вибірку, тому на цих даних можна обрахувати точність моделі. Вибірку, що не ввійшла у навчальну називають – out-of-bag.

Як і в CARD та C4.5, Random Forest також має метод обрахунки найбільш інформативної змінної.

Після побудови моделі на навчальній вибірці кожну змінну тестують на даних out-of-bag та обраховують похибку для кожної змінної. Оскільки змінна задіяна у декількох деревах навчальної вибірки, тому і похибок буде декілька. Щоб отримати одне значення ці похибки усереднюють. Потім значення змінної перемішують на навчальній вибірці і знову рахують похибку на даних out-of-bag. Чим більша різниця похибки до і після перемішання значень змінної, тим більша інформативність змінної. Даний підхід має недолік – завищує значущість категоріальних змінних, якщо великий об’єм вибірки.

Основною перевагою даного методу є точність. Якість краща, ніж у моделі класифікації нейронних мереж та методу опорних векторів(Support Vector Machine).

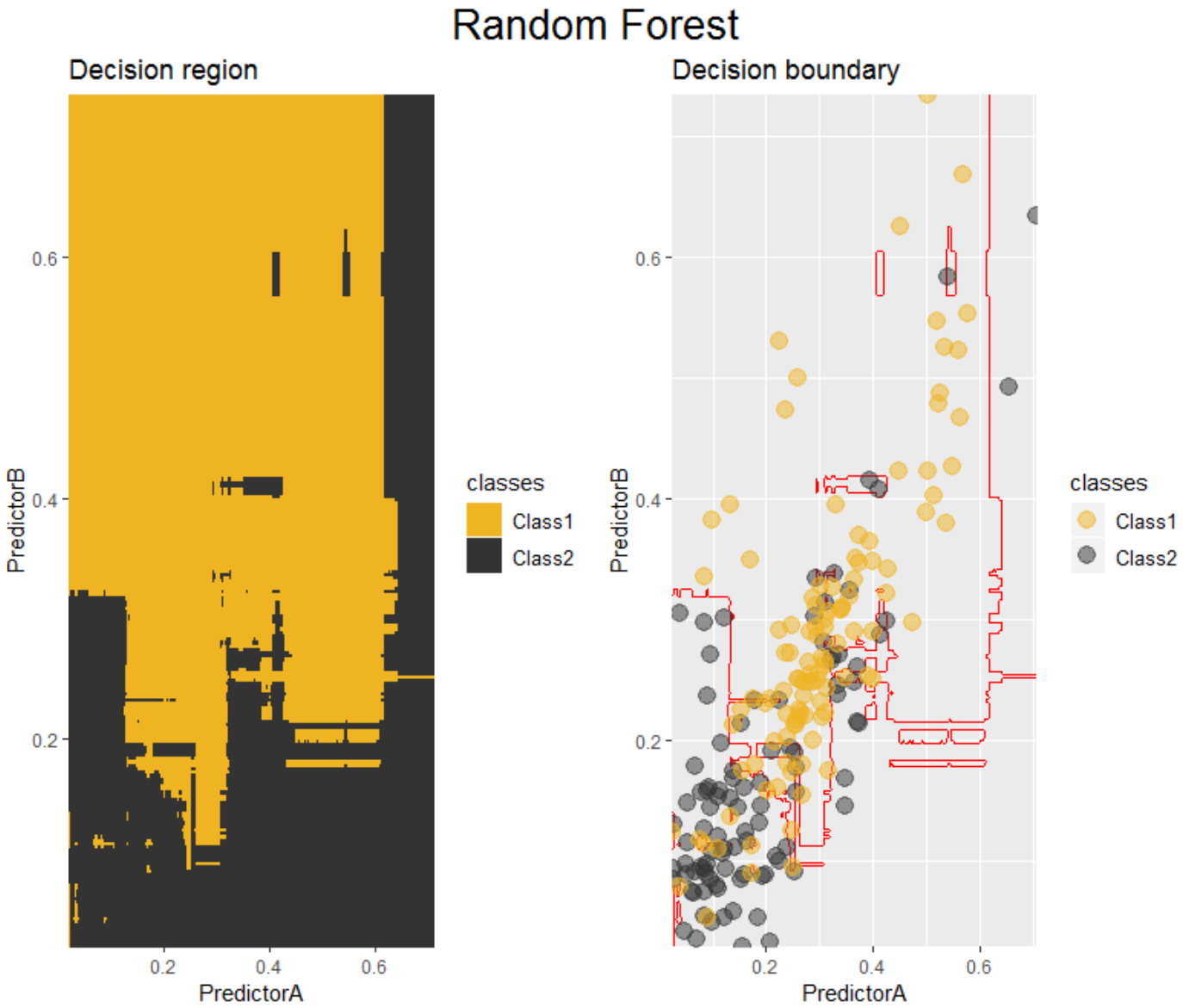
Також здатна ефективно обробляти дані з великим числом ознак і класів.

Однаково добре обробляються як неперервні, так і дискретні ознаки. Метод не чутливий до «викидів» та легко справляється із пропущеними значеннями.

Існують методи оцінювання значущості окремих ознак в моделі.

Але потрібно врахувати, що алгоритм схильний до перенавчання, для обрахунку потрібно більше часу та пам’яті комп'ютеру ніж в CART або C4.5. Також результати погано інтерпретується, на відміну від інших алгоритмів дерев рішень.

Реалізуємо алгоритм на мові R(використовуючи тренувальну вибірку «twoClassData»):



Accuracy	AccuracyCV	AccuracyCVInf	AccuracyCVPAC
1	0.731	0.699	0.644

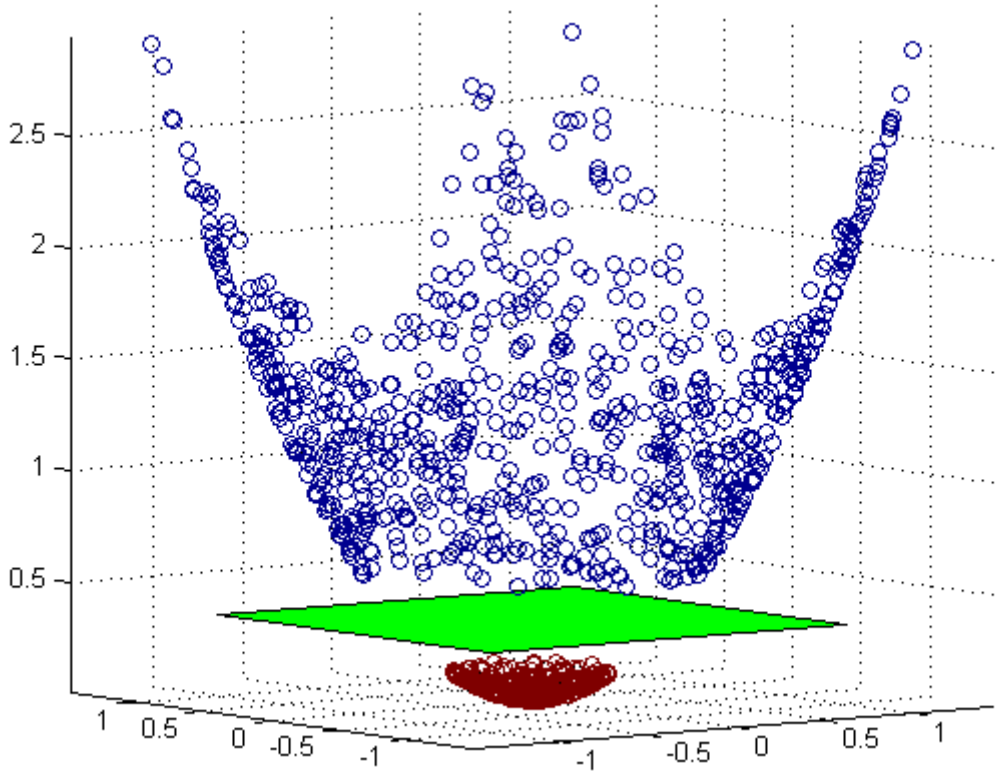
Метод опорних векторів

Метод опорних векторів (SVM - Support vector machine) використовує гіперплощину, щоб класифікувати дані на 2 класи. Взагалі, SVM виконує ті ж операції, що і C4.5, але з однією відмінністю - SVM не використовує дерева рішень.

SVM дозволяє спроектувати дані в простір більшої розмірності. Коли дані спроектовані, то SVM визначає кращу гіперплощину, яка ділить дані на 2 класу.

Наприклад, у нас є купа з синіх і червоних кульок на столі. Якщо кульки не можуть лежать в абсолютному безладі, ви можете взяти палицю і, не змінюючи положення куль, розділити їх палицею. Коли нова куля додається на стіл, ви можете передбачити її колір, знаючи, на яку частину столу вона потрапила. Кулі – це дані, а червоний і синій кольори - два класи. Палка являє собою найпростішу гіперплощину, тобто лінію.

Проте в реальності такий випадок майже неможливий. Зазвичай кулі перемішані, в такому випадку проста палиця не допоможе. Тому ми швидко підіймаємо стіл - кулі злітають у повітря. Поки кулі знаходяться в повітрі і в правильних положеннях, ми розділяємо червоні і сині кулі великим шматком паперу.



Підняття столу в повітря - це еквівалент відображенню даних в просторі з більш високою розмірністю. В цьому випадку ми переходимо від плоскої поверхні столу до тривимірного положенню куль в повітрі. Великий аркуш паперу - це функція площини, а не лінії.

Куля на столі має місце розташування, яке можна визначити за координатами. Наприклад, куля може відстояти на 20 см від лівої межі столу і на 50 - від нижньої. Іншими словами, координати (x, y) кулі мають значення (20,50). Це приклад двовимірного простору.

Повернемося до набору даних про пацієнта, кожен пацієнт може бути описаний різними параметрами, такими як пульс, рівень холестерину, тиск і так далі. Кожен з

цих параметрів є виміром. В результаті: SVM відносить ці параметри до вищого виміру, а потім знаходить гіперплощину, щоб розділити на класи.

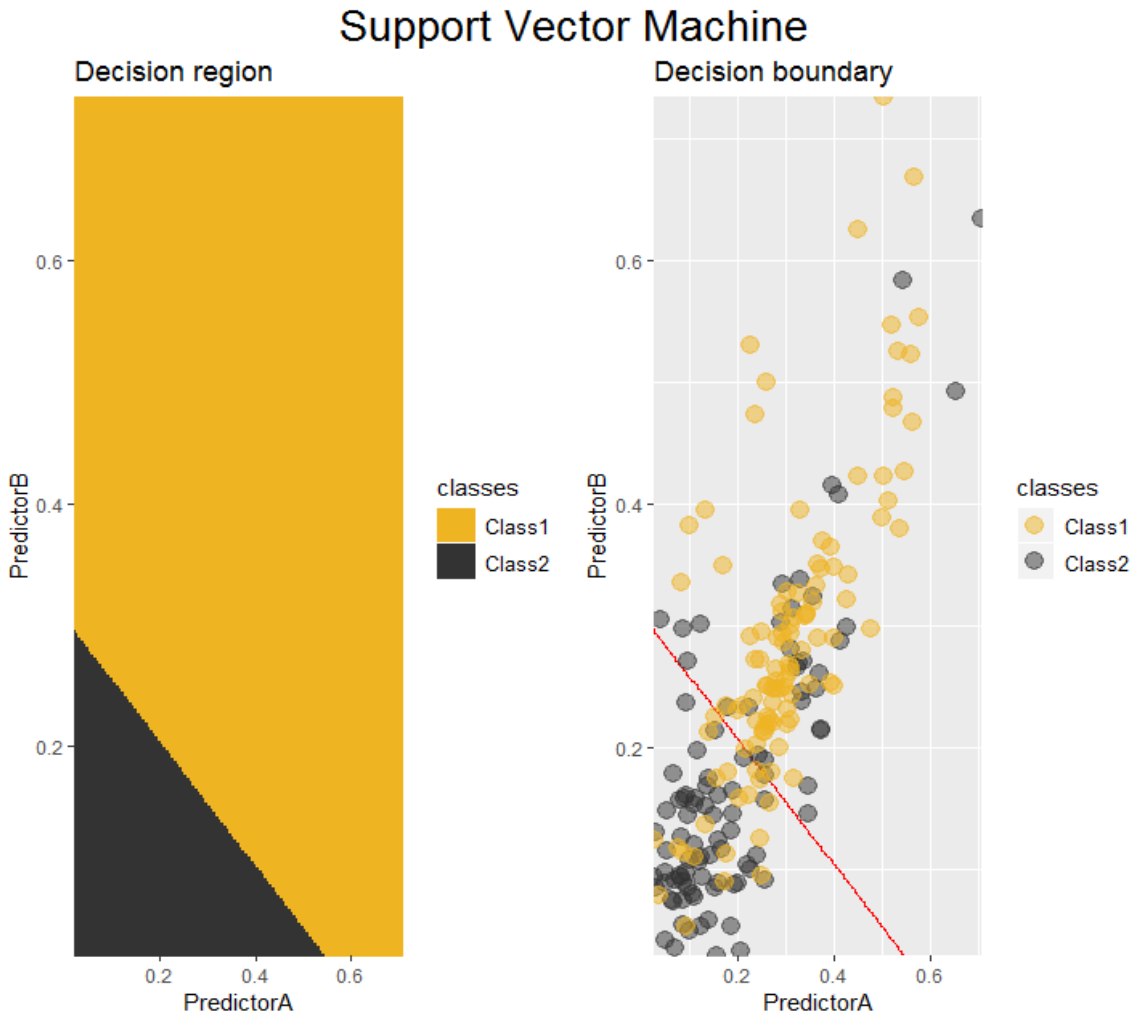
Точність моделі залежить від «відступу»(margin) точок навчальної вибірки до площини, зрозуміло, чим більша відстань, тим більша точність моделі. «Відступ»(margin) - це відстань між гіперплощиною і двома найближчими точками даних кожного класу. У прикладі з кулями і столом «відступом» називається відстань між палицею і найближчою червоною чи синьою кулькою.

Суть в тому, що SVM намагається максимізувати «відступ» так, щоб гіперплощина перебувала приблизно на однаковій відстані від червоних і синіх куль - це знижує шанс помилок класифікації.

Звідки SVM отримав свою назву? Гіперплощина рівновіддалена від червоних і синіх куль. Ці кулі - точки даних, які називаються опорними векторами (support vectors), тому що вони підтримують гіперплощину.

SVM – дуже потужний і універсальний метод машинного навчання, який відмінно проявляє себе на «складних» даних невеликого або середнього об`єму .

Реалізуємо алгоритм на мові R(використовуючи тренувальну вибірку «twoClassData»):



Accuracy	AccuracyCV	AccuracyCVInf	AccuracyCVPAC
0.75	0.749	0.711	0.645

KNN

kNN (k-Nearest Neighbors) - це алгоритм класифікації, проте він відрізняється від попередніх, тому що це - лінивий класифікатор. Це означає, що в процесі навчання він не робить нічого, а тільки зберігає тренувальні дані. Він починає класифікацію тільки тоді, коли з'являються нові немарковані дані. Активний же класифікатор створює класифікаційну модель в процесі навчання. Коли вводяться нові дані, такий класифікатор «згодовує» дані класифікаційної моделі. C4.5, SVM і Random Forest на відміну від kNN, вони все - активні класифікатори. Ось чому:

- C4.5 будує дерево рішень в процесі навчання;
- SVM будує гіперплощину;
- Random Forest будує ансамблеву класифікацію.

kNN не будує ніякої класифікаційну модель. Замість цього він просто зберігає розмічені тренувальні дані. Коли з'являється нові дані, kNN проходить по 2 базовим крокам:

- спочатку він шукає k найближчих розмічених точок даних - іншими словами, k найближчих сусідів;

- використовуючи класи сусідів, kNN вирішує, як краще класифікувати нові дані.

Щоб зрозуміти які точки знаходяться найблище для неперервних даних kNN використовує дистанційну метрику, наприклад, Евклідову. Вибір метрики залежить від типу даних. Деякі радять навіть вибирати дистанційну метрику на підставі тренувальних даних. Є дуже багато нюансів, описаних у багатьох роботах по дистанційним метрик kNN.

При роботі з дискретними даними, вони спочатку перетворюються в неперервні. Ось 2 приклади:

- використання відстані Хеммінга як метрики для визначення «близькості» двох текстових рядків;
- перетворення дискретних даних в бінарні числа.

Якщо дані не належать ні одному із сусідів, тоді використовуються 2 класичні техніки:

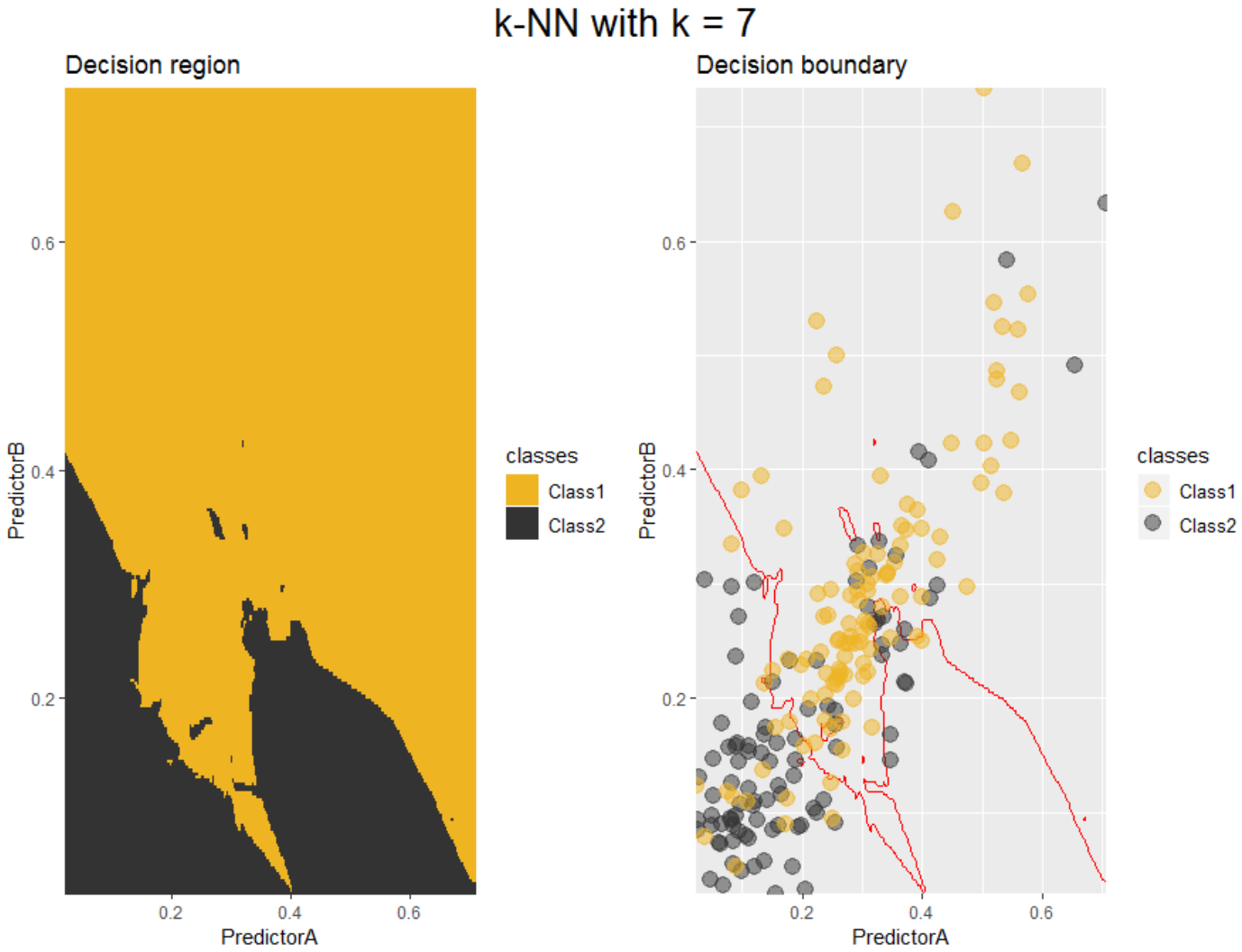
Прийняти за правильне рішення просту більшість. До якого класу належить найбільша кількість сусідів, туди і визначають точку даних.

Виконати те ж саме, але дають найближчому сусіду більшої «ваги». Якщо сусід стоїть на 5 одиниць, то його вага буде 1/5. При збільшенні дистанції вага стає все менше і менше.
kNN легкий в розумінні і його нескладно реалізувати - це дві головні причини йогоо застосувати. Залежно від вибору дистанційної метрики, kNN може показувати досить точні результати.
Проте є 5 речей, за якими потрібно стежити:

- kNN може бути дуже ресурсовитратний, якщо намагатися визначити найближчих сусідів на великому наборі даних;
- «зашумлені» дані можуть «запороти» kNN-класифікацію;
- потрібно враховувати кількість значень. Характеристики з великою кількістю значень можуть впливати на дистанційну метрику, по відношенню до характеристик з меншою кількістю значень;
- оскільки обробка даних «відкладається», kNN зазвичай вимагає більше пам’яті комп’ютеру, ніж активні класифікатори;
- вибір правильної дистанційній метрики дуже важливий для точності kNN.

Реалізуємо алгоритм на мові R(використовуючи тренувальну вибірку «twoClassData»)

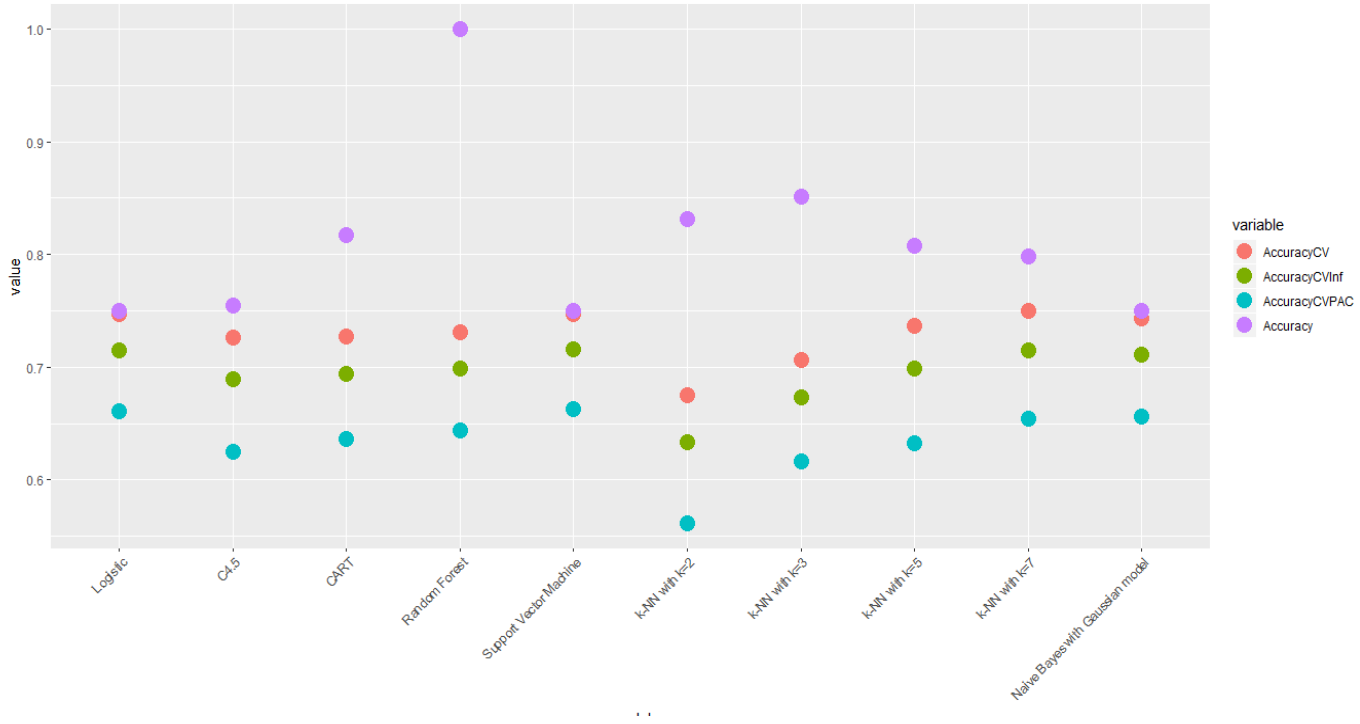
Реалізуємо метод на різній кількості «сусідів»(2, 3, 5, 7):



Accuracy	AccuracyCV	AccuracyCVInf	AccuracyCVPAC
0.798	0.75	0.715	0.654

* З перелічених «сусідів»(2, 3, 5, 7) останній приклад показав найкращу точність(детальна інформація знаходиться у скрипті на мові R, що знаходиться у додатку).

Порівняємо результати різних моделей:



Тепер ми будемо оцінювати нашу модель за 4 критеріями:

Accuracy - емпірична точність, обчислена на початковому наборі даних.

AccuracyCV - середнє значення точності, отримане схемою перехресної перевірки (cross-validation).

AccuracyInf - найнижча межа точності.

AccuracyPAC - середнє значення точності(AccuracyCV) віднімаємо від стандартного відхилення отримане схемою перехресної перевірки (cross-validation).

Процес вибору моделі в машинному навчанні – це пошук компромісу між точністю і його «надійністю» .

Оскільки Random Forest схильний до переневчання, тому не дивно, що Accuracy(емпірична точність) в нього найвище значення, проте інші показники точності не найкращі із запропонованих вище.

Найкрщими моделями виявилися – SVM, KNN - оскільки тренувальна вибірка «twoClassData» невеликого обєму та з невиликою кількістю змінних .

Висновок

Ми розглянули такі методи класифікації як «Naive Bayes», «Logistic regression», «C4.5», «CART», «Random Forest», «Support Vector Machine», «KNN».

Коли ми бачимом всю різноманітність алгоритмів, у нас постає питання: «А який метод слід використовувати мені?» Відповідь на це питання залежить від безлічі факторів:

- Розмір, якість і характер даних;
- Доступне обчислювальний час;
- Терміновість завдання;
- Що ви хочете робити з даними.

Навіть досвідчений data scientist не скаже, який алгоритм буде працювати краще, перш ніж спробує кілька варіантів. Існує безліч інших алгоритмів машинного навчання, але наведені вище - найбільш популярні серед алгоритмів класифікації.

Список використаної літератури:

1. «Алгоритмы машинного обучения: Какой из них выбрать для решения вашей проблемы?» - https://evileg.com/ru/post/300/.
2. «Классификация» - http://www.machinelearning.ru/wiki/index.php?title=%D0%9A%D0%BB%D0%B0%D1%81%D1%81%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F.
3. «Топ-10 data mining-алгоритмов» - https://habr.com/company/iticapital/blog/262155/.
4. «Information Entropy and Information Gain» - https://bambielli.com/til/2017-10-22-information-gain/.
5. «Gini Impurity» - https://bambielli.com/til/2017-10-29-gini-impurity/.
6. «Энтропия источника информации» - http://bourabai.kz/signals/ts0115.htm.
7. «Вибір моделі машинного навчання» - http://www.machinelearning.ru/wiki/images/0/05/BMMO11_4.pdf/.

Скрипт на мові R:

```
library(plyr)
library(dplyr)
library(ggplot2)
library(grid)
library(gridExtra)
library(h2o)
library(doFutur)
library(caret)
library(AppliedPredictiveModeling)
library(klaR)
library(shiny)
library(h2o)
library(rpart)
library(rpart.plot)
library(magrittr)
library(ggpubr)
library(Cubist)
library(C50)

data(twoClassData)

registerDoFuture()
plan(multiprocess)

twoClass=cbind(as.data.frame(predictors),classes)


# predictors distrubution
ggplot(data = twoClass,aes(x = PredictorA, y = PredictorB)) +
  geom_point(aes(color = classes), size = 6, alpha = .5) +
  scale_colour_manual(name = 'classes', values = c("red","blue")) +
  scale_x_continuous(expand = c(0,0)) +
  scale_y_continuous(expand = c(0,0))

# prepare graphics for predictive data
nbp <- 250;
PredA <- seq(min(twoClass$PredictorA), max(twoClass$PredictorA), length = nbp)
PredB <- seq(min(twoClass$PredictorB), max(twoClass$PredictorB), length = nbp)
Grid <- expand.grid(PredictorA = PredA, PredictorB = PredB)
```

```
PlotGrid <- function(pred,title) {
  surf <- (ggplot(data = twoClass, aes(x = PredictorA, y = PredictorB,
                                       color = classes)) +
           geom_tile(data = cbind(Grid, classes = pred), aes(fill = classes)) +
           scale_fill_manual(name = 'classes', values = c("goldenrod2","grey20")) +
           ggtitle("Decision region") + theme(legend.text = element_text(size = 10)) +
           scale_colour_manual(name = 'classes', values = c("goldenrod2","grey20"))) +
    scale_x_continuous(expand = c(0,0)) +
    scale_y_continuous(expand = c(0,0))
  pts <- (ggplot(data = twoClass, aes(x = PredictorA, y = PredictorB,
                                      color = classes)) +
          geom_contour(data = cbind(Grid, classes = pred), aes(z =
as.numeric(classes)),
                       color = "red", breaks = c(1.5)) +
          geom_point(size = 4, alpha = .5) +
          ggtitle("Decision boundary") +
          theme(legend.text = element_text(size = 10)) +
          scale_colour_manual(name = 'classes', values = c("goldenrod2","grey20"))) +
    scale_x_continuous(expand = c(0,0)) +
    scale_y_continuous(expand = c(0,0))
  grid.arrange(surf, pts, top = textGrob(title, gp = gpar(fontsize = 20)), ncol = 2)
}


# create cross validation
V <- 10
T <- 3
TrControl <- trainControl(method = "repeatedcv",
                          number = V,
                          repeats = T)

seed <- 123

# func to compute an accuracy
#postResample can be used to estimate the root mean squared error (RMSE)
ErrsCaret <- function(Model, Name) {
  Errs <- data.frame(t(postResample(predict(Model, newdata = twoClass),
twoClass[["classes"]])),
                     Resample = "None", model = Name)
  rbind(Errs, data.frame(Model$resample, model = Name))
}

Errs <- data.frame()
```

```
CaretLearnAndDisplay <- function (Errs, Name, Formula, Method, ...) {
  set.seed(seed)
  Model <- train(as.formula(Formula), data = twoClass, method = Method, trControl
= TrControl, ...)
  Pred <- predict(Model, newdata = Grid)
  PlotGrid(Pred, Name)
  Errs <- rbind(Errs, ErrsCaret(Model, Name))
}

# Accurany analysis
Acc_analysis1 <- function(Errs) {
  cbind(dplyr::summarize(Errs, mAccuracy = mean(Accuracy, na.rm = TRUE),
mKappa = mean(Kappa, na.rm = TRUE),
                         sdAccuracy = sd(Accuracy, na.rm = TRUE), sdKappa = sd(Kappa,
na.rm = TRUE)))
}

Acc_analysis2 <- function(Errs) {
  ErrCV <- Acc_analysis1(dplyr::filter(Errs, !(Resample == "None")))
  ErrCV <- mutate(ErrCV, AccuracyCV = round(mAccuracy,3), AccuracyCVInf =
round((mAccuracy - 2 * sdAccuracy/sqrt(T * V)),3),
                  AccuracyCVPAC = round((mAccuracy - sdAccuracy),3))
  ErrEmp <- round(dplyr::select(dplyr::filter(Errs, (Resample == "None")),
Accuracy),3)
  Err <- dplyr::select(cbind(ErrCV, ErrEmp), Accuracy, AccuracyCV,
AccuracyCVInf, AccuracyCVPAC)
  Err
}


# Naive Bayes
Errs  <- CaretLearnAndDisplay(Errs, "Naive Bayes with Gaussian model", "classes ~
.", "nb",
                              tuneGrid = data.frame(usekernel = c(FALSE), fL = c(0), adjust =
c(1)))

ggtexttable(Acc_analysis2(Errs[(nrow(Errs)-30):nrow(Errs),]),
            rows = NULL, theme = ttheme("lBlack"))

# Logistic
Errs  <- CaretLearnAndDisplay(Errs, "Logistic", "classes ~ .", "glm")

ggtexttable(Acc_analysis2(Errs[(nrow(Errs)-30):nrow(Errs),]),
```

```
        rows = NULL, theme = ttheme("lBlack"))

# C4.5
Errs <- CaretLearnAndDisplay(Errs, "C4.5", "classes ~ .", "C5.0")

ggtexttable(Acc_analysis2(Errs[(nrow(Errs)-30):nrow(Errs),]),
        rows = NULL, theme = ttheme("lBlack"))


# CART
Errs <- CaretLearnAndDisplay(Errs, "CART", "classes ~ .", "rpart",
                     control = rpart::rpart.control(minsplit = 5, cp = 0), tuneGrid =
data.frame(cp = .02))

Tree <- train(classes ~ ., data = twoClass, method = "rpart", control =
rpart::rpart.control(minsplit = 5, cp = 0),
          tuneGrid = data.frame(cp = .02), trControl = TrControl)

rpart.plot(Tree$finalModel)

ggtexttable(Acc_analysis2(Errs[(nrow(Errs)-30):nrow(Errs),]),
        rows = NULL, theme = ttheme("lBlack"))


# Random Forest
Errs <- CaretLearnAndDisplay(Errs, "Random Forest", "classes ~ .", "rf",
                     tuneLength = 1,
                     control = rpart.control(minsplit = 5))

ggtexttable(Acc_analysis2(Errs[(nrow(Errs)-30):nrow(Errs),]),
        rows = NULL, theme = ttheme("lBlack"))


# SVM
Errs <- CaretLearnAndDisplay(Errs, "Support Vector Machine", "classes ~ .",
"svmLinear")

ggtexttable(Acc_analysis2(Errs[(nrow(Errs)-10):nrow(Errs),]),
        rows = NULL, theme = ttheme("lBlack"))


# KNN
KNNKS <- c(2, 3, 5, 7)
```

```
for (k in KNNKS) {
  Errs <- CaretLearnAndDisplay(Errs, sprintf("k-NN with k=%i", k),
                               "classes ~ .","knn", tuneGrid = data.frame(k = c(k)))

}

ggtexttable(Acc_analysis2(Errs[(nrow(Errs)-30):nrow(Errs),]),
            rows = NULL, theme = ttheme("lBlack"))


## Result
ErrCaretAccuracy <- function(Errs) {
  Errs <- group_by(Errs, model)
  cbind(dplyr::summarize(Errs, mAccuracy = mean(Accuracy, na.rm = TRUE),
mKappa = mean(Kappa, na.rm = TRUE),
                         sdAccuracy = sd(Accuracy, na.rm = TRUE), sdKappa = sd(Kappa,
na.rm = TRUE)))
}


ErrAndPlotErrs <- function(Errs) {
  ErrCV <- ErrCaretAccuracy(dplyr::filter(Errs, !(Resample == "None")))
  ErrCV <- transmute(ErrCV, AccuracyCV = mAccuracy, AccuracyCVInf =
mAccuracy - 2 * sdAccuracy/sqrt(T * V),
                     AccuracyCVPAC = mAccuracy - sdAccuracy,
                     model = model)
  ErrEmp <- dplyr::select(dplyr::filter(Errs, (Resample == "None")), Accuracy,
model)
  Err <- dplyr::left_join(ErrCV, ErrEmp, by = "model")
  print(ggplot(data = reshape2::melt(Err, "model"),
               aes(x = model, y = value, color = variable)) +
          geom_point(size = 5) +
          theme(axis.text.x = element_text(angle = 45, hjust = 1),
                plot.margin = grid::unit(c(1, 1, .5, 1.5), "lines")))
  Err
}

Err <- ErrAndPlotErrs(Errs)


# find best model

FindBestErr <- function(Err) {
```

```
  for (name in names(Err)[!(names(Err) == "model")]) {
    ind <- which.max(Err[, name])
    writeLines(strwrap(paste("Best method according to", name, ":", Err[ind,
"model"])))
  }
}

FindBestErr(Err)

PlotParallels <- function(Errs) {
  pl <- ggplot(data = dplyr::filter(Errs, Resample != "None"),
               aes(x = model, y = Accuracy, color = Resample)) +
    geom_line(aes(x = as.numeric(model))) +
    scale_x_discrete(labels = unique(Errs$model)) +
    scale_color_grey(guide = FALSE)
  Err <- dplyr::summarize(group_by(dplyr::filter(Errs, Resample != "None"), model),
                          Accuracy = mean(Accuracy), Resample = "CV")
  pl <- pl + geom_line(data = Err, aes(x = as.numeric(model)), size = 3) +
    theme(axis.text.x = element_text(angle = 45, hjust = 1),
          plot.margin = grid::unit(c(1, 1, .5, 1.5), "lines"))
  print(pl)
}

PlotParallels(Errs)
```