# Relief feature selection

## Základní myšlenka

Rozšířit + vizualizace + další metody

Hlavní myšlenka relief algoritmů spočívá v odhadu vhodnosti atributů na základě toho, jak jsou si jednotlivé hodnoty v různých instancích blízké.

Mějme náhodný prvek z datové sady. Abychom mohli odhadovat vhodnost atributů, nalezneme nejbližší prvek se stejnou třídou jako náš náhodný prvek a nejbližší prvek s opačnou třídou. Těmto prvkům se také často říká „Nearest hit“ a „Nearest miss“.

Po nalezení těchto prvků porovnáváme jednotlivé atributy. Pokud jsou hodnoty pro daný atribut od sebe velmi vzdálené a jedná se o prvek se stejnou třídou, tak to znamená, že tento atribut nehraje příliš velkou roli v určení dané třídy, a tak se kvalita daného atributu sníží. Naopak, pokud budou hodnoty atributů od sebe vzdálené a bude se jednat o prvek s opačnou třídou, je to žádoucí, protože daný atribut od sebe rozlišuje tyto třídy, a tak se kvalita atributu zvýší. Tento proces je opakován m krát, kde m může být buďto počet instancí nebo uživatelem zvolené.

## Relief extension

Výše zmíněný postup funguje pouze pro datové sady, kde jsou pouze 2 třídy objektů. Algoritmus sice je možné spustit pro datové sady s vícero třídami objektů, ale výsledky mohou být zkreslené.

Z tohoto důvodu byl původní algoritmus rozšířen tak, aby nebyl limitován jen na 2 třídy objektů, a také, aby dokázal pracovat s chybějícími daty nebo se zkreslenými daty.

Stejně jako dříve, je vybrán náhodný prvek, ale poté je nalezeno k nejbližších prvků ze stejné třídy jako náhodný prvek a k nejbližších prvků pro každou další třídu. Odhad kvality se určuje velmi podobně jako v původním algoritmu, akorát se navíc počítá ještě s pravděpodobnostmi pro výskyt dané třídy a průměrují se hodnoty z k nejbližších prvků.

# Použitá implementace

## ReliefJava

Tento algoritmus byl získán z [githubu](https://github.com/ansballard/Relief-Algorithms/blob/master/Relief.java). Byl přepsán z původní Java implementace do C# a jedná se o základní verzi reliefu. Přepočet kvality atributu se určuje následující rovnicí:

Kde n je počet instancí, a je počet atributů, i je konkrétní instance a j je konkrétní atribut

Tento výpočet je poměrně jednoduchý, a tak se výsledky často liší od ostatních algoritmů.

Pseudokód algoritmu:

Pseudokódy do latexu

Scores = List(a)  
for i < n  
 for j < a  
 Scores[j] = Scores[j] – Hit(i, j)^2 + Miss(i, j)^2  
for i < a  
 Scores[i] = Scores[i] / n

Pseudokód funkce Miss:

Miss(sampleIndex, featureIndex):  
 shortestDistance = MAX  
 currentDistance = UNASSIGNED  
 for i < n  
 if class(data[sampleIndex]) != class(data[i])  
 currentDistance =  
 data[sampleIndex][featureIndex] – data[i][featureIndex]  
 if currentDistance^2 < shortestDistance^2  
 shortestDistance = currentDistance  
 return shortestDistance

Pseudokód funkce Hit je téměr totožný, liší se pouze v podmínce:

if class(data[sampleIndex]) == class(data[i]) AND sampleIndex != i

## Relief

Základní verze reliefu vycházející z pseudokódu dostupného [zde](https://link.springer.com/content/pdf/10.1023/A:1025667309714.pdf). Zhruba 7× rychlejší než ReliefJava.  
Rovnice pro přepočet kvality atributu vypadá následovně:

Kde n je počet instancí, a je počet atributů, i[j] je konkrétní atribut z instance i, A[j] je konkrétní atribut, h[j] je atribut z nearest hit instance a m[j] je atribut z nearest miss instance.

Funkce diff(a, i, j) vypadá následovně:

Pro numerické hodnoty:

Kde max(a) a min(a) jsou největší/nejmenší hodnoty ze všech instancí pro atribut a.

Pro nominální hodnoty funkce vrací 0, pokud se hodnoty rovnají a 1, pokud se hodnoty nerovnají.  
Poznámka: nominální atributy v současnosti v projektu nejsou implementované.

Pseudokód algoritmu:

Scores = List(a)  
for i < n  
 ri = data[random]  
 hit = NearestHit(i)  
 miss = NearestMiss(i)  
 for j < a  
 Scores[j] = Scores[j] – Diff(j, ri, hit) + Diff(j, ri, miss)  
for i < a  
 Scores[i] = Scores[i] / n

Pseudokód funkce Diff:

Diff(featureIndex, a, b):  
 return Abs(a[featureIndex] – b[featureIndex]) /  
 Max(data[featureIndex] – Min(data[featureIndex])

Pseudokód funkce NearestMiss:

NearestMiss(sampleIndex):  
 shortestDistance = MAX  
 currentDistance = UNASSIGNED  
 index = UNASSIGNED  
 for i < n  
 if class(data[sampleIndex]) != class(data[i])  
 currentDistance = 0  
 for j < a  
 currentDistance = currentDistance + data[sampleIndex][featureIndex] – data[i][featureIndex]  
 if currentDistance^2 < shortestDistance^2  
 shortestDistance = currentDistance  
 index = i  
 return data[index]

Pseudokód funkce NearestHit je téměr totožný, liší se pouze v podmínce:

if class(data[sampleIndex]) == class(data[i]) AND sampleIndex != i

## ReliefF

Rozšířená verze reliefu pracující s více třídami objektů, vycházející z pseudokódu [zde](https://www.researchgate.net/figure/Pseudo-code-of-the-ReliefF-algorithm_fig2_329963285). Zhruba stejně rychlý jako ReliefJava, i přes to, že je výpočetně daleko náročnější.  
Rovnice pro přepočet kvality atributů vypadá následovně:

Kde n je počet instancí, a je počet atributů, i[j] je konkrétní atribut z instance i, A[j] je konkrétní atribut, h[j] je atribut z nearest hit instance, m[j,c] je atribut z nearest miss instance pro třídu c a P(c) je pravděpodobnost výskytu instance s třídou c.

Funkce diff(a, i, j) vypadá následovně:

Pro numerické hodnoty:

Kde max(a) a min(a) jsou největší/nejmenší hodnoty ze všech instancí pro atribut a.

Pro nominální hodnoty funkce vrací 0, pokud se hodnoty rovnají a 1, pokud se hodnoty nerovnají.

Pseudokód algoritmu:

Scores = List(a)  
for i < n  
 ri = data[random]  
 hits = NearestHits(i, k)  
 misses = NearestMisses(i, k)  
 for j < a  
 hDiff = 0  
 missDiff = 0  
 for l < k  
 hDiff = hDiff + Diff(j, ri, hits[l]) / (n \* k)  
 for l < classes, class(l) != class(ri)  
 prob = classes[l] / 1 – classes[ri]  
 tmp = 0  
 for o < misses, class(misses[o]) == class(l)  
 tmp = tmp + Diff(j, ri, misses[o] / (n \* k)  
 missDiff += prob \* tmp  
 Scores[j] = Scores[j] – hitsDiff + missDiff

Pseudokód funkce NearestMisses:

NearestMisses(sampleIndex, k):  
 currentDistance = UNASSIGNED  
 distances = KeyValues(data.length)  
 for i < n  
 if class(data[sampleIndex]) != class(data[i])  
 currentDistance = 0  
 for j < a  
 currentDistance = currentDistance + data[sampleIndex][j] – data[i][j]  
 distances.push(i, Abs(currentDistance))  
 return first k smallest values from distances

Pseudokód funkce NearestHits je téměr totožný, liší se pouze v podmínce:

if class(data[sampleIndex]) == class(data[i] AND sampleIndex != i)

## ReliefConsistent a ReliefFConsistent

Tyto algoritmy byly upraveny tak, aby nevybíraly náhodný prvek, ale postupně vyhodnotily každou instanci, což vede ke konzistenci výsledků, zatímco algoritmy Relief a ReliefF mohou při opakovaném spuštění vracet rozdílné výsledky, což není žádoucí.

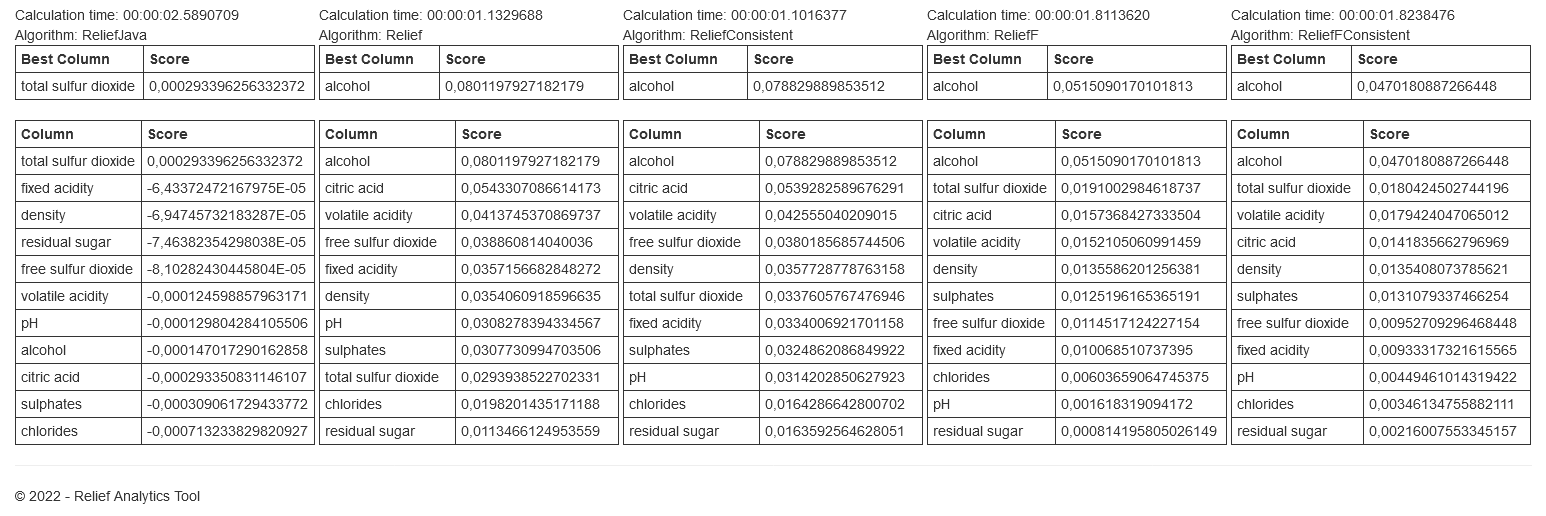
# Srovnání výsledků

Citace url

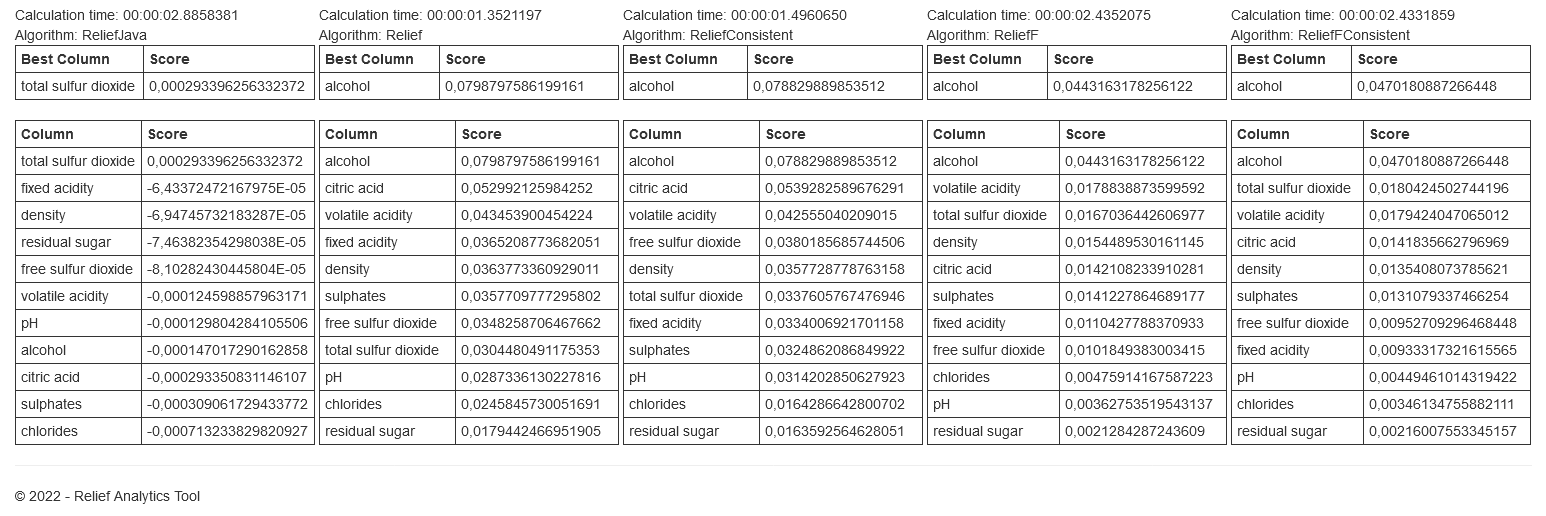
Tabulky smrštit, případně otočit

Popsat aplikaci jako takovou, C#, MVC…

Vyzkoušet víc datasetů s různými časy

Pro demonstraci algoritmů byl použit datový soubor WineQT.csv získaný z webového portálu [kaggle.com](https://kaggle.com).

Můžeme pozorovat, že kromě algoritmu ReliefJava se všechny shodly na nejlepším prvku, a zároveň odlišnosti v dalších atributech jsou si velice blízké, zatímco rozdíly v porovnání s ReliefJava jsou poměrně razantní, což je způsobeno právě odlišným poměrem ohodnocování kvality atributů.

Dále můžeme pozorovat, že konzistentní algoritmy se neshodují s jejich nekonzistentními verzemi, což je pochopitelné, jelikož se porovnávaný prvek volil náhodně. Pokud bychom vyhodnocení opakovali, dostaneme jiné, avšak podobné výsledky.