Universidade Federal de Pernambuco
Centro de Informática

Graduação em Engenharia da Computação

Análise comparativa de técnicas de seleção de protótipos

Dayvid Victor Rodrigues de Oliveira

Trabalho de Graduação

Recife
22 de novembro de 2011

Universidade Federal de Pernambuco
Centro de Informática

Dayvid Victor Rodrigues de Oliveira

Análise comparativa de técnicas de seleção de protótipos

Trabalho apresentado ao Programa de Graduação em Engenharia da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Engenharia da Computação.

Orientador: *Prof. Dr. George Darmiton*

Recife
22 de novembro de 2011

Eu dedico este trabalho a João Rodrigues de Silva, meu avô.

Agradecimentos

Agradeço a Deus.

What can I give back to God, for the blessings You pour out on me?
—BONO (Boston, 2001)

Resumo

RESUMO

Palavras-chave: PORTUGUES

Abstract

ABSTRACT

Keywords: INGLES

Sumário

Lista de Figuras

Lista de Tabelas

CAPÍTULO 1

Técnicas de Seleção de Protótipos

1.1 Motivação e Contextualização

Classificadores são (...)

> *"I have a dream for the Web [in which computers] become capable of analyzing all the data on the Web, the content, links, and transactions between people and computers. A **Semantic Web** which should make this possible has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy and our daily lives will be handled by machines talking to machines. The **intelligent agents** people have touted for ages will finally materialize."Tim Berners-Lee*

Tradução literal: *"Eu tenho um sonho para a Web [em que os computadores] tornam-se capazes de analisar todos os dados na Web, o conteúdo, links, e as transações entre pessoas e computadores. A **Web Semântica** que deve tornar isso possível ainda está para surgir, mas quando isso acontecer, os mecanismos dia-a-dia da burocracia do comércio e nossas vidas diárias serão tratados por máquinas falando com máquinas. Os **agentes inteligentes** que as pessoas têm falado por anos vão finalmente se concretizar."Tim Berners-Lee*

1.1.1 Seleção de Protótipos

1.1.2 Bases Desbalanceadas

1.2 Objetivo

1.3 Estrutura do Trabalho

1.3.1 Sessões

1.3.2 Metodologia Utilizada

1.3.3 Bases de dados

CAPÍTULO 2

Técnicas de Seleção de Protótipos

Neste capítulo, serão mostradas as técnicas de seleção de protótipos abordadas neste trabalho. Cada uma das sessões abaixo abordará uma técnica, será mostrado o conceito da técnica, assim como o pseudo-código e as caracterísicas de cada uma destas técnicas.

2.1 ENN

Edited Nearest Neighbor Rule[CPZ11] é uma técnica de seleção de protótipos puramente seletiva proposta por Wilson em 1976. De uma forma geral, esta técnica foi projetada para funcionar como um filtro de ruídos, ela elimina pontos na região de fronteira, região de alta susceptibilidade a erros, e com isso elimina ruídos.

Por atuar apenas na região de fronteira, esta técnica possui uma baixa capacidade de redução, deixando as instâncias que não se encontram na região de fronteira intactas, exceto pelos ruídos extremos.

Uma desvantagem desta técnica é que ela possui uma baixa capacidade de redução de elementos, visto que ela não elimina redundância.

Segue abaixo o algoritmo da execução do ENN e, logo após, alguns comentários sobre este algoritmo.

Algorithm 1 ENN

Require: *list*: uma lista

1. **for all** instância e_i da base de dados original **do**
2. Aplique o KNN sobre e_i
3. **if** e_i foi classificado erroneamente **then**
4. salve e_i em *list*
5. **end if**
6. **end for**
7. Remova da base de dados todos os elementos que estão em *list*

O valor de K pode variar de acordo com o tamanho da base de dados, porém, tipicamente, utiliza-se o valor de K=3. Tipicamente, O valor de K é inversamente proporcional a quantidade de instâncias que serão eliminadas, ou seja, para que o filtro elimine todos os possíveis ruídos, deve-se utilizar K=1.

Na figura 2.1 pode-se observar uma base de dados com duas classes, no primeiro gráfico da figura, pode-se observar a base de dados original, antes da aplicação do ENN. No segundo

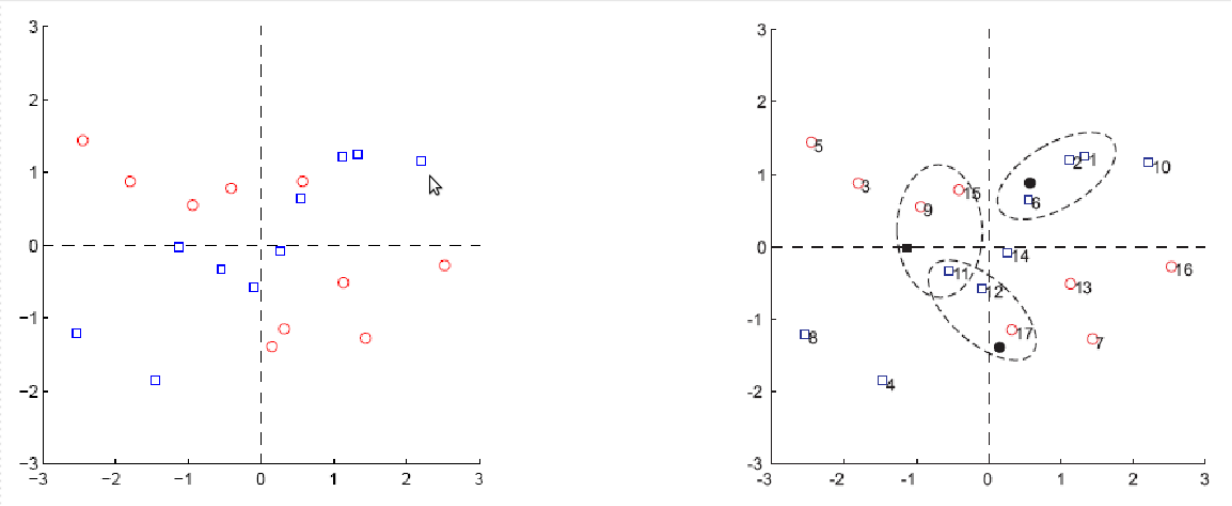

Figura 2.1 ENN aplicado com K=3

gráfico, foi aplicado o 1 com K=3 sobre a base de dados, os pontos pretos representam pontos que foram classificados erroneamente com a aplicação do KNN, a região circulada engloba os K elementos mais próximos. Observa-se que o elemento 11 foi utilizado para eliminar 2 instâncias ruidosas, um próprio ruído poderia ser utilizado para eliminar outro ruído, apesar de ser improvável.

O mais interessante do caso acima é que, após a aplicação do ENN, as classes ficaram bem separadas pelos quadrantes pontilhados, mostrando a eficiência do ENN para a base de dados acima.

Uma vantagem do ENN é que ele independe da ordem que a base de dados foi apresentada, ou seja, o ENN aplicado a uma base de dados, com o mesmo valor de K, sempre terá o mesmo resultado.

Porém, o ENN também apresenta desvantagens, ele possui uma baixa capacidade de redução, pois elimina apenas ruídos, mantendo instâncias que são desnecessárias, que apresentam apenas redundância de informação. No caso da 2.1, a base poderia ser representada por 4 instâncias bem posicionadas ou, por se tratar de uma técnica seletiva, com 8 instâncias, porém, o ENN manteve 13 instâncias, eliminando apenas 3.

Pelas suas características, normalmente o ENN é utilizado como método de pré-processamento da base de dados, eliminando apenas instâncias que apresentam alta probabilidade de serem ruídos.

No caso de bases desbalanceadas, o ENN pode tratar todas as instâncias da base minoritária como ruídos, caso a base seja altamente desbalanceada (isto será demonstrado posteriormente com exemplos). Uma possível adaptação para o ENN em bases altamente desbalanceadas é eliminar apenas os elementos que sejam da base de dados marjoritária. O algoritmo adaptado está demonstrado em 2.

Com este algoritmo, as instâncias da classe minoritária seria mantida, e a região delimitada por ela seria mais bem definida.

Algorithm 2 ENN

Require: $list$: uma lista

1. **for all** instância e_i da base de dados original **do**
2. Aplique o KNN sobre e_i
3. **if** e_i foi classificado erroneamente **then**
4. **if** e_i for da classe marjoritária **then**
5. salve e_i em $list$
6. **end if**
7. **end if**
8. **end for**
9. Remova da base de dados todos os elementos que estão em $list$

2.2 CNN

Condensed Nearest Neighbor [Har68] é uma técnica de seleção de protótipos puramente seletiva que tem como objetivo eliminar informação redundante. Diferentemente do ENN [CPZ11], o CNN não elimina instâncias nas regiões de fronteira, a técnica mantém estes elementos pois estes que "são importantes"para distinguir entre duas classes.

A ideia geral do CNN é encontrar o menor subconjunto da base de dados original que, utilizando o 1-NN, classifica todos os padrões da base de dados original corretamente. Fazendo isso, o algoritmo elimina os elementos mais afastados da região de indecisão, da fronteira de classificação.

O algoritmo do CNN será mostrado abaixo, e logo após, comentários a respeito do mesmo.

Algorithm 3 CNN

Require: $list$: uma lista

1. Escolha um elemento de cada classe $aleatoreamente$ e coloque-os em $list$
2. **for all** instância e_i da base de dados original **do**
3. $KNN(e_i, list)$
4. **if** e_i foi classificado erroneamente **then**
5. salve e_i em $list$
6. **end if**
7. **end for**
8. **return** $list$, os protótipos

Podemos observar que este algoritmo possui uma abordagem diferente do ENN, por exemplo, pois ele começa com um conjunto mínimo de instâncias (uma de cada classe) e depois adiciona instâncias conforme a necessidade de mantê-las para que todos os elementos da base de dados original sejam classificados corretamente.

Uma coisa que pode-se observar no algoritmo, é a palavra $aleatoriamente$, o que significa que o CNN aplicado numa mesma base de dados com um mesmo valor de K para o KNN, nem sempre resulta nos mesmos protótipos. O primeiro fato para que isso ocorra é a seleção aleatória dos protótipos iniciais. Existem algumas adaptações para o CNN, onde os protótipos iniciais

são escolhidos utilizando técnicas como o SGP[FHA07] para obter as instâncias mais centrais. Modificações no CNN são muito comuns [Tom76], porém, mesmo com estas modificações, o CNN ainda não é determinístico, pois a ordem em que as instâncias são classificadas afeta o resultado final.

Para o caso de estudo abordado neste trabalho, o CNN pode ser utilizado de forma adaptada. A adaptação consiste em manter todos os elementos da classe minoritária e o mínimo possível da classe marjoritária. O próprio CNN se encarrega de remover os elementos redundantes da classe marjoritária, assim, basta apenas selecionar todos os elementos da classe minoritária aos protótipos iniciais. Segue abaixo o algoritmo desta adaptação:

Algorithm 4 CNN para bases desbalanceadas

Require: *list*: uma lista

1. Coloque todos os elementos da classe minoritária em *list*
2. **for all** instância e_i da base de dados original **do**
3. Aplique o KNN sobre e_i utilizando os elementos em *list* para treinamento
4. **if** e_i foi classificado erroneamente **then**
5. salve e_i em *list*
6. **end if**
7. **end for**
8. **return** base original - *list*

Com o algoritmo CNN adaptado para bases desbalanceadas, os elementos redundantes da classe marjoritária são removidos, e todos os elementos da classe minoritária são mantidos. Esta adaptação do CNN irá reduzir a base de dados, ocasionando as vantagens de redução, e ainda reduzirá o desbalenceamento da base.

2.3 Tomek Links

Mantendo a mesma linha do ENN, Tomek Links é uma técnica de seleção de protótipos puramente seletiva que elimina os elementos das regiões de fronteiras e instâncias com probabilidade de ser ruído. Tomek Links podem ser definidos da seguinte forma: Dadas duas instâncias e_i e e_j, o par (e_i, e_j) é chamado de Tomek Link se não existe nenhuma instância e_k, tal que, para todo e_k $dist(e_i,e_j) < dist(e_i,e_k)$ e $dist(e_i,e_j) < dist(e_j,e_k)$. Segue o algoritmo detalhado em 5:

Os Tomek Links representam elementos da região de fronteira e prováveis ruídos, e a técnica de seleção de protótipos consiste em remover os Tomek Links da base de dados original. O algoritmo da seleção de protótipos Tomek Links é apresentado em 6:

Enquanto o CNN remove os elementos que estão longe da região de indecisão, o Tomek Links remove os elementos que estão próximos desta região, o que causa uma maior separação entre as classes.

Observa-se facilmente que os Tomek Links pode remove todas as instâncias da fronteira, inclusive as instâncias da classe minoritária, assim sendo, uma possível adaptação dos Tomek Links é eliminar apenas os elementos das classes marjoritárias. Nesse caso, ainda ocorreria uma

Algorithm 5 Seleciona Tomek Links

Require: *list*: uma lista

1. **for all** instância e_i da base de dados original **do**
2. e_j = instância mais próxima de e_i
3. **if** instância mais p¿oxima de e_j for e_i **then**
4. **if** classe de e_i for diferente da classe de e_j **then**
5. salve o par *(e_i, e_j)* em *list*
6. **end if**
7. **end if**
8. **end for**
9. **return** *list*, Tomek Links

Algorithm 6 Tomek Links

Require: *list*: uma lista

1. *list* = *SelecionaTomekLinks* da base original
2. **for all** *(e_i, e_j)* em *list* **do**
3. remova e_i da base original
4. remova e_j da base original
5. **end for**
6. **return** base original filtrada

separação entre as classes, mas apenas as instâncias da classe marjoritária seriam removidos, dimunuindo assim o nível de desbalanceamento. Segue abaixo o algoritmo desta adaptação:

Algorithm 7 Tomek Links

Require: *list*: uma lista

1. *list* = *SelecionaTomekLinks* da base original
2. **for all** *(e_i, e_j)* em *list* **do**
3. **if** e_i for da classe marjoritária **then**
4. remova e_i da base original
5. **end if**
6. **if** e_j for da classe marjoritária **then**
7. remova e_j da base original
8. **end if**
9. **end for**
10. **return** base original - *list*

Com esta adaptação, a classe minoritária é mantida, evitando o aumento do desbalanceamento ou a remoção por alta probabilidade de ruído.

2.4 OSS

One-Sided Selection [KM97] é um método seletivo de seleção de protótipos, surgido pela combinação das técnicas CNN e Tomek Links. O algoritmo consiste na aplicação do CNN e depois da aplicação do Tomek Links como um filtro. O One-Sided Selection combina características das duas técnicas. A aplicação do CNN é feita para eliminar instâncias desnecessárias, redundantes, ou seja, instâncias que estão longe da fronteira de classificação. Já a aplicação do Tomek Links tem a função de remover elementos na fronteira de classificação, fazendo uma aparente separação das classes e removendo ruídos.

O OSS é muito utilizado para bases desbalanceadas, utilizando a adaptação do CNN, como monstrado no algoritmo 8.

Algorithm 8 One-Sided Selection

Require: *list*: uma lista
Require: $tomek_links_list$: uma lista
1. $list = CNNparabasesdesbalanceadas$ sobre a base original
2. $list = TomekLinksparabasesdesbalanceadas$ sobre *list*
3. **return** *list*

Observando o algoritmo, é fácil concluir que o One-Sided Selection é uma técnica apropriada para bases desbalanceadas. A aplicação do CNN adaptado elimina as instâncias redundantes da base marjoritária, colaborando para, além de diminuir a quantidade de instâncias longe da fronteira de classificação, diminuir o nível de desbalanceamento entre as classes. Já a aplicação do Tomek Links adaptado, elimina instâncias da classe marjoritária na fronteira de classificação, colaborando para maior delimitação da classe minoritária.

Uma desvantagem do One-Sided Selection é que ele não é determinístico, o CNN é não-determinístico e como o One-Sided Selection faz a aplicação dele, torna o mesmo não-determinístico. O OSS poderia ser feito aplicando-se outro algoritmo no lugar do CNN, podendo assim, torna-lo determinístico. Este trabalho, porém, não abordará adaptações para o OSS, pois o mesmo já é apropriado para base de dados, e ser ou não determinístico, apesar de ser levado em consideração, não faz parte do escopo deste trabalho.

2.5 LVQ

Learning Vector Quantization proposto por Kohonen [Koh88]. O Learning Vector quantization é um algoritmo supervisionado de síntese de protótipos, ou seja, cria novas instâncias baseadas em instâncias já existentes. A ideia básica do algoritmo é que dado um conjunto inicial de protótipos, o LVQ faz um ajuste dos protótipos, de forma a posicionar cada instância em um ponto que seja possível estabelecer uma função discriminante baseada nestes protótipos.

Uma desvantagem do LVQ é que a ordem das instâncias altera o resultado, ou seja, o algoritmo não é deterministico. Outra desvantagem é que, conforme será mostrado, o LVQ possui vários parâmetros, sendo necessário uma análise empírica dos valores apropriados para esses parâmetros.

Os protótipos iniciais podem ser escolhidos de qualquer forma, a idéia é que sejam protótipos que tenham boa representatividade da base de dados, mas também podem ser selecionados aleatoriamente, pois o próprio LVQ se encarrega de fazer os ajustes nestes protótipos.

2.5.1 LVQ 1

LVQ1 é a primeira versão do Learning Vector Quantization proposto por Kohonen. O algoritmo do LVQ1 basicamente seleciona alguns protótipos iniciais e ajusta esses protótipos utilizando a base original. Quando uma instância da base original é classificada erroneamente pelos protótipos, afasta-se o protótipo mais próximo, e quando é classificada corretamente, aproxima-se. O algoritmo detalhado pode ser visto em 9.

Algorithm 9 LVQ 1

Require: *prototypes*: uma lista para os protótipos
Require: *selection*: um algorítmo para seleção dos protótipos iniciais
1. *prototypes* = *selection* (base original)
2. **while** *prototypes* não estiver sub-ajustado **do**
3. x = *ChooseOne* (base original)
4. e_i = *SelectNearestFrom*(*prototypes*, x)
5. **if** classe de $e_i \neq$ classe de x **then**
6. $e_i = e_i + \alpha(t) \times [x - e_i]$
7. **else**
8. $e_i = e_i - \alpha(t) \times [x - e_i]$
9. **end if**
10. **end while**
11. **return** *prototypes*

A vantagem do LVQ1 é que ele estabiliza durante o treinamento, porém, ele possui um grande número de passos. Para a maioria dos problemas, o LVQ 1 possui um resultado satisfatório, mas além da demora, é necessário escolher os parâmetros corretamente.

Um dos parâmetros é o $\alpha(t)$, uma constante de ajuste, que serve para aproximar ou afastar os protótipos. Este afastamento ou aproximação é regulado pelo valor de $\alpha(t)$, sendo $0 < \alpha(t) < 1$. Percebe-se que $\alpha(t)$ foi colocado como uma função. Normalmente, essa função é uma exponencial descrescente, e o algoritmo termina quando $\alpha(t)$ se torna insignificante.

Outra questão do LVQ1 é escolher a quantidade de protótipos iniciais adequada, visto que, esta quantidade não é alterada durante toda a execução do algoritmo.

No caso de bases desbalanceadas, pode-se utilizar fatores de ajustes diferenciados para cada classe, ou escolher uma quantidade aproximada de cada classe para os protótipos iniciais. Fazendo estas adaptações, o LVQ1 poderá ter resultados melhores para bases desbalanceadas.

2.5.2 Optimized-learning-rate LVQ

Optimized-learning-rate LVQ [colocar referencia aqui http://www.springerlink.com/content/n72865x1t57q187
é uma versão otimizada do LVQ1, proposto para aumentar a velocidade de convergência do

LVQ1. O modelo consiste basicamente em cada protótipo ter taxas de aprendizado individuais, a dinâmica da taxa de aprendizado consiste no aumento da mesma caso o protótipo esteja classificando corretamente e na diminuição, caso contrário.

$$\alpha(t) = \frac{\alpha(t-1)}{1+s(t)\times\alpha(t-1)}$$

$$s(t) = +1, \text{ se } x \text{ é classificado corretamente}$$
$$s(t) = -1, \text{ se } x \text{ é classificado erroneamente}$$
$$0 < \alpha(t) < 1$$

Com esta alteração do valor de $\alpha(t)$ faz com que o LVQ convirja mais rapidamente, tornando o algoritmo OLVQ mais viável em termos de performace e mantendo as características do LVQ1.

2.5.3 LVQ 2.1

Kohonen propos duas novas versões melhoradas do LVQ, uma delas é o LVQ 2.1. Esta nova versão do LVQ faz atualização nos dois protótipos mais próximos desde que as condições de ajuste sejam atendidas.

A ideia do LVQ 2.1 é ajustar apenas os protótipos próximos das fronteiras de classificação, região de indecisão. Para evitar uma divergência entre estes protótipos, foi introduzida a Regra da Janela.

Diz-se que um elemento está na janela quando ele obedece a regra da janela, isso acontece quando um elemento está na região de indecisão.

Dado um elemento x, diz-se que ele está na janela se:

$$\min \frac{d_i}{d_j}\frac{d_j}{d_i} > s, \text{ onde } s = \frac{1-w}{1+w}$$
$$e_i \text{ e } e_j \text{ são os protótipos mais próximos de } x$$
$$d_i \text{ é a distância de } x \text{ para } e_i$$
$$d_j \text{ é a distância de } x \text{ para } e_j$$
$$w \text{ é a largura relativa}$$

A algoritmo de LVQ 2.1 é aplicado depois do LVQ 1, mas ele ajusta dois protótipos a cada iteração. Esta técnica não faz ajustes de protótipos se x não estiver na janela, se nenhum dos protótipos forem da classe de x ou se os protótipos forem da mesma classe. Pode-se ver o algoritmo detalhado em 10.

Enquanto o LVQ1 provoca o afastamento dos protótipos nas regiões de indecisão, o LVQ 2.1 reduz esse afastamento atuando apenas sobre protótipos vizinhos pertencentes a classes diferentes.

Uma desvantagem do LVQ 2.1 é que além do custo ser maior, a aplicação do mesmo pode sobre-ajustar os protótipos nas regiões de indecisão. Para diminuir esse sobre-ajuste, foi criado o LVQ 3, abordado na próxima sessão.

Algorithm 10 LVQ 2.1

Require: *prototypes*: uma lista para os protótipos

1. *prototypes* = $LVQ1$ (base original)
2. **while** *prototypes* não estiver sub-ajustado **do**
3. x = *ChooseOne* (base original)
4. e_i, e_j = $SelectNearestsFrom(prototypes, x)$
5. **if** $CaiuNaJanela(x, e_i, e_j)$ **then**
6. **if** $Classe(e_i) \neq Classe(e_j)$ **then**
7. **if** $Classe(e_i) = Classe(x)$ **then**
8. $e_i = e_i + \alpha(t) \times [x - e_i]$
9. $e_j = e_j - \alpha(t) \times [x - e_i]$
10. **else**
11. $e_i = e_i - \alpha(t) \times [x - e_i]$
12. $e_j = e_j + \alpha(t) \times [x - e_i]$
13. **end if**
14. **end if**
15. **end if**
16. **end while**
17. **return** *prototypes*

2.5.4 LVQ 3

A segunda melhora proposta por Kohonen foi o LVQ 3. Este método tenta evitar o sobre-ajuste do LVQ 2.1 atuando também quando o elemento já está sendo classificado corretamente pelos dois protótipos mais próximos, aproximando ambos da instância utilizada para ajuste. Além disso, a terceira versão do LVQ introduz um fator de estabização ε.

O fator de estabilização serve para suavizar o ajuste quando os protótipos já estão classificando corretamente x. O valor de ε deve ser tal que $0 < \varepsilon < 1$.

Assim como as outras versões do LVQ, o LVQ 3 é robusto, mas seu maior problema é determinar o valor de tantos parâmetros como α, ε e w. Porém, a maior desvantagem é não ser possível saber com certeza quando os protótipos foram ou não sobre-ajustados.

2.6 SGP

Self-Generating Prototypes [FHA07] é uma técnica de síntese de protótipos muito completa. Sua maior vantagem é que, enquanto muitas técnicas de seleção de protótipos dependem da escolha correta do número de protótipos iniciais ou possuem muitos parâmetros, o SGP encontra a quantidade de protótipos e a localização de cada uma em sua fase de treinamento, sem supervisão humana.

A ideia principal do SGP é formar um certo número de grupos e eleger representantes para esses grupos. Conforme necessário, o algorítmo divide os grupos ou move instâncias de um grupo para outro.

Algorithm 11 LVQ 3

Require: *prototypes*: uma lista para os protótipos

1. *prototypes* = $LVQ1$ (base original)
2. **while** *prototypes* não estiver sub-ajustado **do**
3. x = *ChooseOne* (base original)
4. e_i, e_j = *SelectNearestsFrom*(*prototypes*, x)
5. **if** $CaiuNaJanela(x, e_i, e_j)$ **then**
6. **if** $Classe(e_i) \neq Classe(e_j)$ **then**
7. **if** $Classe(e_i) = Classe(x)$ **then**
8. $e_i = e_i + \alpha(t) \times [x - e_i]$
9. $e_j = e_j - \alpha(t) \times [x - e_i]$
10. **else**
11. $e_i = e_i - \alpha(t) \times [x - e_i]$
12. $e_j = e_j + \alpha(t) \times [x - e_i]$
13. **end if**
14. **else if** $Classe(e_i) = Classe(e_j) = Classe(x)$ **then**
15. $e_i = e_i + \varepsilon \times \alpha(t) \times [x - e_i]$
16. $e_j = e_j + \varepsilon \times \alpha(t) \times [x - e_i]$
17. **end if**
18. **end if**
19. **end while**
20. **return** *prototypes*

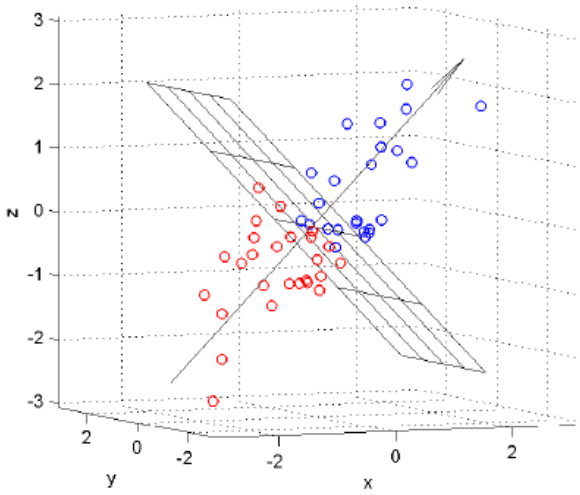

Figura 2.2 Etapa da divisão de grupos. Figura obtida de [dSPC08]

Inicialmente, para cada classe, é criado um grupo contendo todas as instâncias daquela classe. Com os grupos feitos, são obtidos os centróides de cada grupo e estes são chamados representantes do grupo. Depois, para cada grupo, faça os passos abaixo até que não haja mais alterações em nenhum grupo.

- Se para todos os padrões de um grupo o protótipo mais próximo é o centróide do grupo, então nenhuma operação é realizada.
- Se para todos os padrões de um grupo o protótipo mais próximo é de uma classe diferente da do grupo, ele é dividido em dois subgrupos 2.6. Essa divisão é feita separando os padrões pelo hiperplano que passa pelo centroide do grupo, e cujo vetor normal é a primeira componente principal gerada pelos padrões do grupo.
- Se para alguns padrões de um grupo o protótipo mais próximo é diferente do centróide, mas da mesma classe, esses padrões são deslocados do grupo original para o grupo do protótipo mais próximo
- Se para alguns padrões de um grupo o protótipo mais próximo não é o centróide e é de uma classe diferente, estes padrões são removidos do grupo original e formam um novo grupo, sendo o centróide computado como um novo protótipo.

No final de cada iteração, o centróide de cada grupo é computado novamente. O processo se repete até que não haja alterações em mais nenhum grupo. Uma descrição mais formal do algoritmo é descrita em 12

Observando o algoritmo do SGP1 12, percebe-se que apesar do conceito ser bem simples, esta técnica possui alguns passos complexos, sendo necessário conhecimentos sobre extração de características [REFERÊNCIA PCA AQUI]. Principal Component Analysis é a técnica utilizada para traçar o vetor perpendicular ao hiperplano 2.6.

Algorithm 12 SGP 1

Require: PS: uma lista para os representantes
Require: GS: uma lista de grupos de instâncias
Require: T: uma lista de duplas de instâncias

for all classe C **do**
  G = $\bigcup$ instâncias da classe C
  $Adicione(G, GS)$
  $Adicione(Centroide(G), PS)$
end for
$count = 1$
while $count \neq 0$ **do**
  $count$ = $Quantidade(GS)$
  $Limpe(T)$
  for all G em GS **do**
    P = Representante de G
    for all e_i em $group$ **do**
      $NearestP$ = $1NN(e_i, PS)$
      $Adicione((e_i, NearestP), T)$
    end for
    if $\forall\ e_i, NearestP$ em T, $NearestPS$ = P **then**
      $count$ = $count - 1$
    else if $\forall\ e_i, NearestP$ em T, a $Classe(NearestP) \neq Classe(P)$ **then**
      $Vector = PrimeiraComponentePrincipal(G)$
      $Hiperplano$ = hiperplano que passa pelo centróide de G e cujo vetor normal é a $Vector$.
      Divida G em 2 grupos, instâncias acima e abaixo de $Hiperplano$
      Atualize GS e PS.
    else if $\exists\ e_i, NearestP$ em T tal que $NearestP \neq P$ e $Classe(NearestP) = Classe(P)$ **then**
      Remova e_i de G e adicione a grupo de $NearestP$.
      Atualize GPS e PS.
    else if $\exists\ e_i, NearestP$ em T tal que o $Classe(NearestP)! = Classe(P)$ **then**
      Remova e_i de G.
      Crie um novo grupo contendo as instâncias removidas.
      Atualize PS e GS,
    end if
  end for
end while
return PS

A maior vantagem do SGP1 é que, conforme citado anteriormente, ele é não depende de parâmetros como quantidade de protótipos iniciais nem valores específicos. Por ser um algoritmo deterministico, o SGP1 executado numa mesma base de dados, sempre gerará os mesmos protótipos. Estes protótipos gerados possuem excelente representatividade do conjunto de treinamento, tanto que, utilizando-se os protótipos gerados como treinamento de um KNN, e classificando-se todas as instâncias de treinamento do SGP1, a taxa de acerto é de 100%. Claro que o treinamento não pode ser utilizado para teste, mas isto mostra a boa representatividade das instâncias geradas pelo SGP1.

Porém, o SGP1 também apresenta desvantagens. Uma delas é que o SGP1 é muito custoso, exigindo, em geral, um treinamento mais longo que outras técnicas.

Outra desvantagem é que o SGP1 é sensível a ruídos, pois, se necessário, o algoritmo criará um grupo com apenas uma instância. No caso de um ruído, isto será muito desvantajoso, considerando que, em experimentos diversos, o SGP1 conseguiu reduzir em mais de 100 vezes o tamanho da base de dados.

Apesar das desvantagens citadas, bases desbalanceadas podem ser beneficiadas com o SGP1, considerando que ele considera os agrupamentos de classes, e não a quantidade de instâncias em cada classe.

2.7 SGP 2

O Self-Generating Prototypes 2 [FHA07]é uma versão melhorada do SGP1 que reduz ainda mais a quantidade de protótipos e é menos sensível aos ruídos e outliers. Para fazer essas melhorias, o SGP2 possui uma etapa de *merge* e *prunning*.

Quando dois grupos A e B são da mesma classe e para todas as instâncias de A o segundo protótipo mais próximo é o representante de B, e para todas as instâncias de B o segundo protótipo mais próximo é o representante de A, os grupos A e B podem ser fundidos, e será computado um novo protótipo para este novo grupo maior. Esta etapa é chamada de *merge*, e ela é responsável por reduzir ainda mais a quantidade de protótipos do SGP1. O *prunnig* consiste em remover grupos onde o segundo protótipo mais próximo de todos os padrões de um certo grupo possuem a mesma classe, neste caso, tanto as instâncias quanto o representante do grupo são eliminados.

Para evitar o sobre-ajuste e perda de generalização introduziu-se dois parâmetros ao SGP2 chamados de R_n e R_s. O R_n representa um limite para o tamanho relativo de um grupo em relação ao maior grupo. Por exemplo, se R_n é 0.1 e o tamanho do maior grupo é 200, todos os grupos com tamanho menor que $20(0.1 \times 200)$ são descartados. O segundo parâmetro R_s é um limiar para a taxa de classificações incorretas de um grupo. Se o número de padrões classificados erroneamentes em um grupo dividido pelo tamanho do grupo é menor que R_s, então o grupo permanece inalterado. Este parâmetros reduzem o número de protótipos e aumentam a capacidade de generalização. Pereira e Cavalcanti recomendam $0.01 < R_n < 0.2$ e $0.01 < R_s < 0.2$ [dSPC08].

O algoritmo SGP2 13 é mais custoso e demorado que o SGP1, porém, em experimentos práticos, o SGP2 se mostrou mais eficiente, diminuindo a quantidade de protótipos ainda mais que o SGP1 e ainda assim, aumentando a taxa de acerto.

Algorithm 13 SGP 2

Require: PS: uma lista para os representantes
Require: GS: uma lista de grupos de instâncias
Require: T: uma lista de duplas de instâncias

for all classe C **do**
  $G = \bigcup$ instâncias da classe C
  $Adicione(G, GS)$
  $Adicione(Centroide(G), PS)$
end for
$count = 1$
while $count \neq 0$ **do**
  $count = Quantidade(GS)$
  $Limpe(T)$
  for all G em GS **do**
    P = Representante de G
    for all e_i em $group$ **do**
      $NearestP = 1NN(e_i, PS)$
      $Adicione((e_i, NearestP), T)$
    end for
    if Quantidade de tuplas em T onde $NearestP \neq P \leq R_s$ **then**
      $count = count - 1$
    else if $\forall\ e_i, NearestP$ em T, a $Classe(NearestP) \neq Classe(P)$ **then**
      $Vector = PrimeiraComponentePrincipal(G)$
      $Hiperplano$ = hiperplano que passa pelo centróide de G e cujo vetor normal é a $Vector$.
      Divida G em 2 grupos, instâncias acima e abaixo de $Hiperplano$
      Atualize GS e PS.
    else if $\exists\ e_i, NearestP$ em T tal que $NearestP \neq P$ e $Classe(NearestP) = Classe(P)$ **then**
      Remova e_i de G e adicione a grupo de $NearestP$.
      Atualize GPS e PS.
    else if $\exists\ e_i, NearestP$ em T tal que o $Classe(NearestP)! = Classe(P)$ **then**
      Remova e_i de G.
      Crie um novo grupo contendo as instâncias removidas.
      Atualize PS e GS,
    end if
  end for
end while
$LargeGroupCount$ = Quantidade de instâncias do maior grupo em GS
for all G, P in GS, PS **do**
  $CurrentGroupCount$ = Quantidade de instâncias de G.
  if $CurrentGroupCount/LargeGroupCount \leq R_n$ **then**
    Remova G de GS e P de PS.
  end if
end for
return PS

Porém, um defeito do SGP2 é que ele não leva em conta bases desbalanceadas. Dependendo da distribuição das instâncias, pode acontecer de, na fase de *prunning* o SGP2 remover o representante da classe minoritária, podendo até remover todos os protótipos desta classe. Isto é perceptível, considerando o valor de $R_n = 0.15$, bases onde a classe minoritária possui apenas 3% das instâncias da base, mesmo com a base marjoritária multimodal, provavelmente, após o *prunning*, todos os protótipos serão da classe marjoritária.

2.8 CCNN

Referências Bibliográficas

[CPZ11] Ruiqin Chang, Zheng Pei, and Chao Zhang. A modified editing k-nearest neighbor rule. *JCP*, 6(7):1493–1500, 2011.

[dSPC08] Cristiano de Santana Pereira and George D. C. Cavalcanti. Prototype selection: Combining self-generating prototypes and gaussian mixtures for pattern classification. In *IJCNN*, pages 3505–3510. IEEE, 2008.

[EJJ04] Andrew Estabrooks, Taeho Jo, and Nathalie Japkowicz. A multiple resampling method for learning from imbalanced data sets. *Computational Intelligence*, 20(1):18–36, 2004.

[FHA07] Hatem A. Fayed, Sherif Hashem, and Amir F. Atiya. Self-generating prototypes for pattern classification. *Pattern Recognition*, 40(5):1498–1509, 2007.

[Har68] P. E. Hart. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, 14:515–516, 1968.

[HKN07] Jason Van Hulse, Taghi M. Khoshgoftaar, and Amri Napolitano. Experimental perspectives on learning from imbalanced data. In Zoubin Ghahramani, editor, *ICML*, volume 227 of *ACM International Conference Proceeding Series*, pages 935–942. ACM, 2007.

[KM97] Miroslav Kubat and Stan Matwin. Addressing the curse of imbalanced training sets: One-sided selection. In Douglas H. Fisher, editor, *ICML*, pages 179–186. Morgan Kaufmann, 1997.

[Koh86] Teuvo Kohonen. Learning vector quantization for pattern recognition. Report TKK-F-A601, Laboratory of Computer and Information Science, Department of Technical Physics, Helsinki University of Technology, Helsinki, Finland, 1986.

[Koh88] Teuvo Kohonen. Learning vector quantization. *Neural Networks*, 1, Supplement 1:3–16, 1988.

[PI69] E. A. Patrick and F. P. Fischer II. A generalization of the k-nearest neighbor rule. In *IJCAI*, pages 63–64, 1969.

[Sav] Sergei Savchenko. http://cgm.cs.mcgill.ca/ .

[TC67] P.E. Hart T.M. Cover. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, IT-13:21 – 27, 1967.

[Tom76] I. Tomek. Two Modifications of CNN. *IEEE Transactions on Systems, Man, and Cybernetics*, 7(2):679–772, 1976.

[WP01] G. Weiss and F. Provost. The effect of class distribution on classifier learning, 2001.