Relatório do trabalho 3 (parte 2)
Aprendizado baseado em instâncias (IBL 3 e 4)

João Paulo Pizani Flor
Mauricio Oliveira Haensch

24 de abril de 2011

1 Introdução

O objetivo do terceiro trabalho da disciplina era implementar os algoritmos de aprendizado de máquina baseado em instâncias (*Instance-Based Learning - IBL*). Para a segunda etapa do trabalho, eram necessários apenas os algoritmos IBL3 e IBL4. Outros requisitos também foram pedidos:

- Deve ser possível gerar (ou carregar) dados da mesma maneira (ou melhor) que no Trabalho 1.1.
- Após o treinamento, deve ser possível classificar novos padrões.
- Quaisquer que sejam os parâmetros do algoritmo IBL3, eles devem ser configuráveis.

A família de algoritmos IBL faz parte das técnicas de raciocínio indutivo de aprendizado de máquina, e aprendem a categorizar um conjunto de classes de forma incremental com base em exemplos de instâncias dessas classes. O princípio básico de funcionamento desses algoritmos é a utilização de um conjunto de treinamento, que é processado de diferentes maneiras de acordo com o algoritmo em questão (IBL 1, 2, 3 ou 4). Após o treinamento com um conjunto inicial, o algoritmo IBL gera um descritor conceitual, que mapeia padrões a categorias. Esse descritor é utilizado para classificar novos padrões.

Os algoritmos IBL possuem 3 componentes principais:

- Função de similaridade: usada no algoritmo para computar a similaridade entre um dado padrão de treinamento e as instâncias presentes no descritor conceitual;
- Função de classificação: classifica uma dada instância, com base no resultado da função de similaridade e os registros de performance de classificação na descrição conceitual;
- Atualizador do descritor conceitual: registra a performance de classificação e decide quais instâncias serão integradas ao descritor conceitual durante o treinamento;

O pseudo-código do algoritmo básico, IBL 1, pode ser visto logo abaixo. As outras versões do algoritmo são adaptações tentando melhorar a performance de reconhecimento de padrões. O código abaixo foi retirado do material apresentado em sala de aula.

```
1  variavel
2  DC: descricao conceitual
3  ConjuntoTreinamento: conjunto de padroes para treinamento
4  SimX: valores de similaridade entre x e cada um dos y
5  Classificacao: resultado da classificacao
6
7  DC := {}
8  para cada (x em ConjuntoTreinamento) faca
9      para cada (y em DC) faca
10         SimX[y] := FuncaoSim(x, y)
11     fim para
```

```
12
13      ymax := algum y com maior SimX[y]
14
15      se (classe de x = classe de ymax) entao
16          Classificacao = correta
17      senao
18          Classificacao = incorreta
19      fim se
20
21      DC := DC U {x}
22  fim para
```

Pode ser observado que para o algoritmo IBL 1 a classificação estar correta ou incorreta não interfere no resultado final do descritor conceitual, pois todas instâncias do conjunto de treinamento são adicionadas ao descritor. Essa é a única diferença com relação ao algoritmo IBL 2, que guarda apenas os resultados considerados incorretos, na tentiva de manter no descritor os casos que possuem mais chances de dar errado, para que durante a classificação de novos padrões, esse erro não volte a se repetir. A desvantagem dessa estratégia é quando existe um nível de ruído muito grande, que acaba prejudicando a performance do classificador.

Essas são as características principais dos algoritmos IBL 1 e 2, requisitos da primeira etapa do trabalho. A seguir, serão apresentadas as ferramentas utilizadas para a implementação do trabalho e em sequência demonstramos alguns dos resultados alcançados.

O algoritmo IBL 3 foi criado a partir do IBL 2, mas tentando suprir suas deficiências com relação ao ruído. Para isso, é introduzida a ideia de similaridade aceitável, que é um limite mínimo de similaridade entre o padrão sendo classificado e um padrão de referência do conjunto do descritor conceitual. Em outra etapa do algortimo, os padrões que não atingem uma taxa mínima de acertos durante o treinamento são eliminados do descritor conceitual.

Já o algoritmo IBL 4 utiliza uma função de similaridade diferente dos algoritmos anteriores, levando em consideração um vetor de pesos dos atributos de uma categoria. Esses pesos são aprendidos e atualizados durante o treinamento. Isso pois a relevância de um atributo durante o cálculo de similaridade pode variar de acordo com a categoria.

2 Ferramentas utilizadas

A linguagem utilizada para implementação do programa era de livre escolha dos alunos, assim como outras ferramentas que pudessem ser utilizadas para representação gráfica dos padrões, por exemplo. As ferramentas escolhidas para a implementação do trabalho foram:

Linguagem de programação - Python: Escolhida por ser uma linguagem já bem conhecida pela equipe, com boas bibliotecas e documentação, além de agilizar o desenvolvimento.

Interface gráfica - PyQt: *Binding* do framework gráfico Qt para a linguagem Python, escolhida por já ser conhecida pelos integrantes e por haver uma interface desenvolvida para um trabalho anterior que pode ser reaproveitada com poucas modificações.

Algumas funcionalidades da linguagem Python ajudaram muito no desenvolvimento do trabalho e o tornaram bem mais ágil. Principalmente algumas implementações de idéias do paradigma funcional, tais como *list comprehensions*[1], *generator expressions*[2] e a passagem de funções como parâmetro.

Em caráter experimental, foi utilizada uma biblioteca de paralelismo de dados chamada forkmap. Ela implementa uma primitiva de iteração similar ao *map* nativo da linguagem Python, porém onde os itens a serem processados são divididos em n threads, onde n (por padrão) é o número de CPUs disponíveis na máquina do usuário. Essa biblioteca foi utilizada no método *Recognizer.classifyVectors()* da seguinte maneira:

```
1 def classifyVectors(self, vectors):
2     vAndClass = lambda v: (v.coordinates(), self.classify(v))
3     results = forkmap.map(vAndClass, list(vectors))
4     return results
```

A função *vAndClass* toma um vetor como entrada e o classifica. Essa função é aplicada paralelamente (usando o *forkmap.map()*) a cada membro da lista de vetores a serem classificados.

Além disso, no design da interface gráfica, os componentes do *framework* Qt facilitaram muito a tarefa de tratamento de situações não desejadas, ou seja, de entradas inválidas fornecidas pelo usuário. Para fazer a entrada da maioria dos parâmetros para o programa, pudemos utilizar componentes de interface que *proibiam* a entrada de dados inválidos. Por exemplo, no caso do número de pontos para treinamento, nível de ruído das espirais, etc, pudemos definir intervalos aceitáveis de entrada.

[1] PEP202 - List comprehensions: http://www.python.org/dev/peps/pep-0202
[2] PEP289 - Generator expressions: http://www.python.org/dev/peps/pep-0289

O trabalho foi desenvolvido no ambiente Linux/Ubuntu e para executá-lo são necessários alguns pacotes[3] para a interface gráfica:

- pyqt4-dev-tools
- libqtgui4
- libqtcore4

[3]os nomes dos pacotes citados são específicos para a distribuição Ubuntu, para outras distribuições devem ser procurados os pacotes correspondentes.

3 Alguns resultados

Os dados para treinamento do algoritmo IBL podem ser inseridos através da própria interface gráfica (no caso de 2 dimensões), bastando clicar na própria área de desenho para adicionar o ponto. Os pontos também podem ser carregados de um arquivo de dados.

Após feito o treinamento (com qualquer um dos algoritmos IBL disponíveis), o usuário pode adicionar pontos a serem classificados. Nesta primeira entrega do trabalho o treinamento do algoritmo é uma operação atômica. Isso quer dizer que, uma vez feito o treinamento e uma classificação, só é possível voltar a treinar o algoritmo removendo todos os pontos e adicionando todos novamente.

3.1 Considerações sobre o números de dimensões dos padrões

Nossa implementação é capaz de trabalhar com o treinamento e classificação de padrões com um número qualquer de dimensões. Naturalmente, porém, a visualização gráfica do descritor conceitual e dos resultados, assim como a entrada de pontos através de cliques, só é possível para duas dimensões.

Para um número superior de dimensões são utilizados arquivos de texto. Através do menu "Arquivo" o usuário tem à disposição os comandos para carregar arquivos contendo padrões de treinamento e arquivos contendo vetores a serem classificados. O formato tanto para arquivos de entrada como para arquivo de saída gerados é CSV[4]. Um típico fluxo de uso do programa segue:

- Usuário usa "Arquivo - Carregar arquivo de treinamento" para selecionar o arquivo contendo padrões para usar no treinamento. O algoritmo IBL previamente selecionado é usado e o descritor conceitual é armazendo no arquivo conceptualDescriptor.csv
- Usuário usa então "Arquivo - Carregar arquivo de padrões a classificar" para selecionar o arquivo com os padrões a serem classificados. Os resultados da classificação serão armazenados no arquivo classificationResults.csv

Iremos agora demonstrar alguns dos resultados alcançados com o trabalho para os requisitos exigidos. Primeiramente o treinamento:

3.2 Treinamento

A geração de pontos em espiral para treinamento foi facilmente reaproveitada dos trabalhos anteriores. A função de geração de espiral foi implementada em coordenadas polares, com um passo posterior de transformação para coordenadas cartesianas. Alguns parâmetros da espiral podem ser ajustados pelo usuário, como a distância entre 2 voltas sucessivas da espiral, o número de pontos e o nível de ruído. A figura 1 demonstra o uso de uma espiral para treinamento.

A outra maneira gráfica (2D) de se fazer a inserção de dados para treinamento é interativamente, inserindo os pontos um a um através de cliques na própria área de desenho. A figura 2 demonstra pontos de treinamento em que uma classe forma uma espécie de "invólucro" em torno de outra. Esse tipo de situação faz visível

[4] Comma-Separated Values

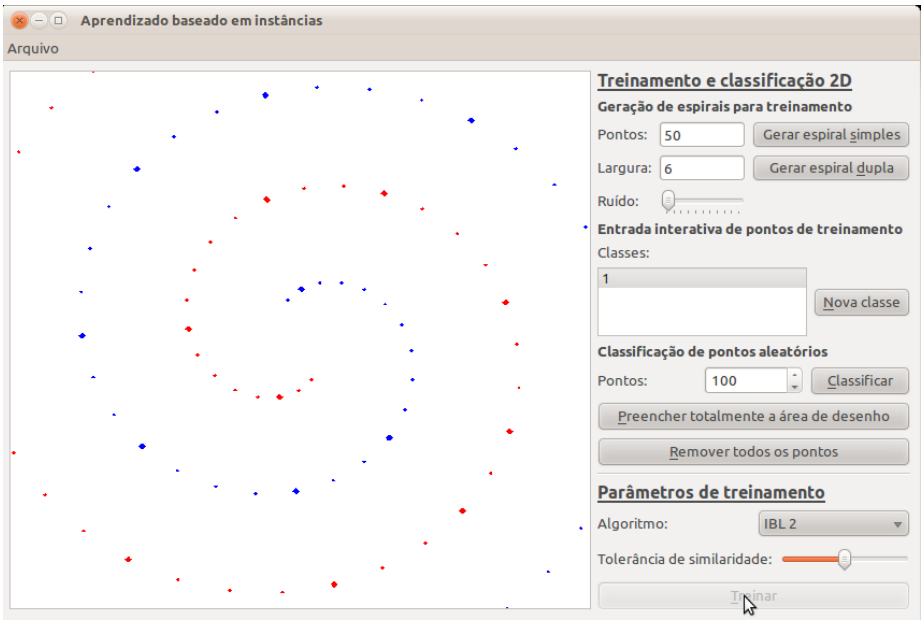

Figura 1: Uma espiral dupla, com 50 pontos, largura 6 e treinada com IBL2

uma vantagem do algoritmo IBL2, que guarda menos pontos para ter eficiência de reconhecimento semelhante.

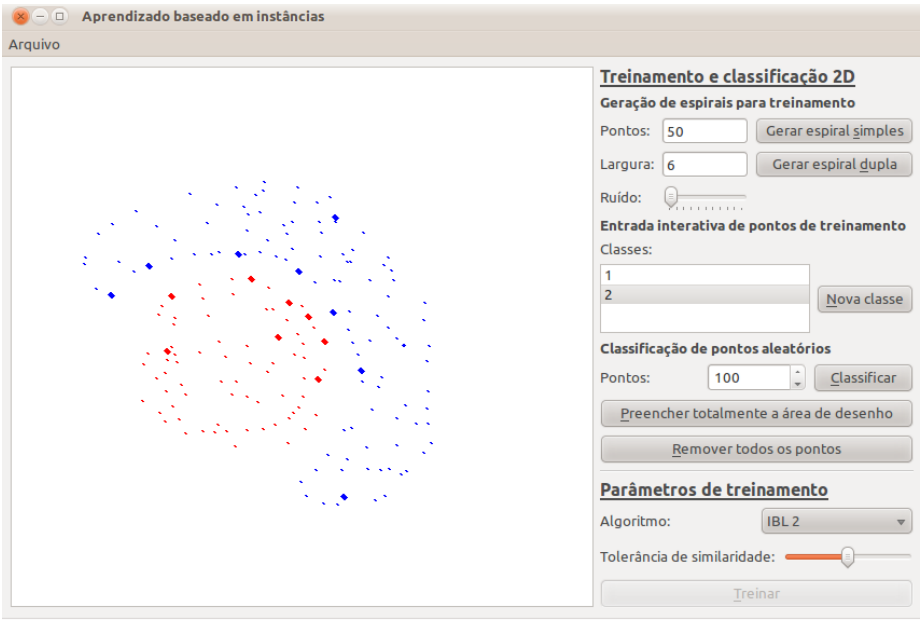

Figura 2: Treinamento com pontos inseridos interativamente, usando IBL2

3.3 Classificação de novos pontos

Na classificação de novos vetores, nossa implementação utiliza como função de similaridade entre atributos a diferença absoluta. Assim, na classificação a métrica é equivalente à distância de Manhattan.

Em nosso programa o usuário pode entrar com vetores a serem classificados de forma textual (para pontos com mais de 2 dimensões) ou gráfica. No caso da entrada através da interface gráfica, as seguintes formas de entrada podem ser utilizadas: aleatória. interativa, ou preenchimento total. Cada uma será melhor explicada a seguir através de uma figura.

3.3.1 Classificação de pontos aleatórios

Nesse caso o usuário apenas seleciona quantos pontos deseja classificar, e então pontos são gerados (com distribuição uniforme), classificados e desenhados. A figura 3 demonstra esse caso de uso.

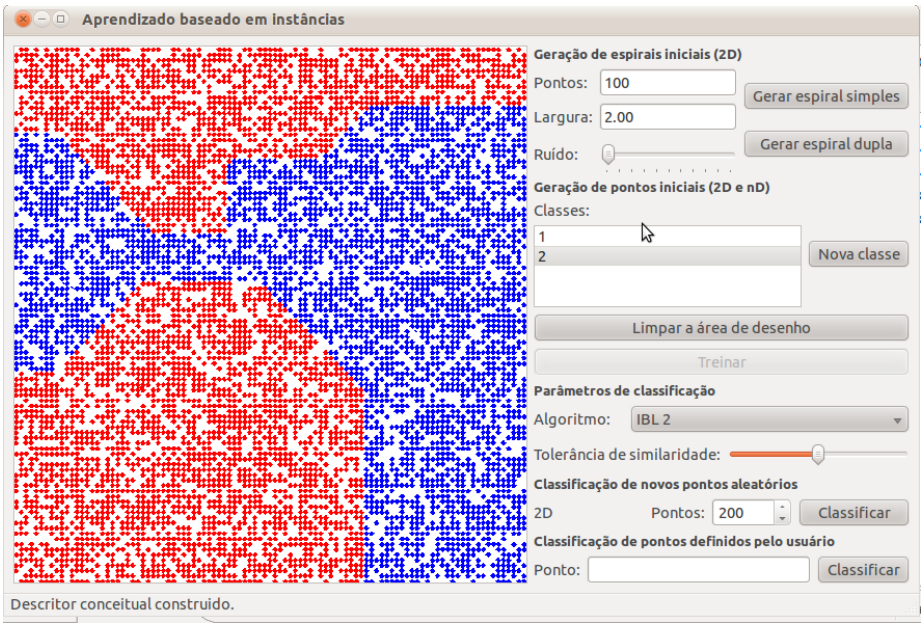

Figura 3: Caso de uso de classificação de pontos aleatórios. Nome artístico: "Cachorro com vuvuzela"

3.3.2 Classificação interativa

O usuário pode também clicar na própria área de desenho para inserir um ponto naquele local e classificá-lo (colorí-lo) imediatamente. O ponto recém-inserido é pintado com um maior tamanho, destacando-se dos demais. Tal situação é ilustrada pela figura 4.

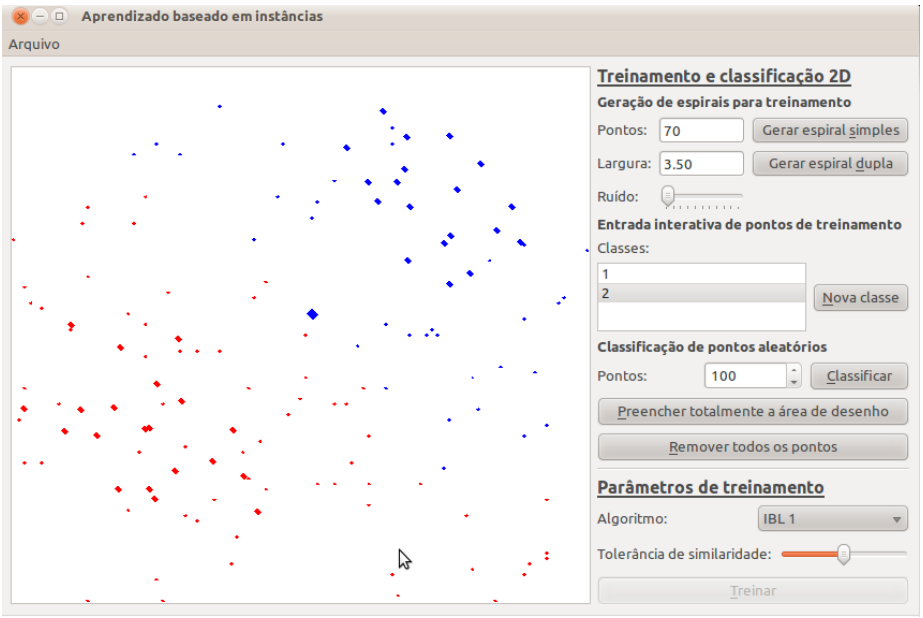

Figura 4: Inserção interativa de um ponto a ser classificado

3.4 Análise comparativa entre IBL 2 e IBL 3

Em 5 foi gerada uma espiral dupla com um nível alto de ruído, treinada com o algoritmo IBL 2 e com a função de preenchimento total da tela aplicada. Já em 6 consta uma imagem de uma espiral com os mesmos parâmetros (número de pontos, distância entre 2 voltas e nível de ruído), porém com o algoritmo IBL 3.

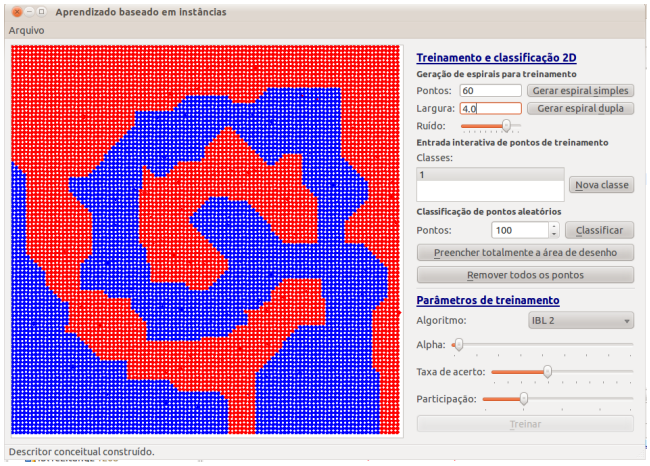

Figura 5: Uma dupla espiral treinada com IBL2.

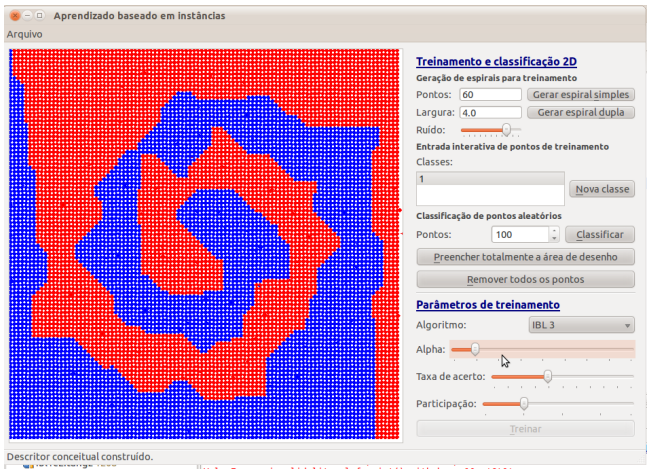

Figura 6: Uma dupla espiral treinada com IBL3.

Pode-se notar que o algoritmo IBL 3 possui uma eficácia superior, pois elimina os pontos ruidosos presentes na espiral que o IBL 2 incluiria em seu descritor conceitual.

3.5 Algoritmo IBL 4

O algoritmo IBL 4 atribui diferentes pesos aos atributos de uma dada categoria durante o treinamento. O resultado dessa estratégia, no entanto, não é facilmente visualizado no caso de vetores bidimensionais, pois há pouca diferença entre a relevância dos atributos.

4 Referências

- Python - http://www.python.org/
- Qt - http://doc.qt.nokia.com/
- pyQt - http://www.riverbankcomputing.co.uk/software/pyqt/intro
- PEP202 - http://www.python.org/dev/peps/pep-0202
- PEP289 - http://www.python.org/dev/peps/pep-0289
- David Aha - "Instance-Based Learning Algorithms"
- Flávia de Oliveira Santos e Maria do Carmo Nicoletti - "A Família de Algoritmos Instance-Based Learning"
- forkmap.py - http://honeypot.net/multi-processing-map-python