USP — UNIVERSIDADE DE SÃO PAULO

Aprendizagem automática de redes soma-produto

Relatório Final de Projeto de Iniciação Científica
CNPq PIBIC Projeto 800585/2016-0

Bolsista: Renato Lui Geh
Orientador: Prof. Dr. Denis Deratani Mauá

São Paulo

2018

Resumo

Modelos probabilísticos baseados em grafo (PGMs) possibilitam modelar distribuições de probabilidade complexas com milhares de varíaveis. Devido a grande expressividade em representabilidade, PGMs mostraram-se viáveis para modelagem de casos reais.

Redes soma-produto (SPNs) são PGMs que restringem-se ao escopo de distribuições de probabilidade tratáveis. SPNs tiveram bons resultados em diversas aplicações, obtendo valores comparáveis a outros modelos estado-da-arte, porém com tempo de execução ordens de magnitude menor.

Apesar dos resultados promissores, atualmente existem poucas bibliotecas para inferência e aprendizado de SPNs. Além disso, não existe atualmente uma comparação detalhada entre diferentes algoritmos de aprendizado de rede-soma produto.

Este projeto teve dois objetivos. O primeiro foi construir uma biblioteca livre e gratuita para inferência e aprendizado de redes soma-produto. O segundo foi fazer uma comparação de alguns algoritmos de aprendizado de SPNs no domínio de classificação e compleição de imagens.

Palavras-chave: Modelos probabilísticos baseados em grafo, redes soma-produto, processamento de imagens

1 Introdução

Modelos probabilísticos baseados em grafos (PGM, do inglês *Probabilistic Graphical Models*) representam uma distribuição de probabilidade de forma compacta. Estes modelos representados por grafos facilitam tanto a compreensão humana ao estudá-los, quanto possibilitam que vários problemas já existentes em Teoria dos Grafos sejam utilizados como solução para problemas em PGMs. Extrair conhecimento de PGMs é análogo a extrair a probabilidade de um certo evento ocorrer dado que eventos distintos tenham ocorrido. Tal extração de conhecimento é chamada de inferência. Fazer inferência exata em PGMs clássicas, ou seja, achar a probabilidade exata de um certo evento, é intratável. Uma solução para este problema é utilizar métodos para inferência aproximada nestes modelos. No entanto, tais algoritmos aproximados são muitas vezes difíceis de analisar. Além disso, como os algoritmos de aprendizado do modelo utilizam inferência como subrotina, por consequência o aprendizado torna-se aproximado.

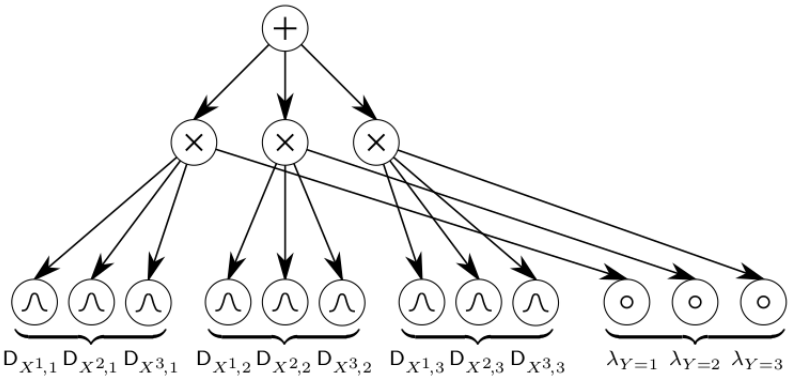


Figura 1: Fonte [12]

Redes soma-produto (SPN, de *Sum-Product Network*) são PGMs que representam uma distribuição de probabilidade tratável. Proposto em 2011, SPNs computam inferência exata em tempo linear ao número de arestas de seu grafo se sua estrutura obedecer a certas propriedades [14]. A Figura 1 mostra um exemplo de rede soma-produto representando o modelo Naïve Bayes com três atributos. SPNs apresentam uma série de características interessantes, como sua arquitetura profunda que permite representar funções de forma mais eficiente quanto mais profundo seu grafo [3]. Outras interessantes propriedades teóricas incluem uma generalização de SPNs para qualquer semianel em que o produto tenha escopo disjunto [7]. Com relação a aplicações, SPNs tiveram resultados impressionantes em diversas áreas, como enovelamento de proteínas [6], modelagem de sinais [13], classificação e reconstrução de imagens [10, 14, 4], reconhecimento de atividade [1] e linguagem natural [2].

Uma SPN pode ser definida como um DAG onde nós são somas ponderadas, produtos, variáveis indicadoras ou distribuições de probabilidade univariadas. Uma folha de uma SPN pode ser uma variável indicadora ou uma distribuição univariada. O escopo de uma SPN é o conjunto de todas variáveis da rede. O escopo de uma folha de uma SPN é a variável descrita pela variável indicadora ou distribuição univariada. Semânticamente, um filho de

um nó soma pode ser interpretado como uma relação de semelhança entre as variáveis de seu escopo, enquanto que o conjunto de filhos de um produto pode ser visto como uma relação de independência entre os escopos dos filhos do nó produto. O valor de um nó soma é a soma ponderada de seus filhos em função dos pesos de suas arestas de saída. O valor de um nó produto é o produto dos valores dos filhos, e o valor de uma folha é o valor da variável indicadora ou a probabilidade da distribuição univariada.

Apesar dos resultados expressivos, atualmente existem poucas bibliotecas para inferência e aprendizado de redes soma-produto. Grande parte dos códigos existentes possuem pouca documentação ou não são mais mantidos ou atualizados. Além disso, não existem comparações detalhadas entre algoritmos de aprendizado de SPNs na literatura ou uma biblioteca que facilite esta tarefa.

Neste projeto, buscou-se criar uma biblioteca livre e gratuita para inferência e aprendizado de redes soma-produto. Foram implementadas subrotinas para computar a probabilidade de evidência exata, a probabilidade *maximum a posteriori* aproximada de uma SPN e três métodos de aprendizado de redes soma-produto. Adicionalmente, foram implementadas funções para auxiliar a comparação entre diferentes arquiteturas de SPNs, verificar propriedades da estrutura da rede e classificar e completar imagens.

2 Objetivos

O projeto teve como objetivos criar uma biblioteca livre e gratuita para inferência e aprendizado de redes soma-produto e gerar dados comparativos de diversos algoritmos de aprendizado de SPNs estado-da-arte.

Originalmente, planejava-se implementar quatro algoritmos de aprendizado. No entanto, preferiu-se gerar um relatório mais detalhado de três algoritmos. Os três algoritmos implementados são listados abaixo.

a. Algoritmo de Poon-Domingos [14]

b. Algoritmo de aprendizado estrutural de Dennis-Ventura [4]

c. Algoritmo de aprendizado estrutural de Gens-Domingos [10]

Após as implementações, foram feitos testes de desempenho em conjuntos de dados reais e artificiais para classificação e compleição de imagens.

3 Metodologia

Neste projeto, buscou-se analisar três artigos de aprendizado de redes soma-produto.

1. *Sum-Product Networks: A New Deep Architecture*, H. Poon e P. Domingos, UAI 2011 [14]
2. *Learning the Structure of Sum-Product Networks*, R. Gens e P. Domingos, ICML 2013 [10]
3. *Learning the Architecture of Sum-Product Networks Using Clustering on Variables*, A. Dennis e D. Ventura, NIPS 2012 [4]

Foram implementados os algoritmos descritos e tentou-se replicar os resultados apresentados nos artigos. Para isso, foi construída a biblioteca GoSPN[1] na linguagem Go. A escolha da linguagem foi dada pelo suporte nativo a programação concorrente e paralela, velocidade de execução, e por ser uma linguagem fortemente tipada e compilada.

Após implementados os métodos de aprendizagem, foram feitos testes e comparações nos desempenhos e acurácias dos três algoritmos nos seguintes conjuntos de dados de imagens.

(a) Caltech-101 [5]

(b) Olivetti Faces Dataset [15]

(c) Digits [8]

(d) DigitsX [9]

(e) MNIST [11]

4 Resultados e discussão

O projeto consistiu na elaboração de três algoritmos de aprendizado. Estes algoritmos foram implementados na biblioteca GoSPN[1]. Duas variações do método de otimização de descida de gradiente foram escritas para os algoritmos de Poon-Domingos e Dennis-Ventura. Para o algoritmo de Gens-Domingos, foram implementados dois métodos de *clustering*, DBSCAN e k-means. Adicionalmente, os algoritmos k-mode e k-medoid foram implementados por Diarmaid Conaty[2] e Cassio P. de Campos[2] como contribuições à biblioteca. Para os testes de independência entre variáveis foram implementados os testes de qui-quadrado e

[1] `https://github.com/RenatoGeh/gospn`

[2] Queen's University Belfast

G-test. A biblioteca também possibilita o uso de distribuições gaussianas ou multinomiais como folhas da rede.

Para o método de otimização de descida de gradiente, foram criadas as versões *soft* e *hard*. A primeira usa as derivadas parciais da SPN em função dos pesos para atualizar os parâmetros da rede. No entanto, para o caso de redes profundas o gradiente tende a se aproximar de zero conforme o sinal é propagado pelos nós. Para evitar este problema, usa-se o segundo método. Seja S uma SPN, define-se M a rede máximo-produto (MPN) de S como o grafo cópia de S em que todos os nós somas são substituidos por nós máximo. O valor de um nó máximo é o maior valor ponderado de seus filhos. O valor de uma MPN é o valor da raíz. O caminho percorrido a partir da raíz de maior valor representa a probabilidade *maximum a posteriori* (MAP). Denotaremos por W o multiconjunto de todas as arestas que pertencem ao caminho descrito pelo MAP. Extraindo as derivadas da MPN M e denotando $\mathrm{Ch}(n)$ e $\mathrm{Pa}(n)$ os conjuntos de filhos e pais do nó n respectivamente, temos os valores descritos nas tabelas 1 e 2.

Método	**Derivada parcial do nó j**
Soft	$\dfrac{\partial S}{\partial S_j} = \sum\limits_{\substack{n \in \mathrm{Pa}(j) \\ n\text{: soma}}} w_{n,j} \dfrac{\partial S}{\partial S_n} + \sum\limits_{\substack{n \in \mathrm{Pa}(j) \\ n\text{: produto}}} \dfrac{\partial S}{\partial S_n} \prod\limits_{k \in \mathrm{Ch}(n) \setminus \{j\}} S_k$
Hard	$\dfrac{\partial M}{\partial M_j} = \sum\limits_{\substack{n \in \mathrm{Pa}(j) \\ n\text{: soma}}} \begin{cases} w_{k,n} \frac{\partial M}{\partial M_k} & \text{se } w_{k,n} \in W \\ 0 & \text{c.c.} \end{cases} + \sum\limits_{\substack{n \in \mathrm{Pa}(j) \\ n\text{: produto}}} \dfrac{\partial M}{\partial M_n} \prod\limits_{k \in \mathrm{Ch}(n) \setminus \{j\}} M_k$

Tabela 1 As derivadas parciais da SPN em função de um nó j no caso geral.

Método	**Derivada parcial do peso**
Soft	$\dfrac{\partial S}{\partial w_{n,j}} = S_j \dfrac{\partial S}{\partial S_n}$
Hard	$\dfrac{\partial M}{\partial w_{n,j}} = M_j \dfrac{\partial M}{\partial M_n}$

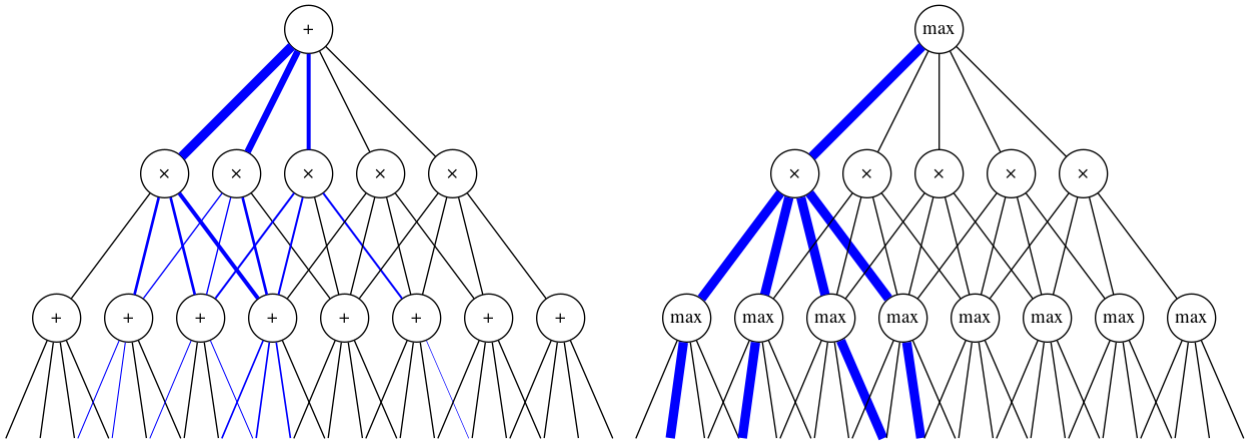
Tabela 2 As derivadas parciais da SPN em função da aresta $n \to j$.

Todas as derivadas são em função de certa evidência. Portanto a notação $\partial S / \partial S_j$ equivale a $\partial S / \partial S_j(X)$, onde X é algum conjunto de valorações válidas para as variáveis de S. Para computar o gradiente, deriva-se a log-verossimilhança da distribuição da SPN em função do conjunto de parâmetros w, ou seja, $\frac{\partial}{\partial w} \log \mathrm{P}(X)$, obtendo os resultados da Tabela 3.

Método	Gradientes
Soft	$\Delta w_{n,j} = \eta \dfrac{\partial S}{\partial w_{n,j}}$
Hard	$\Delta w_{n,j} = \eta \dfrac{c_{n,j}}{w_{n,j}}$

Tabela 3 As atualizações dos pesos da SPN a partir do gradiente para cada aresta $n \to j$.

Onde $c_{n,j}$ denota a contagem de vezes que $w_{n,j}$ pertence a W. No caso do método *soft* de gradiente descendente, a atualização do peso depende da derivada parcial $\partial S/\partial w_{n,j}$, enquanto que na versão *hard*, $\Delta w_{n,j}$ apenas depende do número de passagens pela aresta. É portanto possível perceber o porquê do método *soft* ter problemas com difusão de gradiente e *hard* não. Testes feitos com todos os conjuntos de dados usados mostraram que difusão de gradiente ocorreu em todos eles, o que mostra que, em casos de mundo real é necessário o uso do método de otimização de gradiente descendente *hard*.



(a) Derivação *soft* (b) Derivação *hard*

Figura 2: O gradiente no método *soft* tende a se diluir conforme a altura da rede. Usando a derivação *hard*, o gradiente não depende da derivada parcial diluída.

Para a implementação e replicação dos resultados do artigo de Poon-Domingos, foi necessário implementar o algoritmo de geração de estrutura densa descrito no artigo. O código escrito para este algoritmo é restrito ao domínio de imagens, no entanto, a ideia por trás da arquitetura pode ser aplicada a qualquer domínio em que existe uma forte relação de dependência entre variáveis locais [14].

A estrutura densa é formulada a partir da ideia de que existe uma relação de semelhança entre os pixels de uma região retangular de uma imagem, e uma relação de independência entre outras regiões retangulares que não a contém. Cada região retangular é representada

por um conjunto de m nós somas. Para cada região retangular, decompõe-se a região em duas partições de forma que as partições também sejam regiões retangulares da imagem. Na SPN, representa-se esta decomposição por um conjunto de nós produtos.

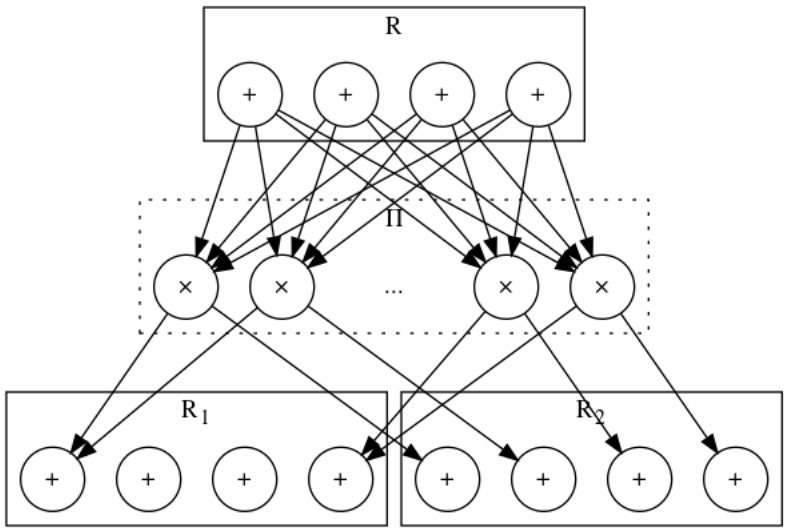


Figura 3: Decomposição de uma região R em duas subregiões R_1 e R_2. O conjunto Π de nós produtos representa a relação de independência entre as duas subregiões.

Quando R representa a própria imagem, ou seja, a região retangular que engloba todos os pixels da imagem, então o conjunto de nós somas contém apenas um nó soma. Para regiões unitárias em que a região é apenas um pixel, ao invés de somas, geram-se g gaussianas de tal forma que cada k gaussiana é o k-ésimo quantil da distribuição dos valores do pixel. Quando a região tem dimensões menores que $r \times r$, a chamamos de região fina. As regiões maiores ou iguais a $r \times r$ são chamadas de regiões grosseiras. Para as regiões grosseiras tomam-se apenas partições que geram subregiões maiores ou iguais a $r \times r$, enquanto que para regiões finas, geram-se partições para todas as possíveis subregiões retangulares. As arestas conectando uma região R ao conjunto de produtos Π são geradas de tal forma que todo nó soma de R está conectada a todo nó produto de Π. Porém, as arestas ligando cada produto em Π a cada nó soma nas subregiões R_1 e R_2 podem ou não existir. Para decidir quais arestas devem existir, para cada instância I do conjunto de treinamento, escolhe-se uma ligação $\pi_k \rightarrow \sigma_j^i$, onde $\pi_k \in \Pi$ e $\sigma_j^i \in R_j$, tal que esta aresta seja a mais provável dado I.

No código disponibilizado[3] pelos autores do artigo, usa-se apenas *hard* Expectation-Maximization (EM) para gerar as arestas mais prováveis e decidir seus pesos, apesar do artigo dar resultados para o método de otimização de gradiente descendente. Com o intuito de gerar resultados mais semelhantes ao artigo possível, foi implementada a parte de se gerar as arestas mais prováveis por *hard* EM, porém a ponderação das arestas foi feita a partir do método de gradiente.

O algoritmo de Dennis-Ventura segue uma ideia similar ao de Poon-Domingos, bus-

[3]http://spn.cs.washington.edu/spn/downloadspn.php

cando achar relações de semelhança e independência entre regiões da imagem. No entanto, generaliza-se a ideia de região para qualquer conjunto de pixels similares, mesmo que não sejam localmente próximas, e podendo tomar formas arbitrárias. Para isso, usa-se métodos de clustering para achar tais regiões.

Seja D o conjunto de dados, onde I é uma instância de D. I então descreve um conjunto ordenado de valorações dos pixels (ou seja, uma imagem) indexado pelos pixels. É possível visualizar D como uma matriz $n \times m$, onde n é o número de imagens do conjunto de dados e m é o número de pixels na imagem. A transposta de $D^\intercal$ é portanto a matriz $m \times n$, onde as colunas são as imagens e as linhas as valorações de um mesmo pixel.

Para o algoritmo de Dennis-Ventura, gera-se inicialmente um "grafo de regiões" que descreve a SPN por meio de um esboço da arquitetura final. Um nó no grafo de regiões é um nó região ou um nó partição. A raíz do grafo é sempre a imagem inteira. O grafo de regiões é gerado particionando-se, recursivamente, cada região em duas subregiões. Seja R um nó região e $D_R^\intercal$ o conjunto de dados transposto contendo apenas variáveis no escopo de R, particiona-se R em duas subregiões R_1 e R_2 usando 2-means clustering em $D_R^\intercal$. O resultado deste clustering são os dois conjuntos de dados transpostos $D_{R_1}^\intercal$ e $D_{R_2}^\intercal$. Cria-se um nó partição P e liga-se $R \to P$, $P \to R_1$ e $P \to R_2$. Em seguida, faz-se a recorrência em R_1 e R_2 se, para cada região, seu escopo for maior que um. Para o nó raíz, repetem-se estes passos k vezes, onde a cada passo i, usa-se o i-ésimo cluster de um k-cluster em D (não transposto).

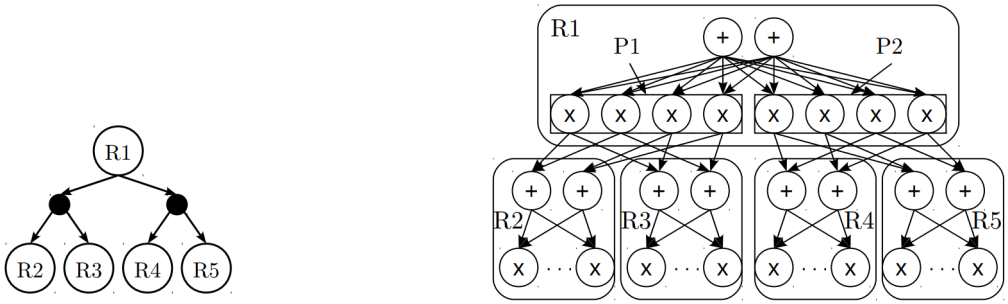


Figura 4: A arquitetura de Dennis-Ventura gera um grafo de regiões (a) que representa um esboço da arquitetura real da rede soma-produto (b). Fonte: [4]

Após gerado o grafo de regiões, faz-se uma transformação para uma rede soma-produto válida. Cada nó região é transformado em um conjunto de m nós somas. Se a região representa apenas um pixel, então geram-se g nós gaussianas, onde cada q-ésima gaussiana é o q-quantil do pixel. Para um nó partição P, sejam R_1 e R_2 as duas subregiões filhas de P, e sejam $|R_1|$ e $|R_2|$ o número de nós em cada filho do nó partição. São então criados $|R_1||R_2|$ nós produtos, onde cada aresta de P é uma combinação dois a dois dos nós R_1^i e R_2^j. O resultado desta transformação é a arquitetura de Dennis-Ventura. Após criada a arquitetura, aprende-se os pesos através do método de otimização de gradiente descendente.

Para o algoritmo de Gens e Domingos, o artigo original [10] não explicíta o algoritmo em

si. Ao invés disso, é dado um esquema de algoritmo que garante uma estrutura profunda. Este esquema é chamado de `LearnSPN`, e segue a ideia de que nós produtos representam independência entre as variáveis envolvidas, enquanto que nós somas representam relações de semelhança entre as instâncias do conjunto de dado.

`LearnSPN`: Algoritmo de Gens-Domingos

Entrada Conjuntos I de instâncias e X de variáveis
Saída SPN representando a distribuição de probabilidade de I sobre as variáveis X

se $|X| = 1$ **então**
    retorna distribuição de probabilidade univariada de I
senão
    Particione X em $P_1, P_2, \ldots, P_m$ tal que todo P_i é independente de P_j, $i \neq j$
    se $m > 1$ **então**
        $\pi \leftarrow$`Produto`()
        para todo $i \leftarrow 1, \ldots, m$ **faça**
            $p_i \leftarrow$`LearnSPN`(I, P_i)
            π`.AdicionaFilho`(p_i)
        retorna π
    senão
        Faça *clustering* em I tal que $Q_1, Q_2, \ldots, Q_n$ são os clusters de I
        $\sigma \leftarrow$`Soma`()
        para todo $i \leftarrow 1, \ldots, n$ **faça**
            $s_i \leftarrow$`LearnSPN`(Q_i, X)
            $w \leftarrow |Q_i|/|I|$
            σ`.AdicionaFilho`(s_i, w) ▷ w é o peso da aresta $\sigma \rightarrow s_i$
        retorna σ

Para o algoritmo Gens-Domingos, separa-se o conjunto de dados D em dois conjuntos I e X de instâncias (por exemplo, imagens) e variáveis (ou seja, o escopo). O algoritmo é recursivo e pode ser dividido em três partes: folha, produto e soma.

Para definir uma folha, apenas verifica-se se o conjunto de dados tem apenas uma variável como escopo. Neste caso, retorna-se a distribuição de probabilidade univariada de I. Foram implementadas duas versões desta parte. A primeira retorna como folha uma distribuição multinomial com as frequências dos valores presentes em I. A segunda retorna uma distribuição gaussiana com média e variância calculadas a partir de I.

Em seguida, o algoritmo tenta construir um nó produto. Tomando-se I e X, tenta-se particionar X em $P_1, P_2, \ldots, P_m$ partições, onde toda variável $Y \in P_i$ e $Z \in P_j$ obedece $Y \perp Z$ se e somente se $i \neq j$, onde $\perp$ representa independência. Foram implementados os testes de χ^2 e G-test para estatisticamente determinar se duas variáveis são independentes. No entanto, testar cada variável par-a-par é intratável. Para resolver este problema, foi gerado um grafo de independência, onde um vértice representa uma variável e uma aresta

indica dependência entre duas variáveis. Achar as partições de X equivale a achar todas as componentes conexas do grafo. Melhor do que isso, é fácil ver que é suficiente achar as árvores geradoras mínimas de cada componente. Para isso, foi implementado o algoritmo de Kruskal para árvores geradoras mínimas. As partições $P_1, \ldots, P_m$ equivalem às componentes conexas do grafo. Se $m > 1$, então foram encontradas relações de independência suficientes para construir um nó produto. Este nó produto tem como filhos as chamadas recursivas de `LearnSPN` onde os escopos foram reduzidos às variáveis de sua respectiva partição.

Caso $m = 1$, então o grafo é conexo. Neste caso, gera-se uma soma. Faz-se o clustering de I em $Q_1, Q_2, \ldots, Q_n$ clusters. Foram implementadas versões que usam k-means, k-mode e k-median para clusterização com número de clusters fixos. O algoritmo DBSCAN também foi implementado. Este método de clustering baseado em densidade acha o número de clusters automaticamente. Assim que os Q_i clusters foram determinados, retorna-se um nó soma cujos filhos são as chamadas recursivas com conjunto de instância restritas aos Q_i. Adicionalmente, o peso das arestas do nó soma são as proporções dos tamanhos dos clusters.

Os testes de desempenho dos três algoritmos cobriram cinco conjuntos de dados.

(a) Caltech-101 [5]

(b) Olivetti Faces Dataset [15]

(c) Digits [8]

(d) DigitsX [9]

(e) MNIST [11]

O conjunto de dados Caltech-101 original[4] contém imagens de 101 categorias de objetos. As imagens têm diferentes dimensões e resoluções. Para os testes de desempenho do projeto, as imagens foram padronizadas e redimensionadas para imagens em escala de cinza com dimensões 150×65 e quantizadas para uma resolução de 4-bits. Adicionalmente, foram escolhidas apenas as categorias moto, carro e rosto para classificação, as mesmas utilizadas para os testes do artigo original [14].

Figura 5: Uma amostra do conjunto de dados Caltech-101.

As imagens do Olivetti Faces Dataset[5] foram redimensionadas para imagens de tamanho 46×56 com resolução 3-bits em escala de cinza. Este conjunto de dados contém 10 fotos de rostos com ângulos e expressões diferentes de 40 pessoas diferentes.

[4]Disponível em `http://www.vision.caltech.edu/Image_Datasets/Caltech101/`.

[5]Disponível em `http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html`

Figura 6: Uma amostra do conjunto de dados Olivetti. A imagem do lado esquerdo é uma amostra antes da redução de resolução. A imagem do lado direito mostra as imagens com resolução de 3-bits.

O par de conjuntos de dados Digits[6] e DigitsX[7] contém imagens de dígitos de zero a nove. Apenas um tipo de caligrafia foi usado, e portanto as imagens são semelhantes e têm baixa variância entre imagens de mesmo rótulo. Digits contém apenas imagens binárias (preto ou branco), onde 1 indica cor branca e 0 indica cor preta. O conjunto DigitsX é uma variação de Digits, onde cada imagem presente no conjunto original é transformada numa imagem de 8-bits. Além disso, é aplicado um filtro de embaçamento gaussiano (gaussian blur) para gerar maior variância e complexidade ao conjunto. São 70 imagens para cada dígito, dando um total de 700 imagens para cada conjunto. Todas imagens de Digits e DigitsX têm dimensões 20×30.

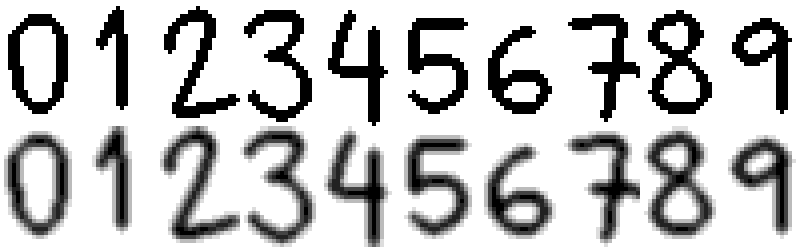


Figura 7: Amostras de cada dígito dos conjuntos Digits e DigitsX. A amostra do topo pertence ao conjunto Digits, onde cada pixel tem valor binário preto ou branco. A de baixo é a transformação das mesmas imagens de cima para o formato DigitsX.

MNIST é um conjunto de dados frequentemente usado para testar o desempenho de classificadores. Assim como Digits, MNIST é um conjunto de dígitos de zero a nove escritos a mão. O conjunto original contém 60000 imagens de treino e 10000 de teste, com aproximadamente 250 tipos de caligrafia no conjunto de treino e 250 no de treino. Cada imagem tem dimensão 28×28 e tem valores binários. Foram escolhidas 2000 imagens de treino e 2000 de teste aleatoriamente para os testes.

[6]Disponível em `https://github.com/RenatoGeh/datasets/tree/master/digits`.
[7]Disponível em `https://github.com/RenatoGeh/datasets/tree/master/digits_x`.

Figura 8: Amostra de dígitos do conjunto MNIST. Ao contrário de Digits, os dígitos são escritos com valores representando branco, enquanto que o fundo é colorido com preto.

Foram feitos testes de classificação e compleição em todos os conjuntos de dados. O algoritmo de Gens-Domingos admite tarefas de classificação e compleição sem alterar sua arquitetura, já que sua estrutura independe do domínio das variáveis. Para os algoritmos de Poon-Domingos e Dennis-Ventura, as redes geradas pelos algoritmos descritos nos artigos representam uma imagem, e adicionar informação externa, como uma variável representando rótulos de classificação, requer modificar a estrutura da rede. Para a tarefa de classificação nestes dois métodos de aprendizado, foi gerada uma estrutura onde cada classe contém uma sub-SPN treinada a partir das instâncias do rótulo correspondente. Variáveis indicadoras da variável de classificação "ligam" e "desligam" as sub-SPNs conforme a valoração da classificação.

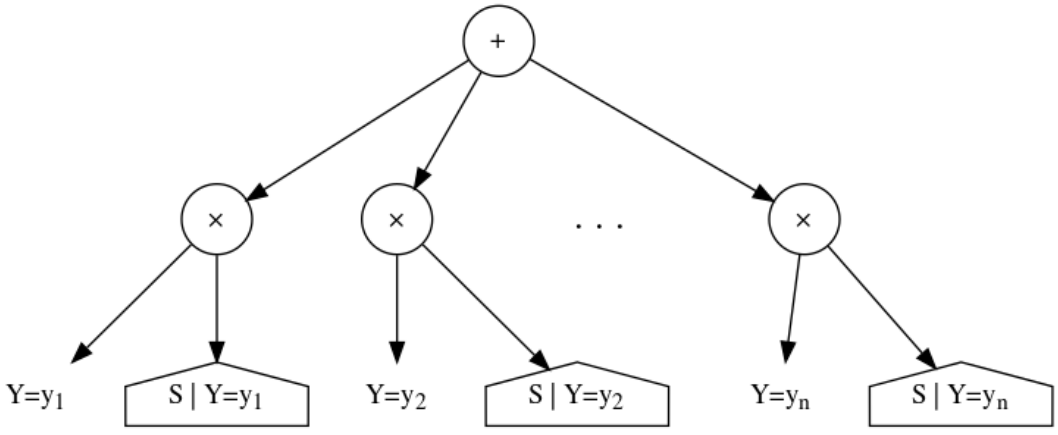


Figura 9: A arquitetura de classificação para os algoritmos de Poon-Domingos e Dennis-Ventura. Para cada valoração da variável de classificação Y, toma-se uma variável indicadora Y_i. O valor do nó produto ligado a Y_i toma um valor 0 se $Y \neq y_i$, e $S|Y_i$ caso contrário. $S|Y = y_i$ representa uma SPN treinada com dados onde Y teve valor y_i.

Para a tarefa de classificação, foram usadas as arquiteturas descritas acima. Para compleição, notou-se que a estrutura de classificação gerou melhores resultados para o algoritmo Dennis-Ventura. No entanto, para o algoritmo Poon-Domingos, esta arquitetura ocupou

mais memória do que o limite de 16GB da máquina usada para os testes. Portanto, para a tarefa de compleição com o algoritmo Poon-Domingos, foi usada a estrutura descrita no artigo original.

No artigo original de Hoifung Poon e Pedro Domingos [14], os testes feitos foram executados em paralelo em um cluster de computadores onde cada máquina tinha 8 CPUs Intel Xeon 2.3GHz e 16GB de memória. Para os testes feitos neste projeto, foi usado apenas uma máquina com 4 CPUs Intel i7–4500U 1.8GHz e 16GM de RAM. Enquanto que no artigo original durou-se 6 minutos para completar o aprendizado do conjunto de dados Caltech-101 para todas as categorias, os testes feitos neste projeto com o algoritmo de Poon-Domingos excederam mais de 24 horas para apenas três categorias. Além disso, enquanto que o artigo original usa 16 somas por região e 8 gaussianas por pixel, por limitação de memória foi necessário usar apenas 4 somas por região e 4 gaussianas por pixel nos testes deste projeto. Devido a esta discrepância em *hardware* e tempo de execução, não foi possível apresentar resultados para todos os testes usando a arquitetura de Poon-Domingos, e quando foi possível, os resultados foram abaixo do esperado. O código do algoritmo de Poon-Domingos foi fortemente baseado no código existente escrito em Java disponibilizado pelos autores do artigo. É possível que existam erros no código, porém é também possível que os resultados ruins e longos tempos de execução se devam à limitação de hardware e ao fato de ter sido necessário usar parâmetros bem menores. Será assinalado por $*$ nas tabelas de teste quando algum algoritmo excedeu o tempo limite de 24 horas no teste. Para os casos em que o algoritmo gastou mais de 16GB de memória, será usado o símbolo $**$ na tabela de testes.

Em questão de tempo de execução, o algoritmo de Gens-Domingos foi o mais rápido comparado aos outros em conjuntos de dados com imagems pequenas. No entanto, sua complexidade aumenta significantemente quando existe um número grande de categorias por variável ou quando a imagem é grande. Isto ocorre pois o passo de determinar as partições de variáveis independentes ocupou 90% do tempo de execução do algoritmo, e os testes estatísticos de independência dependem fortemente do número de categorias por variável e tamanho do escopo dos dados. Por meio da quantização das imagens (de 8-bit para 3 ou 4-bit), o tempo de execução diminuiu significantemente. Em comparação, o algoritmo de Dennis-Ventura manteve-se mais constante em relação ao conjunto de dados, com resultados melhores em tempo de execução ao de Gens-Domingos nos testes de Caltech-101 e Olivetti. Como o método usa otimização de gradiente para determinar os pesos, o tempo de execução dependeu mais fortemente do número de instâncias. O número de categorias para cada variável não teve impacto no tempo de execução. O método de aprendizado de Poon-Domingos teve o maior tempo de execução em todos os testes. Como cada iteração do algoritmo demora cerca de 15 minutos e todos conjuntos de dados contém mais de 300 instâncias, não foi possível extrair resultados de todos os conjuntos para o algoritmo de Poon-Domingos.

As tabelas de acurácia mostram a porcentagem de acertos que cada algoritmo apresentou no conjunto. A linha "Partição p" indica a porcentagem p do conjunto de dados usado para treino e $1-p$ para teste. Portanto, se $p = 0.7$ no conjunto DigitsX, serão usadas 490 imagens para treino e os 210 restantes para teste. Cada partição tem número uniforme de imagens

para cada rótulo de classificação. Para o conjunto MNIST, como existem conjuntos de treino e teste separados, não houve partição. Ao invés disso, foram feitas classificações in-sample, onde faz-se a união do conjunto de treino e teste e em seguida toma-se aleatoriamente metade desta união como treino e o restante como teste, e classificação out-sample, onde toma-se apenas o conjunto de treino para treino e o de teste para teste. Todos os valores de acurácia são em porcentagem de acerto.

DigitsX

Partição p	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Poon-Domingos	10.0	10.0	*	*	*	*	*	*	*
Dennis-Ventura	92.85	98.57	99.18	98.81	99.42	99.28	98.57	93.33	88.75
Gens-Domingos	91.27	96.78	96.93	98.09	97.14	97.85	97.61	92.66	86.25

Caltech-101

Partição p	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Poon-Domingos	**	**	**	**	**	**	**	**	**
Dennis-Ventura	78.58	78.49	80.28	79.88	81.38	81.35	75.45	74.78	75.75
Gens-Domingos	77.40	85.00	84.28	86.11	88.66	90.00	92.22	90.00	84.84

Olivetti

Partição p	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Poon-Domingos	10.0	**	**	**	**	**	**	**	**
Dennis-Ventura	83.78	74.88	89.85	89.93	96.22	97.50	92.89	50.00	60.93
Gens-Domingos	2.50	2.50	93.92	91.25	95.50	98.75	81.93	81.59	100.0

MNIST
(2000 treino / 2000 teste)

Classificação	Poon-Domingos	Dennis-Ventura	Gens-Domingos
In-sample	*	77.85	81.55
Out-sample	*	69.90	76.90

No quesito acurácia na tarefa de classificação, o algoritmo de Gens-Domingos teve os melhores resultados em média. Em seguida, Dennis-Ventura alcançou, em média, resultados comparáveis ao primeiro algoritmo. Por último, o de Poon-Domingos apresentou os piores resultados. Observou-se que Gens-Domingos tem melhores resultados quando existe um grande número de dados para treino. O algoritmo de Dennis-Ventura conseguiu bons resultados com poucos dados. Conjectura-se que, como Dennis-Ventura tem uma estrutura voltada para domínios parecidos com o de imagens, é esperado que tenha bons resultados com poucos dados. Enquanto isso, o algoritmo de Gens-Domingos tem aplicações mais gerais, além de ser um método de aprendizado estrutural, o que requer mais informação para gerar uma estrutura mais informativa e que capture melhor as interações entre variáveis.

Para os testes de compleição de imagem, dois parâmetros foram definidos. O parâmetro de conhecimento de classe (CC) indica se o teste de compleição admite que a SPN tenha conhecimento da classe a ser completada. Por exemplo, no conjunto Olivetti existem 40

classes, onde cada classe corresponde a uma pessoa. Cada classe contém 10 instâncias de rostos. Com conhecimento de classe, o teste de compleição para uma instância I de uma classe c no conjunto D será avaliado com uma SPN treinada com o conjunto de dados $D \setminus \{I\}$, ou seja, a SPN terá conhecimento do rosto da pessoa cuja imagem será completada. Sem conhecimento a priori de classe o teste é feito com uma SPN treinada com o conjunto $D \setminus \{I\}, \forall I$ tal que $I[C] = c$, ou seja, com o conjunto de todas as imagens cujos rostos não pertençam a pessoa c. O segundo parâmetro, chamado de conhecimento de rótulo (CR) indica se, durante a compleição de uma imagem I, foi dada como parte da evidência o rótulo da imagem a ser completada. O parâmetro CC tenta avaliar o quão bem a SPN generaliza os dados quando se é dada informação nunca vista pelo modelo, enquanto que o parâmetro CR avalia se o teste depende da classificação da classe para gerar uma boa compleição.

As compleições foram geradas tomando como evidência metade da imagem e em seguida computando a hipótese mais provável (MPE, de *Most Probable Explanation*) que o modelo assume. A MPE de uma evidência é computada de forma aproximada usando a rede máximo-produto do modelo. O resultado do MPE é um conjunto de valorações das variáveis de todos os pixels da imagem, onde a metade usada como evidência permanece inalterada e a outra metade representa a compleição. Nos testes feitos neste projeto, coloriu-se em escala de verde a compleição feita pela SPN, enquanto que a metade tomada como evidência foi mantida em escala de cinza. Uma linha vermelha de um pixel de largura foi traçada para separar as partições.

Figura 10: Compleição de uma imagem do conjunto de dados Olivetti usando o algoritmo de Dennis-Ventura. A compleição da esquerda usa compleição de classe (CC), enquanto que o da direita não.

Analisando as imagens, foi possível perceber que o parâmetro CR não teve impacto nas compleições, já que na maior parte das vezes o algoritmo já classificava o rótulo correto. O parâmetro CC teve maior impacto. Como a SPN não tem conhecimento de características que apenas uma classe possue, o modelo tentou utilizar características de outras classes, gerando faces desproporcionais, como na Figura 10.

As tarefas de compleição mostraram que a SPN identificou características comuns entre várias classes. Por exemplo, no conjunto Olivetti, a SPN conseguiu reconstruir partes do corpo como nariz, olhos e bigode de forma razoável. Em alguns casos, a rede teve dificuldade com a presença ou ausência de óculos e barba.

Figura 11: Compleição da mesma face da Figura 10 usando o algoritmo Gens-Domingos. A imagem da esquerda possue conhecimento de classe e a da esquerda não.

O algoritmo de Gens-Domingos apresentou melhores resultados na tarefa de compleição. Algumas imagens completadas pelo algoritmo Dennis-Ventura não apresentaram características fortes (como nariz, boca ou olhos) mas tinham o formato certo de rosto e cabelo. Certas compleções parecem ter parte das faces faltando, enquanto que outras possuem mais de dois olhos. A Figura 12 contém algumas das piores compleições.

A boa performance do algoritmo Gens-Domingos mostra que talvez a estrutura da rede seja mais importante do que adequar os pesos do modelo aos dados. Além disso, o algoritmo de Gens-Domingos apresentou resultados bem melhores na compleição comparado ao de Dennis-Ventura, o que mostra que o primeiro algoritmo parece ser melhor em tarefas generativas. Enquanto isso, em tarefas discriminativas como classificação, os dois algoritmos

tiveram resultados semelhantes.

Figura 12: Compleições com características faciais faltando ou fora do padrão. Nestes casos a rede não conseguiu identificar olhos, nariz orelhas ou bocas de forma ideal.

Figura 13: Algumas compleições do algoritmo Gens-Domingos em dez imagens de uma única classe do conjunto de dados Olivetti. A imagem da esquerda foi completada usando o parâmetro CC, enquanto que o da direita não. É possível notar que em alguns casos o modelo corretamente adiciona ou remove a presença de óculos na imagem.

Por questão de tempo e limitação de número de páginas do relatório, foram apenas

mostradas compleições do conjunto de dados Olivetti. No entanto, é possível replicar os testes de classificação e compleição de imagens usando o código presente em `https://github.com/RenatoGeh/benchmarks`. Outras compleições do conjunto Olivetti também podem ser vistas em `https://github.com/RenatoGeh/gospn/tree/dev/results/olivetti_3bit`.

5 Conclusão

Redes soma-produto são modelos probabilísticos baseados em grafos que representam distribuições de probabilidade tratáveis. Neste projeto foram feitas comparações no domínio de classificação e compleição de imagem entre três métodos de aprendizado estado-da-arte de redes soma-produto. Além disso, planejou-se implementar e disponibilizar uma biblioteca livre e gratuita para inferência e aprendizado de redes soma-produto.

Foram implementados três algoritmos de aprendizado. No entanto, apenas dois geraram bons resultados. Devido à limitações de tempo e memória, e possivelmente erros no código, o terceiro algoritmo não rodou de forma ótima. Em tarefas de classificação, os algoritmos que apresentaram resultados mostraram boa acurácia em conjuntos de dados reais e artificiais, como classificação de dígitos manuscritos, reconhecimento de faces e identificação de objetos, mesmo quando os conjuntos de dado continham poucos dados de treino. Na tarefa de compleição, o modelo conseguiu identificar características importantes das imagens, e conseguiu as replicar de forma razoável, mesmo quando as imagens continham informações nunca antes vistas.

Todo o código é aberto e foi implementado e disponibilizado de forma livre e gratuita em `https://github.com/RenatoGeh/gospn` com licensa BSD-3-Clause. A biblioteca inclue documentação, funções para computar inferência, manipular conjuntos de dados e analisar acurácia em classificação e compleição de imagens, além de três algoritmos de aprendizado de redes soma-produto implementados com parâmetros ajustáveis, como diferentes variações para clusterização e testes de independência de variável.

Referências

[1] Mohamed R. Amer e Sinisa Todorovic. "Sum-Product Networks for Activity Recognition". Em: *IEEE Transactions on Pattern Recognition and Machine Intelligence (TPAMI 2015)* (2015).

[2] Wei-Chen Cheng et al. "Language Modelling with Sum-Product Networks". Em: *Annual Conference of the International Speech Communication Association 15 (INTERSPEECH 2014)* (2014).

[3] Olivier Delalleau e Yoshua Bengio. "Shallow vs. Deep Sum-Product Networks". Em: *Advances in Neural Information Processing Systems 24 (NIPS 2011)* (2011).

[4] Aaron Dennis e Dan Ventura. "Learning the Architecture of Sum-Product Networks Using Clustering on Variables". Em: *Advances in Neural Information Processing Systems* 25 (2012).

[5] L. Fei-Fei, R. Fergus e P. Perona. "Learning generative visual models from few training examples: an incremental Bayesian approach tested on 101 object categories". Em: *IEEE CVPR-2014 Workshop on Generative-Model Based Vision* (2014).

[6] Abram L. Friesen e Pedro Domingos. "Recursive Decomposition for Non-convex Optimization". Em: *International Joint Conference on Artificial Intelligence 24 (IJCAI 2015)* (2015).

[7] Abram L. Friesen e Pedro Domingos. "The Sum-Product Theorem: A Foundation for Learning Tractable Models". Em: *International Conference on Machine Learning 33 (ICML 2016)* (2016).

[8] Renato Lui Geh. *Digits Dataset*. URL: `https://github.com/RenatoGeh/datasets/tree/master/digits`.

[9] Renato Lui Geh. *DigitsX: Digits-Expanded Dataset*. URL: `https://github.com/RenatoGeh/datasets/tree/master/digits_x`.

[10] Robert Gens e Pedro Domingos. "Learning the Structure of Sum-Product Networks". Em: *International Conference on Machine Learning* 30 (2013).

[11] Y. LeCunn et al. "Gradient-Based Learning Applied to Document Recognition". Em: *Proceedings of the IEEE* (1998).

[12] Robert Peharz. "Foundations of Sum-Product Networks for Probabilistic Modeling". Tese de dout. Graz University of Technology, 2015.

[13] Robert Peharz et al. "Modeling Speech with Sum-Product Networks: Application to Bandwidth Extension". Em: *IEEE International Conference on Acoustics, Speech, and Signal Processing 39 (ICASSP 2014)* (2014).

[14] Hoifung Poon e Pedro Domingos. "Sum-Product Networks: A New Deep Architecture". Em: *Uncertainty in Artificial Intelligence* 27 (2011).

[15] Ferdinando Samaria e Andy Harter. "Parameterisation of a stochastic model for human face identification". Em: *Proceedings of 2nd IEEE Workshop on Applications of Computer Vision* (1994).