Aprendizagem Supervisionada: Localização de Proteínas em C++ (Tema C2 / Grupo 37)

Bruno Miguel da Silva Barbosa de Sousa (201604145)
MIEIC
FEUP
Porto, Portugal
up201604145@fe.up.pt

Francisco Manuel Canelas Filipe (201604601)
MIEIC
FEUP
Porto, Portugal
up201604601@fe.up.pt

O seguinte artigo está inserido na Unidade Curricular de Inteligência Artificial e aborda o tema Localização de Proteínas (C2) referente a Machine Learning.

Nos próximos parágrafos iremos introduzir um pouco este tipo de problemas, aprofundando com mais detalhe o tema escolhido (Localização de Proteínas). O objetivo deste trabalho é, dados 8 atributos numéricos que ajudam a classificar uma proteína, prever a localização desta.

Desta forma, este artigo não só apresenta e descreve o problema, como também o formula enquanto problema resolúvel utilizando Machine Learning. Também aborda, ainda que numa fase inicial, o seu desenvolvimento.

Machine Learning, C4.5, Redes Neuronais, K-Nearest Neighbor, Localização de Proteínas, C++

I. Introdução

Este artigo aborda o tema de Resolução de Problemas através de Métodos de Pesquisa, um tema já abordado nesta Unidade Curricular. Como tal, e para demonstrar os conhecimentos adquiridos sobre esta área, desenvolvemos um programa que tem por base este mesmo tópico.

A Resolução de Problemas utilizando mecanismos de Machine Learning é uma área bastante importante da Inteligência Artificial pois foi um ponto de partida para a resolução de problemas onde facilmente se encontravam padrões e tendências difíceis de identificar ao olho humano. Muito facilmente consegue analisar uma enorme quantidade de dados o que aumenta a sua eficiência ao longo do tempo. Este tipo de mecanismo é muito utilizado em tudo o que esteja relacionado com previsões.

Este artigo é uma breve síntese daquilo que será a nossa abordagem em relação ao tema escolhido, onde iremos descrever, formular e explicar a implementação do problema escolhido, enquanto um problema resolúvel utilizando métodos e algoritmos da área de Machine Learning.

II. Descrição do Problema

O objetivo deste projeto é poder treinar um modelo que irá prever qual a localização de uma dada proteína ao receber como dados alguns dos atributos da mesma. Este treino deve ser feito recorrendo a alguns dos algoritmos mais utilizados na área de Machine Learning,

Simplificando bastante o problema é possível dizer que, no seu essencial, o objetivo principal é: a partir de um determinado *input*, prever o *output* desejado. Para isso é preciso analisar os dados fornecidos.

Junto com este tema já foram disponibilizados estes dados com a informação num formato predefinido, facilitando assim o parsing dos dados mais relevantes.

O problema consiste então em analisar toda esta informação, construir um modelo e treiná-lo para que consiga prever, com a maior eficácia possível, a localização de uma proteína.

III. Formulação do Problema

O problema consiste em construir um modelo, através de aprendizagem supervisionada, para a previsão da localização de proteínas.

A tarefa a realizar pelo modelo foi então analisar os atributos dados como *input* e prever um determinado *output* em função destes.

Para treinar o modelo, será usado um *dataset* com 8 entradas referentes a características de uma proteína, sendo elas: mcg, gvh, alm, mit, erl, pox, vac, nuc (todos estes atributos de *input*) e o *output* desejado, isto é, a localização tendo em conta estes atributos.

Estes dados serão divididos em duas partes: treino e teste do modelo. As percentagens para cada uma das componentes não será algo estático. Pode ser variável e ajustado pelo utilizador caso este o pretenda fazer.

Para resolver o problema implementou-se uma estrutura a 3 fases:

- Parsing dos dados: executar a leitura dos dados presentes no *dataset* e guardá-los em estruturas de dados adequadas.
- Implementação dos Algoritmos: desenvolvimento de 3 algoritmos de aprendizagem supervisionada: K-Nearest-Neighbous, C 4.5 e Redes Neuronais.
- Construção do output: Após a execução dos algoritmos, tratar os dados de output para, de uma forma mais clara e concisa, apresentar ao utilizador. Desta forma será possível visualizar mais facilmente a performance do modelo implementado.

IV. Trabalho Relacionado

Nesta secção estão presentes alguns dos trabalhos relacionados com aquele que propomos implementar neste artigo assim como alguns dos trabalhos em que nos baseámos para a implementação dos algoritmos:

- Better Prediction of Protein Cellular Localization Sites with the it k Nearest Neighbors Classifier [1].
- Accurate Classification of Protein Subcellular Localization from High-Throughput Microscopy Images Using Deep Learning [2].
- DeepLoc: prediction of protein subcellular localization using deep learning [3].
- Predicting the Subcellular Localization of Human Proteins Using Machine Learning and Exploratory Data Analysis [4].

V. Implementação do Projeto

O desenvolvimento deste projeto dividiu-se em 3 fases: parsing do *dataset*, implementação dos algoritmos e construção do *output* recebido por estes.

Parsing dos dados

Nesta fase apenas foi implementado um parser que lesse os valores do ficheiro e os guardasse numa estrutura de dados. A estrutura de dados escolhida foi um **std:vector<Protein>**, onde Protein é uma struct implementada no âmbito deste trabalho prático, e que contém toda a informação relevante do ficheiro, desde os 8 atributos de *input* ao resultado correspondente. Em relação ao parsing dos dados em si, este foi relativamente simples pois, na estrutura do ficheiro de dados, todos os dados são separados por espaços.

É necessário ter em consideração que dois dos três algoritmos foram baseados em implementações já desenvolvidas que contém um parser embutido. Desta forma, e para estar a complicar em aspetos não relacionados com o objetivo do trabalho prático, procedeu-se à construção de mais dois ficheiros de dados com a mesma informação do ficheiro original. Estes ficheiros, no entanto, encontram-se no formato desejado para poderem ser utilizados nas implementações referidas.

Implementação dos Algoritmos

K-Nearest-Neighbours

O algoritmo k-nearest neighbor foi implementado de raiz por nós, com ajuda da implementação deste mesmo algoritmo pelo GeeksforGeeks[5], implementação esta para a classificação de pontos num plano.

Para o cálculo da distância entre duas entradas de dados, foi utilizada a distância euclidiana entre cada atributo. Dado que todos os atributos são numéricos e percentagens, não foi necessário normalizar os dados.

Foi desenvolvido um parser para o ficheiro com o *dataset* fornecido. Antes de o algoritmo ler o ficheiro, pergunta ao utilizador que percentagem de dados pretende ser utilizado para treino, sendo o restante utilizado para testar. De notar que este algoritmo não necessita de treino, sendo os dados de treino utilizados para calcular a região em que se encontra cada ponto dos dados de teste.

Após o parsing e divisão dos dados fornecidos é pedido ao utilizador que insira o valor k. Este valor k é o número de vizinho do grupo de treino usado para a avaliação da região de cada dado de teste. Após o input do utilizador, é calculada a classe de cada dado de teste, baseado na classe mais frequente dos k vizinhos (pontos dos dados de teste mais próximos) e compara com a classe esperada.

C 4.5

Na implementação deste algoritmo foi utilizada uma outra implementação já desenvolvida[6]. Esta implementação dispõe de um parser e da construção de uma árvore bastante simples. Contém ainda algumas funções de display da árvore e das previsões efetuadas.

Começando pelo parser, este contém uma estrutura ligeiramente diferente da do *dataset* original. A primeira linha deste parser indica os tipos de variáveis (Discretas ou Contínuas) para cada um dos dados de uma proteína. Em segundo existe uma linha com os nomes das variáveis em si. A partir da terceira linha já só existem os dados do *dataset* original sendo que estes se separam agora por vírgulas. Para não estar a complicar demasiado em aspetos que não os objetivos deste trabalho, efetuou-se uma cópia do *dataset*, e adaptou-se ao formato deste parser.

Ao interpretar o ficheiro, os dados são guardados numa matriz em que cada linha corresponde uma linha do ficheiro. Em seguida, são separados os dados de treino e os dados de teste. Utilizando os dados de treino, é recursivamente calculado o atributo com maior ganho de informação e criada uma subárvore com a divisão dos dados pela decisão sobre este atributo. Caso esse atributo seja discreto então é criado um nó para cada valor possível. Por outro lado, caso o atributo seja contínuo então são criados nós que dividem esse atributo em intervalos.

Para efetuar o teste desta árvore, são percorridos todos os dados de teste e, para cada um deles, é percorrida a árvore até encontrar uma folha da mesma (extremidade da árvore). Quando tal acontecer, o valor dessa folha indica a previsão definida pela árvore.

Redes Neuronais

Para a implementação do algoritmo de redes neuronais, teve-se por base uma implementação já desenvolvida[7]. Esta implementação, tal como a anterior, já dispõe de um parser, e uma implementação de uma rede neuronal bastante simples.

Tendo já o parser disponibilizado, optou-se por criar uma cópia do *dataset* e adaptá-la ao mesmo. A sua estrutura consiste numa primeira linha contendo um conjunto de inteiros. O tamanho deste conjunto representa o número de camadas da rede e o valor de cada um indica

o número de nós em cada camada. As restantes linhas contêm uma label "in:" indicando que a linha em questão contém os valores de input, e uma label "out:" indicando o output correspondente.

Neste projeto apenas existem 3 camadas (input, 1 hidden layer e output). Visto que são disponibilizados 8 inputs, a primeira camada contém 8 nós. A segunda camada contém cerca de 4 nós, apenas porque este valor revelou a melhor performance. Em relação ao número de nós de output são cerca de 10, os mesmos que o número de outputs possíveis.

De modo a conseguir guardar o output desejado em forma numérica desenvolveu-se uma função que convertia o output do ficheiro (string) num vetor de doubles de tamanho 10 (número de outputs possíveis). Cada elemento do vetor corresponde a uma classe de output possível. Este vetor contém todas as classes com valor a 0, exceto uma, o output desejado, que contém o seu valor a 1.

Após a leitura dos dados, estes são guardados e é construída a rede neuronal. A partir deste ponto começa a execução do algoritmo em si. A sua estrutura segue o seguinte formato:

- Primeiro é enviado o input de uma proteína para a primeira camada. Esse input passa então a ser o output da própria camada.
- De seguida é percorrida a rede tendo em conta os pesos atuais que esta contém.
- Ao chegar à última camada cada um dos 10 nós de output vai conter um valor. O valor mais elevado vai corresponder ao índice da classe de output prevista pela rede.
- Depois de converter esse índice para string, é obtida a previsão da rede neuronal.
- Após esta previsão, a rede atualiza os seus valores com valores mais coerentes, de modo a melhorar a sua eficácia e demonstrar alguma aprendizagem.

Seguindo esta estrutura, o esperado é que ao longo do tempo vá melhorando as suas previsões, aprendendo então a classificar a localização das proteínas.

Construção do Output

Para mais facilmente conseguir analisar a performance dos algoritmos implementados foi necessário gerar alguns dados de maior relevância tais como: duração do algoritmo, percentagem de previsões corretas e matriz de confusão.

A duração do algoritmo, tal como o nome indica, apenas refere o tempo que o algoritmo demorou a executar.

A percentagem de previsões corretas é um valor, em percentagem, indicativo da quantidade de vezes que acerta as suas previsões.

Por último, a matriz de confusão é uma matriz de valores reais e valores previstos pelo algoritmo. Neste caso esses valores são os valores da localização das proteínas. Visualmente pode ser descrito como uma tabela com 10 entradas nas linhas e colunas (uma para cada classe de output possível). Desta forma é possível indicar de uma forma intuitiva o que foi previsto pelo algoritmo e o que era esperado. Para além dos dados básicos de acerto/falha da previsão do modelo em relação ao output, são mostradas também, para cada classe do output, a taxa de acerto, precisão, sensibilidade e medida f do modelo.

VI. Experiências e Resultados

Estando o código do projeto desenvolvido, é necessário então proceder à recolha e análise dos resultados do mesmo. Este passo é bastante importante pois ajuda no estudo da performance de cada um dos algoritmos implementados, para uma futura comparação entre estes.

Começando pelo algoritmo **K-Nearest-Neighbours** foram feitos dois estudos:

- **Influência dos valores de K nas previsões do algoritmo**

Como é possível verificar pelo gráfico abaixo, o aumento do valor de K vem acompanhado de um aumento do número de previsões corretas por parte deste algoritmo. Este comportamento justifica-se pela diminuição do número de *outliers.* Ao aumentar o valor de K a probabilidade de um outlier persistir diminui, conduzindo a uma menor taxa de erro por parte do algoritmo e, consequentemente, uma maior taxa de acerto. Por outro lado, caso o K seja bastante elevado, o modelo tenderá a prever a classe com mais entradas no *dataset*. Dado que o *dataset* utilizado tem a distribuição de outputs muito pouco uniforme (~60% das entradas pertencem a 2 das 10 classes possíveis), isso é algo que afeta bastante o nosso modelo.

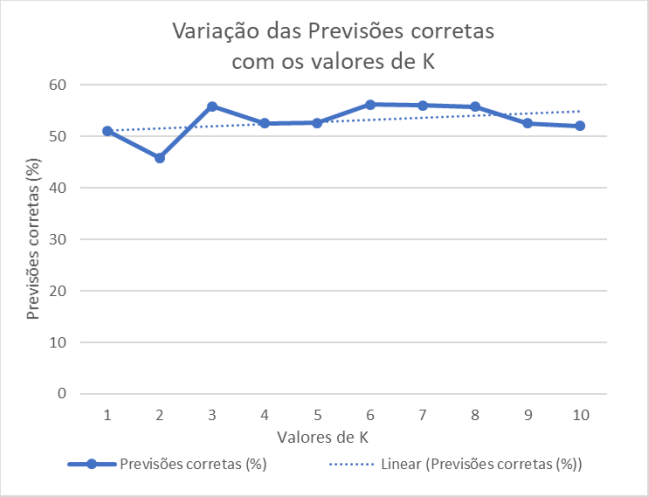

Fig1. Análise K-Nearest-Neighbours-1

- **Influência da quantidade de dados nas previsões do algoritmo**

Analisando o gráfico abaixo é possível verificar que a percentagem de dados utilizados é diretamente proporcional com a percentagem de acerto deste algoritmo. Uma maior percentagem de dados consegue mais facilmente estabelecer os limites entre classes de forma mais precisa, o que é representado por este aumento, quase linear, na percentagem de previsões corretas.

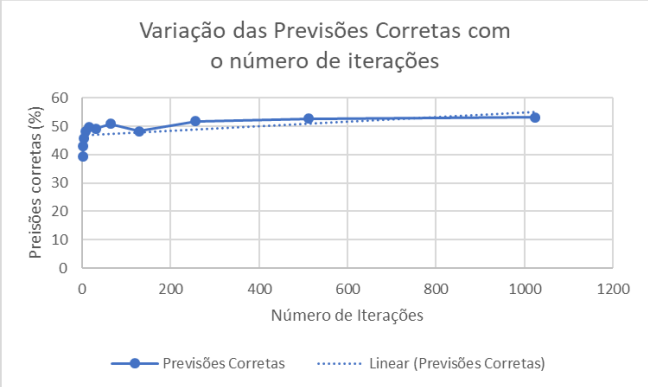

Fig2. Análise K-Nearest-Neighbours-2

Por outro lado, não conseguimos efetuar estudos em relação à utilização do algoritmo **C4.5**, dado que a implantação que utilizamos deste algoritmo era bastante deficiente, não permitindo a utilização de mais de ~100 dados de treino.

Para a utilização deste algoritmo foi necessário decidir, para cada atributo, se era continua ou discreta. Após uma primeira análise dos dados, pensamos que seriam todos atributos contínuos visto que são todos percentagens, tirando o resultado, que seria discreto. Contudo, com uma análise mais profunda percebemos que os atributos erl e pox tinham um número bastante limitado de valores distintos e decidimos classificar estes atributos como discretos.

Para além do problema com o número de dados de treino, encontramos outro problema com a implementação que foi o facto de, por vezes, não conseguir prever nenhum output.

Com todos os problemas já referidos e apenas podendo construir uma árvore de decisão baseada em ~7% do *dataset*, os resultados não foram, como seria de esperar, os melhores, tendo apenas o modelo construído uma taxa de acerto de cerca de 42%. Dado que não foi possível variar a quantidade de dados de treinos, não nos é possível avaliar a sua influência na taxa de acerto do modelo.

No entanto, este algoritmo permite a visualização da árvore de decisão, o que ajuda a perceber os fatores que levam à localização da proteína. Assim sendo, apresentamos aqui a árvore de decisão obtida pelo nosso programa. Dado que esta é bastante complexa, é apresentada também como anexo, para permitir uma melhor análise desta.

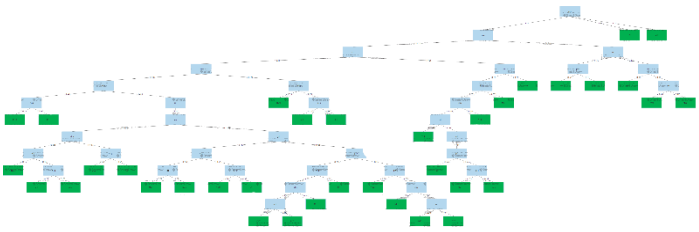

Fig3. Árvore de Decisão do Algoritmo C 4.5

Quanto ao algoritmo utilizado na implementação de **Redes Neuronais** efetuou-se uma análise em relação ao número de iterações efetuadas e à quantidade de dados de treino.

De acordo com os nossos testes, para este algoritmo foi possível atingir os melhores resultados ao utilizar um valor de 0.25 para o learning rate e 0.1 para o momentum.

- **Influência do número de iterações nas previsões do algoritmo**

Nesta análise é possível perceber que, inicialmente, o aumento do número de previsões corretas é bastante significativo com o aumento do número de iterações. No entanto, a partir das 16/32 iterações esse valor torna-se relativamente constante, estagnando por volta dos 53/54%.

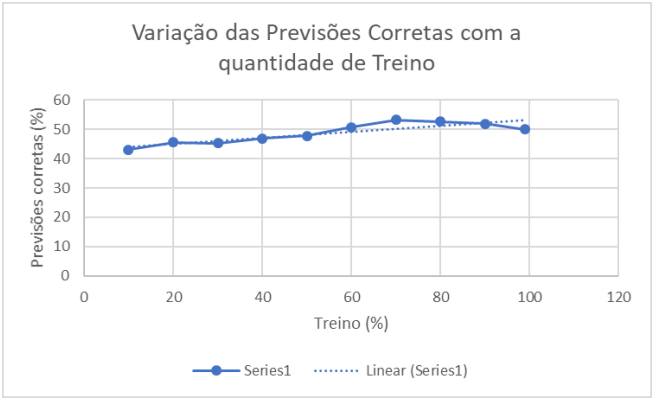

Fig4. Análise Rede-Neuronal-1

- **Influência da quantidade de dados de treino nas previsões do algoritmo**

Pela análise do gráfico abaixo é possível verificar que a tendência é aumentar a performance com o aumento de dados de treino. Apesar de não ser uma diferença bastante considerável é possível verificar que existe uma melhoria, pois quantos mais dados forem estudados, mais completo se torna o treino da rede neuronal, diminuindo assim a probabilidade de erro desta.

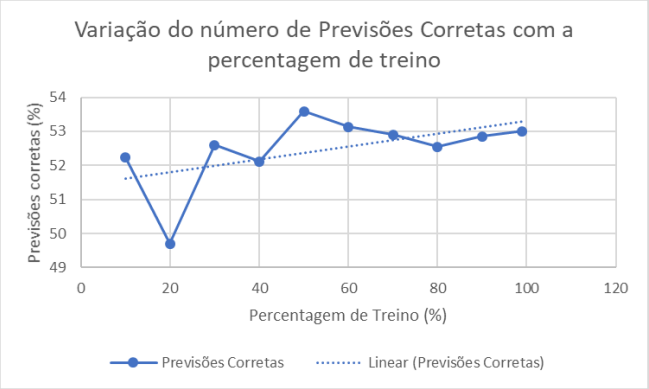

Fig5. Análise Rede-Neuronal-3

Como é possível verificar, em geral, estes valores de previsões corretas não atingem os 60%, o que não é o ideal. No entanto, este problema não está relacionado com a implementação dos algoritmos, mas sim com os dados fornecidos. Visto que estes dados não são uniformes, existem classes de *output* muito mais frequentes que outras. Desta forma, não é possível construir um modelo confiável na previsão das classes com menor taxa de ocorrência nos dados de entrada.

VII. Conclusões e Perspetivas de Desenvolvimento

Em suma, o projeto foi desenvolvido com sucesso tendo sido atingidos todos os objetivos propostos, desde a implementação do número de algoritmos requerido ao correto funcionamento destes.

No entanto, é possível verificar algumas melhorias a realizar futuramente.

A primeira delas seria a utilização de um *dataset* mais completo visto que este não abrange uniformemente todas as opções de output. Isto resulta numa má eficácia de aprendizagem dos dados, não oferecendo assim as previsões mais desejáveis. Para além disso, e apesar de não termos certezas disso, pela análise dos nossos resultados e de outros estudos que utilizaram o mesmo *dataset*, pensamos que não existe uma correlação muito alta entre os atributos de *input* e a classe de *output*.

Ainda uma outra melhoria a ser realizada poderia ser feita ao nível do algoritmo C4.5 de modo a conseguir suportar uma maior quantidade de dados e, consequentemente, construir uma árvore de decisão mais robusta, obtendo assim melhores resultados com este algoritmo.

Referências Bibliográficas

[1] Paul Horton, Kenta Naka, “Better Prediction of Protein Cellular Localization Sites with the it k Nearest Neighbors Classifier”, 1997, consulted on May 2019.

[2] Tanel Pärnamaa, Leopold Parts, “Accurate Classification of Protein Subcellular Localization from High-Throughput Microscopy Images Using Deep Learning”, published on April 2018.

[3] Almagro Armenteros, Sønderby. C.K, Sønderby. S.K, Nielsen. H, Winther. O, “DeepLoc: prediction of protein subcellular localization using deep learning.”, published on November 2017.

[4] George. K. Acquaah-Mensah, Sonia. M. Leach, Chittibabu Guda, “Predicting the Subcellular Localization of Human Proteins Using Machine Learning and Exploratory Data Analysis.”, published on Agust 2006.

[5] GeeksForGeeks. K-Nearest Neighbours. Retrieved from https://www.geeksforgeeks.org/k-nearest-neighbours/ consulted on May 2019

[6] zgyao (github name), “DecisionTree_C4.5”, last updated on June 2015, available at: https://github.com/zgyao/DecisionTree_C4.5/commits/master

[7] Zehao Huang, “Simple Neural Network”, last updated on October 2015, available at: https://github.com/huangzehao/SimpleNeuralNetwork