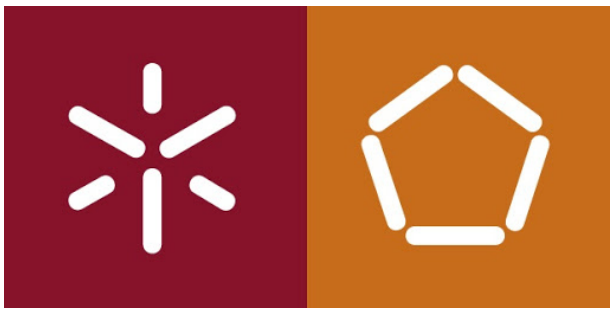


Aprendizagem Automática II
Trabalho Prático
Grupo 9

Daniel Regado (PG42577)

Orientador: Vítor Pereira

Conteúdo

1 | Introdução

Este relatório servirá de apoio ao conteúdo desenvolvido e exposto no *notebook* associado a este trabalho prático. Devido a esse *notebook* já se encontrar devidamente documentado e ser bastante descritivo, este documento irá focar-se mais em explicar de forma geral as decisões tomadas e exposição de principais resultados.

Este trabalho prático tem como objetivo a decisão de *nodes* para processamento de vários serviços, para um dado pedido, num dado estado de rede. Para tal, irão ser exploradas diversas abordagens, de forma a verificar qual destas tem maior qualidade na previsão. Nos próximos capítulos, serão expostos os modelos testados e desenvolvidos, assim como os resultados obtidos. Esta análise tem por base pedidos efetuados sobre a rede *Abilene*, composta por 10 *nodes* e 27 *links*, sendo que nem todos estes *nodes* fazem processamento de serviços.

Figura 1.1: Rede Abilene

2 | Dataset

De forma a não duplicar o conteúdo exposto no *notebook*, esta secção irá apenas salientar os aspetos mais importantes da análise de dados, assim como fazer uma breve descrição do *dataset*.
O *dataset* contém informações relativas a pedidos efetuados na rede exposta na Figura 1.1. Cada pedido, tem a seguinte informação:

- id : Identificador de pedido
- src : *Node* de origem
- dst : *Node* de destino
- bw : Largura de banda pedida
- duration : Tempo necessário para completar o *request*
- S[0-4] : Lista ordenada com os serviços pedidos (-1 representa ignorar serviço)
- MLU : Utilização máxima de um *link* (conexão entre *nodes*)
- MNU : Utilização máxima de um *node*
- E[0-27] : Utilização de um *link*
- N[0-10] : Utilização de um *node*
- RS[0-4] : *Node* sobre o qual cada pedido descrito em S[0-4] irá ser tratado

Foi verificado que nenhum campo tem valores nulos, e que existe uma distribuição equilibrada entre valores de *nodes* de origem e chegada, serviços pedidos e número de serviços por pedido. Por essas razões, o tratamento de dados tornou-se mais simples, passando apenas pela normalização dos valores de largura de banda, codificação das variáveis categóricas e eliminação de colunas que não contribuíam ou prejudicavam a qualidade das previsões.

3 | Modelos de *Machine Learning*

Nesta secção iremos abordar os modelos de *machine learning* que foram testados, assim como explicar como o desenvolvimentos destes foi evoluindo, visto que trata-se de uma secção consideravelmente extensa do *notebook*. Por fim, os melhores resultados serão expostos e posteriormente comparados, de forma a perceber qual o modelo mais adequado para previsões, seguindo esta abordagem.

3.1 Modelos *multi-output*

Estes modelos representam o ponto de partida, no ponto de vista de exploração de modelos de previsão. Estes modelos foram escolhidos numa fase inicial devido a não precisarem de uma grande reestruturação do *dataset*, visto que como foi indicado, são modelos que fornecem a previsão para os cinco serviços simultaneamente. Dos modelos testados, o que forneceu melhores resultados foi o *RandomForestClassifier*, com uma *accuracy* de 41.64%.

Figura 3.1: Resultados para RandomForestClassifier com multi-output

Sendo um modelo multi-output, o valor de *accuracy* considera uma previsão correta apenas se todos os valores de output estiverem corretos, ou seja, caso todos os *nodes* de processamento tenham sido previdos correta-

mente, sendo que caso um destes esta incorreto, todo o output é considerado incorreto. Este comportamento não é desejado, pelo que analisando a tabela *Model Accuracies* na Figura 3.1, conseguimos observar que a *accuracy* acumulada, ou seja, analisando serviço a serviço, está mais perto dos 80%. No entanto, este valor continua a não ser representativo da qualidade da previsão, visto que existe uma maioria dos serviços com valor -1, sobre os quais não são necessárias previsões. De forma a combater todos estes problemas descritos, o *dataset* foi transformado, como iremos ver seguidamente.

3.2 Modelos *single-output*

Estes modelos são identificados como *single-output* de forma a contrastar com os modelos expostos anteriormente. Tratam-se de modelos de classificação, sendo que estão preparados para a classificação numa única variável de interesse, neste caso o *node* de processamento (RS).

3.2.1 Transformação do *dataset*

De modo a treinar estes modelos, foi necessário um processamento adicional do *dataset*, sendo aplicado um tipo *flat map* aos dados existentes. O objetivo principal deste processamento foi a extração das variáveis comuns relativas ao estado da rede, juntamente com cada tuplo S_i e RS_i, $i \in [0, 4]$. Desta forma, já seria possível também excluir serviços com valor -1. Adicionalmente, foram adicionadas as variáveis $\#\_N_i, i \in \{2, 4, 7, 8, 9, 10\}$ e $\#\_S$. As variáveis $\#\_N_i$ denotam a percentagem de serviços, presentes no pedido, que podem ser tratadas no *node i*. A variável $\#\_S$ denota a percentagem de serviços pedidos, tendo em conta o máximo de cinco serviços (ou seja, 1 no caso de um pedido necessitar dos cinco serviços). Estas variáveis são explicadas com mais algum detalhe no *notebook*, mas de forma sucinta, o objetivo destas variáveis é de minimizar a perca de informação global do pedido, ao ser feita a redução a cada serviço.

3.2.2 Modelos orientados a serviços

Uma abordagem que também foi explorada e detalhada no *notebook*, foi a criação de estruturas (classes) que fizessem a separação do *dataset* por serviço pedido, e treinar um modelo para cada serviço. Ao treinar um modelo para cada serviço individualmente, está-se efetivamente a minimizar a necessidade do modelo de generalizar para todos os serviços, resultando em modelos especificamente preparados para aprender e evoluir de acordo com os detalhes específicos de cada serviço.

3.2.3 Modelos únicos

Naturalmente, foram implementadas também as versões mais tradicionais de previsão, aonde apenas é treinado um modelo. Com isto, conseguiremos comparar resultados, e verificar se realmente existe vantagem na complexidade adicional da separação dos dados e treino de vários modelos.

3.2.4 Melhores resultados

Aqui expostos, estão os melhores resultados para as duas abordagens, orientado a serviços e modelo singular. Em ambos os casos, o melhor modelo foi, mais uma vez, o *RandomForestClassifier*. Observando os valores entre parêntesis retos, que correspondem à *accuracy*, verificamos que os valores em termos globais (Figura 3.3 e *All services* na Figura 3.2) são basicamente iguais, tendo apenas uma diferença de 0.001, provavelmente devido a arredondamentos. Portanto, tendo em conta que os resultados são praticamente idênticos, a divisão dos dados para geração adicional de modelos específicos a serviços não se justifica, pelo menos nos casos observados.

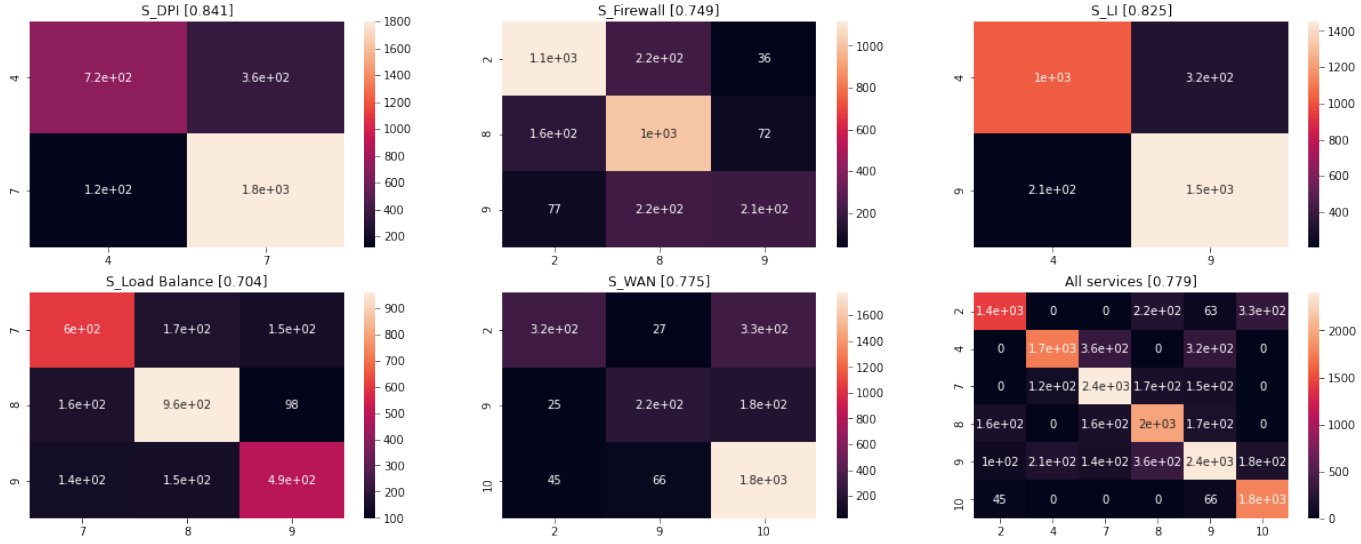


Figura 3.2: Resultados para RandomForestClassifier com um modelo por serviço

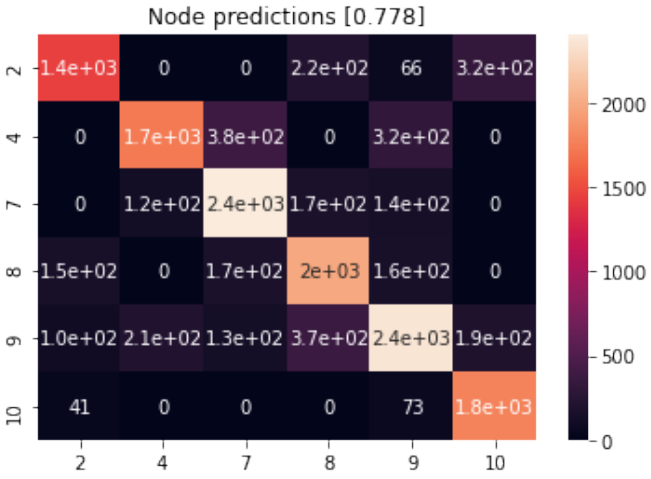


Figura 3.3: Resultados para RandomForestClassifier para previsão de todos os serviços

4 | Deep Learning

Nesta secção, iremos abordar os procedimentos para o desenvolvimento de *Deep Neural Networks* (DNNs), assim como os resultados obtidos. Tal como foi implementado no caso de *machine learning*, também foram implementadas classes para a separação dos dados por serviços, que por sua vez irão treinar um rede para cada serviço. Para o treino destas redes, serão utilizados também os dados transformados, onde cada observação corresponde a um serviço apenas.

4.1 Redes orientadas a serviços

Como referido anteriormente, este objetivo é alcançado recorrendo a uma classe que faz a divisão dos dados para treino de redes diferentes, e faz também a divisão posterior para os casos de testes. Visto que uma abordagem por *deep learning* é a que promete resultados melhores, de forma geral, esta abordagem com treino de várias redes será comparada com uma abordagem com treino de uma rede apenas, analogamente à comparação feita no caso dos modelos de *machine learning*. A abordagem com melhores resultados será a escolhida para aprofundar e desenvolver funcionalidades adicionais.

4.2 Redes únicas

As redes treinadas com esta abordagem utilizam o *dataset* completo sem divisão, em contraste com a abordagem anterior, que é dividido por serviços para treino e posterior previsão. Esta abordagem é mais simples e portanto um pouco mais rápido para treinar, devido a não necessitar de tratamentos intermédios de dados e treino de várias redes.

4.3 Comparação entre abordagens

Foram definidas duas DNNs básicas que foram utilizadas por ambas as abordagens, sendo uma delas com três *hidden layers* e outra com duas *hidden layers*. Assim, podemos comparar resultados, visto que usam a mesma rede para treinar os seus modelos. Iremos expor aqui os resultados correspondentes à rede *Exemplo 2*, que se trata da rede com duas *hidden layers*.

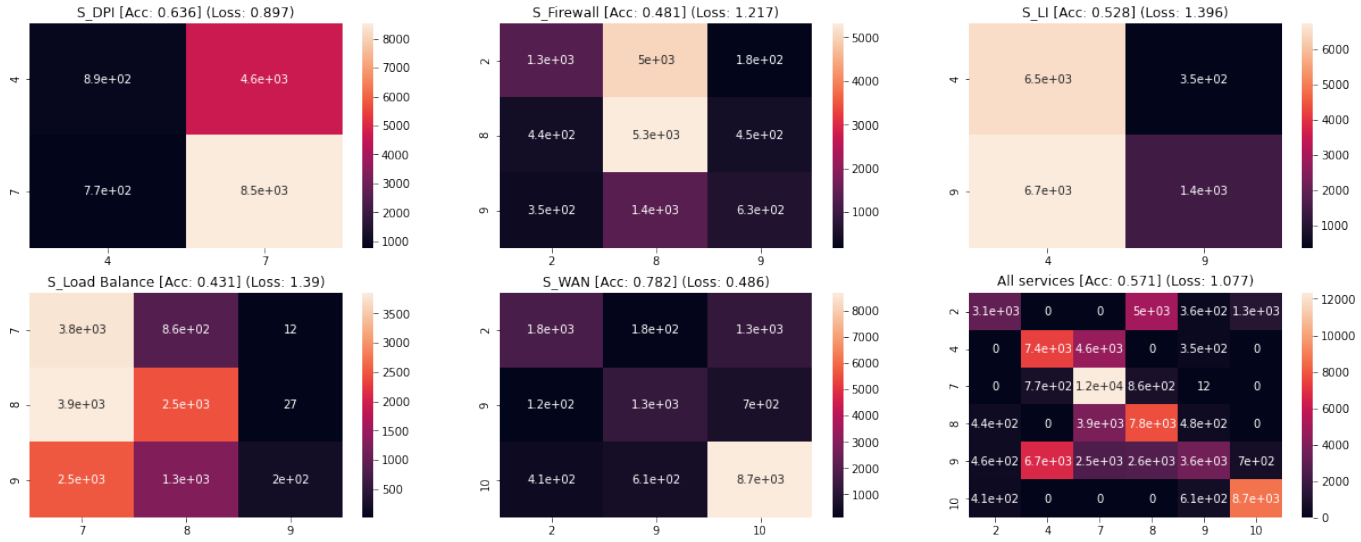


Figura 4.1: Resultados para DNNs orientadas a serviços

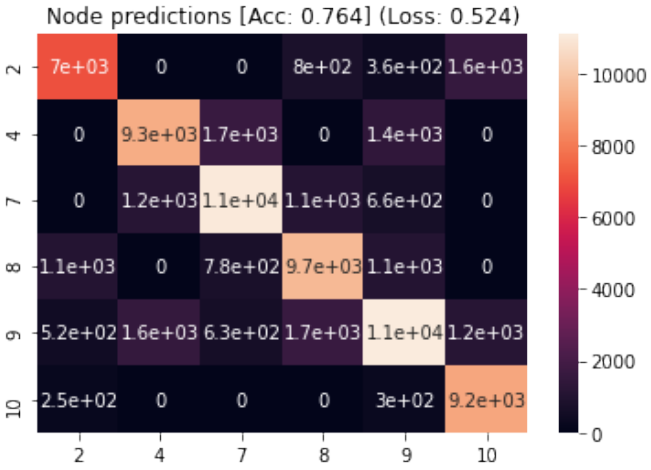
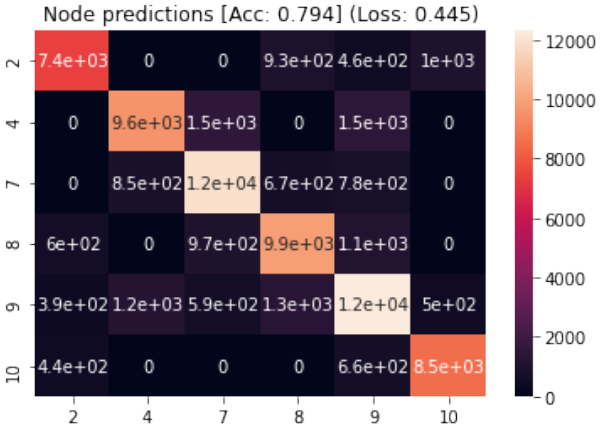


Figura 4.2: Resultados para DNN para previsão de todos os serviços

Para comparar os resultados, devemos comparar a matriz de confusão exposta na Figura 4.2, com a matriz de confusão global da Figura 4.1, com o título *All services*. Neste caso, as diferenças de resultados entre as abordagens já são bastante acentuadas (57.1% vs 76.4% de *accuracy*), sendo que o treino de DNNs separadas por serviço obteve resultados piores. Portanto, tendo em conta estes resultados, foi decidido aprofundar e estender o desenvolvimento do tipo de DNNs treinadas com o *dataset* sem separação.

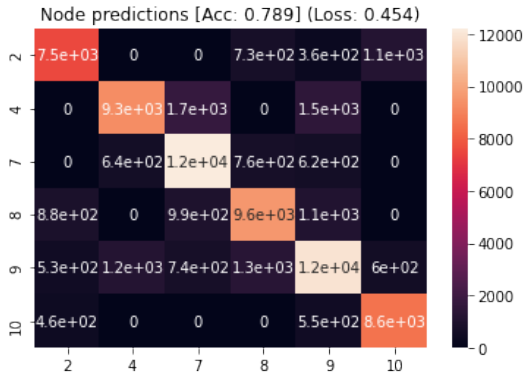
4.4 Otimização de hiper-parâmetros

A implementação de otimização de hiper-parâmetros, recorrendo ao *keras-tuner*, tem como objetivo enriquecer o conteúdo deste trabalho prático, visto que é uma implementação pertinente no âmbito desta UC. É baseado em *random search*, onde é definido um *search space* e são testadas combinações de parâmetros durante algumas *epochs*, fazendo um tipo de teste de rapidez de convergência. Para isso, foram definidas duas redes base sobre as quais serão aplicadas a otimização.



Hyperparameter	Best Value
units_1	416
activation_1	relu
dropout_rate_1	0.2
units_2	32
activation_2	relu
dropout_rate_2	0
units_3	416
activation_3	relu
learning_rate	0.001

Figura 4.3: Resultados de otimização para DNN com 3 *hidden layers*



Hyperparameter	Best Value
units_1	352
activation_1	relu
dropout_rate_1	0
units_2	736
activation_2	relu
learning_rate	0.001

Figura 4.4: Resultados de otimização para DNN com 2 *hidden layers*

Nestes exemplos, cada combinação de parâmetros foi treinada com 4 *epochs*, e foram testadas 15 tentativas diferentes, para cada DNN base. Como podemos observar, foram obtidos melhores resultados para a DNN com 3 *hidden layers*, embora as diferenças não sejam significativas.

5 | Redes recorrentes

Uma das sugestões de implementação, de modo a testar mais abordagens, que possivelmente alcancem melhores resultados, passou pela definição de redes com camadas LSTMs, que tenham em conta a natureza sequencial do *dataset*. Para tal, foi necessário um processamento dedicado dos dados para segmentos, estando este processamento presente no *notebook*.

5.1 Previsão de *nodes* de processamento

De modo a alcançar este objetivo, o *dataset* foi processado para uma instância de *WindowGenerator*, posteriormente fornecido para uma rede desenvolvida com *LSTMs*. Devido ao treino destas redes ser computacionalmente intensivo, foram executadas apenas duas *epochs*, de modo a observar a convergência da rede. Foram obtidos os seguintes resultados:

```
loss: 1.2628 - sparse_categorical_accuracy: 0.4533
```

Portanto, tendo em conta o tempo de treino e a *accuracy* obtida, uma abordagem recorrendo a DNNs será uma escolha mais acertada.

5.2 Previsão de utilização de *nodes*

Após já estarem implementadas as funções de construção de redes recorrentes e a class *WindowGenerator*, foi decidido tentar aplicar estes desenvolvimentos a variáveis que façam mais sentido, como por exemplo o caso dos valores de utilização de *nodes*.

Ao observar a utilização dos *nodes* ao longo do tempo, repara-se num comportamento cíclico, o que parece mais apropriado para a aplicação de redes recorrentes. Após algumas modificações na função para construção da RNN, foram obtidos os seguintes resultados, para a previsão das variáveis *N2*, *N4*, *N7*, *N8*, *N9* e *N10*:

```
loss: 0.1143 - mean_absolute_error: 0.2720
```

Embora não sejam resultados excelentes, talvez sejam possível alcançar resultados melhores com alterações na RNN e mais tempo de treino. No entanto, estando esta exploração foram dos objetivos do trabalho prático, não se justifica a alocação excessiva de recursos para tal.

6 | Conclusão

Após os desenvolvimentos apresentados e descritos no *notebook* e neste relatório, foram abordados vários modelos e arquiteturas de redes neuronais artificiais, de modo a procurar implementações que melhor satisfaçam os objetivos do trabalho prático. A melhor implementação obtida foi baseada numa DNN, e resultou numa *accuracy* de cerca de 80%, treinada com o *dataset* completo e 25 *epochs*. De modo a obter melhores resultados, um procedimento futuro poderá passar pelo uso da função de otimização de hiper-parâmetros associada a um *search space* maior e com DNNs base diferentes, possivelmente com mais *hidden layers*. Tendo em conta o conteúdo desenvolvido e os resultados apresentados, os objetivos deste trabalho prático podem ser considerados como atingidos, conforme o que foi planeado originalmente.