UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA

INF01017 - Aprendizado de Máquina

Prof. Bruno Castro da Silva
Prof. Anderson Rocha Tavares

Relatório do Trabalho Prático

Redes Neurais

Alunos
Aline Weber - **00274720**
Felipe Zorzo - **00261760**
William Vargas - **00274692**

Introdução

O trabalho desenvolvido consiste na implementação de uma rede neural com o algoritmo de *backpropagation*, bem como a validação numérica do gradiente da rede e os opcionais de vetorização, mini-batch e método do momento. A implementação foi avaliada utilizando a técnica de validação cruzada estratificada.

Implementação

A implementação do algoritmo foi realizada com Python 3, usando as bibliotecas *pandas* (para leitura e filtragem dos dados), *numpy* (para facilitar alguns cálculos) e *sklearn* (para realizar a normalização de atributos).

Estruturas de dados

Rede neural: representada pela classe NeuralNetwork. É a classe única do trabalho, contendo a rede neural em si e os métodos aplicados sobre ela. Seus principais atributos são:

- *thetas:* lista de matrizes que contém os pesos das conexões entre os neurônios da rede;
- *struct:* dicionário Python usado para armazenar o valor do parâmetro *lambda* para regularização e a quantidade de neurônios existente em cada camada da rede, sem contar os neurônios de *bias*;
- *a:* valor de ativação de cada neurônio da rede após a propagação de uma entrada até a saída;
- *deltas:* valores usados no *backpropagation* que representam a culpa de cada neurônio de cada camada no erro da rede;
- *layers:* quantidade de camadas da rede;
- *alpha:* parâmetro para atualização dos pesos *theta* da rede, quanto maior o *alpha*, mais vai ser considerado o gradiente atual em detrimento do histórico dos pesos *theta*;
- *beta*: parâmetro usado no método do momento para determinar a direção da variação do gradiente.
- *max_iteration*: quantidade máxima de iterações, usada como critério de parada para o *backpropagation*;
- *batch_size*: tamanho dos *mini-batches* usados no treinamento;
- *epsilon*: valor pequeno usado para alterar os pesos da rede durante a verificação numérica do gradiente.

Nessa classe também estão implementadas as funções para a execução do *backpropagation*, validação numérica do gradiente, cálculo da função de custo e funções auxiliares para fornecer a saída no formato esperado, definido na especificação do trabalho.

Implementação da rede

A rede foi implementada de forma vetorizada, ou seja, tanto os pesos da rede quanto os valores delta, as ativações dos neurônios e a entrada são representados por matrizes ou vetores. Desse modo a propagação de uma entrada na rede ocorre pela multiplicação das ativações da camada anterior acrescidas do valor do neurônio de *bias*, com a matriz dos pesos da rede para essa camada. Essa multiplicação é seguida pela aplicação da função sigmóide. Assim como para a propagação, o *backpropagation* também usa vetorização para representação dos valores *delta* e dos gradientes. A condição de parada para o algoritmo de *backpropagation* é o número de iterações no conjunto de treinamento.

Implementamos o treinamento em *mini-batch*, permitindo a parametrização da quantidade de exemplos considerada em cada atualização dos pesos da rede. Também implementamos a heurística do método do momento, para aceleração da convergência. O parâmetro *beta*, que indica o quanto considerar o histórico dos pesos da rede durante a atualização, foi fixado como 0.9. Ainda, antes de o conjunto de treinamento ser passado para a rede, ele é normalizado utilizando o MinMaxScaler da biblioteca *sklearn*, que transforma os dados para o intervalo [0,1].

Regularização

A regularização usa o valor do parâmetro *lambda*, presente na estrutura da rede, para adicionar custos aos pesos da rede, com objetivo de evitar *overfitting*. Durante o *backpropagation* e durante a verificação numérica a regularização é calculada sobre os pesos atuais da rede e esses valores são considerados nos

gradientes para a atualização da rede. A regularização também é usada no cálculo da função de custo para considerar os valores *thetas* da rede.

Validação cruzada

Dado um conjunto de dados de entrada, o primeiro passo do programa é dividi-lo nos *folds* a serem utilizados na validação cruzada. Como essa separação deve ser estratificada (manter a proporção de classes do conjunto original em cada um dos grupos), a técnica adotada foi: 1) separar os dados pelas suas classes; 2) dividir as instâncias de cada classe em *k* grupos, podendo sobrar instâncias sem grupo caso a divisão não seja inteira; 3) pegar um grupo de cada classe e juntá-los novamente, formando um dos *k-folds* agora com instâncias de todas as classes; e 4) distribuir as instâncias "resto", uma em cada *fold*. Esta implementação está na função *cross_validation* no arquivo *main*.

Com os *k-folds* construídos, ocorre a validação cruzada de fato, onde treinamos e avaliamos *k* redes neurais diferentes. A cada rede, *k-1 folds* são agrupados e formam o conjunto de treinamento, que é passado para o método de backpropagation da rede, e 1 *fold* é utilizado para o teste, de onde vêm os dados para o cálculo da *F-measure* de cada rede.

Após classificar cada instância do *fold* de teste, obtém-se uma matriz onde encontram-se a classificação verdadeira da instância (no formato *one-hot encode*) e a que foi predita pelo modelo sendo avaliado. A partir dessas informações, é possível contabilizar os dados que compõem a matriz de confusão e então calcular as métricas de *precision* e *recall*, que são necessárias para o cálculo da *F-measure*. Esta implementação está na função *Fmeasure* no arquivo *main*.

Instruções para rodar o programa

Para executar o programa na funcionalidade de avaliação do *backpropagation*, deve-se chamar o arquivo *main.py*, utilizando os seguintes parâmetros:

-d: recebe como argumento o caminho para o arquivo que contém o conjunto de dados;

-n: recebe como argumento o caminho para o arquivo contendo a estrutura da rede neural. Essa estrutura deve conter na primeira linha o valor do parâmetro *lambda* para a regularização e em seguida a quantidade de neurônios em cada camada da rede;

-w: recebe como argumento o caminho para o arquivo contendo os pesos iniciais da rede neural;

-v: se recebe True executa a verificação numérica do gradiente.

Exemplo:

python main.py -n examples/network.txt -d examples/dataset.txt -w examples/initial_weights.txt

Esse comando irá criar um arquivo com os gradientes finais do *backpropagation* na pasta *output*. Caso se deseje visualizar a verificação numérica dos gradientes, basta adicionar o argumento ***-v True*** ao comando anterior: um outro arquivo, com os gradientes calculados com a verificação numérica, será criado na pasta *output*. O nome dos arquivos começa com a data de criação da rede (logo, para uma mesma execução, tanto o *backpropagation* quanto a verificação numérica possuem o mesmo prefixo no nome do arquivo) e termina com "_backpropagation.txt" ou "_numerical_verification.txt", de acordo com qual função calculou os gradientes.

Para rodar o programa com os *datasets* fornecidos, o argumento **-n** não deve mais conter o número de neurônios da primeira camada, que agora é calculado automaticamente. Além disso, para esses *datasets* pode-se usar o argumento **-i NUM** para definir o número de iterações do *backpropagation* a serem rodadas, com o valor padrão sendo 1.

Com base nisso, as linhas de comando adequadas para criar a rede neural com os *datasets* fornecidos são (sem informar os pesos iniciais da rede nem o caminho completo para o *dataset*):

- **wine:** python main.py -n datasets/wine.txt -d wine -i 100
- **pima:** python main.py -n datasets/pima.txt -d pima -p 8 -i 100
- **iono:** python main.py -n datasets/iono.txt -d iono -p 34 -i 100

- **wdbc:** python main.py -n datasets/wdbc.txt -d wdbc -p 1 -i 100 --drop_column 0

O argumento “--drop_column” é usado para ignorar colunas do conjunto de dados; para o conjunto wdbc é ignorada a coluna dos IDs das instâncias. No entanto, não é obrigatório para o funcionamento da rede, uma vez que o treinamento deve aproximar os pesos da entrada do ID para zero. Ainda assim, a remoção dessa coluna facilita o treinamento da rede, por isso mantém-se a funcionalidade. É possível variar o arquivo com a estrutura da rede (argumento -n, com arquivo no formato descrito acima), mas o argumento -p deve ser mantido dessa forma pois é o que indica qual a coluna do *dataset* que possui a informação a ser predita.

Avaliação dos *datasets*

Após a verificação da corretude dos algoritmos implementados, a partir dos dados de teste fornecidos, iniciamos o treinamento e ajustes de parâmetros para os quatro conjuntos de dados indicados.

Inicialmente, variamos o número de neurônios em apenas uma camada, sendo redes com 1, 3, 5, 8, 10, 15, 20, 30, 40 e 50 neurônios. Começando os testes pelos dados **Wine**, os parâmetros utilizados foram *mini-batches* de tamanho 16, *lambda = 0.25* e *alpha = 0.8*. O treinamento ocorreu por 50 iterações. Os resultados podem ser visualizados no gráfico da Figura 1. Depois, testamos essa mesma configuração no **Ionosphere**, entretanto, os resultados não foram os desejados (F-measure variando entre 0.65 e 0.7). Treinamos, então, por mais iterações, até fixarmos o número de iterações em 500, pois percebemos que, apesar de ainda diminuir o custo J, não era significativo o suficiente pelo tempo necessário para cada iteração. Ainda assim, o valor de F-measure ficava em torno de 0.85. Variamos então o fator de regularização (0.25, 0.1 e 0.0), e observamos que o melhor para esse conjunto de dados era *lambda = 0*. Variamos também alpha (0.8, 0.5 e 0.1), e o melhor valor foi *alpha = 0.1*. Como esse conjunto de dados é maior, mudamos o tamanho do *mini-batch* para 32. Dessa forma, os valores finais para os parâmetros foram *mini-batches* de tamanho 32, *lambda = 0.0* e *alpha = 0.1*, em 500 iterações, e os resultados também podem ser visualizados no gráfico da Figura 1. No conjunto

de dados **Pima** foi onde encontramos maior dificuldade para ajustar os parâmetros. Tentamos as mesmas variações que usamos no Wine e no Ionosphere, mas nada teve uma melhora significativa. Os resultados para os parâmetros *mini-batches* de tamanho 32, *lambda = 0.0* e *alpha = 0.1*, em 500 iterações, estão no gráfico da Figura 1. Por fim, testamos também o conjunto de dados **wdbc**, que facilmente atingiu F-measure de 0.93, com *lambda = 0.0*, *alpha = 0.5* e *mini-batches* de tamanho 16, em 25 iterações. Aumentando o número de iterações, não ocorreu melhora significativa no desempenho.

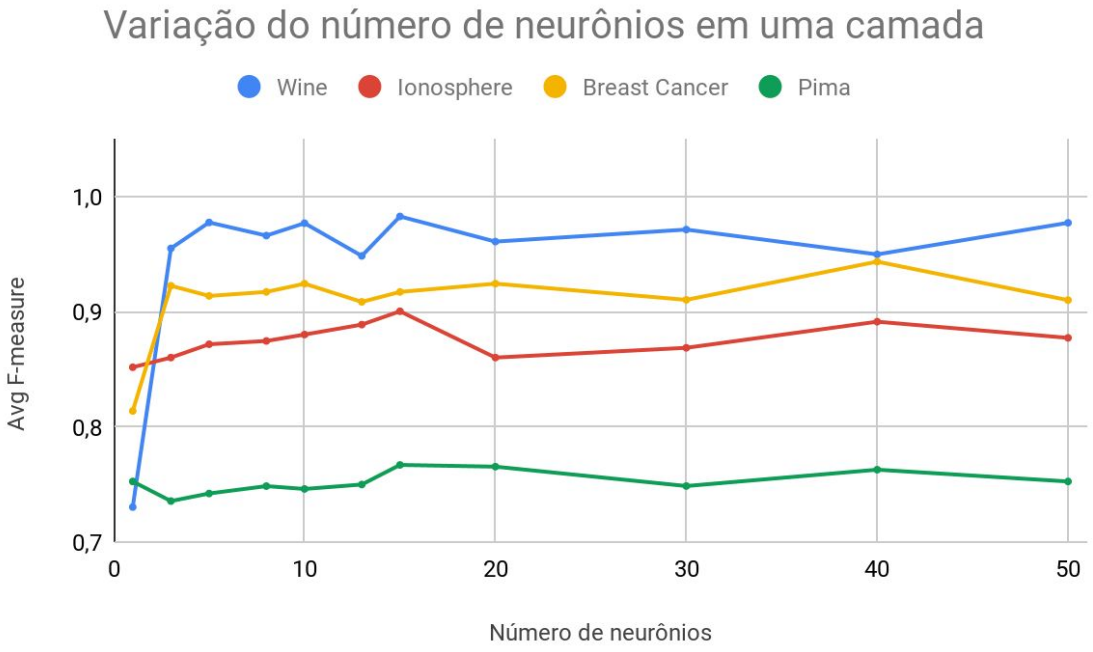


Figura 1: Média dos valores da *F-measure* obtidos com as variações de número de neurônios em uma camada

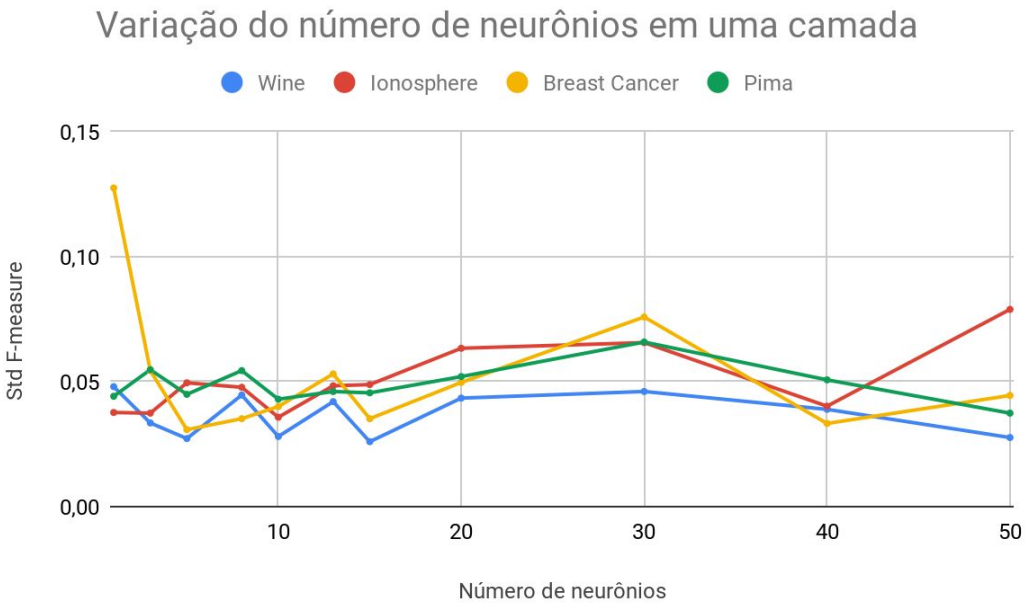


Figura 2: Desvio padrão dos valores da *F-measure* obtidos com as variações de número de neurônios em uma camada

Apesar do **Wine** e **wdbc** terem atingido bom desempenho com apenas uma camada e cinco neurônios, **Pima** e **Ionosphere** ainda poderiam melhorar, portanto testamos também variações no número de camadas. Inicialmente, mantemos fixo o número de neurônios em 5, e testamos com 1, 2, 3 e 5 camadas. Os valores dos outros parâmetros são os mesmos utilizados na avaliação da variação de número de neurônios. Os resultados para os quatro conjuntos de dados podem ser visualizados no gráfico da Figura 3. Conseguimos um melhor resultado para o **Ionosphere**, em comparação com os obtidos no primeiro experimento de variação, com duas camadas de 5 neurônios, mas em geral, adicionar camadas não trouxe melhora significativa na F-measure. Além de precisar mais tempo para o treinamento, também tende a acontecer *overfitting* dos dados em algumas das divisões entre treino e teste (o que pode ser observado dado o desvio padrão maior do que o observado até então em casos como três camadas no **Wine** e **Ionosphere**).

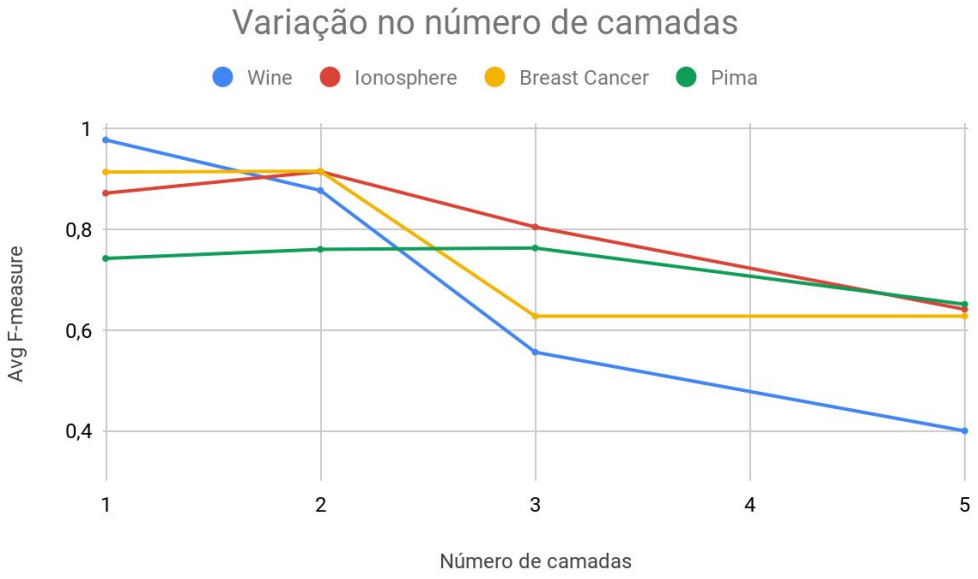


Figura 3: Média dos valores da *F-measure* obtidos com as variações no número de camadas com cinco neurônios em cada

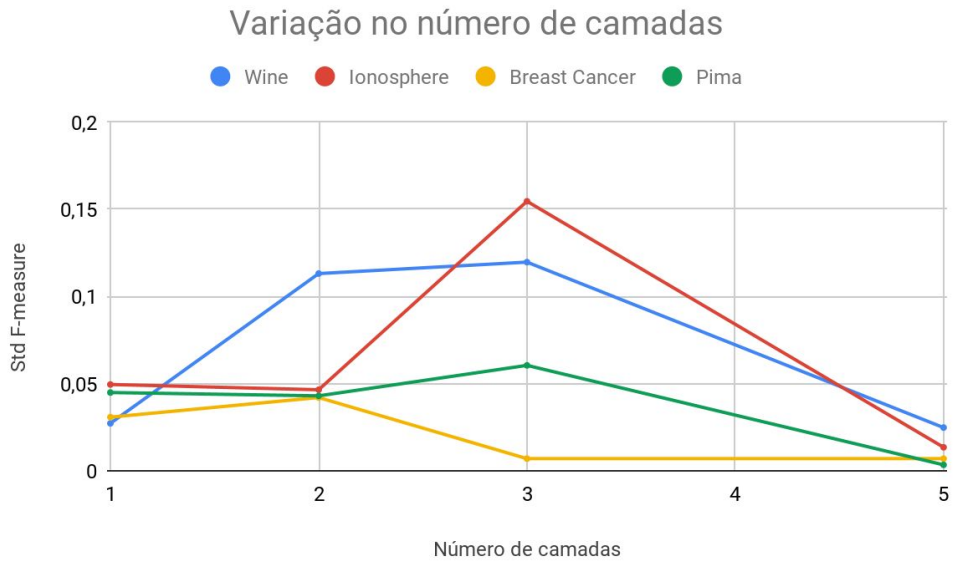


Figura 4: Desvio padrão dos valores da *F-measure* obtidos com as variações no número de camadas com cinco neurônios em cada

Utilizando a melhor configuração encontrada até o momento para cada um dos conjuntos de dados, com exceção do **Pima**, para o qual escolhemos a melhor configuração em relação aos resultados obtidos e o tempo levado para treinar a rede. Fizemos, então, testes variando os parâmetros *lambda* (0.0, 0.1, 0.25 e 0.5) e *alpha* (0.1, 0.3, 0.5 e 0.8). Estes resultados podem ser visualizados nos gráficos das

Figuras 5 e 6. Para o conjunto **Wine**, a rede utilizada foi uma camada com cinco neurônios, com *lambda* fixo em 0.0 (quando variando *alpha*), *alpha* fixo em 0.8 (quando variando *lambda*) e *mini-batches* de tamanho 16, por 50 iterações. Para o conjunto **Pima**, a rede utilizada foi uma camada com oito neurônios, com *lambda* fixo em 0.0 (quando variando *alpha*), *alpha* fixo em 0.1 (quando variando *lambda*) e *mini-batches* de tamanho 32, por 500 iterações. Para o conjunto **Ionosphere**, a rede utilizada foi duas camadas com cinco neurônios em cada, com *lambda* fixo em 0.0 (quando variando *alpha*), *alpha* fixo em 0.1 (quando variando *lambda*) e *mini-batches* de tamanho 32, por 500 iterações. Para o conjunto **wdbc**, a rede utilizada foi uma camada com 40 neurônios, com *lambda* fixo em 0.0 (quando variando *alpha*), *alpha* fixo em 0.5 (quando variando *lambda*) e *mini-batches* de tamanho 16, por 25 iterações. Percebemos o efeito da regularização principalmente no conjunto de dados **Ionosphere**, onde a cada aumento no *lambda*, a qualidade de predição da rede diminuía. Isso nos mostra que todos os pesos da rede de duas camadas são necessários para a melhor predição. O **Wine**, entretanto, foi o *dataset* que sofreu a mudança mais drástica, quando o lambda passou de 0.25 para 0.5. As variações do *alpha,* por sua vez, tiveram efeito significativo apenas para o **Wine**.

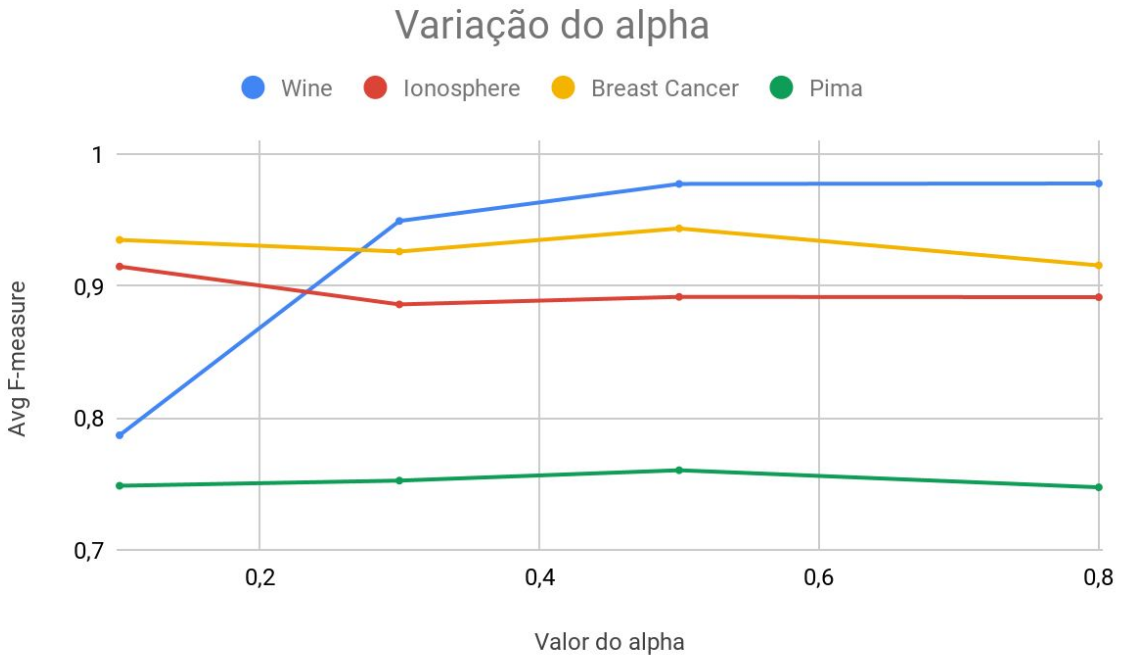


Figura 5: Média dos valores da *F-measure* obtidos com as variações no parâmetro *alpha*

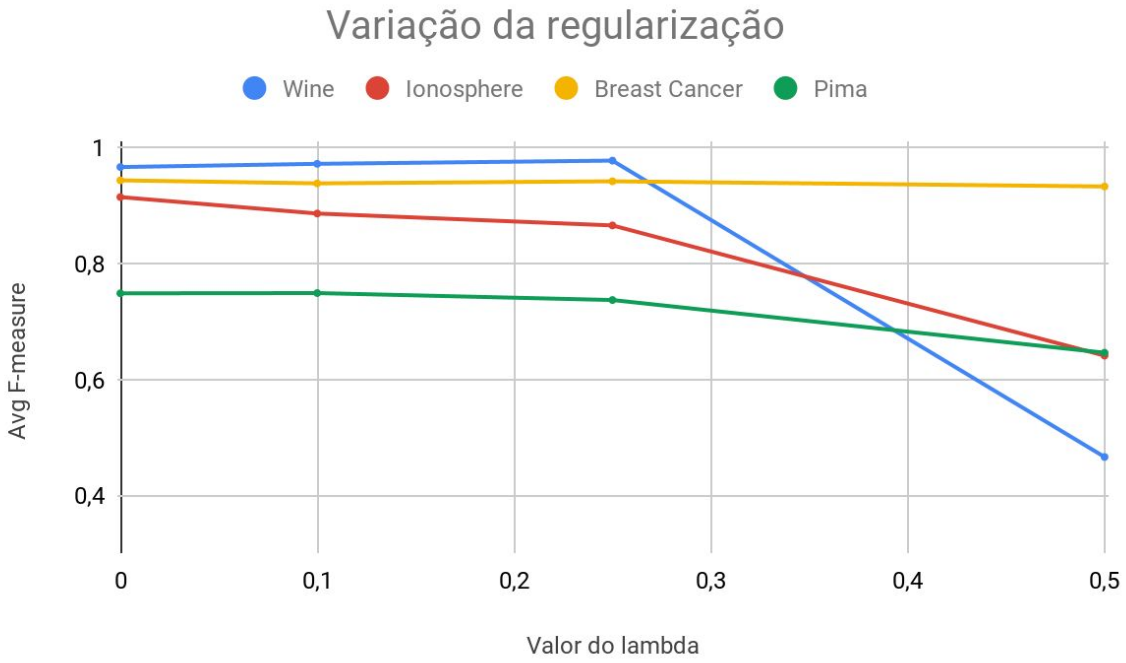
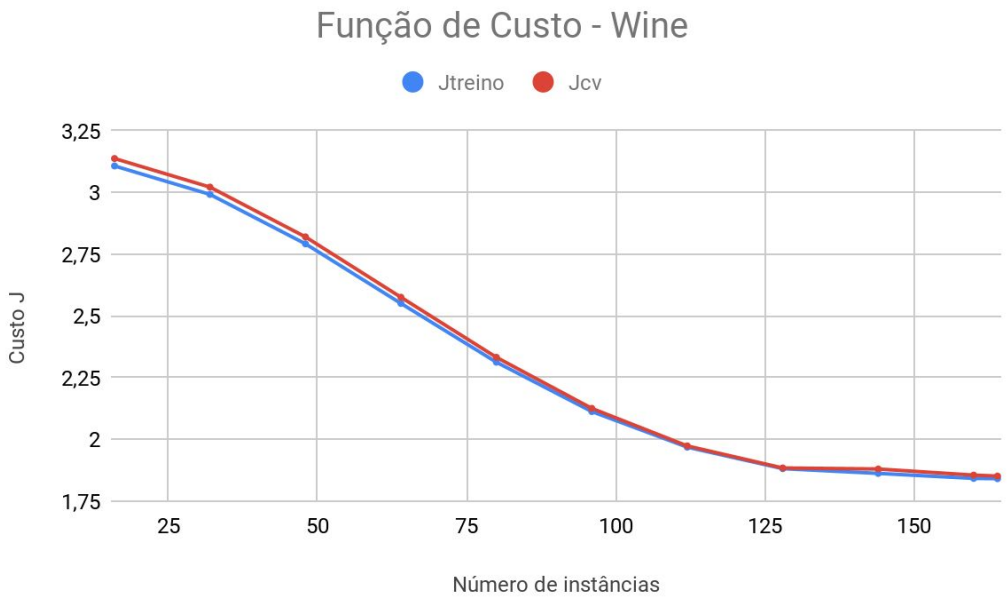


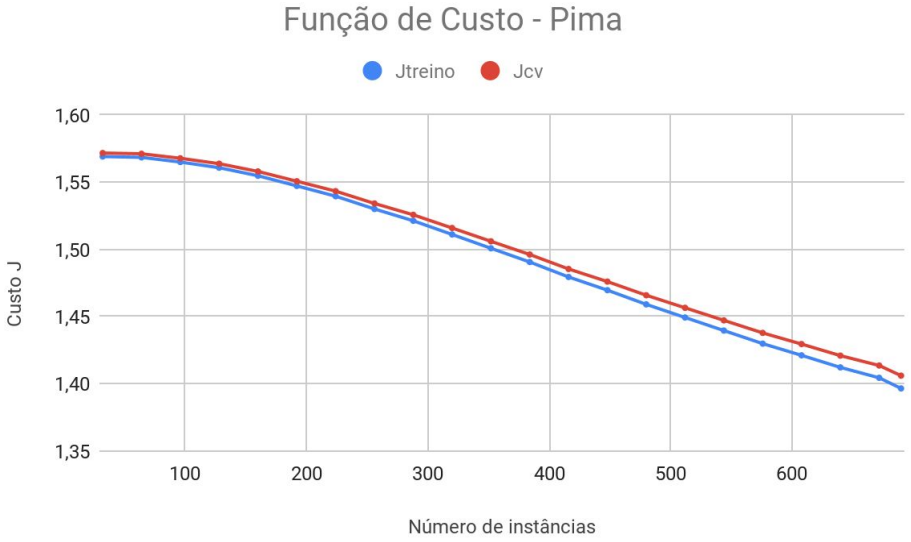
Figura 6: Média dos valores da *F-measure* obtidos com as variações no parâmetro *lambda*

Concluindo, então, utilizamos a melhor arquitetura encontrada para cada conjuntos de dados (novamente com exceção do **Pima**) a fim de observarmos os valores J na primeira iteração do algoritmo de *backpropagation*, onde o conjunto de treino eram nove *folds* e o conjunto de validação era o *fold* restante. Os resultados obtidos foram:

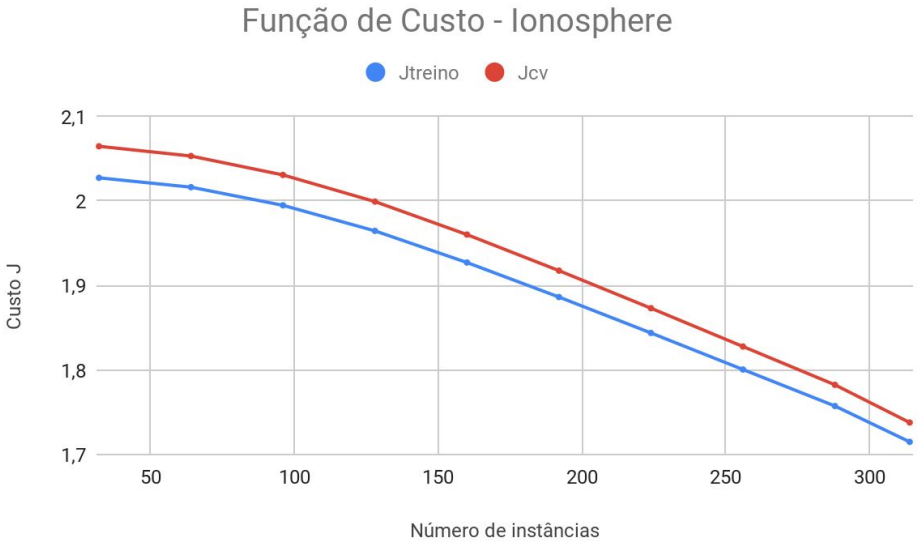
Wine: Uma camada com cinco neurônios, *mini-batches* de tamanho 16, *lambda = 0.25* e *alpha = 0.8*.



Pima: Uma camada com oito neurônios, *mini-batches* de tamanho 32, *lambda = 0.0* e *alpha = 0.1*.



Ionosphere: Duas camadas com cinco neurônios em cada, *mini-batches* de tamanho 32, *lambda = 0.0* e *alpha = 0.1*.



Breast Cancer: Uma camada com 40 neurônios, *mini-batches* de tamanho 16, *lambda = 0.0* e *alpha = 0.5*.