E3: Redes Neuronais para a predição de bancarrota em empresas polacas

Relatório Final

Inteligência Artificial

Mestrado Integrado em Engenharia Informática e Computação

2016-2017

Membros:

- Diogo Duque (up201406274@fe.up.pt)
- João Costa (up201403967@fe.up.pt)
- José Cruz (up201403526@fe.up.pt)

Objetivo

O objetivo deste trabalho consiste na aplicação de Redes Neuronais artificiais à previsão de bancarrota em empresas polacas.

O programa criado deve treinar apropriadamente uma Rede Neuronal Artificial, usando o algoritmo "Back-Propagation", tendo por base um conjunto de dados disponibilizado para o efeito. Os dados devem ser cuidadosamente analisados de forma a verificar a eventual necessidade de pré-processamento. O modelo obtido deve poder depois ser utilizado no diagnóstico de novos casos.

Este projeto engloba os seguintes procedimentos:

- Concepção de uma rede neuronal multi-camada: a camada de entrada contém os atributos ou variáveis de identificação dos dados, a camada de saída contém a classificação obtida e a(s) camada(s) intermédia(s) auxilia(m) no funcionamento da rede neuronal. Devem ser testadas várias configurações da rede (nº de camadas, nº de células nas diferentes camadas, variáveis de entrada, parâmetros do algoritmo de aprendizagem), sendo analisados e comparados os seus resultados com vista à definição da melhor arquitectura.
- Implementação/aplicação do algoritmo "Back-Propagation".
- Medição detalhada de resultados nos dados de treino e de teste.

Índice

Especificação

Análise do tema

Objectivo

Este projecto, desenvolvido no âmbito da unidade curricular de Inteligência Artificial, tendo como objectivo criar um sistema capaz de prever a bancarrota de empresas polacas, utilizando uma Rede Neuronal Artificial, treinada através do algoritmo "Back-Propagation".

A rede neuronal deve aplicar o algoritmo de treino sobre um conjunto de dados-exemplo previamente fornecidos. Através deste processo, a rede neuronal deve tornar-se capaz de aceitar conjuntos de dados desconhecidos sobre uma empresa polaca, e prever com considerável exatidão, se esta irá entrar em bancarrota ou não.

Detalhes do Tema

Polónia

A Polónia, conhecida oficialmente por República da Polónia, possui uma área total de 312 679 km quadrados, e a capital é Varsóvia. Em 2008, era a sexta nação com mais população na União Europeia, e na data de escrita deste documento, está classificada como a vigésima-quarta nação em termos do Produto Interno Bruto, segundo o World Bank.

O maior setor econômico é o setor terciário, seguido do setor secundário e do setor primário. Foi o único país europeu que conseguiu escapar à recessão económica de 2004, é um importante foco de investimento.

Bancarrota

Bancarrota é um estatuto legal, no qual a entidade em questão não tem capacidade de pagar aos credores. Este estatuto é decidido pelos tribunais após um análise sobre a capacidade de pagar as dívidas.

Apesar de ser possível prever a bancarrota de uma empresa, nem sempre é fácil, devido à existência de diversos fatores e relações difíceis de identificar.

Cenários de Uso

Esta ferramenta pode ser usada tanto por investidores da bolsa como pelas próprias empresas.

Os investidores podem usar-la para obterem uma segunda análise, facilitando decisões a tomar no momento de investir ou vender.

As empresas podem usar-la para obterem uma análise objectiva do estado atual e do currente rumo de si mesmas.

Datasets

O dataset usado foi criado por Sebastian Tomczak do Departamento of Investigação Operacional, da Universidade de Ciência e Tecnologia de Wrocław.

Os dados foram recolhidos do Emerging Markets Information Service (EMIS), que contêm informação acerca de mercados emergentes de todo o mundo. As empresas foram analisadas entre 2000 a 2013.

Os dados foram fornecidos através de cinco ficheiros no formato .arrf, consoante a diferença entre a data de análise da empresa e a data de aferição do estado de bancarrota.

Cada conjunto de dados correspondente a uma empresa é constituído por 64 taxas, calculados a partir de várias medidas de carácter financeiro como as vendas totais, inventários, etc; e por um valor de zero ou um, consoante a empresa sobreviveu ou entrou em bancarrota, respetivamente. É importante mencionar que alguns destes dados se encontram incompletos, existindo campos em falta.

Abordagem

De modo a resolver os diversos problemas com que nos fomos deparando, fomos obrigados a fazer alguma pesquisa para descobrir soluções para estes mesmos problemas.

Uma das maiores dificuldades observadas foi obter convergência na framework Neuroph. Como tal, migrámos o projeto para Encog onde a convergência já era possível, mas sem o mecanismo ideal para lidar com valores em falta. No entanto, deixamos aqui as técnicas usadas tanto com uma framework como com outra. Isto será mais explicado na secção “Detalhes relevantes da implementação”.

Nesta secção, são mostrados os algoritmos e técnicas que são parte integrante do programa elaborado.

Extrair um dataset a partir de um ficheiro .arff

Os dados obtidos a partir do repositório encontram-se incorporados num ficheiro de extensão .arff. Uma biblioteca que permite a leitura destes ficheiros é a WEKA.

Através das funções da WEKA, é possível obter todas as instâncias do ficheiro. Uma instância representa os dados de uma empresa, pelo que para cada instância é criado um *array* que guarda cada um dos valores presentes nessa instância. Este *array* é depois inserido numa lista, que reúne todas as instâncias.

Com os valores isolados já guardados, distinguem-se os dados que representam o input na rede neuronal do output, para depois serem utilizados na aprendizagem e/ou teste.

Normalização

A aprendizagem de uma rede neuronal é mais eficiente se os seus dados estiverem normalizados. Apesar de a normalização não produzir resultados diferentes, a convergência ocorre em menos iterações para um conjunto de dados normalizados do que para dados "puros".

Para normalizar os dados, decidimos usar uma técnica de escala, que os redimensiona para um determinado intervalo `[a, b]`. O valor mínimo de um determinado tipo de dados é igualado a '**a**' e o valor máximo para esse tipo é igualado a '**b**'. Os restantes valores são calculados segundo a seguinte fórmula:

$$valorNormalizado = (b - a) * \frac{valorOriginal - valorMin}{valorMax - valorMin} + a$$

Os valores máximos e mínimos de cada atributo são calculados a partir dos valores que existem, não havendo nenhuma dedução sobre os valores que estão em falta.

Esta normalização é aplicada a todos os 64 tipos de dados de input diferentes que existem no repositório.

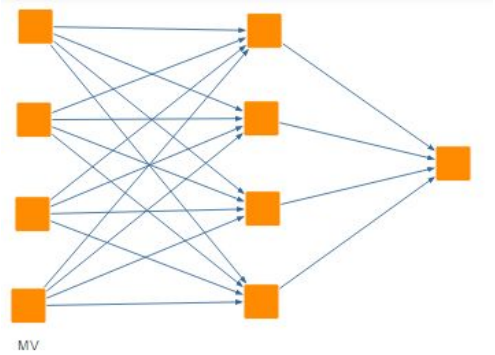
A motivação para o uso desta normalização consiste na discrepância que existe entre os valores de que cada tipo de dados.

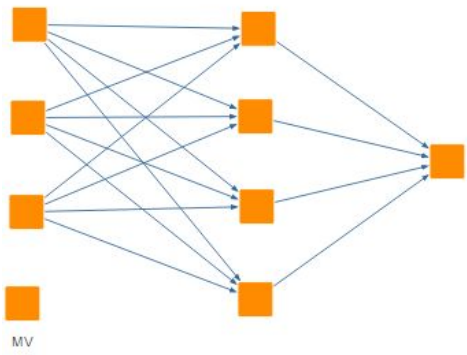
Por exemplo, um dos dados de entrada é o "*Working Capital*" de uma empresa, ou seja, a diferença entre os ativos e os passivos, representa uma quantidade de dinheiro. Já a razão entre lucro líquido e valor total dos ativos representam uma relação e não uma quantidade. Como tal, uma variação de 1.0 nestes dois valores não é equiparável. Com o reescalonamento pretende-se diminuir esta diferença.

Lidar com os valores em falta relativos a algumas empresas (Neuroph)

Para lidar com os valores em falta, baseamo-nos no NSIM (*Neural Selective Input Model*[1]). Na nossa implementação, sempre que se introduz um MV (*Missing Value*) num neurónio da Rede Neuronal, removemos todas as ligações desse neurónio e damos-lhe um valor arbitrário. Tendo em conta que esse neurónio não tem ligações às camadas seguintes, não vai ser capaz de influenciar a RN (independentemente do valor que lhe inserimos como *input*). No fim de ocorrer o *BackPropagation*, são devolvidas as ligações ao neurónio.

No exemplo apresentado abaixo, suponhamos que existe um valor em falta (MV) para o 4º neurónio. Podemos ver na imagem da direita o que acontece no momento antes de inserirmos os *inputs* na rede: o 4º neurónio fica sem ligações, ficando temporariamente fora da rede e não influenciando os seus pesos.





Lidar com os valores em falta relativos a algumas empresas (Encog)

A framework *Encog* não nos permite aceder à sua arquitetura do mesmo modo que a *Neuroph*, pelo que a estratégia para lidar com os valores em falta teve de ser alterada.

De forma a obter o efeito mais próximo do método NSIM, foi calculada a média para cada tipo de atributo. Os valores em falta são substituídos com a média do seu tipo antes de serem normalizados, de forma a minimizar os efeitos nos pesos da rede neuronal.

Treino da rede neuronal

Para cada instância do conjunto de dados existe uma série de valores de entrada e um valor de saída que foi influenciado pelos valores de entrada. Desta forma, o método mais indicado para treinar a rede neuronal é o supervisionado.

Backpropagation

O *backpropagation* pertence aos métodos supervisionados de treino de redes neuronais e é um método de propagação de erro. Isto significa que a rede neuronal recebe um determinado conjunto como entrada, processa esse conjunto e calcula a diferença entre o valor que calculou e o valor ideal, sendo essa diferença o erro da rede neuronal.

A partir do erro, é estimada a função que define o seu gradiente, ou seja, como é que o erro varia consoante os parâmetros recebidos. Este gradiente é calculado na camada final da rede neuronal e retro-propagado em direção à camada de entrada, de forma a alterar os pesos dos neurónios com vista em diminuir o erro.

Uma rede neuronal com *Backpropagation* é, no fundo, um algoritmo de otimização que procura os mínimos da função de erro.

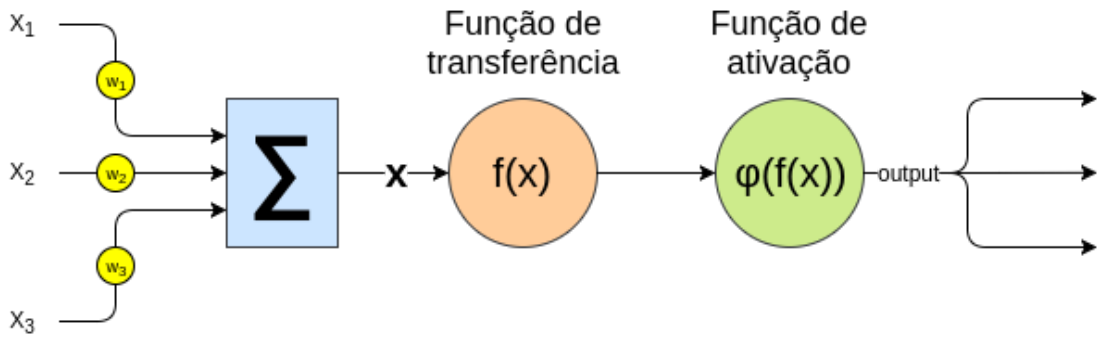
Uso de Backpropagation com *momentum* (Neuroph)

O uso do *momentum* associado ao Backpropagation, permite atenuar as oscilações no *gradient descent*. O *momentum* define o quanto a alteração de pesos da iteração anterior deve afetar a alteração de pesos na iteração atual. Isto permite que, à medida que a rede "acerta" consecutivamente nos valores, vá ganhando mais *momentum* e evolua mais rapidamente no sentido em que os valores apontam. No caso de a rede em algum momento falhar a previsão, isto impedirá que ela se desvie demasiado do rumo de evolução em que já estava.

Uso da heurística RProp (Resilient Backpropagation) (Encog)

Esta heurística tem como principal vantagem a rápida atualização dos pesos da rede. O seu funcionamento prende-se com o facto de, em vez de usar os valores obtidos através das derivadas parciais usadas para calcular o gradiente de erro (BackProp tradicional) para atualizar os pesos das ligações, usa apenas o sinal destes valores. Se, numa certa iteração, o sinal da derivada parcial for igual ao sinal da derivada parcial da iteração anterior, aumenta-se o *delta* e atualiza-se o peso da ligação usando este novo *delta*; se os sinais forem diferentes,diminui-se o valor do *delta* e atualiza-se o peso da ligação usando este novo *delta*.

Uso da função de ativação sigmóide



A capacidade de aprendizagem de uma rede neuronal é altamente influenciada pela função de ativação escolhida para os seus neurónios. Uma função linear limita a capacidade de adaptação dos neurónios, pelo que funções sigmóide tendem a ser escolhidas como funções de ativação.

Segundo Lecun[4], a função de ativação deve produzir resultados próximos do zero, de forma a otimizar o processo de aprendizagem. Assim, na nossa rede neuronal, é uma função sigmóide com declive igual a 1.

Lidar com o *overfitting*

Uma vantagem do uso da framework *Encog* é que esta possibilita um ajuste automático do número de neurónios na *hidden layer*. Este ajuste, por si só, é já o suficiente para assegurar a inexistência de *overfitting* ao ajustar o número de neurónios escondidos para o indicado.

Lidar com um dataset desequilibrado

Um dos problemas dos nossos datasets centra-se no facto de estes serem bastante desequilibrados, o que afeta o treino da rede e a maneira como esta deteta casos de bancarrota ou o oposto. Assim, criamos um dataset “test2” (com entradas extraídas de um dos datasets originais, das quais são eliminadas entradas de não-bancarrota até que o dataset esteja equilibrado) para treinar corretamente a rede.

Desenvolvimento

Ambiente de Desenvolvimento

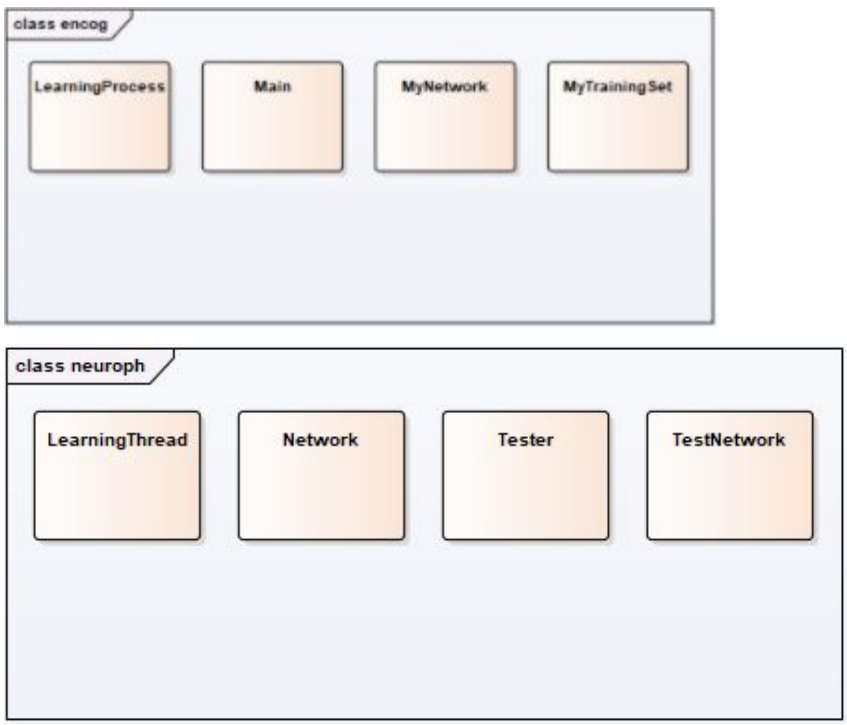
Este programa foi desenvolvido tanto em Linux como em Windows, usando os IDEs IntelliJ e Eclipse. Recorremos também às frameworks *Encog* e *Neuroph* para abstrair algum do funcionamento da rede neuronal.

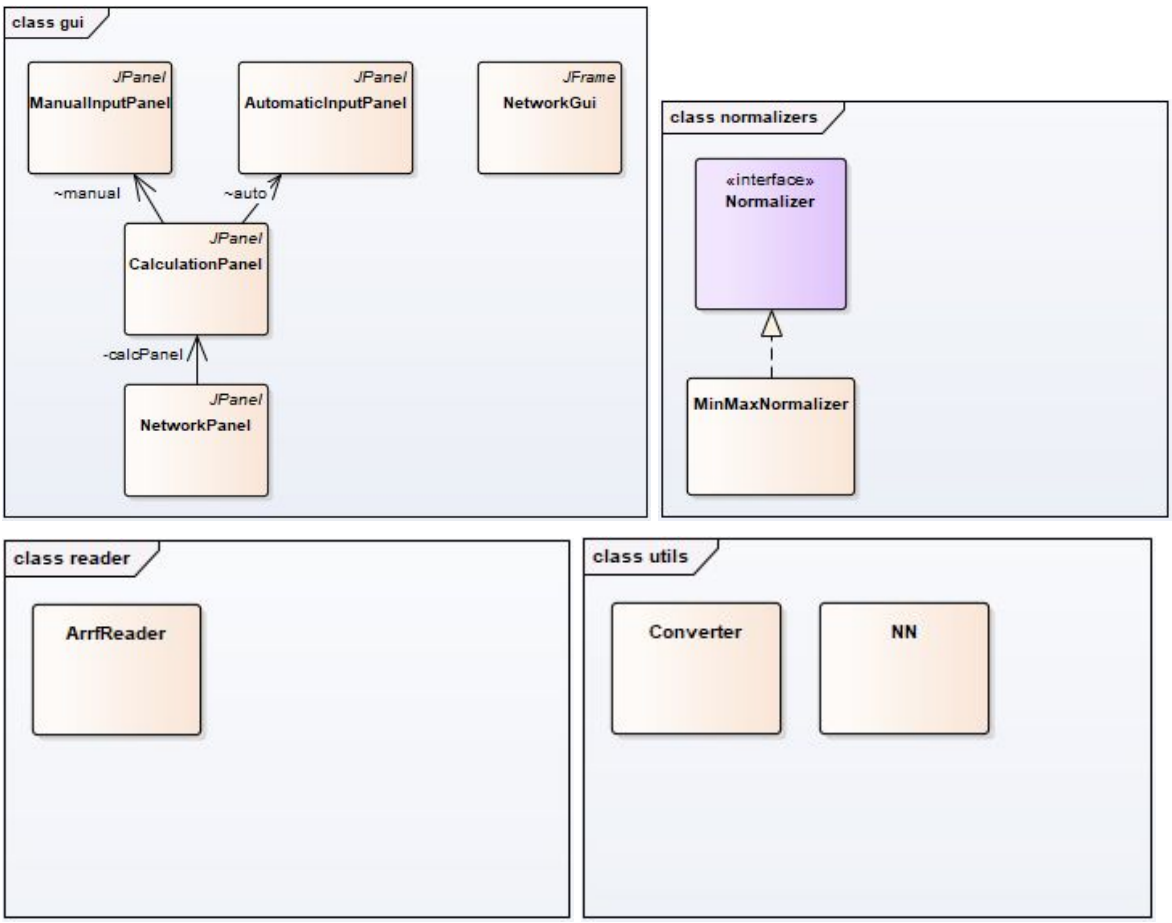
Estrutura da aplicação

Packages

- **encog:** lógica geral da RN usando a framework *encog*.
- **neuroph:** lógica geral da RN usando a framework *neuroph*.
- **gui:** interfaces gráficas.
- **normalizers:** classes para normalizar *datasets*.
- **reader:** *parsing* do *dataset* num ficheiro para uma estrutura de dados.
- **utils:** outras classes que fornecem suporte para os *packages* anteriores.

Diagrama de classes





Detalhes relevantes da implementação

Para a implementação das redes neuronais e funcionalidades a estas associadas, optamos por recorrer a frameworks para nos permitir dar maior atenção aos outros problemas mostrados na secção “Abordagem”.

Inicialmente usamos a framework *Neuroph*. Com esta framework, fazíamos a extração do dataset, a sua normalização (com MinMax) e o tratamento dos valores em falta para depois usarmos Backpropagation com *momentum*. No entanto, visto que não estávamos a atingir a convergência desejada com esta framework, migramos o código para a *Encog*. Com esta framework, replicamos todas as técnicas excetuando 2: em vez de usarmos Backpropagation com *momentum*, usamos RProp pois dava-nos melhores resultados; não conseguimos implementar o mecanismo baseado no NSIM para lidar com os valores em falta devido à maneira como a framework constrói a rede (no entanto, este ainda é visível no código que usa neuroph).

Experiências

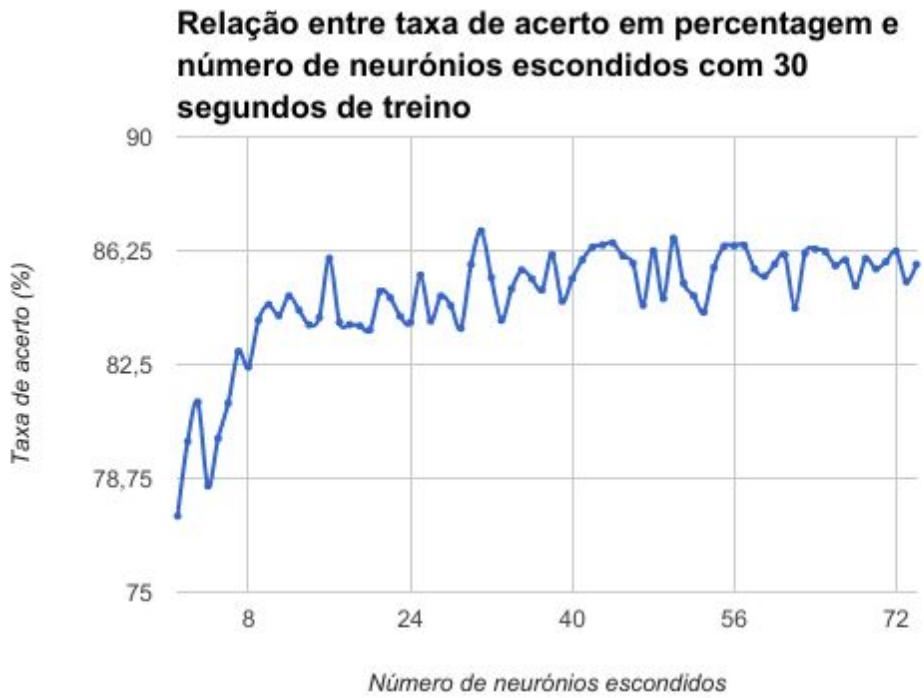
Número de neurónios na camada escondida

Segundo Jeff Heaton, em "*Introduction to Neural Networks for Java, Second Edition*", o tamanho da camada escondida de uma rede neuronal deve ser guiado por uma das seguintes regras:

- Estar entre o tamanho da camada de input e de output;
- Ser cerca de dois terços da soma do tamanho da camada de input e de output;
- Nunca ser superior ao dobro do tamanho da camada de input.

Desta forma, decidimos testar os nossos dados em redes neuronais com diferentes números de neurónios na sua única camada escondida. Segundo Heaton, o tamanho ideal deverá estar entre 1 (tamanho da camada de output) e 64 (tamanho da camada de input).

Como tal, testámos para valores entre 1 e 72 e obtivemos os seguintes resultados.



A própria framework *Encog* possui um mecanismo capaz de determinar o número ideal de neurónios para a camada escondida, dependendo do conjunto de dados de treino. Para o nosso conjunto de treino, o *Encog* geralmente escolhe usar 64 neurónios escondidos. Como este número se encontra em acordo com as nossas observações, a estrutura da nossa rede é criada utilizando este mecanismo.

Normalização

Para verificar se a normalização afeta ou não o desempenho de uma rede neuronal, cronometramos o treino de uma rede neuronal com dados normalizados e sem dados normalizados e verificámos que a primeira chega à taxa de acerto mais depressa.

Conclusões

As redes neuronais são sistemas que não têm em conta o conteúdo da informação que recebem. Apenas tentam fazer sentido daquilo que lhes é fornecido. Como tal, a qualidade dos dados inseridos numa rede neuronal é determinante para a sua aprendizagem. Daí que se tenha verificado que a normalização de dados tenha ajudado a rede neuronal a melhorar a sua taxa de erro.

Além disso, é um sistema baseado numa matriz de pesos, verificando-se que a dimensão desta matriz importa na aprendizagem da rede neuronal.

Melhoramentos

O desempenho da rede apresenta-se satisfatório, no entanto, há um problema em particular que poderá receber uma melhor solução: os valores em falta presentes no *dataset*. Apesar de termos resolvido este problema na nossa implementação com *Neuroph*, com *Encog* isto não foi possível devido à maneira como a framework representa a RN e os neurónios. Caso este problema fosse resolvido (com *Encog*), acreditamos que a rede seria muito mais fiável.

Recursos

Bibliografia

[1] Handling Missing Values Via A Neural Selective Input Model:
http://www.nnw.cz/doi/2012/NNW.2012.22.021.pdf
[2] Dealing with missing values in neural network-based diagnostic systems
: https://link.springer.com/article/10.1007/BF01421959
[3] Neural Learning from Unbalanced Data:
http://sci2s.ugr.es/keel/pdf/specific/articulo/NL-Unbalanced-data.pdf
[4] Resilient Backpropagation
https://en.wikipedia.org/wiki/Rprop
[5] Efficient BackProp
http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf

Outras fontes

Repositório de dados:
http://archive.ics.uci.edu/ml/datasets/Polish+companies+bankruptcy+data

Documentação Java: https://docs.oracle.com/javase/8/docs/api/

Documentação Encog:
http://heatonresearch-site.s3-website-us-east-1.amazonaws.com/javadoc/encog-3.3

Moodle: https://moodle.up.pt/

StackOverflow: http://stackoverflow.com

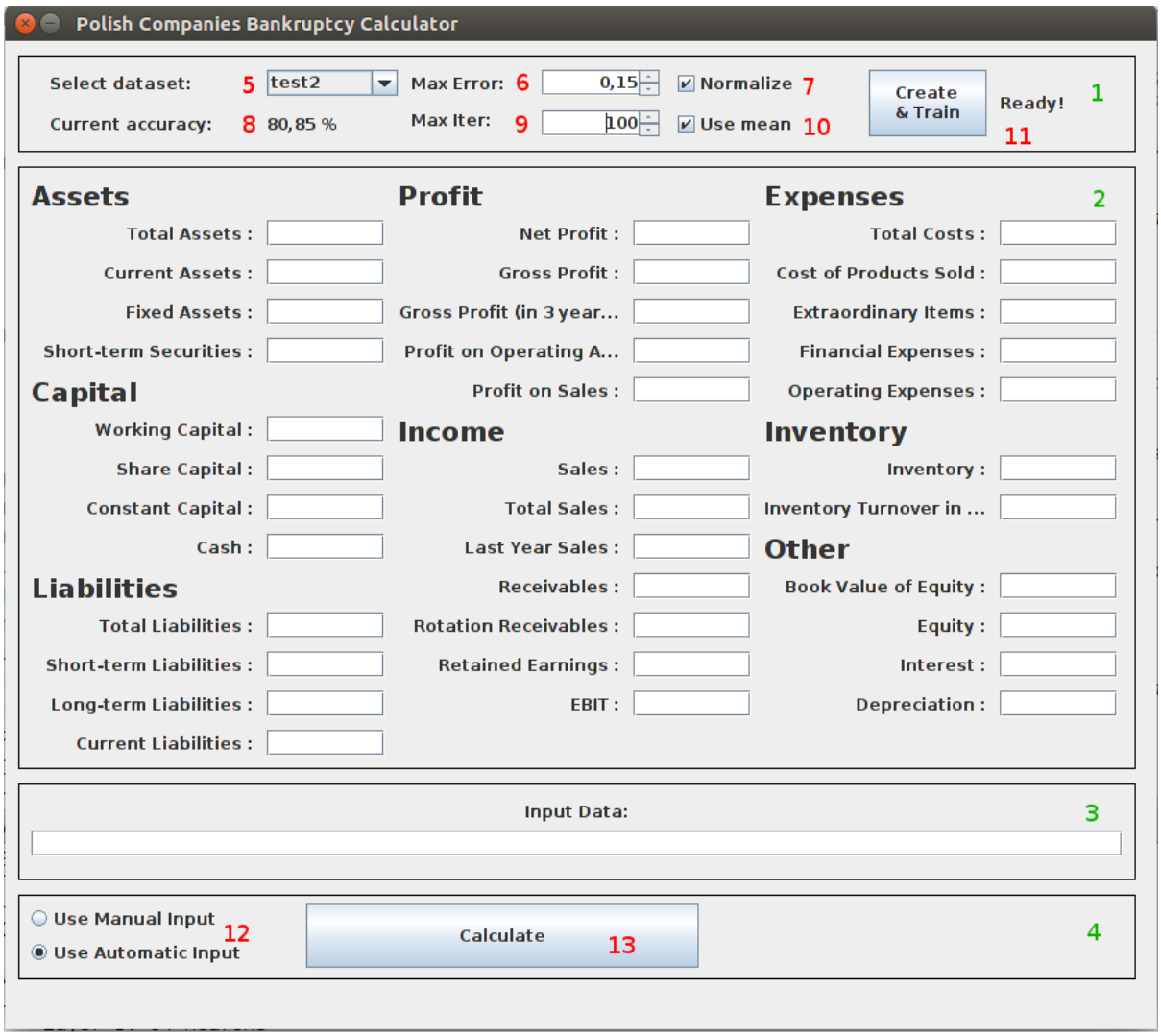
Software utilizado

IDEs: IntelliJ IDEA e Eclipse
Frameworks: Neuroph e Encog

Contribuição

- Diogo Duque - 1/3
- João Costa - 1/3
- José Cruz - 1/3

Apêndice



Manual de Utilização

Interface

1. **Bloco de Treino da Rede Neuronal**
   Contém opções acerca da criação e treino da rede neuronal.
2. **Bloco de Introdução de Dados Manual**
   Contém vários campos de entrada para os valores financeiros referentes a uma empresa.
3. **Bloco de Introdução de Dados Automático**
   Contém um campo de entrada para colar os dados referentes a uma empresa.
4. **Bloco de Cálculo**

Contém opções acerca do cálculo da previsão de bancarrota.

5. **Selector do Dataset**
   Permite escolher que dataset usar.
6. **Erro Máximo**
   Permite escolher quando o treino deve parar, através da limitação do erro.
7. **Switch "Normalize"**
   Permite ligar/desligar a normalização dos valores.
8. **Precisão**
   Indica a percentagem de testes que a última rede neuronal criada acertou.
9. **Número máximo de iterações**
   Número máximo de iterações (*epochs*) que a rede neuronal executará durante o seu treino.
10. **Switch "Use mean"**
    Se estiver selecionado, valores em falta são substituídos pela média dos dados. Caso contrário, é usado o ponto médio entre os dois casos mais extremos.
11. **Botão "Create and Train"**
    Inicia o treino da rede neuronal.
12. **Seletor do Modo de Entrada de Dados**
    Use Manual Input - usa os valores do bloco de introdução de dados manual(2).
    Use Automatic Input - - usa os valores do bloco de introdução de dados automático(3).
13. **Botão "Calculate"**
    Inicia o processo de questionar a rede neuronal.

Procedimento

- Escolher o dataset desejado(1), e indicar o valor máximo do erro(2).
- Indicar se os valores devem ser normalizados ou não(9).
- Indicar se os valores devem os valores em falta devem ser substituídos pela média ou não(10).
- Iniciar o treino da rede neuronal(7). Este processo pode demorar algum tempo, consoante os valores indicados anteriormente.
- Manual
  - Introduzir os valores pedidos no Bloco de Introdução de Dados Manual.
  - Selecionar "Use Manual Input" no Seletor do Modo de Entrada de Dados.
- OU Automático
  - Introduzir os dados no Bloco de Introdução de Dados Automático, no mesmo formato dos ficheiros .arff (valor de saída não é necessário).
  - Selecionar "Use Automatic Input" no Seletor do Modo de Entrada de Dados.
- Calcular o resultado(12).