Gliwice, 11.09.2017r.

Kalibracja stanu zbiorników paliw płynnych

Hurtownie Danych i Systemy Eksploracji Danych

Projekt zasadniczy

Autor:

Sebastian Oprzędek

1. Temat

Celem projektu jest opracowanie metody kalibracji stanu zbiorników paliw płynnych oraz jej implementacja. Objętość paliwa w zbiorniku paliwa wyznaczana jest na podstawie pomiarów jego wysokości, które są przeliczane na objętość na podstawie charakterystycznego dla danego zbiornika przekształcenia: wysokość-objętość. Na podstawie pomiarów wysokości i objętości paliwa w zbiorniku oraz objętości sprzedanego paliwa, a także wykorzystując modele teoretyczne zbiorników paliwa należy znaleźć rzeczywiste odwzorowanie wysokość-objętość, które służy do obliczania faktycznej objętości paliwa w zbiorniku.

2. Analiza zadania

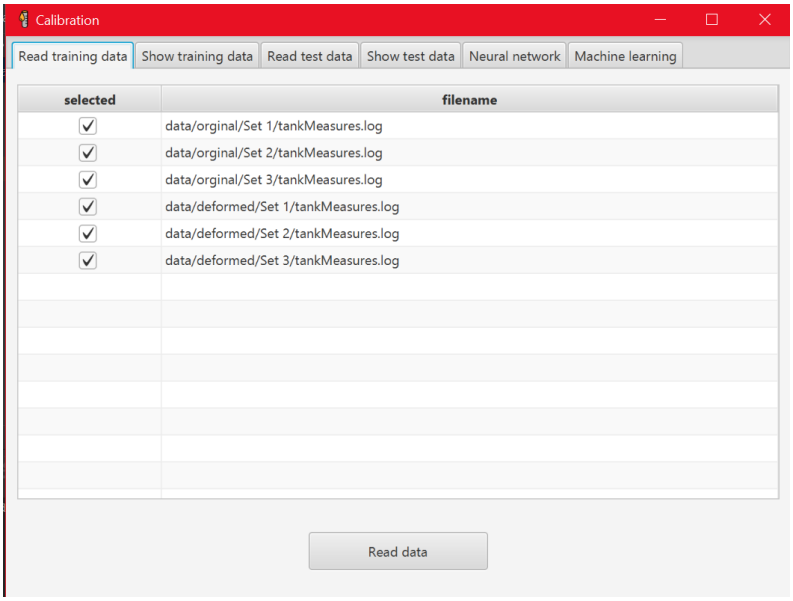
Na podstawie wcześniejszych doświadczeń wybrane zostało podejście oparte o własną sieć neuronową. Inną metodą wartą sprawdzenia jest uczenie maszynowe.
Aplikacja powinna umożliwiać przetestowanie skuteczności wyznaczania objętości, w tym wyznaczenie błędu przewidywania na podstawie dostarczonych pomiarów. Implementacja zostanie napisana w języku Java. Projekt nie wymaga szczególnie zaawansowanego GUI, jednak dla rozwoju własnego użyta zostanie technologia Java FX.

3. Specyfikacja zewnętrzna

Aplikacja składa się z 6 kart. Do nawigacji między nimi służą ich zakładki w górnym pasku.
Karty:

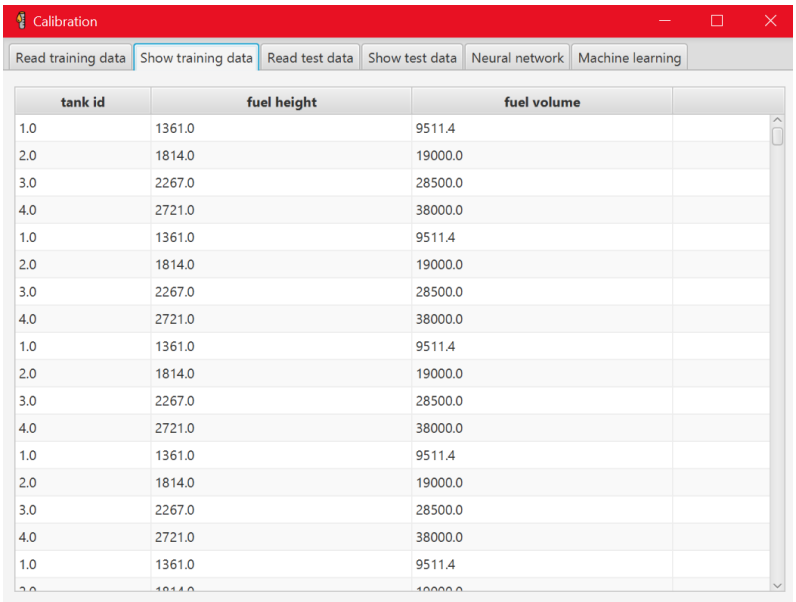
a) *Read training data*

Karta umożliwia wczytanie pomiarów z przykładowych plików .log (ścieżki do plików są konfigurowane w pliku properties.xml). Pliki można zaznaczać za pomocą dołączonych checkbox'ów. Wczytane pomiary zostają zapisane w pamięci i są używane jako dane uczące. Po wczytaniu danych użytkownik zostaje przeniesiony na kolejną kartę.



b) *Show training data*

Karta umożliwia podgląd wczytanych danych uczących.



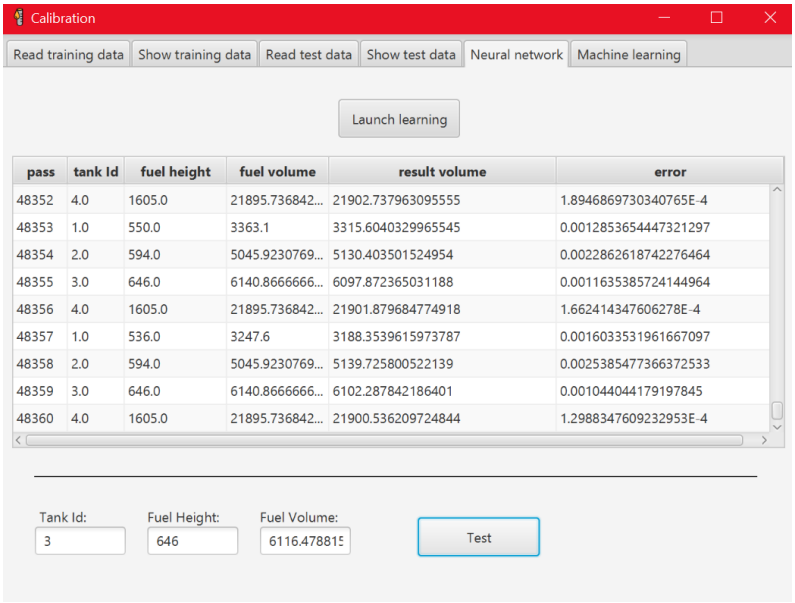
c) *Read test data*

Karta wygląda i zachowuje się tak samo jak *Read training data*, jednak wczytane w ten sposób pomiary używane są jako dane testowe.

c) *Show test data*

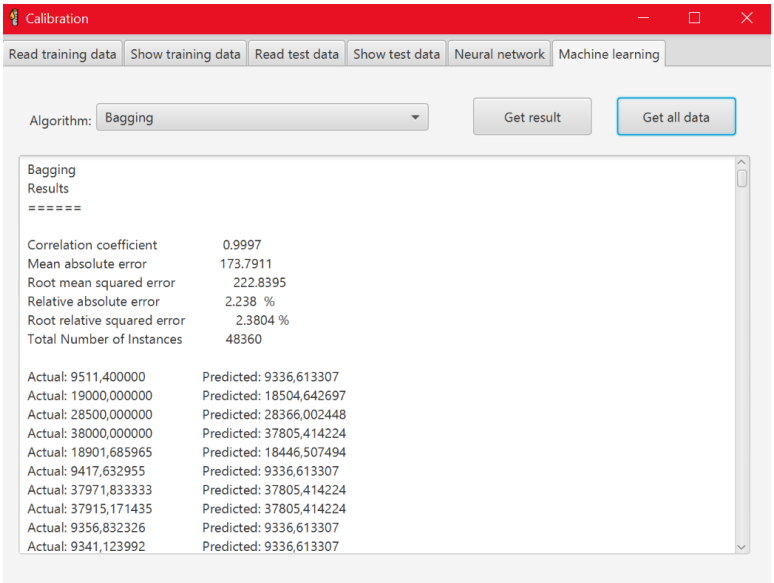
Karta wygląda tak samo jak *Show training data*, jednak na tej karcie prezentowane są pomiary z danych testowych.

e) *Neural network*



Karta umożliwia nauczenie sieci neuronowej danymi uczącymi, a następnie testowanie jej za pomocą danych prowadzanych w polach *Tank Id* oraz *Fuel Height*. Po wciśnięciu przycisku *Test* pole *Fuel Volume* wypełni się wartością wyznaczoną przez sieć.

f) *Machine learning*



Karta umożliwia sprawdzenie skuteczności algorytmów uczenia maszynowego. Po wybraniu algorytmu z listy i kliknięciu przycisku *Get result* na podstawie danych uczących dla wszystkich pomiarów z danych testowych wyznaczona zostanie objętość. Wyznaczone objętości zostaną porównane z faktycznymi, a błąd wyznaczania wypisany.
Przycisk *Get all data* umożliwia także wypisanie wszystkich otrzymanych objętości, za wyjątkiem tych które już wystąpiły.
W celu porównania wyników do listy algorytmów dodana została także sieć neuronowa.

4. Specyfikacja wewnętrzna

4.1 Sieć neuronowa

Sieć neuronowa została przepisana na Javę w oparciu o https://github.com/krzykun/biai - projekt, w którym testowaliśmy możliwości i parametry sieci neuronowej.

Parametry wybrane tam jako optymalne zostały zaimplementowane tutaj jako domyślne i nie są konfigurowalne z poziomu użytkownika.

Model sieci składa się z 4 klas:

- *Net* – klasa przechowująca strukturę sieci oraz umożliwiająca operacje na neuronach
- *Neuron* – klasa przechowująca wartości dla poszczególnego neuronu
- *Connection* – klasa reprezentująca połączenie między neuronami
- *NeuralNetworkDataPostion* – klasa reprezentująca pojedynczy rekord uczący bądź testowy

Dołączone są także klasy *TrainingDataWriter* i *TrainingDataReader* umożliwiające zapis i odczyt danych uczących. Klasy te nie są użyte w projekcie, jednak zostały przeniesione z całością by wykorzystać je w przyszłości. Skorzystanie z nich może wymagać poprawek, ponieważ nie były one testowane.

Oprócz klas modelu paczka zawiera także klasę pomocniczą – *TopologyHelper*, służącą do tworzenia topologii oraz klasę kontrolującą sieć *NeuralNetworkController*.

Wszystkie metody powyższych klas, oraz wszystkich innych z paczki *common*, są dokładniej opisane formie JavaDoc.

index.html

4.2 Uczenie maszynowe

Do przetestowania możliwości uczenia maszynowego wybrana została biblioteka weka. Biblioteka została dołączona w folderze *lib,* w wersji 3.8.

Możliwości biblioteki zostały sprawdzone tylko pobieżnie. Z wszystkich dostępnych algorytmów wybranych zostało tylko 11 działających. Pozostałe zostały odrzucone, ponieważ ich użycie wymagało by głębszego zbadania tematu.

Wybór algorytmów odbywa się w klasie *pl.hdised.calibration.util.algorithm.ClassifierCreator*.

Dokładna dokumentacja biblioteki jest dostępna pod:
http://www.cs.waikato.ac.nz/ml/weka/documentation.html

4.3 GUI

Oprócz wymienionych wyżej źródeł aplikacja posiada także GUI w postaci widoków i kontrolerów. Par widok-kontroler jest 5, z czego jedna para to jest główna, a pozostałe odpowiadają za poszczególne karty.

Widoki:

- machineLearningTab.fxml
- mainScene.fxml
- neuralNetworkTab.fxml
- readDataTab.fxml
- showDataTab.fxml

Kontrolery:

- *MachineLearningTabController*
- *MainSceneController*
- *NeuralNetworkTabController*
- *ReadDataTabController*
- *ShowDataTabController*

Kart jest o dwie więcej, ponieważ trainingData i testData są obsługiwane przez te same klasy.
Pozostałe klasy aplikacji są pomocnicze, więc ich opis zostanie pominięty.

5. Wnioski

5.1 Sieć neuronowa

Sieć neuronowa okazała się niewystarczająco skuteczna. Dokładność na poziomie 5-10% w zależności od wybranych danych jest raczej niewystarczająca do wyznaczania objętości zbiorników.

Możliwe, że parametry wybrane w jako domyślne (na podstawie doświadczeń z https://github.com/krzykun/biai) okazały się niewłaściwe i należałoby przetestować inne.

Domyślnie użyte parametry to:

- Jedna warstwa wewnętrzna sieci o rozmiarze 4 neuronów
- Alpha - momentum sieci = 0.3
- Eta- współczynnik uczenia = 0.2

Dokładniejsze informacje o parametrach sieci dostępne są pod adresem https://github.com/krzykun/biai/blob/master/Dokumentacja/Raport.pdf.

5.1 Uczenie maszynowe

Uczenie maszynowe okazało się zdecydowanie bardziej skuteczne. Należy tutaj zauważyć, że możliwości biblioteki weka zostały sprawdzone tylko pobieżnie, a otrzymane błędy predykcji są zdecydowanie mniejsze niż w przypadku sieci neuronowej.

Minusem uczenia maszynowego (zwłaszcza bardziej dokładnych algorytmów) jest czas obliczeń. Dodatkowo trzeba pamiętać, że uczenie maszynowe działa na zasadzie klasyfikacji, zatem dla zbliżonych pomiarów predykcje będą takie same.

Poniżej przykład takiego zachowania.

Przykład 1.

```
Bagging
Results
======

Correlation coefficient                  0.9997
Mean absolute error                    173.7911
Root mean squared error                222.8395
Relative absolute error                  2.238  %
Root relative squared error              2.3804 %
Total Number of Instances            48360

Actual: 9511,400000             Predicted: 9336,613307
Actual: 19000,000000            Predicted: 18504,642697
Actual: 28500,000000            Predicted: 28366,002448
Actual: 38000,000000            Predicted: 37805,414224
Actual: 18901,685965            Predicted: 18446,507494
Actual: 9417,632955             Predicted: 9336,613307
Actual: 37971,833333            Predicted: 37805,414224
Actual: 37915,171435            Predicted: 37805,414224
Actual: 9356,832326             Predicted: 9336,613307
Actual: 9341,123992             Predicted: 9336,613307
Actual: 9241,204193             Predicted: 9336,613307
Actual: 28490,000000            Predicted: 28366,002448
```

Jak widać dla zbliżonych wyników oczekiwanych predykcje są takie same.

Wśród algorytmów uczenia maszynowego wybranych do sprawdzenia w tym projekcie najlepsze okazały się algorytmy *RandomizableFilteredClassifier* oraz *IBk* z otrzymanymi błędami mniejszymi niż 0.03%.

Pełne wyniki testów w plikach:
- *Testy dla danych pierwotnych.pdf*
- *Testy dla danych zniekształconych.pdf*

Jednocześnie trzeba zauważyć, że wyniki są mocno związane z danymi, dlatego czasem możliwe jest otrzymanie błędu na poziomie 0%, co nie świadczy o nieomylności algorytmu, tylko tych samych danych powtarzających się w danych uczących i testowych.