Exploring survival on the Titanic

Nikita Tchayka

1. Introduction

This is a direct port of the Exploring survival on the titanic notebook by Megan Risdal. The intention of this is to have an example of what can be achieved with the data science tools we have in Haskell as for **February 27th, 2017**.

```
{-# LANGUAGE ConstraintKinds #-}
{-# LANGUAGE DataKinds #-}
{-# LANGUAGE FlexibleContexts #-}
{-# LANGUAGE GADTs #-}
{-# LANGUAGE OverloadedStrings #-}
{-# LANGUAGE PatternSynonyms #-}
{-# LANGUAGE QuasiQuotes #-}
{-# LANGUAGE ScopedTypeVariables #-}
{-# LANGUAGE TemplateHaskell #-}
{-# LANGUAGE TypeOperators #-}
{-# LANGUAGE ViewPatterns #-}
module Lib where

import Control.Applicative
import qualified Control.Foldl as L
import qualified Data.Foldable as F
import Data.Proxy (Proxy(..))
import Lens.Family
import Frames
import Frames.CSV (readTableOpt, rowGen, RowGen(..))
import Pipes hiding (Proxy)
import qualified Pipes.Prelude as P
import Text.Regex
import Text.Regex.Base
```

1.1 Load and check data

We begin by loading our data from the `input/train.csv` file. `Frames` generates a data type in *compile time* for our CSV file.

```
tableTypes "TrainingSet" "input/train.csv"
```

Lets check what's the generated data type, from our REPL:

```
*Lib> :i TrainingSet
type TrainingSet =
  Record
    '[ "PassengerId" :-> Int
     , "Survived"    :-> Bool
     , "Pclass"      :-> Int
     , "Name"        :-> Text
     , "Sex"         :-> Text
     , "Age"         :-> Double
     , "SibSp"       :-> Int
     , "Parch"       :-> Int
     , "Ticket"      :-> Int
     , "Fare"        :-> Double
     , "Cabin"       :-> Text
     , "Embarked"    :-> Text
     ]
```

We will define a **stream** for representing our dataset. This is useful as it might not fit in our RAM:

```
trainingSetStream :: Producer TrainingSet IO ()
trainingSetStream = readTableOpt trainingSetParser "input/train.csv"
```

As our data set is small, we will fit our whole data set in RAM (called in-core) through an *Array of Structures* representation:

```
loadTrainingSet :: IO (Frame TrainingSet)
loadTrainingSet = inCoreAoS trainingSetStream
```

Now we can load the data in our REPL with:

```
*Lib> ts <- loadTrainingSet
*Lib> take 2 $ F.toList ts
[{PassengerId :-> 4
, Survived :-> True
, Pclass :-> 1
, Name :-> "Futrelle, Mrs. Jacques Heath (Lily May Peel)"
, Sex :-> "female"
, Age :-> 35.0
, SibSp :-> 1
, Parch :-> 0
, Ticket :-> 113803
, Fare :-> 53.1
, Cabin :-> "C123"
, Embarked :-> "S"
}]
```

We can check the number of observartions too:

```
*Lib> length $ F.toList ts
521
```

From here we see what we have to deal with:

Variable Name	Description
Survived	True or False
Pclass	Passenger's class
name	Passenger's name
sex	Passenger's sex
age	Passenger's age
SibSp	Number of siblings/spouses aboard
Parch	Number of parents/children aboard
Ticket	Ticket number
Fare	Fare
Cabin	Cabin
Embarked	Port of embarkation

2. Feature Engineering

2.1 What's in a name?

We can see that in the passenger name we have the *passenger title*, so we can break it down into additional variables to have better predictions. Also, we can break it into *surname* too to represent families.

```
getTitleFromName :: String -> Maybe String
getTitleFromName name =
  extractMatch <$> matchOnceText titleRegex name
 where
  titleRegex             = mkRegex "(.*, )|(\\..*)"
  extractMatch (m, _, _) = m
```

Now we can use this function in our REPL to extract the title:

```
*Lib> getTitleFromName "Capt. Obvious"
Just "Capt"
*Lib> getTitleFromName "No title here"
Nothing
```