haggressive-0.1.0.0: Aggression analysis for Tweets on Twitter

Aggression analysis for Tweets on Twitter

Contents

Chapter 1

Evaluation

```
module Evaluation (
    main
  ) where
```

```
 main :: IO ()
```

Chapter 2

Hag

```
module Hag (
    parseCsv,  getFiles,  countItem,  frequency,  iFrequency,  idftf,
    intersectDistance,  tweetToMiniDict,  insertInMap,  validate,
    crossCheckBetterK,  featureIntersectionBetterK,  queueTake,  queueTake',
    mergeTweetFeatures,  crossCheckReal,  crossCheckRealK,  getCategoryK,  end,
    endList,  main
  ) where
```

This module is the main interface for Tweet classification.

```
parseCsv :: Text -> Either String (Vector Tweet)
```

IO and Parsing

`parseCsv` parses a `Text` input for fields in CSV format and returns a `Vector` of `Tweets`

```
getFiles :: FilePath -> IO [FilePath]
```

Get directory contents of `FilePath`. A better variant is at: `http://book.realworldhaskell.org/read/systems-programming-in-haskell.html`

```
countItem :: Ord a => Map a Float -> a -> Map a Float
```

Dictionary operations

For convenice, I refer to two dictionaries: * Mini Dictionary The bag of words for *one* Tweet * Grand Dictionary The bag of words for the entire Corpus **TODO**: Could be defined as type.

Insert an item into a `Map`. Default value is 1 if the item is not existing. If the item is already existing, its frequency will be increased by 1.

```
frequency :: Ord a => Vector a -> Map a Float
```

Calculate the `frequency` of items in a `Vector` and return them in a `Map`.

```
 iFrequency :: Map String Float -> String -> Float -> Float
```

```
idftf :: Map String Float -> Map String Float -> Map String Float
```

Takes a mini dictionary (frequency of words in one Tweet) and a dictionary and calculates the idftf values for all words in the mini dictionary.

```
intersectDistance :: Num a => Map String a -> Map String a -> a
```

Take the bag of words of two `Tweet`s and return the distance as `Num`. *TODO*: Forgot it.

```
tweetToMiniDict :: Tweet -> Map String Float
```

Extract features (for the bag of words) for one Tweet. Thereby, the Tweet will be (in order of application): * tokenized * converted to a `Vector` * `String`s will be converted to lowercase * `String`s that are not `isAlpha` are removed * `String`s that are element of `stopWords` are removed * Empty `String`s will be removed

```
insertInMap :: Map Tweet (Map String Float)
            -> Tweet -> Map Tweet (Map String Float)
```

Take a 'Grand Dictionary'

```
validate :: Int
         -> (Vector Tweet, Vector Tweet) -> Vector (Tweet, [Tweet])
```

Specify k (the number of neighbors) and compare two vectors of `Tweet`s and return the k nearest neighbors for each `Tweet`.

```
crossCheckBetterK :: Int                     -> Map Tweet (Map String Float) -> Tweet -> (Tweet, [Tweet])

featureIntersectionBetterK :: Int                              -> Map Tweet (Map String Float) -> Tweet ->

queueTake :: Int -> PSQ Tweet Float -> [Tweet]

queueTake' :: Int -> PSQ Tweet Float -> [Tweet] -> [Tweet]

mergeTweetFeatures :: (Map String Float                        -> Map String Float -> Float)

crossCheckReal :: Vector (Tweet, Tweet, Float) -> Vector Float

crossCheckRealK :: Vector (Tweet, [Tweet]) -> Vector Float

getCategoryK :: [Tweet] -> String

end :: Vector Float -> Float

endList :: [Float] -> Float

main :: IO ()
```

Chapter 3

Preprocess

```
module Preprocess (
    preprocess
  ) where
```

```
 preprocess :: Text -> Text
```

Chapter 4

Tweethelpers

```
module Tweethelpers (
    mkCrossValScheme,  stopWords,  filterByLabel,
    Tweet(Tweet, tLabel, tUser, tDate, tTime, tMessage)
  ) where
```

```
mkCrossValScheme :: Eq a => [Vector a] -> [(Vector a, Vector a)]
```

```
stopWords :: [String]
```

```
filterByLabel :: Vector Tweet -> String -> Vector Tweet
```

```
data Tweet
```

```
     =                  Tweet
     tLabel ::  String  tUser ::  String     tDate ::  String
     tTime ::  String   tMessage ::  String
```

```
instance Eq Tweet
instance Ord Tweet
instance Show Tweet
instance FromRecord Tweet
```