UNIVERSIDAD DE GRANADA

GENERACIÓN DE REPRESENTACIONES ALTERNATIVAS DE DATOS MEDIANTE TÉCNICAS NO SUPERVISADAS DE DEEP LEARNING

FRANCISCO DAVID CHARTE LUQUE

Trabajo Fin de Máster

Máster en Ciencia de Datos e Ingeniería de Computadores

Tutores
Francisco Herrera Triguero
Francisco Charte Ojeda

ESCUELA INTERNACIONAL DE POSGRADO
E.T.S. INGENIERÍAS INFORMÁTICA Y DE TELECOMUNICACIÓN

Granada, a 28 de mayo de 2018

ÍNDICE GENERAL

RESUMEN

[Tur36]

Parte I

TEORÍA

En esta parte se describen los modelos de aprendizaje estudiados desde un punto de vista teórico.

1

INTRODUCCIÓN

2

SECCIÓN SEGUNDA

Parte II

DESARROLLO PRÁCTICO

Documentación y explicación del software desarrollado que incluye las técnicas descritas.

3

SECCIÓN TERCERA

El siguiente código es un ejemplo de coloreado de sintaxis e inclusión directa de código fuente en el texto usando `minted`.

```
-- From the GHC.Base library.
class  Functor f  where
    fmap        :: (a -> b) -> f a -> f b

    -- | Replace all locations in the input with the same value.
    -- The default definition is @'fmap' . 'const'@, but this may be
    -- overridden with a more efficient version.
    (<$)        :: a -> f b -> f a
    (<$)        =  fmap . const

-- | A variant of '<*>' with the arguments reversed.
(<**>) :: Applicative f => f a -> f (a -> b) -> f b
(<**>) = liftA2 (\a f -> f a)
-- Don't use \$ here, see the note at the top of the page

-- | Lift a function to actions.
-- This function may be used as a value for `fmap` in a `Functor` instance.
liftA :: Applicative f => (a -> b) -> f a -> f b
liftA f a = pure f <*> a
-- Caution: since this may be used for `fmap`, we can't use the obvious
-- definition of liftA = fmap.

-- | Lift a ternary function to actions.
liftA3 :: Applicative f => (a -> b -> c -> d) -> f a -> f b -> f c -> f d
liftA3 f a b c = liftA2 f a b <*> c


{-# INLINABLE liftA #-}
{-# SPECIALISE liftA :: (a1->r) -> IO a1 -> IO r #-}
{-# SPECIALISE liftA :: (a1->r) -> Maybe a1 -> Maybe r #-}
{-# INLINABLE liftA3 #-}
```

```
{-# SPECIALISE liftA3 :: (a1->a2->a3->r) -> IO a1 -> IO a2 -> IO a3 -> IO r #-}
{-# SPECIALISE liftA3 :: (a1->a2->a3->r) ->
                                Maybe a1 -> Maybe a2 -> Maybe a3 -> Maybe r #-}

-- | The 'join' function is the conventional monad join operator. It
-- is used to remove one level of monadic structure, projecting its
-- bound argument into the outer level.
join              :: (Monad m) => m (m a) -> m a
join x            =  x >>= id
```

Vivamus fringilla egestas nulla ac lobortis. Etiam viverra est risus, in fermentum nibh euismod quis. Vivamus suscipit arcu sed quam dictum suscipit. Maecenas pulvinar massa pulvinar fermentum pellentesque. Morbi eleifend nec velit ut suscipit. Nam vitae vestibulum dui, vel mollis dolor. Integer quis nibh sapien.

BIBLIOGRAFÍA

[Tur36] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 1936.