Facultat d'Informàtica de Barcelona

Aprenentatge Automàtic

Anàlisi de sentiment

Josep Ciurana Herrera (josep.ciurana)
Oriol Closa Márquez (oriol.closa)

Tutoritzat per

Lluís A. Belanche Muñoz

11 de gener de 2019

Sumari

1 Introducció

Aquest és l'informe escrit associat a la pràctica de l'assignatura d'Aprenentatge Automàtic (APA) de la FIB pel primer semestre del curs 2018/19, creat per Josep Ciurana Herrera i Oriol Closa Márquez.

Tal i com s'indica en el document proveït[1], l'objectiu de la pràctica consisteix en desenvolupar un conjunt de models de predicció per solucionar un problema concret i determinar quin d'ells és el més adient. Aquest problema pot ser obtingut d'algun dels dipòsits especificats en el document o proposat pels propis estudiants. En el nostre cas s'ha seleccionat el problema de classificació d'opinions a partir de les ressenyes de pel·lícules, obtingut del dipòsit de la universitat d'Edimburg[2]. En les següents pàgines es procedirà a explicar detalladament el problema triat, els diversos models seleccionats, els resultats experimentals obtinguts i les conclusions a les que hem arribat.

1.1 El problema

El problema seleccionat consisteix en predir l'opinió resultant d'una frase donada. Disposem d'un conjunt de dades[3] format per 215154 frases de ressenyes de pel·lícules publicades a Rotten Tomatoes[4] (coneguda web dedicat a la crítica i la informació sobre pel·lícules i videojocs). Per cada frase tenim el grau d'opinió d'aquesta, en forma d'escala amb 5 intervals que va de [0, 0.2] o molt negativa a (0.8, 1] o molt positiva.

Les principals raons que ens van portar a triar aquest problema van ser la seva complexitat i el sorprenent de la proposta. Les oracions inicials es descomponen en arbres de parseig (vist en altres assignatures de l'especialitat), a cada àtom se li assigna un valor en funció de la paraula i, finalment, es combinen *bottom up* per obtenir l'opinió de la frase.

1.2 Treball previ

En el paper *Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank*[5] de Stanford del 2013 s'explica que gràcies a l'ús d'una *Recursive Neural Tensor Network* (RNTN) aconsegueixen millorar algunes mètriques respecte a models predictius anteriorment utilitzats. En concret, impulsa la tècnica de classificació positiu/-negatiu en una sola frase del 80% al 85,4% i també incrementa fins el 80,7% la precisió de predir les etiquetes d'opinió amb alta granularitat per a totes les frases.

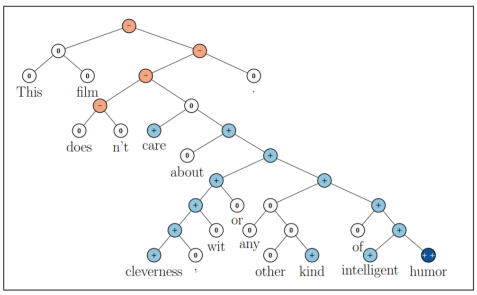


Figura 1: Exemple d'una solució amb RNTN

El mateix document també descriu procediments usats anteriorment; NB, SVM, BiNB, VecAvg, RNN i MV-RNN, els quals hem enumerat ordenats de pitjor a millor segons les mètriques d'alta granularitat i predicció binaria. Els pitjors resultats són de Naive Bayes amb una precisió per tots els nodes amb alta granularitat de 67.2% i de 82.6% per la predicció binaria de tots els nodes i els més propers a la RNTN són de Matrix-Vector Recursive Neural Network amb 78.7% i 86.8%.

2 Exploració de les dades

Disposem de diversos fitxers que componen la totalitat de les fonts de dades. En concret, disposem dels següents continguts.

- `datasetSentences.txt`: Conté una llista d'identificadors i frases separats per un tabulador.
- `dictionary.txt`: Conté una llista de parts de frases amb un identificador separat per una barra vertical.
- `sentiment_labels.txt`: Conté una llista d'identificadors de parts de frases amb un valor real entre el `0` i l'`1` separat per una barra vertical.
- `datasetSplit.txt`: Conté una llista d'identificadors de frases amb un valor indicatiu sobre si forma part del conjunt d'entrenament, prova o desenvolupament separat per una coma.

Realitzem la lectura dels fitxers utilitzant la rutina `read.csv` i els desem en les variables `sentences`, `phrases`, `sentiments` i `split`. Aquests conjunts són *datasets* que contenen una primera columna amb un identificador i una segona amb el valor corresponent.

Donat que aquests fitxers van estar creats ja amb els conjunts de parts de paraules però nosaltres utilitzarem les frases senceres, haurem d'obtenir el valor del *sentiment* a través d'aquell element en el conjunt de parts de frases que si ug igual a la frase que cerquem.

```
#Assign the sentiment to each sentence
cat("Computing sentence sentiments...\n")
for(i in 1:nrow(sentences)){
   try(.Object@data <- rbind(.Object@data,
      data.frame(id=sentences[i,]$id,
         sentence=I(strsplit(as.character(sentences[i,]$sentence), "\\s+")),
         sentiment=sentiments[which(sentiments$id ==
            phrases[which(phrases$phrase ==
               sentences[i,]$sentence),]$id),]$sentiment)),
   silent=TRUE)
 }
```

Finalment, dividim les dades en els tres conjunts corresponents segons el fitxer `datasetSplit.txt`. A més a més, unim els dos conjunts d'entrenament i prova en un altre conjunt anomenat `dataMix` per tal de facilitar l'accés al conjunt sencer que utilitzarem en el futur.

```
#Split data
cat("Splitting data...\n")
for(i in 1:nrow(.Object@data)){
   label = split[which(split$id == .Object@data[i,]$id),]$label
   if(label==1){
      .Object@dataTrain <- rbind(.Object@dataTrain, .Object@data[i,])
   }
   else if(label==2){
      .Object@dataTest <- rbind(.Object@dataTest, .Object@data[i,])
   }
   else if(label==3){
      .Object@dataDev <- rbind(.Object@dataDev, .Object@data[i,])
   }
}
.Object@dataMix <- rbind(.Object@dataTrain, .Object@dataTest)
```

Donem primer de tot un cop d'ull a les dades, per fer-ho, utilitzem la funció `summary`. A continuació es mostra el resultat escapçat en la setena línia del resultat per evitar mostrar totes les llargades de les frases. Comprovem a part, la llargada mínima i màxima d'aquestes amb valor `2` i `52` respectivament. Si mirem els resultats del *summary* del valor del *sentiment* veurem que ja estan normalitzats i que no hi ha absolutament cap valor per sota de `0` o per sobre d'`1`. A més, veiem que el primer quantil se situa molt a prop del 25% mentre que el tercer també ho fa del 75%. Per acabar, la mitjana i mediana són vora del 50%. Així doncs, sembla que tenim una bona mostra de dades.

```
summary(sentiment@dataMix)
id                    sentence.Length      sentiment
Min.:       1         36                   Min.:       0.0000
1st Qu.:    3215      37                   1st Qu.:    0.2778
Median:     5898      39                   Median:     0.5139
Mean:       6051      19                   Mean:       0.5114
3rd Qu.:    9141      8                    3rd Qu.:    0.7361
Max.:       11855     32                   Max.:       1.0000
```

Una vegada vists aquests nombres, ens fixem en les distribucions de les llargades de les frases i del *sentiment*. Veiem com tot i ser el mínim per la llargada de les primeres `2` i el màxim `52`, la majoria es troben entre `5` i `35` essent la mitjana prop del 20. Pel que fa al *sentiment*, podem observar el que semblen ser dues campanes de Gauss solapades i centrades en el primer quantil i el tercer respectivament. Això podria ser així donat que les dades representen puntuacions de pel·lícules i a la gent en general li ha agradat o no, d'aquí els dos pics que podria ser que representessin la mitjana de la gent que ha puntuat malament una pel·lícula o bé respectivament. Fora de les especulacions sobre si és una distribució bimodal o no, ve ei en gris les dadem d'entrenament i prova, en marró les d'entrenament i en blau les de prova. Així doncs, sembla que la partició no és esbiaixada i que tots els conjunts segueixen si més no la mateixa distribució.

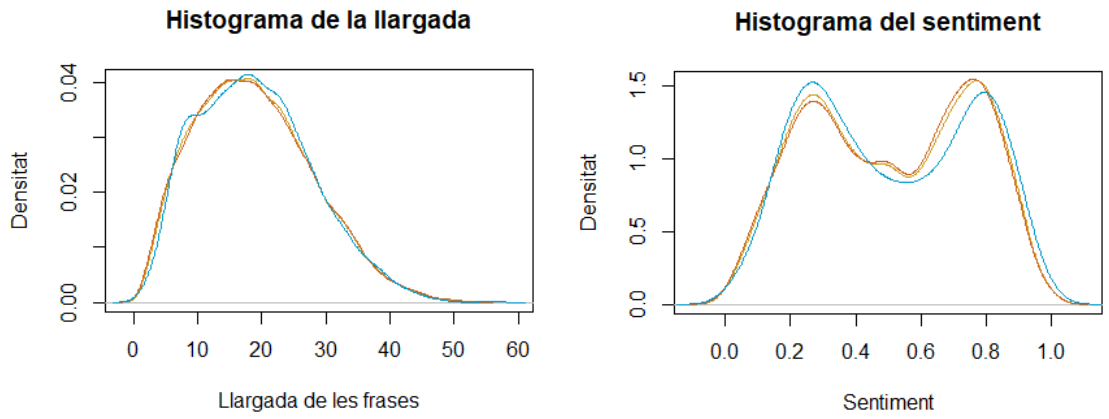


Figura 2: Histogrames de les dades

Com que depenent del cas es realitza un tipus de processat extraordinari de les dades, aquests d'explicaran en les seccions corresponents. Tot i això, els dos més utilitzats són l'eliminació de paraules molt poc freqüents o molt repetides, l'eliminació de les anomenades *stop words*, o la transformació de totes les paraules a minúscules i eliminació de signes de puntuació. Cal dir però, que les dades que tenim ja han tingut un preprocessat anterior de manera que el que realitzarem nosaltres serà lleu.

3 Mètodes lineals i/o quadràtics

Donat que les dades de les que disposem es poden interpretar d'ambdues maneres, com un problema de regressió o com un de classificació, s'han desenvolupat diversos mètodes en cadascuna de les dues àrees.

3.1 Regressió

3.1.1 LASSO

Comencem unint les frases altra vegada donat que la font de dades que tenim en aquest punt després del processat és de llistes de paraules. Les unim primer de tot amb un espai i determinem el tipus dels seus elements. Així doncs, creem un *dataframe* consistent en tres columnes, l'identificador, la frase i el valor del *sentiment* [6].

```
data <- data.frame(id = sentiment@dataMix$id, sentiment = sentiment@dataMix$sentiment,
                                              sentence = sentiment@dataMix$sentence)
for(i in 1:nrow(data)){
  data$sentence[i] <- paste(unlist(data$sentence[i]), collapse = ' ')
}
data$id <- as.character(data$id)
data$sentiment <- as.numeric(data$sentiment)
data$sentence <- as.character(data$sentence)
```

Utilitzem la llibreria `text2vec` [7] per transformar les frases a conjunts numèrics representant les paraules. D'aquesta manera, crearem un conjunt de *tokens* que seran els que substituiran els mots a través d'un vocabulari. Tot això ho fem per poder crear la DTM, més coneguda com a *Document Term Matrix*.

```
tokens <- data$sentence %>% tolower %>% word_tokenizer
it <- itoken(tokens, ids = data$id)
v <- create_vocabulary(it) %>%
   prune_vocabulary(term_count_min = 5)
vectorizer <- vocab_vectorizer(v)
```

Amb això, podrem utilitzar el `vectorizer` creat després de tornar a crear el *tokenitzador* per les dades d'entrenament i prova. D'aquesta manera i amb la rutina `create_dtm` obtindrem la DTM de *training* i `test`. Finalment, ja només ens caldrà crear el model i entrenar-lo [10].

```
glmnet_classifier <- cv.glmnet(x = dtm.train, y = data.train$sentiment)

train.pred <- predict(glmnet_classifier, dtm.train)
train.actual <- data.train$sentiment

test.pred <- predict(glmnet_classifier, dtm.test)
test.actual <- data.test$sentiment

train.accuracy <- 1-rmse(train.pred, train.actual)
test.accuracy <- 1-rmse(test.pred, test.actual)
```

Una vegada fet això, calcularem l'*acuracy* de les prediccions utilitzant `RMSE`. Aquesta funció ens calcularà la mitjana dels errors al quadrat entre els valors predits i els reals.

Abans de continuar amb els experiments però, donarem un cop d'ull al model. Per fer-ho, podem realitzar directament un `plot` d'aquest una vegada entrenat. En la figura del costat podem observar la corba de *crossvalidation* que ha emprat la GLMNET per tal d'entrenar el model. A més a més, també ens indica les desviacions estàndards superiors i inferiors dels valors juntament amb la seqüència λ (les barres d'error). En el nostre cas, s'han triat dos valors de lambda, tal i com podem veure amb les línies discontínues en vertical. Els valors han estat de `0.001977895` i de `0.002869586` per `glmnet_classifier$lambda.min` i `glmnet_classifier$lambda.min` respectivament. La primera representa el mínim error de *crossvalidation* mentre que la segona representa el model més regularitzat on l'error es troba molt properament del mínim. Podem veure els valors mínims de λ per totes les paraules del vocabulari amb la crida `coef(glmnet_classifier, s="lambda.min")` [8].

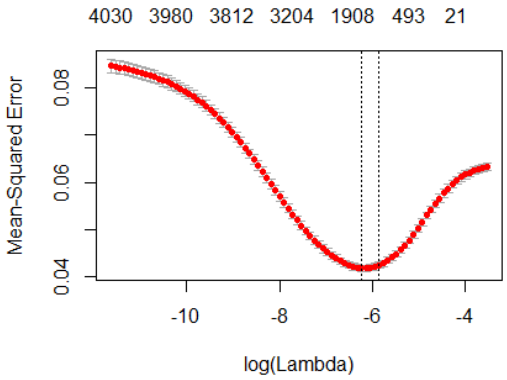


Figura 3: Corba de validació creuada

Donem un cop d'ull també a les gràfiques de les traces de les variables lambda que ja hem vist, i l'L1*norm*, accedint al contingut de `glmnet_classifier$glmnet.fit` segons una o l'altra.

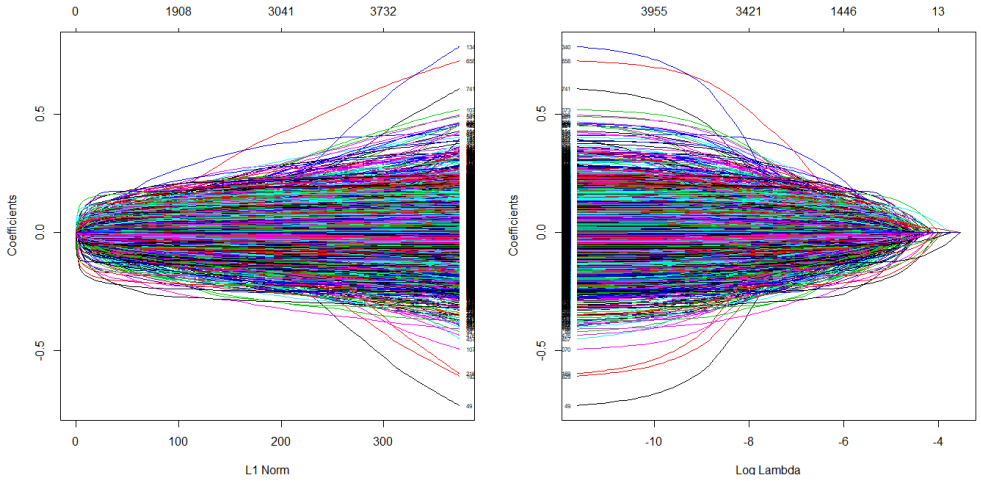


Figura 4: Traces de les variables segons la norma L1 i λ

Si ens fixem, en ambdues gràfiques, cada línia de color representa el valor pres per un coeficient diferent en el model. La λ, el terme de regularització, que tal i com veiem, quan s'aproxima a zero, ho fa cap a una funció d'OLS (*Ordinary least squares*). Com que el terme de regularització és tant petit, la solució que donaríem en aquest apartat, la LASSO, s'aproximaria moltíssim a l'OLS. Com més creix la lambda, més importància té el terme al que multiplica. Pel que fa la la norma L1, veiem que quan és un valor petit, tenim molta regularització, i al contrari. Això és així perquè l'eix de les X ens indica el valor més gran que pot arribar a prendre. Així doncs, un valor de norma igual a zero, seria equivalent a un model buit, donat que el terme de regularització vindria a ser tota la funció. Com més gran és però, més coeficients adquireixen valors majors a zero i per tant entren al model. En definitiva, les dues gràfiques ens estan representant el mateix però en diferents escales [9].

Tot i que hem realitzat diferents experiments canviant el paràmetre `nfolds` i `weights`, els resultats han estat pràcticament idèntics aconseguint millores de dècimes.

Primer de tot hem volgut provar variant el valor del nombre de *it* que realitza a l'hora d'entrenar el model per la *crossvalidation*. Per defecte, el valor és de `10` i tal i com podem veure en la següent taula, el seu valor no fa variar molt el resultat.

Folds	10	15	20	25	50	100
Training accuracy	0.8229277	0.8265820	0.8191026	0.8229277	0.8229277	0.8191026
Test accuracy	0.7929200	0.7935963	0.7918842	0.7929200	0.7929200	0.7918842

Taula 1: Precisió d'entrenament i de prova segons el nombre de *folds*

A part, hem provat de donar diversos valors de `weight`, és a dir, diferents pesos segons cada frase. Hem provat de donar aquests valors d'acord i segons la llargada de la frase corresponent. D'aquesta manera, una frase de llargada `0` tindria un pes `0` i una de llargada màxima, un pes d'`1`.

```
centerLength <- max(lengths(sentiment@dataTrain$sentence))
w <- list()
for(i in 1:n.train){
  w[i] <- 1-(abs(length(sentiment@dataTrain$sentence[[i]])-centerLength)/centerLength)
}
w <- matrix(unlist(w), ncol=1, byrow=TRUE)
```

Altra vegada, hem realitzat diverses execucions però el valor final no ha alterat de l'esperat. Així doncs, hem decidit que els paràmetres òptims en aquest cas, són els que ja vénen per defecte. El resultat final és el següent.

Training acuracy	***Test acuracy***
0.8265820	0.7935963

Taula 2: Valors finals de la precisió utilitzant LASSO

3.1.2 SVM

Comprovem a continuació el rendiment de les *Support Vector Machines* en el nostre problema. Primer de tot, unirem les frases altra vegada tal i com hem fet en l'apartat anterior. Aquesta vegada però, utilitzarem el paquet `e1071`[11] per crear les matrius de termes als documents[12].

```
dtMatrix <- create_matrix(data$sentence)

container <- create_container(dtMatrix, data$sentiment,
   trainSize=1:n.train, virgin=FALSE)
```

Una vegada arribat a aquest punt, disposem d'un objecte especial de dades, el `container`, que conté totes les frases i els seus *sentiments* assignats, però també quina part d'elles és per l'entrenament. Ara doncs, només ens cal entrenar el model utilitzant l'algorisme de l'SVM de la llibreria anomenada anteriorment.

```
model <- train_model(container, "SVM", kernel="linear", cost=1)
```

Finalment, haurem de crear el contenidor de *test* per tal de poder validar el mètode. Ho farem de la següent manera[1].

```
predTestMatrix <- create_matrix(data$sentence[n.test:n.all], originalMatrix=dtMatrix)
predTestSize = length(data$sentence[n.test:n.all])
predictionTestContainer <- create_container(predTestMatrix,
   labels=rep(0, predTestSize), testSize=1:predTestSize, virgin=FALSE)
resultsTest <- classify_model(predictionTestContainer, model)
```

Si tenim en compte que només utilitzem les frases senceres sense crear n-grames (explicats en l'apartat corresponent a l'RNN), el resultat final és el següent.

Training acuracy	*Test acuracy*
0.7918136	0.7255655

Taula 3: Valors finals de la precisió utilitzant SVM per regressió

3.2 Classificació

3.2.1 SVM

Provem a continuació una *Support Vector Machine* per classificació enlloc de regressió.

Per tal de validar-ne els resultats, provem de predir les dades sent més o menys estrictes calculant dos valors per cada conjunt d'entrenament i de prova. En el primer cas, seria aquell on donarem per vàlida la predicció si la classe que prediu l'SVM és la correcta. En el segon cas però, la donarem també per vàlida en cas que hagi predit una classe adjacent. Per exemple, si la xarxa prediu la classe `4` quan la classe real era `5` la donarem per vàlida, en canvi, si prediu `3` pel mateix cas, no. Això ho determinem així perquè al ser classes que representen uns valors de *sentiment* correlatius, té lògica pensar que no hauria de tenir el mateix pes predir que la pel·lícula ha agradat quan el cas és que ha agradat molt que no pas predir que no ha agradat gens.

```
test.notprecise.accuracy <- c()
for(i in 1:length(test.pred)){
  test.notprecise.accuracy[i] <- ifelse(abs(test.pred[i] - test.actual[i]) <= 1, 1, 0)
}
test.notprecise.accuracy <- sum(test.notprecise.accuracy)/length(test.pred)
```

Com podem veure, només cal determinar si el valor absolut de la resta és inferior o igual a `1`. Una vegada fetes les prediccions, donem un cop d'ull a la llargada de les frases per classes com a simple curiositat per veure si hi podem trobar alguna correlació.

[1] Degut a un problema en el paquet `RTextTool`, és possible que l'execució del codi a l'hora de crear la matriu de *train* doni error. Per solucionar-ho, només cal executar `trace("create_matri", edit=T)` i canviar la paraula `Acronym` per `acronym` a la línia 42, desar i continuar amb l'execució. Aquesta *issue* es troba documentada a l'enllaç <https://github.com/timjurka/RTextTools/issues/4>.

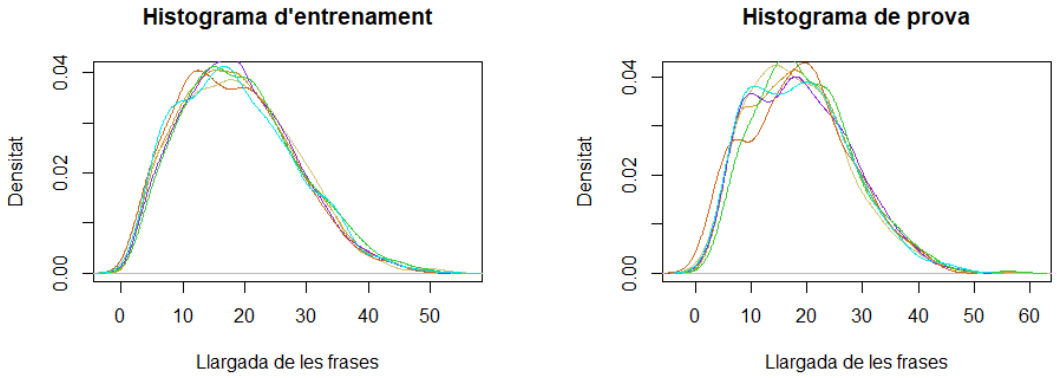


Figura 5: Histogrames de les llargades de els frases separades per classes segons l'SVM

Després de veure els histogrames però, sembla ser que les *reviews* tenen de mitjana la mateixa llargada segons si representen un *sentiment* positiu o negatiu. En cas de veure algun biaix, l'haguéssim pogut aprofitar per tal de veure si potenciava la precisió en alguna classe, però no s'ha donat el cas. A continuació podem veure els resultats finals.

Precisió	***Training acuracy***	***Test acuracy***
Exacta	0.8063324	0.3868235
Propera	0.9928545	0.8018824

Taula 4: Valors finals de la precisió utilitzant SVM per classificació

3.2.2 Naive bayes

Pel que fa al cas de naive bayes, ens pot donar la sensació inicial que els resultats no serien gaire bons. La implementació segueix principalment el mateix mètode anterior, però en aquest cas, amb la crida homònima al títol de la secció.

```
1 system.time(classifier <- naiveBayes(trainNB, trainY, laplace = 0.5))
2 system.time(train.pred <- predict(classifier, newdata=trainNB))
3 system.time(test.pred <- predict(classifier, newdata=testNB))
```

Per tal de determinar la precisió, utilitzem la mateixa definició dual de precisió pel cas actual. Així doncs, el resultat final és el següent.

Precisió	***Training acuracy***	***Test acuracy***
Exacta	0.6790686	0.4042353
Propera	0.8843169	0.8112941

Taula 5: Valors finals de la precisió utilitzant Naive bayes per classificació

4 Mètodes no lineals

Segons el *paper* d'Stanford, en aquest conjunt és on torbem els models que millors resultats experimentals originen. Tot i que ells proven les dades amb diferents tècniques i models, el que millor resultats dóna és una RNTN[2]. En el nostre cas, provarem amb Random forest i una RNN (de l'anglès, *Recurrent Neural Network*).

4.1 Classificació

4.1.1 Random forest

Com a primer mètode no lineal, provem amb `Random Forest` utilitzant la llibreria de nom homònim. Partim de les dades anteriors amb la separació en conjunts de *train* i *test*. Apliquem els filtres habituals a les paraules de `data$sentence` i discretitzant els valors de `data$sentiment` amb `convert_class` i deixant-ho en tipus `factor` per tal d'indicar que es tracta d'una classe.

```
convert_class <- function(x){
  y <- ifelse(x >= 0.8, 5,
         ifelse(x >= 0.6, 4,
          ifelse(x >= 0.4, 3,
           ifelse(x >= 0.2, 2, 1))))
  y <- factor(y, levels=c(1, 2, 3, 4, 5), labels=c("1", "2", "3", "4", "5"))
  return(y)
}
dataSentiment <- sapply(as.numeric(as.character(data$sentiment)), convert_class)
```

A continuació creem el model amb `randomForest()` i mesurem l'*accuracy* comprovant el nombre de prediccions que coincideixen amb el conjunt de *test* respecte el total d'aquest. Per tal d'assegurar-nos de tenir el valor òptim pel paràmetre `ntree`, comprovem la precisió de diversos valors.

```
test.accuracy.randomforest <- c()
for(i in 1:length(test.pred.randomforest)){
  test.accuracy.randomforest[i] <- ifelse(test.pred.randomforest[i]
    == test.actual[i], 1, 0)
}
test.accuracy.randomforest <- sum(test.accuracy.randomforest)/length(test.pred)
```

Val a dir que tot i que hem pogut executar aquest algorisme en diversos altres jocs de dades, no hem estat capaços de realitzar-ho en el nostre en concret, utilitzant els *embeddings*. El motiu ha estat el gran cost computacional que, sorprenentment i comparant-ho amb el següent apartat, ens ha generat.

[2]Una RNTN (de l'anglès, *Recursive Neural Tensor Network*), és un tipus de xarxa neuronal utilitzada pel processament del llenguatge. Aquesta permet determinar grups de paraules que tenen un *sentiment* positiu o negatiu dins d'una mateixa frase. En aquest treball però, ens centrarem en els mètodes vists a classe i en les RNN, ja que creiem que és un pas important per aproximar-nos a les RNTN que hem de fer primer.

4.1.2 RNN

Finalment, implementem una Xarxa Tensorial Recurrent (en anglès, *Recurrent Neural Network*). Aquestes xarxes són una família de xarxes neurals on el seu objectiu és processar dades seqüencials. Es caracteritzen en que les connexions entre unitats formen un cicle dirigit. Això crea un estat intern de la xarxa que li permet adoptar un comportament dinàmic segons el temps. Així doncs, podem aprofitar aquest comportament per tal de processar seqüències de diferent longitud.

En el nostre cas però, no hem implementat una simple xarxa tensorial, pel simple fet que no és del tot adequada pel nostre cas. Això és així perquè aquestes tenen problemes de dependències a llarg termini. És a dir, poden predir molt fàcilment quina serà per exemple la propera paraula tenint en compte el context immediat anterior, però en cas que hi hagi un espai important entre on s'ha après la dada i on es necessita, no funciona. En teoria ho haurien de poder fer, però s'ha vist que no acaba de ser cert. Així doncs, disposem d'un tipus d'unitats pertanyents a les RNN, les LSTM[3] (de l'anglès, *Long Short-Term Memory*). Aquest tipus d'unitats, solucionen el problema esmentat anteriorment així com introdueixen diverses contribucions com ara cicles a si mateix per tal que el descens de gradient pugui fluir per un temps més llarg. Una de les parts més importants però, és el fet que els pesos de les connexions d'aquest cicle ja no són fixes sinó que canvien en funció del context, fet que afegeix dinamisme.

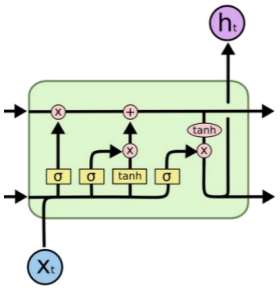


Figura 6: Detall d'una LSTM

Comencem indicant el nombre màxim de paraules que podrà reconèixer la nostra xarxa i creem a continuació l'objecte que ens transformarà les paraules a valors numèrics que la xarxa pugui entendre. A continuació, tot el que hem de fet és convertir les dades de *train* i *test* a matrius amb vectors numèrics. Com que les frases tenen llargades diferents, el que farem serà agafar la més llarga de totes i determinar aquesta com l'amplada de la nostra matriu. Així doncs, anirem tr asformant les frases introdnint-les en la matriu amb *padding*, és a dir, alineades a un extrem amb la resta `0`s.

```
vocabularySize <- 100000
tokenizer <- text_tokenizer(num_words=vocabularySize)
tokenizer %>% fit_text_tokenizer(unlist(sentiment@data$sentence))

maxLength <- max(lengths(sentiment@data$sentence))

x_train <- tokenizer %>% texts_to_sequences(sentiment@dataTrain$sentence)
x_train <- pad_sequences(x_train, maxLength)
```

Finalment, transformem els valors numèrics de *sentiment* a classes, com ja hem fet anteriorment, tindrem una matriu d'amplada `5` on per una classe donada, totes les posicions contindran `0`s menys un `1` en aquella columna corresponent a la classe en qüestió.

En el nostre cas, hem utilitzat `Keras`, un paquet originalment disponible per `Python` i molt utilitzat arreu del món en el camp de *machine* i *deep learning*. Crearem la xarxa seguint el model seqüencial per tal de poder afegir capes fàcilment. El problema d'aquesta part és que no existeix cap mètode conegut per construir les xarxes. Si bé és cert que hi ha pistes sobre com fer la seva construcció, no existeix cap fórmula

[3]Una variant de les LSTM sorgida més recentment, és la GRU (*Gated Recurrent Unit*) que incorpora alguns canvis al respecte com la combinació de la *forget* i *input gate* en una de sola o la unió de l'estat i del *hidden* simplificant-ho tot. Avui dia aquest tipus d'unitats estan creixent en popularitat, nosaltres però, només farem incís en les LSTM.

que ens indica per un cas donat quina és la millor estructura. Així doncs, procedim a mode de prova i error a provar moltes, moltes, estructures diferents[4].

Per tal de model la xarxa hem utilitzat diversos tipus d'unitats del paquet `Keras`, en concret, hem experimentat amb les següents capes.

- `Embedding`: Capa de conversió d'enters positius en vectors densos de llargada fixa.
- `RNN`: Capa d'unitats recurrents.
- `Dense`: Capa clàssica de xarxa neuronal densa connectada.
- `Dropout`: Capa per rebaixar l'*overfitting* posant `0`s segons un *ratio* en l'*input*.
- `LSTM`: Capa amb unitats *Long Short-Term Memory*.
- `GRU`: Capa amb unitats *Gated Recurrent Unit*.

Així doncs, tot i que hem estat experimentant i provant amb moltíssims models diferents, al final ens hem decidit pel model següent. Primer de tot, consta d'una capa d'`embedding` completament obligatòria en tots els models que hem provat per tal de passar de les paraules codificades com a enters a valors que es puguin donar d'*input* a la xarxa. A continuació hem afegit un `dropout` per intentar mitigar una mica l'*overfitting* que hem vist que obteníem sense aquesta capa. A continuació la capa important, l'LSTM que s'encarregarà de les prediccions. Finalment, utilitzem una capa densa com en la resta de models amb els que hem experimentat amb 5 neurones de sortida amb la funció d'activació `softmax`.

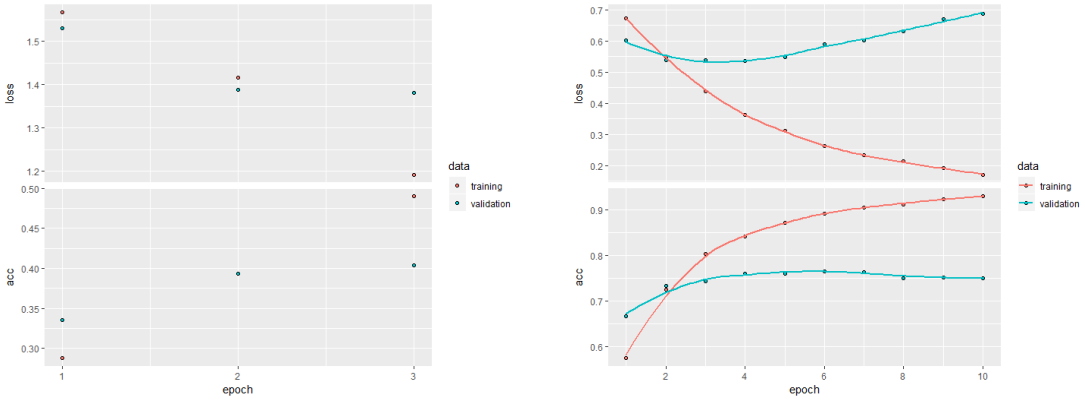
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 32)          3200000
_________________________________________________________________
dropout_25 (Dropout)         (None, None, 32)          0
_________________________________________________________________
lstm_38 (LSTM)               (None, 64)                24832
_________________________________________________________________
dense_33 (Dense)             (None, 5)                 325
=================================================================
Total params: 3,225,157
Trainable params: 3,225,157
Non-trainable params: 0
_________________________________________________________________
```

A continuació hem volgut provar la xarxa amb dues fonts de dades. La primera és com ho hem acabat fent, i és només amb les frases senceres, sense els anomenats n-grames. Aquests són els elements del conjunt de totes les particions d'`n` paraules consecutives possibles en la frase donada. Aquesta és una tècnica que permet utilitzar moltes més dades d'entrada i millora molt el model a l'hora de predir. Com que la divisió de les dades donada s'especifica només per les frases senceres i no pels n-grames d'aquestes, utilitzarem el primer mètode que només considera les frases senceres. En un futur però, s'hauria d'ampliar i abordar amb la segona font de dades.

Comencem amb la classificació en 5 classes diferents. Segons el document original, s'aconsegueix un 43,2% d'encert en l'entrenament utilitzant només les frases senceres i una RNN. En el nostre cas, hem

[4]Cal comentar que abans d'utilitzar les unitats LSTM vam provar-ho amb simples capes d'RNN i denses però els resultats no van ser molt satisfactoris per aquest problema donat. A més, també vam experimentar una mica amb les GRU però no hi vam veure la millora o la diferència per aquest cas en particular.

aconseguit prop del 41%, aproximant-nos molt als resultats obtinguts a Stanford. Pel que fa a la classificació en 2 classes, sense modificar cap paràmetre de la xarxa a excepció de l'última capa que passa a tenir una sola neurona de sortida i la funció d'activació ara és `sigmoid`, la xarxa obté pràcticament un 76% vers el 82,4% del document original[3].



(a) 5 classes (b) 2 classes

Figura 7: Procés d'aprenentatge a través de diferents èpoques amb la funció de *loss* i precisió

El resultat final doncs és el següent. Hem de tenir en compte però que no hem utilitzat n-grames ni la resta d'informació proporcionada, pel que, segurament i tal com es mostra al document original, ens podríem arribar a aproximar al 90% per classificació en 2 classes.

Classes	***Training acuracy***	***Test acuracy***
5	0.6341013	0.4057647
2	0.9236189	0.7506568

Taula 6: Valors finals de la precisió utilitzant l'RNN per classificació sense n-grames

5 Conclusions

En aquest últim apartat presentem les reflexions més rellevants obtingudes durant el desenvolupament de la pràctica. A notació personal, creiem, per una banda, que el problema seleccionat no ha sigut tan fàcil de tractar com d'altres que vam estar considerant abans de decidir-nos i en aquests moments desconeixem si aquesta dificultat extra serà valorada o no i, per l'altra, valorem positivament la feina de recerca extraordinària i de consolidació que hem agut de fer per poder aplicar els diversos models, la qual ens ha ajudat a entendre millor el concepte de *machine learning* i ens ha obert les portes al *deep learning*.

També voldríem deixar constància d'algunes propostes d'ampliació que es podrien desenvolupar en el futur. Una d'elles és aprofundir en les diferències entre entendre aquest problema com un de classificació o de regressió realitzant més models del segon pels dos conjunts vists. Una altra seria provar més models de xarxes neuronals ja que són les que se suposa que proporcionen millors resultats experimentals.

Per acabar, presentem una síntesi dels resultats obtinguts per a cada un dels diferents models de les classes lineal/quadràtic i no lineal.

Classe	**Tipus**	**Mètode**	***Training acuracy***	***Test acuracy***
Lineal/quadràtic	Regressió	LASSO	**0.8265820**	**0.7935963**
		SVM	0.7918136	0.7255655
	Classificació	SVM	0.8063324	0.3868235
		Naive bayes	0.6790686	0.4042353
No lineal	Classificació	Random forest	–	–
		RNN	**0.6341013**	**0.4057647**

Taula 7: Valors finals comparatius de tots els mètodes

Com podem observar els valors obtinguts són, en general, semblants als que trobem al *paper* d'Stanford. Els models que ens han donat millors resultats són LASSO per regressió amb 79,36% de precisió en *test* i RNN per classificació amb 40,58% també de prova. Val a dir, tal i com hem comentat en el seu apartat, que aquest valor de prova no té en compte els n-grames. Així doncs, creiem que el millor model de tots és aquest darrer, tenint en compte que mancaria la implementació d'aquest aspecte. De fet, creiem que segurament el podria millorar també utilitzant les unitats GRU que es comenten en comparació amb les LSTM.

Finalment, volem donar les gràcies al professor Richard Röttger del Departament de Matemàtiques i Computació de la Syddansk Universitet (Odense, Dinamarca), per la seva ajuda en la comprensió de les xarxes neuronals i en concret de les recurrents que de ben segur que ha estat de molta ajuda a l'hora de fer aquest treball.

Bibliografia

[1] A. Belanche Muñoz, Lluís. *Aprenentatge automàtic*. «Practical work». Barcelona, Catalunya: Universitat Politècnica de Catalunya, octubre de 2018.

[2] The School of Informatics (University of Edinburgh). *IRDS: Datasets for Mini-Projects*. [En línia]. <http://www.inf.ed.ac.uk/teaching/courses/irds/miniproject-datasets.html#prj3>. [Data 28/12/2018].

[3] *Dataset*. [En línia]. <http://nlp.stanford.edu/~socherr/stanfordSentimentTreebank.zip>.

[4] *Rotten Tomatoes*. [En línia]. <https://www.rottentomatoes.com/>. [Data: 18/12/2018].

[5] Socher, Richard; Perelygin, Alex; Wu, Jean; Chuang, Jason; Manning, Christopher; Ng, Andrew i Potts, Christopher. *Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank*. «Conference on Empirical Methods in Natural Language Processing (EMNLP)». Stanford, Califòrnia (EUA): Stanford University, 2013. [En línia]. <https://nlp.stanford.edu/~socherr/EMNLP2013_RNTN.pdf>. [Data 15/12/2018].

[6] Selivanov, Dmitriy. *Analyzing Texts with the text2vec package*. [En línia]. <https://cran.r-project.org/web/packages/text2vec/vignettes/text-vectorization.html>.

[7] Selivanov, Dmitriy. *Package 'text2vec'*. [En línia]. <https://cran.r-project.org/web/packages/text2vec/text2vec.pdf>.

[8] Sycorax. *Stack Exchange*. «How to interpret cv.glmnet() plot?». [En línia]. <https://stats.stackexchange.com/questions/253963/how-to-interpret-cv-glmnet-plot>.

[9] Marx, David. *Stack Exchange*. «Interpretting LASSO variable trace plots». [En línia]. <https://stats.stackexchange.com/questions/68431/interpretting-lasso-variable-trace-plots>.

[10] Hastie, Trevor. *Package 'glmnet'*. [En línia]. <https://cran.r-project.org/web/packages/glmnet/glmnet.pdf>.

[11] Meyer, David. *Package 'e1071'*. [En línia]. <https://cran.r-project.org/web/packages/e1071/e1071.pdf>.

[12] *SVM Tutorial*. «How to classify text in R». [En línia]. <https://www.svm-tutorial.com/2014/11/svm-classify-text-r/>.

A Codis en R

A.1 Regressió

A.1.1 LASSO

```r
#Libraries
#devtools::install_github("rstudio/keras")
library(jsonlite)
library(reticulate)
#use_condaenv('tf-keras')
#library(keras)
library(purrr)
library(reshape2)
library(RNiftyReg)
library(tm)
library(RTextTools)
library(e1071)
library(dplyr)
library(caret)
library(Metrics)
library(text2vec)
library(glmnet)

#Sentiment class
setClass("Sentiment",
       representation(data="list",
                  dataTrain="list",
                  dataTest="list",
                  dataDev="list",
                  dataMix="list",
                  sentencesPath="character",
                  phrasesPath="character",
                  sentimentsPath="character",
                  splitPath="character"),
       prototype(data=data.frame(id=numeric(),
                            sentence=list(),
                            sentiment=double()),
              dataTrain=data.frame(id=numeric(),
                                   sentence=list(),
                                   sentiment=double()),
              dataTest=data.frame(id=numeric(),
                                  sentence=list(),
                                  sentiment=double()),
              dataDev=data.frame(id=numeric(),
                                 sentence=list(),
                                 sentiment=double()),
              dataMix=data.frame(id=numeric(),
                                 sentence=list(),
                                 sentiment=double()),
              sentencesPath="datasetSentences.txt",
              phrasesPath="dictionary.txt",
              sentimentsPath="sentiment_labels.txt",
              splitPath="datasetSplit.txt"))

setMethod("initialize", "Sentiment", function(.Object, ...) {
  #Check validity
  cat("Checking validity...\n")
  .Object <- callNextMethod()
```

```
54    validObject(.Object)
55
56    #Read sentences
57    cat("Reading sentences...\n")
58    sentences <- read.csv(.Object@sentencesPath, header=TRUE, sep="\t", colClasses=c('
                                                numeric', 'character'))
59    colnames(sentences) <- c("id", "sentence")
60
61    #Read phrases
62    cat("Reading phrases...\n")
63    phrases <- read.csv(.Object@phrasesPath, header=FALSE, sep="|", colClasses=c('
                                                character', 'numeric'))
64    phrases <- phrases[, c(2, 1)]
65    colnames(phrases) <- c("id", "phrase")
66
67    #Read sentiments
68    cat("Reading sentiments...\n")
69    sentiments <- read.csv(.Object@sentimentsPath, header=TRUE, sep="|", colClasses=c('
                                                numeric', 'double'))
70    colnames(sentiments) <- c("id", "sentiment")
71
72    #Read split
73    cat("Reading split...\n")
74    split <- read.csv(.Object@splitPath, header=TRUE, sep=",", colClasses=c('numeric', '
                                                numeric'))
75    colnames(split) <- c("id", "label")
76
77    #Assign the sentiment to each sentence
78    cat("Computing sentence sentiments...\n")
79    for(i in 1:nrow(sentences)){
80      try(.Object@data <- rbind(.Object@data, data.frame(id=sentences[i,]$id, sentence=I(
                                                  strsplit(as.character(sentences[i,]$
                                                  sentence), "\\s+")), sentiment=
                                                  sentiments[which(sentiments$id ==
                                                  phrases[which(phrases$phrase ==
                                                  sentences[i,]$sentence),]$id),]$
                                                  sentiment)), silent=TRUE)
81    }
82
83    #Split data
84    cat("Splitting data...\n")
85    for(i in 1:nrow(.Object@data)){
86      label = split[which(split$id == .Object@data[i,]$id),]$label
87      if(label==1){
88        .Object@dataTrain <- rbind(.Object@dataTrain, .Object@data[i,])
89      }
90      else if(label==2){
91        .Object@dataTest <- rbind(.Object@dataTest, .Object@data[i,])
92      }
93      else if(label==3){
94        .Object@dataDev <- rbind(.Object@dataDev, .Object@data[i,])
95      }
96    }
97
98    .Object@dataMix <- rbind(.Object@dataTrain, .Object@dataTest)
99
100   #Return object
101   .Object
102 })
103
104 toClasses <- function(data){
105   dataClasses <- list()
```

```
  for(i in 1:length(data)){
    if((data[i]>=0)&&(data[i]<=0.2)){
      dataClasses[[i]] <- list(1, 0, 0, 0, 0)
    }
    else if((data[i]>0.2)&&(data[i]<=0.4)){
      dataClasses[[i]] <- list(0, 1, 0, 0, 0)
    }
    else if((data[i]>0.4)&&(data[i]<=0.6)){
      dataClasses[[i]] <- list(0, 0, 1, 0, 0)
    }
    else if((data[i]>0.6)&&(data[i]<=0.8)){
      dataClasses[[i]] <- list(0, 0, 0, 1, 0)
    }
    else if((data[i]>0.8)&&(data[i]<=1)){
      dataClasses[[i]] <- list(0, 0, 0, 0, 1)
    }
  }
  return(matrix(unlist(dataClasses), ncol=5, byrow=TRUE))
}

convert_class <- function(x) {
  y <- ifelse(x >= 0.8, 5, ifelse(x >= 0.6, 4, ifelse(x >= 0.4, 3, ifelse(x >= 0.2, 2,
                                                                           1))))
  y <- factor(y, levels=c(1, 2, 3, 4, 5), labels=c("1", "2", "3", "4", "5"))
  y
}

sentiment <- new("Sentiment")

data <- data.frame(id = sentiment@dataMix$id, sentiment = sentiment@dataMix$sentiment,
                                                 sentence = sentiment@dataMix$sentence)
for(i in 1:nrow(data)){
  data$sentence[i] <- paste(unlist(data$sentence[i]), collapse = ' ')
}
data$id <- as.character(data$id)
data$sentiment <- as.numeric(data$sentiment)
data$sentence <- as.character(data$sentence)

n.train = nrow(sentiment@dataTrain)
n.test = n.train + 1
n.all = nrow(sentiment@dataMix)

data.train <- data[1:n.train,]
data.test <- data[n.test:n.all,]

tokens <- data$sentence %>% tolower %>% word_tokenizer
it <- itoken(tokens, ids = data$id)
v <- create_vocabulary(it) %>%
  prune_vocabulary(term_count_min = 5)
vectorizer <- vocab_vectorizer(v)

tokens.train <- data.train$sentence %>% tolower %>% word_tokenizer
it.train <- itoken(tokens.train, ids = data.train$id)
dtm.train <- create_dtm(it.train, vectorizer)

tokens.test <- data.test$sentence %>% tolower %>% word_tokenizer
it.test <- itoken(tokens.test, ids = data.test$id)
dtm.test <- create_dtm(it.test, vectorizer)

#data.train$sentiment <- sapply(as.numeric(as.character(data.train$sentiment)), convert
                                                 _class)

```

```
centerLength <- max(lengths(sentiment@dataTrain$sentence))
#centerLength <- 25
w <- list()
for(i in 1:n.train){
  w[i] <- 1-(abs(length(sentiment@dataTrain$sentence[[i]])-centerLength)/centerLength)
}
w <- matrix(unlist(w), ncol=1, byrow=TRUE)

glmnet_classifier <- cv.glmnet(x = dtm.train, y = data.train$sentiment)
#glmnet_classifier <- cv.glmnet(x = dtm.train, y = data.train$sentiment, weights = w)

plot(glmnet_classifier)

op <- par(mfrow=c(1, 2))
plot(glmnet_classifier$glmnet.fit, "norm", label=TRUE)
plot(glmnet_classifier$glmnet.fit, "lambda", label=TRUE)
par(op)

train.pred <- predict(glmnet_classifier, dtm.train)
train.actual <- data.train$sentiment

test.pred <- predict(glmnet_classifier, dtm.test)
test.actual <- data.test$sentiment

train.accuracy <- 1-rmse(train.pred, train.actual)
test.accuracy <- 1-rmse(test.pred, test.actual)
cat("Train: ", train.accuracy, "\nTest: ", test.accuracy, "\n")
```

A.1.2 SVM

```
#Libraries
#devtools::install_github("rstudio/keras")
library(jsonlite)
library(reticulate)
#use_condaenv('tf-keras')
#library(keras)
library(purrr)
library(reshape2)
library(RNiftyReg)
library(tm)
library(RTextTools)
library(e1071)
library(dplyr)
library(caret)
library(Metrics)

#Sentiment class
setClass("Sentiment",
       representation(data="list",
                  dataTrain="list",
                  dataTest="list",
                  dataDev="list",
                  dataMix="list",
                  sentencesPath="character",
                  phrasesPath="character",
                  sentimentsPath="character",
                  splitPath="character"),
       prototype(data=data.frame(id=numeric(),
```

```
                            sentence=list(),
                            sentiment=double()),
                dataTrain=data.frame(id=numeric(),
                                     sentence=list(),
                                     sentiment=double()),
                dataTest=data.frame(id=numeric(),
                                    sentence=list(),
                                    sentiment=double()),
                dataDev=data.frame(id=numeric(),
                                   sentence=list(),
                                   sentiment=double()),
                dataMix=data.frame(id=numeric(),
                                   sentence=list(),
                                   sentiment=double()),
                sentencesPath="datasetSentences.txt",
                phrasesPath="dictionary.txt",
                sentimentsPath="sentiment_labels.txt",
                splitPath="datasetSplit.txt"))

setMethod("initialize", "Sentiment", function(.Object, ...) {
  #Check validity
  cat("Checking validity...\n")
  .Object <- callNextMethod()
  validObject(.Object)

  #Read sentences
  cat("Reading sentences...\n")
  sentences <- read.csv(.Object@sentencesPath, header=TRUE, sep="\t", colClasses=c('
                                                numeric', 'character'))
  colnames(sentences) <- c("id", "sentence")

  #Read phrases
  cat("Reading phrases...\n")
  phrases <- read.csv(.Object@phrasesPath, header=FALSE, sep="|", colClasses=c('
                                                character', 'numeric'))
  phrases <- phrases[, c(2, 1)]
  colnames(phrases) <- c("id", "phrase")

  #Read sentiments
  cat("Reading sentiments...\n")
  sentiments <- read.csv(.Object@sentimentsPath, header=TRUE, sep="|", colClasses=c('
                                                numeric', 'double'))
  colnames(sentiments) <- c("id", "sentiment")

  #Read split
  cat("Reading split...\n")
  split <- read.csv(.Object@splitPath, header=TRUE, sep=",", colClasses=c('numeric', '
                                                numeric'))
  colnames(split) <- c("id", "label")

  #Assign the sentiment to each sentence
  cat("Computing sentence sentiments...\n")
  for(i in 1:nrow(sentences)){
    try(.Object@data <- rbind(.Object@data, data.frame(id=sentences[i,]$id, sentence=I(
                                                  strsplit(as.character(sentences[i,]$
                                                  sentence), "\\s+")), sentiment=
                                                  sentiments[which(sentiments$id ==
                                                  phrases[which(phrases$phrase ==
                                                  sentences[i,]$sentence),]$id),]$
                                                  sentiment)), silent=TRUE)
  }

```

```
  #Split data
  cat("Splitting data...\n")
  for(i in 1:nrow(.Object@data)){
    label = split[which(split$id == .Object@data[i,]$id),]$label
    if(label==1){
      .Object@dataTrain <- rbind(.Object@dataTrain, .Object@data[i,])
    }
    else if(label==2){
      .Object@dataTest <- rbind(.Object@dataTest, .Object@data[i,])
    }
    else if(label==3){
      .Object@dataDev <- rbind(.Object@dataDev, .Object@data[i,])
    }
  }

  .Object@dataMix <- rbind(.Object@dataTrain, .Object@dataTest)

  #Return object
  .Object
})

toClasses <- function(data){
  dataClasses <- list()
  for(i in 1:length(data)){
    if((data[i]>=0)&&(data[i]<=0.2)){
      dataClasses[[i]] <- list(1, 0, 0, 0, 0)
    }
    else if((data[i]>0.2)&&(data[i]<=0.4)){
      dataClasses[[i]] <- list(0, 1, 0, 0, 0)
    }
    else if((data[i]>0.4)&&(data[i]<=0.6)){
      dataClasses[[i]] <- list(0, 0, 1, 0, 0)
    }
    else if((data[i]>0.6)&&(data[i]<=0.8)){
      dataClasses[[i]] <- list(0, 0, 0, 1, 0)
    }
    else if((data[i]>0.8)&&(data[i]<=1)){
      dataClasses[[i]] <- list(0, 0, 0, 0, 1)
    }
  }
  return(matrix(unlist(dataClasses), ncol=5, byrow=TRUE))
}

sentiment <- new("Sentiment")

data <- data.frame(sentiment = sentiment@dataMix$sentiment, sentence =
                                          sentiment@dataMix$sentence)
for(i in 1:nrow(data)){
  data$sentence[i] <- paste(unlist(data$sentence[i]), collapse = ' ')
}
data$sentiment <- as.numeric(data$sentiment)
data$sentence <- as.character(data$sentence)

n.train = nrow(sentiment@dataTrain)
n.test = n.train + 1
n.all = nrow(sentiment@dataMix)

dtMatrix <- create_matrix(data$sentence)

container <- create_container(dtMatrix, data$sentiment, trainSize=1:n.train, virgin=
                                          FALSE)

```

```
model <- train_model(container, "SVM", kernel="radial", cost=1, epsilon=0.5)

predTrainMatrix <- create_matrix(data$sentence[1:n.train], originalMatrix=dtMatrix)
predTrainSize = length(data$sentence[1:n.train])
predictionTrainContainer <- create_container(predTrainMatrix, labels=rep(0,
                                             predTrainSize), testSize=1:predTrainSize,
                                             virgin=FALSE)
resultsTrain <- classify_model(predictionTrainContainer, model)

predTestMatrix <- create_matrix(data$sentence[n.test:n.all], originalMatrix=dtMatrix)
predTestSize = length(data$sentence[n.test:n.all])
predictionTestContainer <- create_container(predTestMatrix, labels=rep(0, predTestSize)
                                            , testSize=1:predTestSize, virgin=FALSE)
resultsTest <- classify_model(predictionTestContainer, model)

train.pred <- as.double(as.character(resultsTrain$SVM_LABEL))
train.actual <- as.double(as.character(data$sentiment[1:n.train]))

test.pred <- as.double(as.character(resultsTest$SVM_LABEL))
test.actual <- as.double(as.character(data$sentiment[n.test:n.all]))

train.accuracy <- 1-rmse(train.pred, train.actual)
test.accuracy <- 1-rmse(test.pred, test.actual)
cat("Train: ", train.accuracy, "\nTest: ", test.accuracy, "\n")
```

A.2 Classificació

A.2.1 SVM

```
#Libraries
#devtools::install_github("rstudio/keras")
library(jsonlite)
library(reticulate)
#use_condaenv('tf-keras')
#library(keras)
library(purrr)
library(reshape2)
library(RNiftyReg)
library(tm)
library(RTextTools)
library(e1071)
library(dplyr)
library(caret)
library(Metrics)

#Sentiment class
setClass("Sentiment",
       representation(data="list",
                  dataTrain="list",
                  dataTest="list",
                  dataDev="list",
                  dataMix="list",
                  sentencesPath="character",
                  phrasesPath="character",
                  sentimentsPath="character",
                  splitPath="character"),
       prototype(data=data.frame(id=numeric(),
```

```
                          sentence=list(),
                          sentiment=double()),
                dataTrain=data.frame(id=numeric(),
                                     sentence=list(),
                                     sentiment=double()),
                dataTest=data.frame(id=numeric(),
                                    sentence=list(),
                                    sentiment=double()),
                dataDev=data.frame(id=numeric(),
                                   sentence=list(),
                                   sentiment=double()),
                dataMix=data.frame(id=numeric(),
                                   sentence=list(),
                                   sentiment=double()),
                sentencesPath="datasetSentences.txt",
                phrasesPath="dictionary.txt",
                sentimentsPath="sentiment_labels.txt",
                splitPath="datasetSplit.txt"))

setMethod("initialize", "Sentiment", function(.Object, ...) {
  #Check validity
  cat("Checking validity...\n")
  .Object <- callNextMethod()
  validObject(.Object)

  #Read sentences
  cat("Reading sentences...\n")
  sentences <- read.csv(.Object@sentencesPath, header=TRUE, sep="\t", colClasses=c('
                                              numeric', 'character'))
  colnames(sentences) <- c("id", "sentence")

  #Read phrases
  cat("Reading phrases...\n")
  phrases <- read.csv(.Object@phrasesPath, header=FALSE, sep="|", colClasses=c('
                                              character', 'numeric'))
  phrases <- phrases[, c(2, 1)]
  colnames(phrases) <- c("id", "phrase")

  #Read sentiments
  cat("Reading sentiments...\n")
  sentiments <- read.csv(.Object@sentimentsPath, header=TRUE, sep="|", colClasses=c('
                                              numeric', 'double'))
  colnames(sentiments) <- c("id", "sentiment")

  #Read split
  cat("Reading split...\n")
  split <- read.csv(.Object@splitPath, header=TRUE, sep=",", colClasses=c('numeric', '
                                              numeric'))
  colnames(split) <- c("id", "label")

  #Assign the sentiment to each sentence
  cat("Computing sentence sentiments...\n")
  for(i in 1:nrow(sentences)){
    try(.Object@data <- rbind(.Object@data, data.frame(id=sentences[i,]$id, sentence=I(
                                                strsplit(as.character(sentences[i,]$
                                                sentence), "\\s+")), sentiment=
                                                sentiments[which(sentiments$id ==
                                                phrases[which(phrases$phrase ==
                                                sentences[i,]$sentence),]$id),]$
                                                sentiment)), silent=TRUE)
  }

```

```
  #Split data
  cat("Splitting data...\n")
  for(i in 1:nrow(.Object@data)){
    label = split[which(split$id == .Object@data[i,]$id),]$label
    if(label==1){
      .Object@dataTrain <- rbind(.Object@dataTrain, .Object@data[i,])
    }
    else if(label==2){
      .Object@dataTest <- rbind(.Object@dataTest, .Object@data[i,])
    }
    else if(label==3){
      .Object@dataDev <- rbind(.Object@dataDev, .Object@data[i,])
    }
  }

  .Object@dataMix <- rbind(.Object@dataTrain, .Object@dataTest)

  #Return object
  .Object
})

toClasses <- function(data){
  dataClasses <- list()
  for(i in 1:length(data)){
    if((data[i]>=0)&&(data[i]<=0.2)){
      dataClasses[[i]] <- list(1, 0, 0, 0, 0)
    }
    else if((data[i]>0.2)&&(data[i]<=0.4)){
      dataClasses[[i]] <- list(0, 1, 0, 0, 0)
    }
    else if((data[i]>0.4)&&(data[i]<=0.6)){
      dataClasses[[i]] <- list(0, 0, 1, 0, 0)
    }
    else if((data[i]>0.6)&&(data[i]<=0.8)){
      dataClasses[[i]] <- list(0, 0, 0, 1, 0)
    }
    else if((data[i]>0.8)&&(data[i]<=1)){
      dataClasses[[i]] <- list(0, 0, 0, 0, 1)
    }
  }
  return(matrix(unlist(dataClasses), ncol=5, byrow=TRUE))
}

sentiment <- new("Sentiment")

data <- data.frame(sentiment = sentiment@dataMix$sentiment, sentence =
                                                  sentiment@dataMix$sentence)
for(i in 1:nrow(data)){
  data$sentence[i] <- paste(unlist(data$sentence[i]), collapse = ' ')
}
data$sentiment <- as.factor(data$sentiment)
data$sentence <- as.character(data$sentence)

n.train = nrow(sentiment@dataTrain)
n.test = n.train + 1
n.all = nrow(sentiment@dataMix)

dtMatrix <- create_matrix(data$sentence)

convert_class <- function(x) {
  y <- ifelse(x >= 0.8, 5, ifelse(x >= 0.6, 4, ifelse(x >= 0.4, 3, ifelse(x >= 0.2, 2,
                                                  1))))
```

```
  y <- factor(y, levels=c(1, 2, 3, 4, 5), labels=c("1", "2", "3", "4", "5"))
  y
}

dataSentiment <- sapply(as.numeric(as.character(data$sentiment)), convert_class)

container <- create_container(dtMatrix, dataSentiment, trainSize=1:n.train, virgin=
                                          FALSE)

model <- train_model(container, "SVM", kernel="linear", cost=1)

predTrainMatrix <- create_matrix(data$sentence[1:n.train], originalMatrix=dtMatrix)
predTrainSize = length(data$sentence[1:n.train])
predictionTrainContainer <- create_container(predTrainMatrix, labels=rep(0,
                                          predTrainSize), testSize=1:predTrainSize,
                                          virgin=FALSE)
resultsTrain <- classify_model(predictionTrainContainer, model)

predTestMatrix <- create_matrix(data$sentence[n.test:n.all], originalMatrix=dtMatrix)
predTestSize = length(data$sentence[n.test:n.all])
predictionTestContainer <- create_container(predTestMatrix, labels=rep(0, predTestSize)
                                          , testSize=1:predTestSize, virgin=FALSE)
resultsTest <- classify_model(predictionTestContainer, model)

train.pred <- as.double(as.character(resultsTrain$SVM_LABEL))
train.actual <- as.double(as.character(dataSentiment[1:n.train]))

test.pred <- as.double(as.character(resultsTest$SVM_LABEL))
test.actual <- as.double(as.character(dataSentiment[n.test:n.all]))

train.accuracy <- c()
for(i in 1:length(train.pred)){
  train.accuracy[i] <- ifelse(train.pred[i] == train.actual[i], 1, 0)
}
train.accuracy <- sum(train.accuracy)/length(train.pred)

test.accuracy <- c()
for(i in 1:length(test.pred)){
  test.accuracy[i] <- ifelse(test.pred[i] == test.actual[i], 1, 0)
}
test.accuracy <- sum(test.accuracy)/length(test.pred)

train.notprecise.accuracy <- c()
for(i in 1:length(train.pred)){
  train.notprecise.accuracy[i] <- ifelse(abs(train.pred[i] - train.actual[i]) <= 1, 1,
                                          0)
}
train.notprecise.accuracy <- sum(train.notprecise.accuracy)/length(train.pred)

test.notprecise.accuracy <- c()
for(i in 1:length(test.pred)){
  test.notprecise.accuracy[i] <- ifelse(abs(test.pred[i] - test.actual[i]) <= 1, 1, 0)
}
test.notprecise.accuracy <- sum(test.notprecise.accuracy)/length(test.pred)

cat("[EXACT CLASS]\nTrain: ", train.accuracy, "\nTest: ", test.accuracy, "\n[NEAR CLASS
                                          ]\nTrain: ", train.notprecise.accuracy, "\
                                          nTest: ", test.notprecise.accuracy, "\n")
```

A.2.2 Naive bayes

```
#Libraries
#devtools::install_github("rstudio/keras")
library(jsonlite)
library(reticulate)
#use_condaenv('tf-keras')
#library(keras)
library(purrr)
library(reshape2)
library(RNiftyReg)
library(tm)
library(RTextTools)
library(e1071)
library(dplyr)
library(caret)
library(Metrics)

#Sentiment class
setClass("Sentiment",
       representation(data="list",
                   dataTrain="list",
                   dataTest="list",
                   dataDev="list",
                   dataMix="list",
                   sentencesPath="character",
                   phrasesPath="character",
                   sentimentsPath="character",
                   splitPath="character"),
       prototype(data=data.frame(id=numeric(),
                             sentence=list(),
                             sentiment=double()),
               dataTrain=data.frame(id=numeric(),
                                 sentence=list(),
                                 sentiment=double()),
               dataTest=data.frame(id=numeric(),
                                sentence=list(),
                                sentiment=double()),
               dataDev=data.frame(id=numeric(),
                               sentence=list(),
                               sentiment=double()),
               dataMix=data.frame(id=numeric(),
                               sentence=list(),
                               sentiment=double()),
               sentencesPath="datasetSentences.txt",
               phrasesPath="dictionary.txt",
               sentimentsPath="sentiment_labels.txt",
               splitPath="datasetSplit.txt"))

setMethod("initialize", "Sentiment", function(.Object, ...) {
  #Check validity
  cat("Checking validity...\n")
  .Object <- callNextMethod()
  validObject(.Object)

  #Read sentences
  cat("Reading sentences...\n")
  sentences <- read.csv(.Object@sentencesPath, header=TRUE, sep="\t", colClasses=c('
                                                numeric', 'character'))
  colnames(sentences) <- c("id", "sentence")
```

```
58
59    #Read phrases
60    cat("Reading phrases...\n")
61    phrases <- read.csv(.Object@phrasesPath, header=FALSE, sep="|", colClasses=c('
                                                character', 'numeric'))
62    phrases <- phrases[, c(2, 1)]
63    colnames(phrases) <- c("id", "phrase")
64
65    #Read sentiments
66    cat("Reading sentiments...\n")
67    sentiments <- read.csv(.Object@sentimentsPath, header=TRUE, sep="|", colClasses=c('
                                                numeric', 'double'))
68    colnames(sentiments) <- c("id", "sentiment")
69
70    #Read split
71    cat("Reading split...\n")
72    split <- read.csv(.Object@splitPath, header=TRUE, sep=",", colClasses=c('numeric', '
                                                numeric'))
73    colnames(split) <- c("id", "label")
74
75    #Assign the sentiment to each sentence
76    cat("Computing sentence sentiments...\n")
77    for(i in 1:nrow(sentences)){
78      try(.Object@data <- rbind(.Object@data, data.frame(id=sentences[i,]$id, sentence=I(
                                                  strsplit(as.character(sentences[i,]$
                                                  sentence), "\\s+")), sentiment=
                                                  sentiments[which(sentiments$id ==
                                                  phrases[which(phrases$phrase ==
                                                  sentences[i,]$sentence),]$id),]$
                                                  sentiment)), silent=TRUE)
79    }
80
81    #Split data
82    cat("Splitting data...\n")
83    for(i in 1:nrow(.Object@data)){
84      label = split[which(split$id == .Object@data[i,]$id),]$label
85      if(label==1){
86        .Object@dataTrain <- rbind(.Object@dataTrain, .Object@data[i,])
87      }
88      else if(label==2){
89        .Object@dataTest <- rbind(.Object@dataTest, .Object@data[i,])
90      }
91      else if(label==3){
92        .Object@dataDev <- rbind(.Object@dataDev, .Object@data[i,])
93      }
94    }
95
96    .Object@dataMix <- rbind(.Object@dataTrain, .Object@dataTest)
97
98    #Return object
99    .Object
100 })
101
102 toClasses <- function(data){
103   dataClasses <- list()
104   for(i in 1:length(data)){
105     if((data[i]>=0)&&(data[i]<=0.2)){
106       dataClasses[[i]] <- 1
107     }
108     else if((data[i]>0.2)&&(data[i]<=0.4)){
109       dataClasses[[i]] <- 2
110     }
```

```
111     else if((data[i]>0.4)&&(data[i]<=0.6)){
112       dataClasses[[i]] <- 3
113     }
114     else if((data[i]>0.6)&&(data[i]<=0.8)){
115       dataClasses[[i]] <- 4
116     }
117     else if((data[i]>0.8)&&(data[i]<=1)){
118       dataClasses[[i]] <- 5
119     }
120   }
121   return(matrix(unlist(dataClasses), ncol=1, byrow=TRUE))
122 }
123
124 convert_count <- function(x) {
125   y <- ifelse(x > 0, 1, 0)
126   y <- factor(y, levels=c(0,1), labels=c("No", "Yes"))
127   y
128 }
129
130 convert_class <- function(x) {
131   y <- ifelse(x >= 0.8, 5, ifelse(x >= 0.6, 4, ifelse(x >= 0.4, 3, ifelse(x >= 0.2, 2,
                                                                              1))))
132   y <- factor(y, levels=c(1, 2, 3, 4, 5), labels=c("1", "2", "3", "4", "5"))
133   y
134 }
135
136 sentiment <- new("Sentiment")
137
138 data <- data.frame(sentiment = sentiment@dataMix$sentiment, sentence =
                                                    sentiment@dataMix$sentence)
139 for(i in 1:nrow(data)){
140   data$sentence[i] <- paste(unlist(data$sentence[i]), collapse = ' ')
141 }
142 data$sentiment <- sapply(as.numeric(as.character(data$sentiment)), convert_class)
143 data$sentence <- as.character(data$sentence)
144
145 corpus <- Corpus(VectorSource(data$sentence))
146 corpus
147
148 corpus.clean <- corpus %>%
149   tm_map(content_transformer(tolower)) %>%
150   tm_map(removePunctuation) %>%
151   tm_map(removeNumbers) %>%
152   tm_map(removeWords, stopwords(kind="en")) %>%
153   tm_map(stripWhitespace)
154
155 dtm <- DocumentTermMatrix(corpus.clean)
156
157 n.train = nrow(sentiment@dataTrain)
158 n.test = n.train + 1
159 n.all = nrow(sentiment@dataMix)
160
161 data.train <- data[1:n.train,]
162 data.test <- data[n.test:n.all,]
163
164 dtm.train <- dtm[1:n.train,]
165 dtm.test <- dtm[n.test:n.all,]
166
167 corpus.clean.train <- corpus.clean[1:n.train]
168 corpus.clean.test <- corpus.clean[n.test:n.all]
169
170 dim(dtm.train)
```

```
fivefreq <- findFreqTerms(dtm.train, lowfreq = 5)
length(fivefreq)

dtm.train.nb <- DocumentTermMatrix(corpus.clean.train, control=list(dictionary =
                                        fivefreq))
dim(dtm.train.nb)

dtm.test.nb <- DocumentTermMatrix(corpus.clean.test, control=list(dictionary = fivefreq
                                        ))
dim(dtm.test.nb)

trainNB <- apply(dtm.train.nb, 2, convert_count)
testNB <- apply(dtm.test.nb, 2, convert_count)

trainY <- data.train$sentiment
testY <- data.test$sentiment

system.time(classifier <- naiveBayes(trainNB, trainY, laplace = 0.5))
system.time(train.pred <- predict(classifier, newdata=trainNB))
system.time(test.pred <- predict(classifier, newdata=testNB))

train.pred <- as.numeric(as.character(train.pred))
train.actual <- as.numeric(as.character(trainY))

test.pred <- as.numeric(as.character(test.pred))
test.actual <- as.numeric(as.character(testY))

train.accuracy <- c()
for(i in 1:length(train.pred)){
  train.accuracy[i] <- ifelse(train.pred[i] == train.actual[i], 1, 0)
}
train.accuracy <- sum(train.accuracy)/length(train.pred)

test.accuracy <- c()
for(i in 1:length(test.pred)){
  test.accuracy[i] <- ifelse(test.pred[i] == test.actual[i], 1, 0)
}
test.accuracy <- sum(test.accuracy)/length(test.pred)

train.notprecise.accuracy <- c()
for(i in 1:length(train.pred)){
  train.notprecise.accuracy[i] <- ifelse(abs(train.pred[i] - train.actual[i]) <= 1, 1,
                                        0)
}
train.notprecise.accuracy <- sum(train.notprecise.accuracy)/length(train.pred)

test.notprecise.accuracy <- c()
for(i in 1:length(test.pred)){
  test.notprecise.accuracy[i] <- ifelse(abs(test.pred[i] - test.actual[i]) <= 1, 1, 0)
}
test.notprecise.accuracy <- sum(test.notprecise.accuracy)/length(test.pred)

cat("[EXACT CLASS]\nTrain: ", train.accuracy, "\nTest: ", test.accuracy, "\n[NEAR CLASS
                                        ]\nTrain: ", train.notprecise.accuracy, "\
                                        nTest: ", test.notprecise.accuracy, "\n")
```

A.2.3 Random forest

```
#Libraries
#devtools::install_github("rstudio/keras")
library(jsonlite)
library(reticulate)
use_condaenv('tf-keras')
library(keras)
library(purrr)
library(reshape2)
library(RNiftyReg)
library(tm)
library(RTextTools)
library(e1071)
library(dplyr)
library(caret)
library(Metrics)

#Sentiment class
setClass("Sentiment",
       representation(data="list",
                  dataTrain="list",
                  dataTest="list",
                  dataDev="list",
                  dataMix="list",
                  sentencesPath="character",
                  phrasesPath="character",
                  sentimentsPath="character",
                  splitPath="character"),
       prototype(data=data.frame(id=numeric(),
                            sentence=list(),
                            sentiment=double()),
              dataTrain=data.frame(id=numeric(),
                              sentence=list(),
                              sentiment=double()),
              dataTest=data.frame(id=numeric(),
                             sentence=list(),
                             sentiment=double()),
              dataDev=data.frame(id=numeric(),
                            sentence=list(),
                            sentiment=double()),
              dataMix=data.frame(id=numeric(),
                            sentence=list(),
                            sentiment=double()),
              sentencesPath="datasetSentences.txt",
              phrasesPath="dictionary.txt",
              sentimentsPath="sentiment_labels.txt",
              splitPath="datasetSplit.txt"))

setMethod("initialize", "Sentiment", function(.Object, ...) {
  #Check validity
  cat("Checking validity...\n")
  .Object <- callNextMethod()
  validObject(.Object)

  #Read sentences
  cat("Reading sentences...\n")
  sentences <- read.csv(.Object@sentencesPath, header=TRUE, sep="\t", colClasses=c('
                                          numeric', 'character'))
  colnames(sentences) <- c("id", "sentence")

  #Read phrases
  cat("Reading phrases...\n")
```

```
61    phrases <- read.csv(.Object@phrasesPath, header=FALSE, sep="|", colClasses=c('
                                                  character', 'numeric'))
62    phrases <- phrases[, c(2, 1)]
63    colnames(phrases) <- c("id", "phrase")
64
65    #Read sentiments
66    cat("Reading sentiments...\n")
67    sentiments <- read.csv(.Object@sentimentsPath, header=TRUE, sep="|", colClasses=c('
                                                  numeric', 'double'))
68    colnames(sentiments) <- c("id", "sentiment")
69
70    #Read split
71    cat("Reading split...\n")
72    split <- read.csv(.Object@splitPath, header=TRUE, sep=",", colClasses=c('numeric', '
                                                  numeric'))
73    colnames(split) <- c("id", "label")
74
75    #Assign the sentiment to each sentence
76    cat("Computing sentence sentiments...\n")
77    for(i in 1:nrow(sentences)){
78      try(.Object@data <- rbind(.Object@data, data.frame(id=sentences[i,]$id, sentence=I(
                                                    strsplit(as.character(sentences[i,]$
                                                    sentence), "\\s+")), sentiment=
                                                    sentiments[which(sentiments$id ==
                                                    phrases[which(phrases$phrase ==
                                                    sentences[i,]$sentence),]$id),]$
                                                    sentiment)), silent=TRUE)
79    }
80
81    #Split data
82    cat("Splitting data...\n")
83    for(i in 1:nrow(.Object@data)){
84      label = split[which(split$id == .Object@data[i,]$id),]$label
85      if(label==1){
86        .Object@dataTrain <- rbind(.Object@dataTrain, .Object@data[i,])
87      }
88      else if(label==2){
89        .Object@dataTest <- rbind(.Object@dataTest, .Object@data[i,])
90      }
91      else if(label==3){
92        .Object@dataDev <- rbind(.Object@dataDev, .Object@data[i,])
93      }
94    }
95
96    .Object@dataMix <- rbind(.Object@dataTrain, .Object@dataTest)
97
98    #Return object
99    .Object
100 })
101
102 toClasses <- function(data){
103   dataClasses <- list()
104   for(i in 1:length(data)){
105     if((data[i]>=0)&&(data[i]<=0.2)){
106       dataClasses[[i]] <- list(1, 0, 0, 0, 0)
107     }
108     else if((data[i]>0.2)&&(data[i]<=0.4)){
109       dataClasses[[i]] <- list(0, 1, 0, 0, 0)
110     }
111     else if((data[i]>0.4)&&(data[i]<=0.6)){
112       dataClasses[[i]] <- list(0, 0, 1, 0, 0)
113     }
```

```
114     else if((data[i]>0.6)&&(data[i]<=0.8)){
115       dataClasses[[i]] <- list(0, 0, 0, 1, 0)
116     }
117     else if((data[i]>0.8)&&(data[i]<=1)){
118       dataClasses[[i]] <- list(0, 0, 0, 0, 1)
119     }
120   }
121   return(matrix(unlist(dataClasses), ncol=5, byrow=TRUE))
122 }
123
124 sentiment <- new("Sentiment")
125
126 data <- data.frame(sentiment = sentiment@dataMix$sentiment, sentence =
                                                  sentiment@dataMix$sentence)
127 for(i in 1:nrow(data)){
128   data$sentence[i] <- paste(unlist(data$sentence[i]), collapse = ' ')
129 }
130 data$sentiment <- as.factor(data$sentiment)
131 data$sentence <- as.character(data$sentence)
132
133 n.train = nrow(sentiment@dataTrain)
134 n.test = n.train + 1
135 n.all = nrow(sentiment@dataMix)
136
137 dtMatrix <- create_matrix(data$sentence)
138
139 convert_class <- function(x) {
140   y <- ifelse(x >= 0.8, 5, ifelse(x >= 0.6, 4, ifelse(x >= 0.4, 3, ifelse(x >= 0.2, 2,
                                                    1))))
141   y <- factor(y, levels=c(1, 2, 3, 4, 5), labels=c("1", "2", "3", "4", "5"))
142   y
143 }
144
145 dataSentiment <- sapply(as.numeric(as.character(data$sentiment)), convert_class)
146
147 container <- create_container(dtMatrix, dataSentiment, trainSize=1:n.train, virgin=
                                                  FALSE)
148
149 model <- train_model(container, "RF")
150
151 predTrainMatrix <- create_matrix(data$sentence[1:n.train], originalMatrix=dtMatrix)
152 predTrainSize = length(data$sentence[1:n.train])
153 predictionTrainContainer <- create_container(predTrainMatrix, labels=rep(0,
                                                  predTrainSize), testSize=1:predTrainSize,
                                                  virgin=FALSE)
154 resultsTrain <- classify_model(predictionTrainContainer, model)
155
156 predTestMatrix <- create_matrix(data$sentence[n.test:n.all], originalMatrix=dtMatrix)
157 predTestSize = length(data$sentence[n.test:n.all])
158 predictionTestContainer <- create_container(predTestMatrix, labels=rep(0, predTestSize)
                                                  , testSize=1:predTestSize, virgin=FALSE)
159 resultsTest <- classify_model(predictionTestContainer, model)
160
161 train.pred <- as.double(as.character(resultsTrain$SVM_LABEL))
162 train.actual <- as.double(as.character(dataSentiment[1:n.train]))
163
164 test.pred <- as.double(as.character(resultsTest$SVM_LABEL))
165 test.actual <- as.double(as.character(dataSentiment[n.test:n.all]))
166
167 train.accuracy <- c()
168 for(i in 1:length(train.pred)){
169   train.accuracy[i] <- ifelse(train.pred[i] == train.actual[i], 1, 0)
```

```
}
train.accuracy <- sum(train.accuracy)/length(train.pred)

test.accuracy <- c()
for(i in 1:length(test.pred)){
  test.accuracy[i] <- ifelse(test.pred[i] == test.actual[i], 1, 0)
}
test.accuracy <- sum(test.accuracy)/length(test.pred)

train.notprecise.accuracy <- c()
for(i in 1:length(train.pred)){
  train.notprecise.accuracy[i] <- ifelse(abs(train.pred[i] - train.actual[i]) <= 1, 1,
                                         0)
}
train.notprecise.accuracy <- sum(train.notprecise.accuracy)/length(train.pred)

test.notprecise.accuracy <- c()
for(i in 1:length(test.pred)){
  test.notprecise.accuracy[i] <- ifelse(abs(test.pred[i] - test.actual[i]) <= 1, 1, 0)
}
test.notprecise.accuracy <- sum(test.notprecise.accuracy)/length(test.pred)

cat("[EXACT CLASS]\nTrain: ", train.accuracy, "\nTest: ", test.accuracy, "\n[NEAR CLASS
                                        ]\nTrain: ", train.notprecise.accuracy, "\
                                        nTest: ", test.notprecise.accuracy, "\n")
```

A.2.4 RNN

```
#Libraries
library(jsonlite)
library(reticulate)
use_condaenv('tf-keras')
library(keras)
library(purrr)
library(reshape2)
library(RNiftyReg)
library(nnet)

#Sentiment class
setClass("Sentiment",
       representation(data="list",
                  dataTrain="list",
                  dataTest="list",
                  dataDev="list",
                  sentencesPath="character",
                  phrasesPath="character",
                  sentimentsPath="character",
                  splitPath="character"),
       prototype(data=data.frame(id=numeric(),
                         sentence=list(),
                         sentiment=double()),
              dataTrain=data.frame(id=numeric(),
                           sentence=list(),
                           sentiment=double()),
              dataTest=data.frame(id=numeric(),
                          sentence=list(),
                          sentiment=double()),
              dataDev=data.frame(id=numeric(),
```

```
                              sentence=list(),
                              sentiment=double()),
                sentencesPath="datasetSentences.txt",
                phrasesPath="dictionary.txt",
                sentimentsPath="sentiment_labels.txt",
                splitPath="datasetSplit.txt"))

setMethod("initialize", "Sentiment", function(.Object, ...) {
  #Check validity
  cat("Checking validity...\n")
  .Object <- callNextMethod()
  validObject(.Object)

  #Read sentences
  cat("Reading sentences...\n")
  sentences <- read.csv(.Object@sentencesPath, header=TRUE, sep="\t", colClasses=c('
                                                numeric', 'character'))
  colnames(sentences) <- c("id", "sentence")

  #Read phrases
  cat("Reading phrases...\n")
  phrases <- read.csv(.Object@phrasesPath, header=FALSE, sep="|", colClasses=c('
                                                character', 'numeric'))
  phrases <- phrases[, c(2, 1)]
  colnames(phrases) <- c("id", "phrase")

  #Read sentiments
  cat("Reading sentiments...\n")
  sentiments <- read.csv(.Object@sentimentsPath, header=TRUE, sep="|", colClasses=c('
                                                numeric', 'double'))
  colnames(sentiments) <- c("id", "sentiment")

  #Read split
  cat("Reading split...\n")
  split <- read.csv(.Object@splitPath, header=TRUE, sep=",", colClasses=c('numeric', '
                                                numeric'))
  colnames(split) <- c("id", "label")

  #Assign the sentiment to each sentence
  cat("Computing sentence sentiments...\n")
  for(i in 1:nrow(sentences)){
    try(.Object@data <- rbind(.Object@data, data.frame(id=sentences[i,]$id, sentence=I(
                                                  strsplit(as.character(sentences[i,]$
                                                  sentence), "\\s+")), sentiment=
                                                  sentiments[which(sentiments$id ==
                                                  phrases[which(phrases$phrase ==
                                                  sentences[i,]$sentence),]$id),]$
                                                  sentiment)), silent=TRUE)
  }

  #Split data
  cat("Splitting data...\n")
  for(i in 1:nrow(.Object@data)){
    label = split[which(split$id == .Object@data[i,]$id),]$label
    if(label==1){
      .Object@dataTrain <- rbind(.Object@dataTrain, .Object@data[i,])
    }
    else if(label==2){
      .Object@dataTest <- rbind(.Object@dataTest, .Object@data[i,])
    }
    else if(label==3){
      .Object@dataDev <- rbind(.Object@dataDev, .Object@data[i,])
```

```
    }
  }

  #Return object
  .Object
})

toClasses <- function(data){
  dataClasses <- list()
  for(i in 1:length(data)){
    if((data[i]>=0)&&(data[i]<=0.2)){
      dataClasses[[i]] <- list(1, 0, 0, 0, 0)
    }
    else if((data[i]>0.2)&&(data[i]<=0.4)){
      dataClasses[[i]] <- list(0, 1, 0, 0, 0)
    }
    else if((data[i]>0.4)&&(data[i]<=0.6)){
      dataClasses[[i]] <- list(0, 0, 1, 0, 0)
    }
    else if((data[i]>0.6)&&(data[i]<=0.8)){
      dataClasses[[i]] <- list(0, 0, 0, 1, 0)
    }
    else if((data[i]>0.8)&&(data[i]<=1)){
      dataClasses[[i]] <- list(0, 0, 0, 0, 1)
    }
  }
  return(matrix(unlist(dataClasses), ncol=5, byrow=TRUE))
}


toClass <- function(data){
  dataClasses <- list()
  for(i in 1:length(data)){
    if((data[i]>=0)&&(data[i]<0.5)){
      dataClasses[[i]] <- 0
    }
    else if((data[i]>=0.5)&&(data[i]<=1)){
      dataClasses[[i]] <- 1
    }
  }
  return(matrix(unlist(dataClasses), ncol=1, byrow=TRUE))
}


#setwd("~/stanfordSentimentTreebank")

sentiment <- new("Sentiment")

vocabularySize <- 100000
tokenizer <- text_tokenizer(num_words=vocabularySize)
tokenizer %>% fit_text_tokenizer(unlist(sentiment@data$sentence))

maxLength <- max(lengths(sentiment@data$sentence))

x_train <- tokenizer %>% texts_to_sequences(sentiment@dataTrain$sentence)
x_train <- pad_sequences(x_train, maxLength)

x_test <- tokenizer %>% texts_to_sequences(sentiment@dataTest$sentence)
x_test <- pad_sequences(x_test, maxLength)

y_train <- matrix(sentiment@dataTrain$sentiment)
y_train <- toClass(y_train)
```

```
145
146 y_test <- matrix(sentiment@dataTest$sentiment)
147 y_test <- toClass(y_test)
148
149 model <- keras_model_sequential()
150 model %>%
151   layer_embedding(
152     input_dim = vocabularySize,
153     output_dim = 32,
154     name = "embedding") %>%
155   layer_dropout(0.5) %>%
156   layer_lstm(
157     units = 32,
158     dropout = 0.5,
159     recurrent_dropout = 0.5,
160     return_sequences = FALSE
161   ) %>%
162   layer_dense(
163     units = 1,
164     activation = "sigmoid"
165   )
166
167 model %>% summary()
168
169 model %>% compile(
170   optimizer = 'adam',
171   loss = 'binary_crossentropy',
172   metrics = list('accuracy')
173 )
174
175 history <- model %>% fit(
176   x_train,
177   y_train,
178   epochs = 5,
179   batch_size = 8,
180   #validation_split = 0.1,
181   validation_data = list(x_test, y_test),
182   verbose=1
183 )
184
185 train.pred <- model %>% predict(x_train)
186 train.actual <- y_train
187
188 test.pred <- model %>% predict(x_test)
189 test.actual <- y_test
190
191 train.accuracy <- c()
192 for(i in 1:nrow(train.pred)){
193   train.accuracy[i] <- ifelse(which.is.max(train.pred[i,]) == which.is.max(train.actual
                                                    [i,]), 1, 0)
194 }
195 train.accuracy <- sum(train.accuracy)/nrow(train.pred)
196
197 test.accuracy <- c()
198 for(i in 1:nrow(test.pred)){
199   test.accuracy[i] <- ifelse(which.is.max(test.pred[i,]) == which.is.max(test.actual[i
                                                   ,]), 1, 0)
200 }
201 test.accuracy <- sum(test.accuracy)/nrow(test.pred)
202
203 cat("Train: ", train.accuracy, "\nTest: ", test.accuracy, "\n")
```