Επεξεργασία Φυσικής Γλώσσας – 2022/20223 - Ο.Π.Α

Εργασία 3
Φοίβος-Αστέριος Νταντάμης (f3312204) (phivos93@yahoo.com)

Άσκηση 9 notebook url:
https://colab.research.google.com/drive/1_qvSOYCJyMEG4u0OF5JtmVKuTPbjpIiY?usp=sharing

Περιγραφή Dataset

Το dataset όπως και η προεπεξεργασία του είναι τα ίδια με την προηγούμενη άσκηση. Για λόγους πληρότητας ωστόσο περιγράφουμε συνοπτικά ξανά τη διαδικασία.

Χρησιμοποιούμε το Cornell Movie Review Data από το ομώνυμο πανεπιστήμιο. Το dataset αυτό περιέχει κριτικές ταινιών από το αρχείο του IMDb κατηγοριοποιημένες ως θετικές(pos) ή αρνητικές(neg). Το μέγεθος του συνόλου όλων των κριτικών είναι 2000, στις οποίες οι 1000 ανήκουν στην κατηγορία pos και οι υπόλοιπες 1000 στην κατηγορία neg.

```
[3]  !wget http://www.cs.cornell.edu/people/pabo/movie-review-data/review_polarity.tar.gz
     !tar xvzf review_polarity.tar.gz

     from sklearn.datasets import load_files
     movies = load_files(container_path="txt_sentoken", encoding="utf-8")
     x, y, target_names = movies.data, movies.target, movies.target_names
```

Ο μέσος όρος χαρακτήρων ανά κριτική είναι 3893.002, ενώ ο μέσος όρο των λέξεων είναι 746.3405.

```
char_sum = word_sum = n = 0

for i in x:
  char_sum += len(i)
  word_sum += len(i.split())
  n += 1

avg_chars = char_sum/n
avg_words = word_sum/n

print(f'Average characters per document (including whitespacs characeter): {avg_chars}')
print(f'Average words per document: {avg_words}')
```

```
Average characters per document (including whitespacs characeter): 3893.002
Average words per document: 746.3405
```

Προεπεξεργασία

Το πρώτο βήμα που εφαρμόζουμε είναι να αφαιρέσουμε τα stopwords της αγγλικής γλώσσας, φιλτράροντας όλα τα κείμενα για τις λέξεις που θεωρεί το nltk ως stopwords.

Στη συνέχεια με τη βοήθεια του WordNetLemmatizer αντικαθιστούμε όλες τις λέξεις με τα λήμματά τους.

Συνεχίζοντας, μετατρέπουμε όλους τους χαρακτήρες σε μικρούς.

Σε αυτήν την φάση έχουν προκύψει λέξεις οι οποίες ανήκουν στα stopwords, οπότε θα ξαναφιλτράρουμε τα κείμενα.
Έπειτα αφαιρούμε τα σημεία στίξης και τα σύμβολα που ουσιαστικά δεν αποτελούν κάποια λέξη.

Αφαιρούμε όλες τις λέξεις που αποτελούνται από έναν χαρακτήρα μόνο, μιας και δεν προσφέρουν κάποια ιδιαίτερη πληροφορία για την κατηγοριοποίηση.

Σε αυτό το σημείο και πάλι έχουν προκύψει κάποιες λέξεις που είναι stopwords οπότε φιλτράρουμε εκ νέου.

Αφαιρούμε τώρα όλους τους αριθμούς.

Αφαιρούμε τέλος τα honorifics

Διαχωρισμός Dataset

Σπάσαμε το Dataset σε Train, Development και Test sets με ποσοστά 70%, 20%, 10% αντίστοιχα. Προκύπτουν έτσι το Train Dataset με 1400 κριτικές και μέγεθος λεξιλογίου 30635. Το Development Dataset με 400 κριτικές και μέγεθος λεξιλογίου 17726. Το Testing Dataset με 200 κριτικές και μέγεθος λεξιλογίου 12148.

```
from sklearn.model_selection import train_test_split

x_train, x_temp, y_train, y_temp = train_test_split(docs, y, test_size=0.3, random_state=17)
x_dev, x_test, y_dev, y_test = train_test_split(x_temp, y_temp, test_size=0.333, random_state=25)

print('Sizes of train, dev and test sets are:')
print(len(y_train), len(y_dev), len(y_test))

train_vocab = set()
dev_vocab = set()
test_vocab = set()

for i in x_train:
  for j in i.split():
    train_vocab.add(j)

for i in x_dev:
  for j in i.split():
    dev_vocab.add(j)

for i in x_test:
  for j in i.split():
    test_vocab.add(j)

print(f'Train set vocabulary size is: {len(train_vocab)}')
print(f'Dev set vocabulary size is: {len(dev_vocab)}')
print(f'Test set vocabulary size is: {len(test_vocab)}')
```

```
Sizes of train, dev and test sets are:
1400 400 200
Train set vocabulary size is: 30635
Dev set vocabulary size is: 17726
Test set vocabulary size is: 12148
```

Feature Exctraction

Προκειμένου να μετατρέψουμε τα κείμενα σε διανύσματα, χρησιμοποιούμε TF-IDF features.

```
# Use unigram & bi-gram tf*idf features
vectorizer = TfidfVectorizer(
    ngram_range = (1, 2),
    max_features = 5000,
    #Apply sublinear tf scaling, i.e. replace tf with 1 + log(tf).
    sublinear_tf = True,
)


# Fit Vectorizer on train data
# Transform on all data (train - test)
x_train_tfidf = vectorizer.fit_transform(x_train)
x_dev_tfidf = vectorizer.transform(x_dev)
x_test_tfidf = vectorizer.transform(x_test)

print('Shape of training data: {}'.format(x_train_tfidf.shape))
print('Shape of dev data: {}'.format(x_dev_tfidf.shape))
print('Shape of test data: {}\n'.format(x_test_tfidf.shape))
```

```
Shape of training data: (1400, 5000)
Shape of dev data: (400, 5000)
Shape of test data: (200, 5000)
```

Ορίζουμε τώρα τη συνάρτηση get_ig_features, η οποία δέχεται ως όρισμα τον αριθμό των features που θέλουμε να κρατήσουμε και επιλέγει εκείνα για τα οποία μεγιστοποιείται το information gain.

```
def get_ig_features(k): # k = Number of features to keep
# The function relies on nonparametric methods
# based on entropy estimation from k-nearest neighbors distances
  mutual_information = mutual_info_classif(x_dev_tfidf_array, y_dev, n_neighbors=3,
                                           random_state=4321)


  # Indexes of the feature columns
  indexes = np.arange(len(mutual_information))

  # Sort by mutual information values --> (value, idx)
  sorted_indexes = sorted(zip(mutual_information, indexes), reverse=True)
  print('Sorted indexes: {}'.format(sorted_indexes[:4]))

  # keep the indexes of the best k features
  best_idxs = [idx for val, idx in sorted_indexes[:k]]
  print('Best indexes: {}\n'.format(best_idxs[:4]))

  # Keep only the colums of the best features
  x_train_tfidf_ig = x_train_tfidf_array[:, best_idxs]
  x_dev_tfidf_ig = x_dev_tfidf_array[:, best_idxs]
  x_test_tfidf_ig = x_test_tfidf_array[:, best_idxs]
  return (x_train_tfidf_ig, x_dev_tfidf_ig, x_test_tfidf_ig)
```

Ως σημείο αναφοράς παίρνουμε τα αποτελέσματα του LogisticRegression με τις καλύτερες παραμέτρους όπως προέκυψαν στην προηγούμενη άσκηση.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
import time

train, dev, test = get_ig_features(2500)

clf = LogisticRegression(solver="saga", penalty='elasticnet', l1_ratio=0.5, C=0.5)
start_time = time.time()
clf.fit(train, y_train)
print("Training took: {} seconds \n".format(time.time() - start_time))
```

```
predictions_train = clf.predict(train)
print(classification_report(y_train, predictions_train, target_names=target_names))
```

```
              precision    recall  f1-score   support

         neg       0.84      0.67      0.74       690
         pos       0.73      0.87      0.79       710

    accuracy                           0.77      1400
   macro avg       0.78      0.77      0.77      1400
weighted avg       0.78      0.77      0.77      1400
```

MLP

Κλάση callback

Αρχίζοντας να χτίζουμε το MLP μοντέλο μας, ορίζουμε αρχικά μία callback συνάρτηση, η οποία θα καλείται στο τέλος της κάθε εποχής του νευρωνικού μας δικτύου και θα υπολογίζει τα f1, precision και recall.

```
import numpy as np
import os
import tensorflow as tf
from sklearn.metrics import f1_score, recall_score, precision_score


class Metrics(tf.keras.callbacks.Callback):
    def __init__(self, valid_data):
        super(Metrics, self).__init__()
        self.validation_data = valid_data

    def on_epoch_end(self, epoch, logs=None):
        logs = logs or {}
        val_predict = np.argmax(self.model.predict(self.validation_data[0]), -1)
        val_targ = self.validation_data[1]

        if len(val_targ.shape) == 2 and val_targ.shape[1] != 1:
            val_targ = np.argmax(val_targ, -1)
        val_targ = tf.cast(val_targ,dtype=tf.float32)

        _val_f1 = f1_score(val_targ, val_predict,average="weighted")
        _val_recall = recall_score(val_targ, val_predict,average="weighted")
        _val_precision = precision_score(val_targ, val_predict,average="weighted")

        logs['val_f1'] = _val_f1
        logs['val_recall'] = _val_recall
        logs['val_precision'] = _val_precision
        print(" — val_f1: %f — val_precision: %f — val_recall: %f" % (_val_f1, _val_precision, _val_recall))
        return
```

Συνεχίζοντας, οδηγούμενοι από τον τρόπο που ζητάνε τις κατηγορίες οι συναρτήσεις του tensorflow, δημιουργούμε 1-hot vectors για τις δύο κλάσεις του προβλήματός μας.

```
from sklearn.preprocessing import LabelBinarizer

lb = LabelBinarizer()
target_list = target_names

y_train_1_hot = lb.fit_transform([target_list[x] for x in y_train])

y_dev_1_hot = lb.transform([target_list[x] for x in y_dev])
```

Στη συνεχεία ορίζουμε μία παραμετροποιήσιμη συνάρτηση στην οποία θα υλοποιείται το βασικό pipeline του αλγορίθμου που σχεδιάζουμε. Η συνάρτηση αυτή δέχεται τα εξής ορίσματα

number_of_features_ig: ο αριθμός των features που θέλουμε να κρατήσουμε σύμφωνα με το information gain

activation: το είδος της συνάρτησης ενεργοποίησης για τα κρυφά επίπεδα του νευρωνικού μας δικτύου

input_size: ο αριθμός των νευρώνων στο επίπεδο εισόδου του νευρωνικού μας δικτύου

hidden_layers: το πλήθος των κρυφών επιπέδων που θα περιέχεται στο νευρωνικό μας δίκτυο

hidden_layers_size: ο αριθμός των νευρώνων στα κρυφά επίπεδα του νευρωνικού μας δικτύου

dropout_prob: η πιθανότητα με την οποία κάθε νευρώνας κάνει drop out

batch_size: το μέγεθος του batch που χρησιμοποιείται για την εκπαίδευση

Μέσα στη συνάρτηση χτίζουμε το νευρωνικό μας δίκτυο ανάλογα με τις παραμέτρους. Λόγω του ότι το πρόβλημα είναι δυαδικό, το επίπεδο εξόδου χρησιμοποιεί sigmoid συνάρτηση ενεργοποίησης , όπως επίσης για loss χρησιμοποιείται η binary_crossentropy.

```
def myModel(number_of_features_ig, activation,input_size, hidden_layers, hidden_layers_size, dropout_prob, batch_size):
  train, dev, test = get_ig_features(number_of_features_ig)

  model = Sequential()
  model.add(Dense(input_size, input_dim=train.shape[1] , activation=activation))
  model.add(Dropout(dropout_prob))

  for i in range(hidden_layers):
    model.add(Dense(hidden_layers_size, activation=activation))
    model.add(Dropout(dropout_prob))

  model.add(Dense(1,  activation='sigmoid'))

  # print(model.summary())

  model.compile(
    loss='binary_crossentropy',
    optimizer=SGD(learning_rate=0.01),
    metrics=["accuracy"]
    )

  if not os.path.exists('./checkpoints'):
    os.makedirs('./checkpoints')
```

```
checkpoint = ModelCheckpoint(
  'checkpoints/weights.hdf5',
  monitor='val_accuracy',
  mode='max',
  verbose=2,
  save_best_only=True,
  save_weights_only=True
  )

start_training_time = time.time()
history = model.fit(
    train,
    y_train_1_hot,
    validation_data=(dev, y_dev_1_hot),
    batch_size=batch_size,
    epochs=40,
    shuffle=True,
    callbacks=[Metrics(valid_data=(dev, y_dev_1_hot)), checkpoint]
    )
end_training_time = time.time()

print(f'\nTraining time: {time.strftime("%H:%M:%S", time.gmtime(end_training_time - start_training_time))} sec\n')

return history, model
```

Στη συνέχεια ορίζουμε τον χώρο αναζήτησης στον οποίο θα ψάξουμε τις βέλτιστες τιμές των υπερπαραμέτρων. Ο χώρος είναι λίγο περιορισμένος διότι έτσι είναι και ο χρόνος :) . Το collab από ό,τι φαίνεται έχει περιορισμούς στην χρήση της gpu, οπότε η διαδικασία αυτή επαναλήφθηκε

αρκετές φορές με αποτέλεσμα να μειώσουμε το εύρος αναζήτησης ώστε να βγει κάποιο αποτέλεσμα.

```
number_of_features_ig = [2500, 3000]
activation = ['relu', 'tanh']
input_size = [256, 512]
hidden_layers = [0, 1, 2]
dropout_prob = [0.4, 0.5]
batch_size = [1]
```

Συνεχίζοντας, με brute force θα ψάξουμε τον καλύτερο συνδυασμό υπερπαραμέτρων. Όλες οι παράμετροι επιτρέπεται να πάρουν όλες τις διαθέσιμες τιμές, εκτός από μέγεθος των κρυφών επιπέδων που στη βιβλιογραφία αναφέρεται ότι καλό είναι να κυμαίνεται μεταξύ του μεγέθους του επιπέδου εισόδου και το μέγεθος το επιπέδου εξόδου. Μετά το τέλος κάθε επανάληψης αποθηκεύουμε το καλύτερο μοντέλο με κριτήριο το accuracy.

```
best_acc = 0.0

start_searching_time = time.time()
for n in number_of_features_ig:
  for a in activation:
    for i in input_size:
      for h in hidden_layers:
        for hs in filter(lambda ins : ins <= i, input_size):
          for d in dropout_prob:
            for b in batch_size:
              history, model = myModel(n, a, i, h, hs, d, b)
              if history.history['accuracy'][-1] > best_acc:
                best_acc = history.history['accuracy'][-1]
                model.save('./checkpoints/best_model')
                best_n = n
                best_a = a
                best_i = i
                best_h = h
                best_hs = hs
                best_d = d
                best_b = b

end_searching_time = time.time()
print(f'\nSearching time: {time.strftime("%H:%M:%S", time.gmtime(end_searching_time - start_searching_time))} sec\n')
print(f'\n Best parameters: n{best_n}, a{best_a}, i{best_i}, h{best_h}, hs{best_hs}, d{best_d}, b{best_b}')
```

Από ό,τι φαίνεται το brute force δεν μπορεί να χρησιμοποιηθεί στο collab, μιας και πάλι σταμάτησε την εκτέλεση λόγω χρησιμοποίησης πόρων. Θα χρησιμοποιήσουμε για τον λόγο αυτόν ένα από ενδιάμεσα αποτελέσματα και όχι το πραγματικά καλύτερο, ώστε να μπορούμε να συνεχίσουμε.

Παραθέτουμε τα αποτελέσματα των μετρικών των προβλέψεων ώστε να συγκριθούν με τα αντίστοιχα του logisticRegression. Παραθέτουμε εδώ τόσο αυτά τα αποτελέσματα όσο και του Dummy Classifier που ζητούνται

Scores for Test

```
df = pd.DataFrame([[precision_test_dummy_boolean[0], precision_test_dummy_boolean[1], recall_test_dummy_boolean[0], recall_test_dummy_boolean[1], f1_test_dummy_bool
                   [precision_test[0], precision_test[1], recall_test[0], recall_test[1], f1_test[0], f1_test[1], auc_test_pos, auc_test_neg, macro_precision_test,
                   [precision_test_tfidf[0], precision_test_tfidf[1], recall_test_tfidf[0], recall_test_tfidf[1], f1_test_tfidf[0], f1_test_tfidf[1], auc_test_pos_t
                  index=pd.Index(['Dummy Classifier', 'Logistic Regression (Boolean)', 'Logistic Regression (Tf-Idf)'], name='Model:'),
                  columns=pd.MultiIndex.from_product([['Precision', 'Recall', 'F1', 'AUC', 'Macro Precision', 'Macro Recall', 'Macro F1', 'Macro AUC'],['Positive',
df.style
```

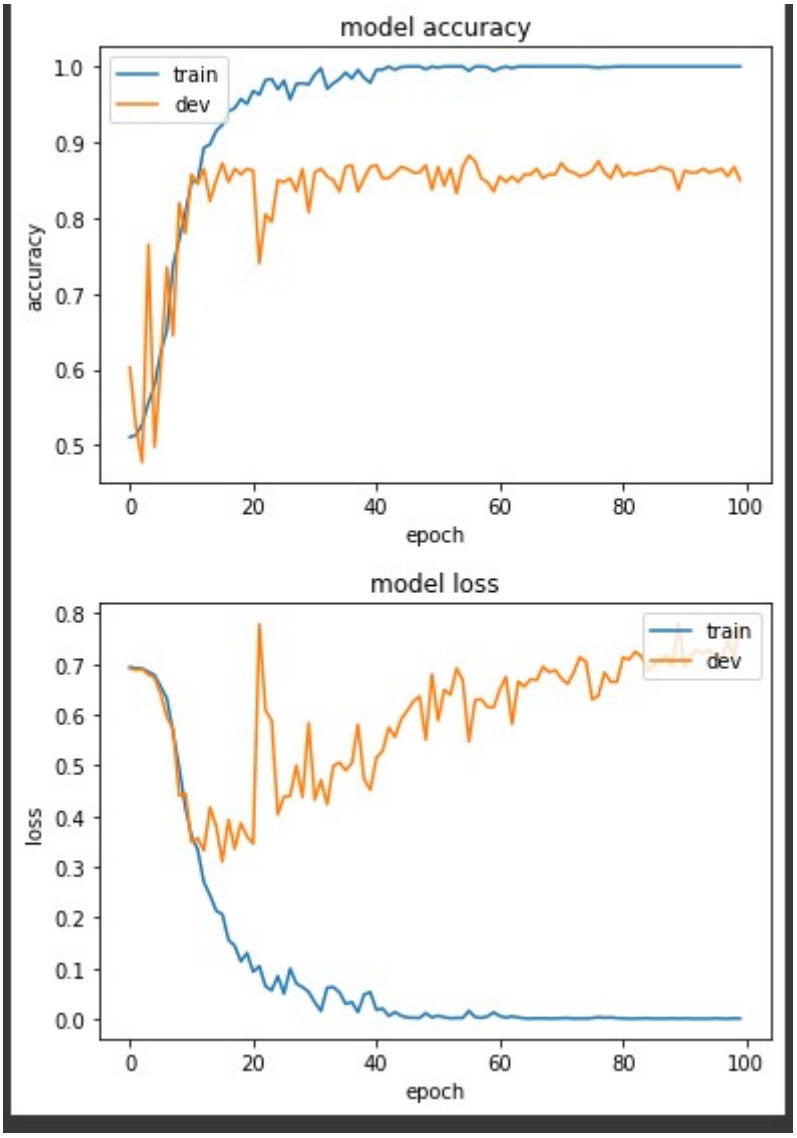
Scores:	Precision		Recall		F1		AUC		Macro Precision		Macro Recall		Macro F1		Macro AUC	
Class:	Positive	Negative	Positive	Negative	Positive	Negative	Positive	Negative	Positive	Negative	Positive	Negative	Positive	Negative	Positive	Negative
Model:																
Dummy Classifier	0.000000	0.495000	0.000000	1.000000	0.000000	0.662207	0.747500	0.752500	0.247500	0.247500	0.500000	0.500000	0.331104	0.331104	0.750000	0.750000
Logistic Regression (Boolean)	0.849462	0.794393	0.782178	0.858586	0.814433	0.825243	0.909950	0.321752	0.821927	0.821927	0.820382	0.820382	0.819838	0.819838	0.615851	0.615851
Logistic Regression (Tf-Idf)	0.590909	0.537313	0.386139	0.727273	0.467066	0.618026	0.574792	0.445113	0.564111	0.564111	0.556706	0.556706	0.542546	0.542546	0.509953	0.509953

Τέλος σχεδιάζουμε τις καμπύλες για το accuracy και loss συναρτήσει του αριθμού των εποχών για τα training και development δεδομένα.

```
%matplotlib inline
import matplotlib.pyplot as plt

# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'dev'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'dev'], loc='upper right')
plt.show()
```



Παρατηρούμε ότι περίπου στην εποχή 12 είναι το βέλτιστο σημείο, όπου ναι μεν τα loss μειώνονται και τα accuracies αυξάνονται, αλλά δεν συμβαίνει ακόμη overfitting. Η μη ομαλή αύξηση ή μείωση κατά την αύξηση των εποχών, ίσως να οφείλεται στο ότι χρειαζόταν κάποιο μικρότερο learning rate για τα συγκεκριμένα δεδομένα. Οπότε θα ξανατρέξουμε τον αλγόριθμο εκπαίδευσης για τόσες εποχές.

Παραθέτουμε τα αποτελέσματα για κάθε κλάση

```
model.evaluate(train, y_train)
predictions = np.argmax(model.predict(train), -1)
print(classification_report(y_train, predictions, target_names=target_names))
```

```
==============================] - 0s 4ms/step - loss: 0.1817 - accuracy: 0.9586
==============================] - 0s 2ms/step
              precision    recall  f1-score   support

         neg       0.49      1.00      0.66       690
         pos       0.00      0.00      0.00       710

    uracy                           0.49      1400
o avg       0.25      0.50      0.33      1400
d avg       0.24      0.49      0.33      1400
```

```
model.evaluate(dev, y_dev)
predictions = np.argmax(model.predict(dev), -1)
print(classification_report(y_dev, predictions, target_names=target_names))
```

```
==============================] - 0s 3ms/step - loss: 0.3267 - accuracy: 0.8726
==============================] - 0s 2ms/step
              precision    recall  f1-score   support

         neg       0.52      1.00      0.69       209
         pos       0.00      0.00      0.00       191

    uracy                           0.52       400
o avg       0.26      0.50      0.34       400
d avg       0.27      0.52      0.36       400
```

```
model.evaluate(test, y_test)
predictions = np.argmax(model.predict(test), -1)
print(classification_report(y_test, predictions, target_names=target_names)
```

```
==========================] - 0s 3ms/step - loss: 0.3889 - accuracy: 0.8438
==========================] - 0s 3ms/step
              precision    recall  f1-score   support

         eg       0.51      1.00      0.67       101
         os       0.00      0.00      0.00        99

         cy                           0.51       200
        vg       0.25      0.50      0.34       200
        vg       0.26      0.51      0.34       200
```

Τα νούμερα του classification report είναι προφανώς λάθος, είναι σαν προβλέπεται πάντα μόνο η αρνητική κλάση. Βλέπουμε από το evaluate όμως ότι το accuracy στα testing δεδομένα για το μοντέλο που δημιουργήθηκε με οδηγό τα development δεδομένα είναι 84.38. Δεν έχουμε υπολογίσει τα auc διότι δεν έχουν νόημα αν τα παραπάνω αποτελέσματα δεν είναι σωστά.