lab04

April 17, 2023

```
[51]: from datasets import load_dataset
      import pandas as pd
      from sklearn.model_selection import train_test_split

      import fasttext as ft

      SEED = 42
```

```
[ ]: dataset = load_dataset('imdb')
```

1 FastText

1. Turn the dataset into a dataset compatible with Fastext (see the Tips on using FastText section a bit lower). For pretreatment, only apply lower casing and punctuation removal.

```
[45]: from string import punctuation
      import re

      def preprocess(df: pd.DataFrame) -> pd.DataFrame :
          """
          Preprocess the dataset by lowercasing the text and removing the punctuation␣
       ↪manually

          Parameters
          ----------
          df : pd.DataFrame
              The dataset to preprocess

          Returns
          -------
          pd.DataFrame
              The preprocessed dataset
          """
          punctuation_to_remove = '|'.join(map(re.escape, sorted(list(filter(lambda p:
       ↪ p != "'" and p != '-', punctuation)), reverse=True)))
          df["class"] = df["class"].apply(lambda x: "__label__" + ("positive" if x ==␣
       ↪0 else "negative"))
```

```
    df["document"] =  df["class"] + " " + df["document"].apply(lambda x: re.
↪sub(' +', ' ',
                                                                  re.
↪sub(punctuation_to_remove, " ",
                                                                         x.lower()
                                                                         .replace('<br /
↪>', "")))
                                                                  .strip())
    return df
```

[46]:
```
train_raw = pd.DataFrame(dataset["train"], columns=["text", "label"]).
↪rename(columns={"text": "document", "label": "class"})
preprocessed_train = preprocess(train_raw)
test_raw = pd.DataFrame(dataset["test"], columns=["text", "label"]).
↪rename(columns={"text": "document", "label": "class"})
preprocessed_test = preprocess(test_raw)
preprocessed_train.head()
```

[46]:
```
                                            document              class
0  __label__positive i rented i am curious-yellow…  __label__positive
1  __label__positive i am curious yellow is a ris…  __label__positive
2  __label__positive if only to avoid making this…  __label__positive
3  __label__positive this film was probably inspi…  __label__positive
4  __label__positive oh brother after hearing abo…  __label__positive
```

[18]:
```
# save the preprocessed data to a file
preprocessed_train.to_csv("data/train.txt", index=False, header=False)
preprocessed_test.to_csv("data/test.txt", index=False, header=False)
```

2. Train a FastText classifier with default parameters on the training data, and evaluate it on the test data using accuracy

[20]:
```
# train fasttext model with default parameters on the preoprocessed train␣
↪dataset
model = ft.train_supervised(input="data/train.txt", label_prefix="__label__")

# evaluate the model on the preprocessed test dataset using accuracy
print("Accuracy: ", model.test("data/test.txt")[1])
```

```
Read 5M words
Number of words:  116671
Number of labels: 2
Progress: 100.0% words/sec/thread: 4939539 lr: -0.000007 avg.loss:  0.343260
ETA:   0h 0m 0s

Accuracy:  0.87812

 lr:  0.000000 avg.loss:  0.343260 ETA:   0h 0m 0s
```

The accuracy on the test data is 0.87812

3. Use the hyperparameters search functionality of FastText and repeat step 2.

- To do so, you'll need to split your training set into a training and a validation set.
- Let the model search for 5 minutes (it's the default search time).
- Don't forget to shuffle (and stratify) your splits. The dataset has its entry ordered by label (0s first, then 1s). Feeding the classifier one class and then the second can mess with its performances.

[25]:
```
train, val = train_test_split(preprocessed_train, test_size=0.2, shuffle=True,␣
↪random_state=SEED, stratify=preprocessed_train["class"])
train.to_csv("data/train.txt", index=False, header=False)
val.to_csv("data/val.txt", index=False, header=False)
```

[26]:
```
model_autotune = ft.train_supervised(input='data/train.txt',␣
↪autotuneValidationFile='data/val.txt')
```

```
Progress: 100.0% Trials:   19 Best score:  0.991400 ETA:   0h 0m 0s
Training again with best arguments
Read 4M words
Number of words:  103775
Number of labels: 2
Progress: 100.0% words/sec/thread:  919488 lr:  0.000000 avg.loss:  0.041763
ETA:   0h 0m 0s
```

4. Look at the differences between the default model and the attributes found with hyperparameters search. How do the two models differ?

[31]:
```
# evaluate the model on the preprocessed test dataset using accuracy
print("Accuracy (default model):", model.test("data/test.txt")[1])
print("Accuracy (Autotuned model):", model_autotune.test("data/test.txt")[1])
```

```
Accuracy (default model): 0.87812
Accuracy (Autotuned model): 0.99144
```

Results with default parameters vs autotuned parameters: - Accuracy (default model): 0.87812 - Accuracy (Autotuned model): 0.99144

We see that the autotuned model is almost 100% right, which is a huge improvement. Let's look at the differences between the two models:

[50]:
```
print("Default model lr attribute:", model.lr)
print("Autotuned model lr attribute:", model_autotune.lr, end="\n\n")

print("Default model loss attribute:", model.loss)
print("Autotuned model loss attribute:", model_autotune.loss, end="\n\n")

print("Default model epoch attribute:", model.epoch)
print("Autotuned model epoch attribute:", model_autotune.epoch, end="\n\n")
print("Default model lrUpdateRate attribute:", model.lrUpdateRate)
print("Autotuned model lrUpdateRate attribute:", model_autotune.lrUpdateRate,␣
 ↪end="\n\n")

print("Default model ws attribute:", model.ws)
print("Autotuned model ws attribute:", model_autotune.ws, end="\n\n")

print("Default model neg attribute:", model.neg)
print("Autotuned model neg attribute:", model_autotune.neg, end="\n\n")

print("Default model minn attribute:", model.minn)
print("Autotuned model minn attribute:", model_autotune.minn, end="\n\n")

print("Default model maxn attribute:", model.maxn)
print("Autotuned model maxn attribute:", model_autotune.maxn)
```

```
Default model lr attribute: 0.1
Autotuned model lr attribute: 4.348128173351435

Default model loss attribute: loss_name.softmax
Autotuned model loss attribute: loss_name.softmax

Default model epoch attribute: 5
Autotuned model epoch attribute: 14

Default model lrUpdateRate attribute: 100
Autotuned model lrUpdateRate attribute: 100

Default model ws attribute: 5
Autotuned model ws attribute: 5

Default model neg attribute: 5
Autotuned model neg attribute: 5

Default model minn attribute: 0
Autotuned model minn attribute: 3

Default model maxn attribute: 0
Autotuned model maxn attribute: 6
```

Recap of the values: | Attribute | Default Model | Autotuned Model | | ———————————— | ————- | ————— | | lr | 0.1 | 4.348 | | loss_name | softmax | softmax | | epoch | 5 | 14 | | lrUpdateRate | 100 | 100 | | ws | 5 | 5 | | neg | 5 | 5 | | Dimension of a lookup vector | 100 | 66 | | minn | 0 | 3 | | maxn | 0 | 6 |

We see that the autotuned model has a very high learning rate (4.35) compared to the default model (0.1). They both use the same loss fonction as well as the same update rate for the learning rate. One of the big difference can also be seen in the number of epoch, where the autotuned model has 3x more epochs than the default model. This could be one of the elements explaining the better accuracy of the autotuned model. Finally, the `minn` and `maxn` attributes go from 0 to 3 and 6 respectively in the autotuned model, which means that the autotuned model uses subword information.

5. Using the tuned model, take at least 2 wrongly classified examples from the test set, and try explaining why the model failed.

[48]:
```python
wrongly_classified = []
predictions = []
for i in range(len(preprocessed_test)):
    pred = model.predict(preprocessed_test.iloc[i]["document"])[0][0]
    if pred != preprocessed_test.iloc[i]["class"]:
        wrongly_classified.append(preprocessed_test.iloc[i]["document"])
        predictions.append((pred, preprocessed_test.iloc[i]["class"]))
    if len(wrongly_classified) == 2:
        break

print(f"First wrongly classified example (pred={predictions[0][0]},␣
 ↪label={predictions[0][1]}):")
print(wrongly_classified[0])
print(f"Second wrongly classified example (pred={predictions[1][0]},␣
 ↪label={predictions[1][1]}):")
print(wrongly_classified[1])
```

```
First wrongly classified example (pred=__label__negative,
label=__label__positive):
__label__positive first off let me say if you haven't enjoyed a van damme movie
since bloodsport you probably will not like this movie most of these movies may
not have the best plots or best actors but i enjoy these kinds of movies for
what they are this movie is much better than any of the movies the other action
guys segal and dolph have thought about putting out the past few years van damme
is good in the movie the movie is only worth watching to van damme fans it is
not as good as wake of death which i highly recommend to anyone of likes van
damme or in hell but in my opinion it's worth watching it has the same type of
feel to it as nowhere to run good fun stuff
Second wrongly classified example (pred=__label__negative,
label=__label__positive):
__label__positive ben rupert grint is a deeply unhappy adolescent the son of his
unhappily married parents his father nicholas farrell is a vicar and his mother
laura linney is well let's just say she's a somewhat hypocritical soldier in
jesus' army it's only when he takes a summer job as an assistant to a foul-
mouthed eccentric once-famous and now-forgotten actress evie walton julie
walters that he finally finds himself in true 'harold and maude' fashion of
course evie is deeply unhappy herself and it's only when these two sad sacks
find each other that they can put their mutual misery aside and hit the road to
happiness of course it's corny and sentimental and very predictable but it has a
hard side to it too and walters who could sleep-walk her way through this sort
of thing if she wanted is excellent it's when she puts the craziness to one side
```

```
and finds the pathos in the character like hitting the bottle and throwing up in
the sink that she's at her best the problem is she's the only interesting
character in the film and it's not because of the script which doesn't do
anybody any favours grint on the other hand isn't just unhappy he's a bit of a
bore as well while linney's starched bitch is completely one-dimensional still
she's got the english accent off pat the best that can be said for it is that
it's mildly enjoyable - with the emphasis on the mildly
```

The two wrongly classified examples are the following: 1. *first off let me say if you haven't enjoyed a van damme movie since bloodsport you probably will not like this movie most of these movies may not have the best plots or best actors but i enjoy these kinds of movies for what they are this movie is much better than any of the movies the other action guys segal and dolph have thought about putting out the past few years van damme is good in the movie the movie is only worth watching to van damme fans it is not as good as wake of death which i highly recommend to anyone of likes van damme or in hell but in my opinion it's worth watching it has the same type of feel to it as nowhere to run good fun stuff*

2. *ben rupert grint is a deeply unhappy adolescent the son of his unhappily married parents his father nicholas farrell is a vicar and his mother laura linney is well let's just say she's a somewhat hypocritical soldier in jesus' army it's only when he takes a summer job as an assistant to a foul-mouthed eccentric once-famous and now-forgotten actress evie walton julie walters that he finally finds himself in true 'harold and maude' fashion of course evie is deeply unhappy herself and it's only when these two sad sacks find each other that they can put their mutual misery aside and hit the road to happiness of course it's corny and sentimental and very predictable but it has a hard side to it too and walters who could sleep-walk her way through this sort of thing if she wanted is excellent it's when she puts the craziness to one side and finds the pathos in the character like hitting the bottle and throwing up in the sink that she's at her best the problem is she's the only interesting character in the film and it's not because of the script which doesn't do anybody any favours grint on the other hand isn't just unhappy he's a bit of a bore as well while linney's starched bitch is completely one-dimensional still she's got the english accent off pat the best that can be said for it is that it's mildly enjoyable - with the emphasis on the mildly*

They are both classified as negative, but they are positive.

The model could have failed because of some of the same reasons models of the previous labs failed. We can see that the first example uses negative terms to say that people who didn't like this type of movie will probably not like this one, but this is not a negative point towards the movie itself. The second one contains a description of the movie, which seems to have a negative emphasis, but again, this is not related to the movie review itself, so this could be confusing for the classifier. Also, these two examples are mixed reviews, the author didn't enjoy the movie at 100%. The review it still positive but not 100% defending the movie

6. Why is it likely that the attributes minn and maxn are at 0 after an hyperparameter search on our data? Hint: on what language are we working?

We are working on English, which is a language with a lot of words. This means that the words are not composed of subwords, so the model doesn't need to use subword information to classify the examples. It is possible that during the hyperparameter search, the autotuned model found that the use of character n-grams did not significantly improve the model's performance on the given data, and thus set the `minn` and `maxn` parameters to 0. This could be due to the fact that the text data used for training did not contain significant patterns that could be captured by character n-grams, or that the use of character n-grams led to overfitting on the training data. However, the model ended with these two attributes at 3 and 6, which means that the model did use subword information.