Exercise 4 - Classification

January 15, 2020

1 Decision Trees

a) ID3 Decision Tree Construction (3 points)

You are given the following labeled data set. Construct a decision tree using pen and paper. Apply the information gain splitting. Constructing the first two levels of the tree is sufficient. Draw the tree and indicate each splitting attribute. Show your calculations.

Outlook	Temperature	Humidity	Wind	Which sport?
sunny	cold	high	weak	soccer
cloudy	cold	low	strong	soccer
sunny	warm	low	weak	soccer
rainy	cold	high	weak	squash
sunny	cold	high	weak	squash
rainy	warm	high	strong	squash
cloudy	cold	high	weak	squash
rainy	warm	high	weak	squash
cloudy	warm	high	weak	tennis
cloudy	cold	low	strong	tennis
sunny	cold	low	strong	tennis
cloudy	cold	high	weak	tennis

2 Neural Networks

a) Hyperparameter exploration (2 points)

Try varying the hyper-parameters of a multi-layer perceptron on the TensorFlow Playground webpage (`https://playground.tensorflow.org`) using the *Circle* classification dataset (see figure below). Try different network architectures (number of hidden layers, number of neurons per layer). Does it work better to a) use more neurons near the input layer, b) more neurons towards the last hidden layer or c) use the same number of neurons in each hidden layer? Provide a justification for why a), b) or c) might work better. Report the best configuration which you found. Can a perceptron (without hidden layers) separate the circular dataset? Why or why not?

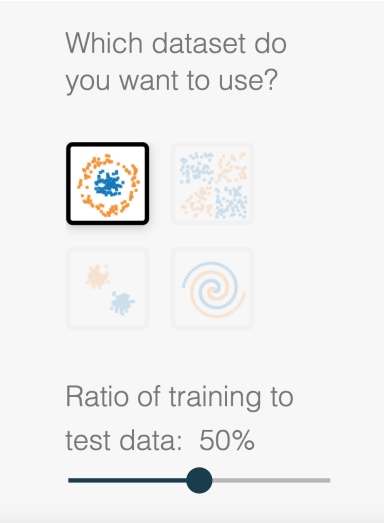

Figure 1: Mandatory settings on TensorFlow Playground for this exercise

b) Multi-Layer Perceptron for Sentiment Analysis (5 points)

Implement a multi-layer perceptron for classifying the movie reviews into positive or negative sentiment using PyTorch (`https://pytorch.org/`). The data is stored in the folder *data*. Each line i in *labels.txt* contains the label for the i-th movie review in *reviews.txt*. You have to map each of the movie review texts to a fixed-size embedding vector, which you can use as input to your multi-layer perceptron. You can do so by using the `flair` library[1]. Install it using `pip install flair`. You can use it to encode text as follows (there are multiple ways - pooling of pre-trained word vectors, recurrent neural nets, etc.):

```
from flair.embeddings import WordEmbeddings, DocumentRNNEmbeddings,
    FlairEmbeddings, DocumentPoolEmbeddings, Sentence

# instantiate pre-trained word embeddings
word_embeddings = WordEmbeddings("glove")

# document pool embeddings - try to exchange this with DocumentRNNEmbeddings!
document_embeddings = DocumentPoolEmbeddings( \
  [word_embeddings], fine_tune_mode="none")

# create an example sentence object
sentence = Sentence("Colorless green ideas sleep furiously.")

# embed the sentence with the document embeddings (needed for each movie review)
document_embeddings.embed(sentence)

# check out the embedded sentence - it's a torch.Tensor object :-)
print(sentence.get_embedding())
```

Listing 1: Embedding documents using flair

Perform a 3-fold cross validation and report precision and recall. Report your hyper-parameter configuration (learning rate, batch size, network structure, etc.).

[1] https://github.com/flairNLP/flair

c) Bonus Question: Word Embeddings (1 point)

Read the paper "Deep contextualized word representations" (Peters et al.)[2] and answer the following questions: What is the most important difference between word2vec or GloVe word embeddings and the ELMo model proposed in the paper? Why does this difference have a large effect on the quality of the resulting word embeddings?

[2] https://arxiv.org/pdf/1802.05365.pdf