Machine Learning HW#5

Ella Pavlechko
Due 12/13/19

1 Keras has several datasets you can load, one of them is the IMDB dataset for sentiment classification of reviews (see https://keras.io/datasets/). Load the data using $num\_words = 5000$, note that these data also have training and out of sample test data already separated. Use the function keras.preprocessing.sequence.pad_sequences (see https://keras.io/preprocessing/sequence/) to pad each sequence of characters, and use $maxlen = 500$. Create a Keras sequential model, or use the functional API, to create an LSTM model for predicting sentiment. For the first layer, use the Keras embedding layer (see https://keras.io/layers/embeddings/) to map words to a vector of length 16. Use one LSTM layer (64 hidden states) followed by one dense layer with a sigmoid output, where the outputs 0 and 1 represent good and bad sentiment, respectively. Compile the model with binary cross entropy and the adam optimizer with default values. Train for 10 epochs and report the accuracy on the out of sample test set. Repeat all of the above by setting the embedding vector length to 8, 16, or 32, and the number of hidden states equal to 16, 32, 64, or 128. Report a table of results for the out of sample test accuracy. Note that larger models may require more epochs for training.

This dataset contains 25,000 movie reviews, labeled as either positive or negative. In this model we considered the 5000 most used words, and chunks of 500 word strings, and used this to find long-term dependencies inside the strings of words.

Using an embedding layer, then an LSTM layer (included in the Keras Sequential model package), and a final dense layer we found the following accuracies when we varied Hidden States and Vector Length:

Table 1: Out of Sample Testing Accuracy

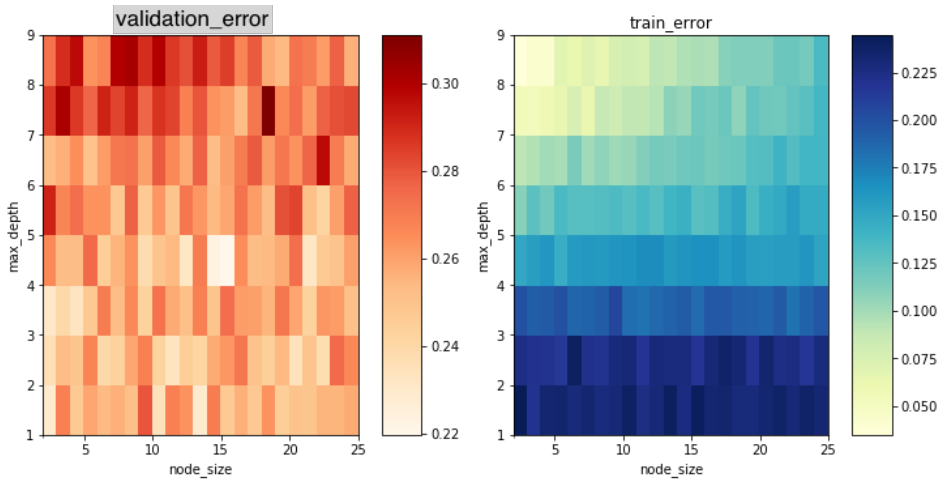
		Hidden States			
		16	32	64	128
Vector Length	8	0.85628	0.86432	0.85848	0.86516
	16	0.86448	0.86448	0.85692	0.87156
	32	0.87184	0.86748	0.8522	0.86656

Thus, all the Accuracies of the vectors and hidden states were in the range 85.2% to 87.2%, which is fairly high and indicates this LSTM model did a good job telling whether a string of words had a good or bad sentiment. One thing that may have improved computation time when running this model, was the use of mini-batching, although it was not requested in the problem.

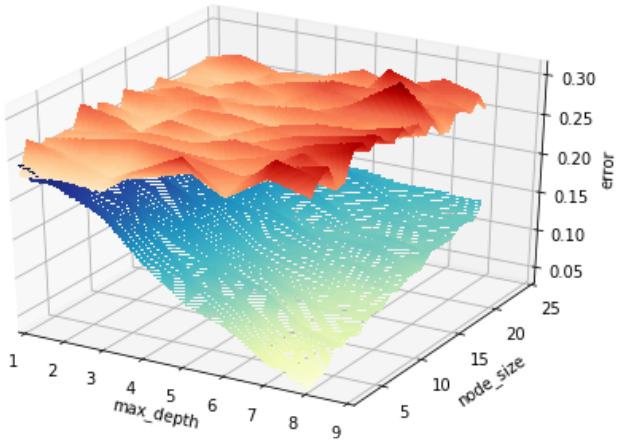
2 For this problem you will use the scikit learn package in python to train decision trees. Include python code in your assignment write-up. You will train decision trees using the Pima Indian Diabetes dataset downloaded from the UC Irvine Machine Learning Database at https://archive.ics.uci.edu/ml/datasets/pima+indians+diabetes. See the website for a description of the columns, in particular the last column represents the class label and the first 8 columns are the features. The data are in the same folder as this assignment in the file *Pima-Data-Adjusted.mat*. Note that these data are adjusted from the database to remove entries with missing data, and these data have been normalized. Use 5-fold cross validation to tune hyperparameters, e.g., by using a 70/30 train/val split.

2a) Use the function "sklearn.tree.DecisionTreeClassifier" to train a decision tree. Tune the hyperparameters: maximum depth (max_depth), minimum node size (min_samples_split), gain in error reduction (you will need to write your own function for this).

2b) Use the function "sklearn.ensemble.RandomForestClassifier" to train a random forest model. Tune the hyperparameter "n_estimators", i.e., the number of trees. Use the optimal hyperparameters for 2a) to define the trees.



(a)



(b)

Figure 1: The Red surfaces correspond to the validation error, and the Blue surfaces to the training error

Since there are 8 features on the data, at maximum we could expect 8 layers in a Decision Tree. At minimum it should be 1, which would correspond to one root node and 8 subsequent nodes in the single layer. For minimum node size the smallest amount of data accepted is two samples, and we continued to look up to 25 samples required

to split the node.

For each combination of maximum depth and minimum node size, we performed a 5-fold cross-validation. At each cross-validation this entails randomly splitting the training and validation data 70% and 30% respectively. We then trained our decision tree with the set parameters. After 5 cross-validations we averaged the errors.

After looking at all the depth and node combinations, we plot the errors on Figure 1a. To determine the best hyperparameters we are looking for a gain in error reduction, or a minimum validation error while having our training error decreasing. This is more easily seen in Figure 1b, where the best hyperparameters are:

$$(node\_size, max\_depth) = (15, 4)$$

For part b) we use n number of 4-layer trees, with minimum node size 15. After training the n trees, we average their results and compare errors. To determine the best n, we again perform 5-fold cross-validation. The result of which can be summarized in Figure 2. The gain in error reduction for this hyperparameter occurs at 137, so if we were to apply our Random Forest model to test data we may want to set

$$(node\_size, max\_depth, numer\_of\_trees) = (15, 4, 137)$$

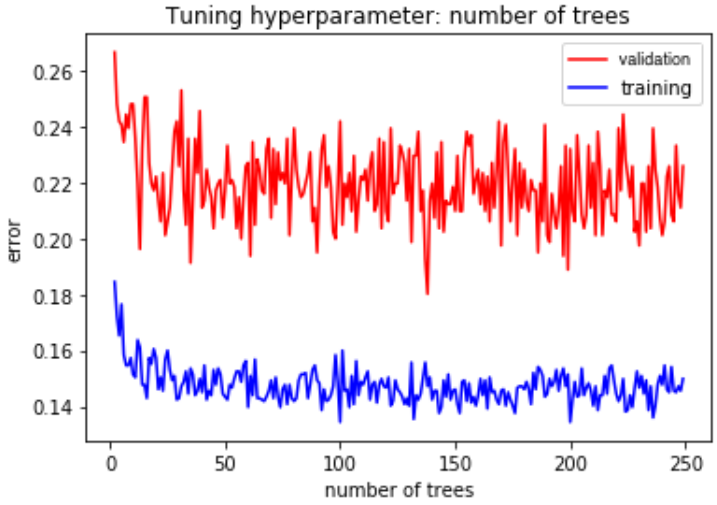


Figure 2: Random Forest Classifier hyperparameter tuning