CS771

ML Class of '14

Problem 2.1

`(The Code Corrector)`

Your task is to learn a multi-class classification model which, when given the BoW representation of a test line, can tell us the error class associated with that line. Please note that you do not have access to the exact code line – you only have access to the BoW encoding of that line. However, your model is allowed to output 5 error classes which it thinks may be the correct error class. Given a test line, your method should return a vector `pred` of length 5 with each entry between 1 and 50. The first entry of the vector i.e. `pred[0]` will be interpreted by us as the error class your method thinks is most likely to be the correct error class for that line, the second entry of the vector i.e. `pred[1]` is the next most likely error class, and so on. Thus, if you think the most suitable error class for the line is the ninth error class, you must set `pred[0]` = 9 (error classes are numbered 1 to 50 and not 0 to 49). Thus, we expect a *ranking* of the 5 most suitable error classes for that line. Please do not repeat error classes in your prediction vector e.g. by setting `pred[0] = pred[1] = ...`. Our evaluation code will automatically remove duplicates from your predictions (see `eval.py` and `utils.py`).

1. Suggest a method that you would use to solve the problem. Describe your method in great detail including any processing you do on the features, what classifier(s) you use to obtain the final predictions, what hyperparameters you had to tune to get the best performance, how you tuned those hyperparameters (among what set did you search for the best hyperparameter and how) e.g. you may say that we tuned the depth of a decision tree and I tried 5 depths $\{2, 3, 4, 5\}$ and found 3 to be the best using held out validation. (40 marks)

Answer: Our Model is a combination of **3** different ML models. We first train 2 base models over the entire training data. First model is an *MLP Classifier* from `sklearn` that uses *neural networks* to do the multiclassification. Second is an *XGboost model* which is a gradient-boosted decision tree model. XGboost works by creating an ensemble of decision trees which work together to get a very high performance.

Testing both the models individually reveals that gradient boosted trees are powerful in classification of samples which belong to classes which have small number of samples, while Neural Networks are good in classifications of classes which have good amount of samples. As a result XGBoost gives very high performance on mprec@5, hence it can identify top 5 classes the sample may belong to with high accuracy.

To get a model which combines the best parts of both the models, we do the following. We first ask the class probabilities from both the models for each test sample. Then we take a weighted average of both the probabilities.

$$p^{new} = 0.3 \cdot p^{MLP} + 0.7 \cdot p^{XGBoost}$$

This ensures that if a particular model is very confident in a particular prediction then the final prediction is majorly affected by this model. Now, we take the top 5 predictions using the newly obtained probabilities. Further, in order to isolate these 5 classes from rest and

Preprint. Under review.

to increase the contribution of MLP model specifically in these 5 classes (since the MLP model had higher overall prec scores), we update the probabilities of only these 5 classes as

$$p^{new} = 3 \cdot (p^{old} \cdot 0.5 + p^{MLP} \cdot 0.2)$$

Note that it doesn't matter that these numbers may not add to 1.

The final model in the battery is also an *XGBoost model* which is trained to identify just between the classes 2, 3 and rest (i.e. ternary classification). This model has a very high accuracy and since these classes contain half the total samples, it is expected that contribution of this model can increase the precision scores.

So, we finally update

$$p^{new} = (p^{old} + p_{2,3}^{XGBoost} \cdot 0.1)$$

where

$$p_{2,3}^{XGBoost} = 0.7 \cdot p^{XGBoost} + 0.3 \cdot p^{MLP}$$

for all classes (except 2, 3) and probability from the third model. Otherwise, we report the predictions in the decreasing order of probabilities.

Note: The classes 2 and 3 were specifically chosen out of all the available ones because these two classes alone comprise (almost) half of the samples in the given training data.

Note all these hyperparameters had to be tuned like weights of $p^{XGBoost}$, $p^{XGBoost}$ in the calculation of p^{new}. However, searching for hyperparameters is not adhoc. For example in

$$p^{new} = 0.3 \cdot p^{MLP} + 0.7 \cdot p^{XGBoost}$$

It makes sense to have lower weight for MLP and a higher weight for XGboost since we want to capture top 5 precisely with the obtained probabilities. Likewise in

$$p^{new} = 3 \cdot (p^{old} \cdot 0.5 + p^{MLP} \cdot 0.2)$$

we needed that $3 \cdot 0.5 > 1$ in order to isolate the top 5 samples. A similar analysis works for other hyperparameters.

2. Discuss at least two advantages of your method over some other approaches you may have considered. Discuss at least two disadvantages of your method compared to some other method (which either you tried or wanted to try but could not). Advantages and disadvantages may be presented in terms of prediction, macro precision, training time, prediction time, model size, ease of coding and deployment etc. (10 marks)

Answer: We tried various methods and selected the one producing best results. Some of the models we tried were

* Decision Tree Classifier
* XGBoost Classifier
* One vs All SVM Classifier
* K-Nearest Neighbour Classifier
* Linear SVM Classifier
* Neural Network Classifier
* Nearest Centroid Classifies
* Random Forest Classifier

Overall, we observed that the *Decision Tree* classifies the data pretty well for all the classes even if they have very little training data, but overall it produces low prediction scores. *Neural Network* on the other hand, classifies the data and predicts the correct label with very high confidence producing high prediction results overall, but works well for the classes having large training data only. In our model solution we use the combination of both Neural Networks and XGBoost Classifier, and the contribution of each to the model is a hyperparameter, which has been tuned.

Apart from the basic advantages of our model solution that includes faster training (small training time) and better prediction scores (both P-prec and M-prec) the model has a few disadvantages like,

1. It produces a model of size greater than that of a normal Neural Network Classifier
2. It has more number of hyper-parameters to tune, which is difficult to do.

3. Train your chosen method on the train data we have provided (using any validation technique you feel is good and also splitting the train data into validation split(s) in any way you wish). Store the model you obtained in the form of any number of binary/text/pickled/compressed files as is convenient and write a prediction method in Python in the file `predict.py` that can take a new data point and use the model files you have stored to make predictions on that data point. Include all the model files, the `predict.py` file, as well as any library files that may be required to run the prediction code, in your ZIP submission. You are allowed to have subdirectories within the archive. (50 marks)

Answer: The output containing accuracy has been provided below:

```
prec@1:  0.818 prec@3:  0.947 prec@5:  0.972
mprec@1:  6.911e-01 mprec@3:  8.598e-01 mprec@5:  9.018e-01
```

The test time for the model was **0.287 s.** with a test train split of 20-80.

Dependencies: sklearn, numpy, xgboost