Report intro to ML 29-11

Lorenzo Beltrame

23 December 2021

Note: During this assignment I did not have access to my usual PC, therefore the computational power that I had access to was limited. This mainly lead me to implement the code but not being able to run it in its entirety (and get the relative results!).

1 Task 1

1.1 Subtask 1

In the first subtask I loaded the movie data. The data were separated into 3 different tables: "movies.dat", "ratings.dat", "users.dat".

Subsequently, I merged the ratings data frame with the movies data frame. I chose to do a SQL left outer join-like merge of the two dataset to associate to each rating (i.e. "UserID", "MovieID", "Rating", "Timestamp") to the corresponding movie information (i.e. "MovieID", "Title", "Genres"). The merging was done on the "MovieID" feature[1].

Subsequently I wanted to associate the information known about each user (i.e. "UserID", "Gender", "Age", "Occupation", "Zip-code") to each of the review. We, therefore, merged the precedent dataset with the users dataset performing a SQL left outer join-like merge on the feature "UserID".

Finally; I dropped the "Timestamp" and the "Zip-Code" feature.

I also created a new feature column since the title feature contained also the year the movie was made. Then, I applied a filter to choose the users that made only more that 200 reviews, this was handily done using the "groupby" method. This procedure eliminated 343228 reviews (1/3 of the total!).

In the end, the data-frame had the following features:

- 'UserID'
- 'MovieID'
- 'Rating'
- 'Title'
- 'Genres'
- 'Gender'
- 'Age'
- 'Occupation'
- 'Year'

Subsequently, I encoded the categorical attributes.

To create the train and the test sets for the learning problem I built the test data from all the ratings with "UserID" equal to 1, . . . , 1000 (unless a user was removed). I then used the remaining ratings to construct the training data. The train set has 561388 elements. The test set has 95593 elements.

[1] Note that the same notation is consistent in all the three datasets.

Figura 1: This plot presents the precision/recall curve for both the tuned classifiers

After that I differentiated the design matrix and the target, by choosing as the target the "Rating" column of the dataset and dropping the column in the design matrix.

1.2 Subtask 1.2

First I tried to tune the SVC's regularisation hyperparameter using a logarithmic-ally spaced grid: 1.26 e-02, 1.78 e-01, 2.51, 3.55 e+01, 5.012 e+02, 7.08 e+03 and 1.00 e+05.

It is important to note that the 7 hyperparameter raise warnings because they needed more iterations to converge (I increased the iteration maximum from 1000 to 2000). Probably, by increasing the number of iterations done, it was possible to obtain better results. On the other hand, this would have highered up the computational time by a lot.

Accuracies for the tuned model an the train set: 0.4746577193956574 With the following hyperparameters: 'C': 501.19

The accuracy score (SVC) on the test is: 0.47

Subsequently, I tuned the MLP.

First I tuned some random shapes, in order to get an idea of what a good hyperparameter was, subsequently I chose to restrict to the following hidden layer shapes: [(20,20), (10, 10)].

Accuracies for the tuned model an the train set: 0.5423193255209094 With the following hyperparameters: 'hidden_layer_sizes': (20, 20)

I did not have the computational capability to create deep networks, therefore I just created simple nets. On this basis, I included just those two features for the sake of computational time. Anyway, during development I tried several of them.

Therefore, I could not go above the accuracy reference stated in the assignment.

Probably, by acquiring the "Genres" features and creating new boolean values associated to the mined features we could have achieved better scores.

We can definitely see that the MLP is definitely better than the SVC in this application. Me might have obtained different results In any case, both the models are better than the "predict all 0" classifier (which give a 0.46 accuracy on the test set). This mean that we are learning some pattern from the data.

1.3 Subtask 1.3

First, I implemented a function to compute the precision, the recall using the manually computed TP, FN, FP values. To do so I used the "predict_proba" method. More details are available in code's comment.

The precision/recall curves that I obtained are the following:

1.4 Subtask 1.4

In this subsection I repeated the steps done in task 1.2 regarding data preparation without applying the label binarizer.

I also implemented a confusion matrix that adapts itself to the number of target possible.

I trained a linear SVC (a SVC would have taken too long. I left the computer on for one whole night and id did not get the results.) and got the following confusion matrix:

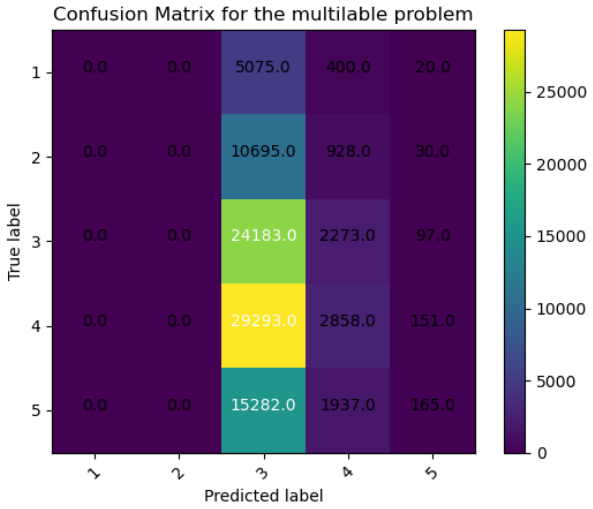

Figura 2: This is the confusion matrix computed for the multilable problem using a linear SVC.

We can see that the LinearSVC tends to predict just 3 star movie and never predicts 1,2 star movies.

Since the IDE states that with the vanilla configuration (the requested ones) the algorithm fails to converge, we might need more iterations to get a better result with a linear SVC. Note that I did not fix a random_state for the confusion matrix, therefore the plot might be different.

Probably I could have implemented a SVC and obtained better results and 1,2 star predictions. Another option would have been implementing a version considering a subset of the training data, but I did not have enough time to do it.

1.5 Subtask 2.1

The statistics of the data set are the following:

Number of features: 2

Number of instances in the training dataset: 500

Number of instances in the testing: 10000

1.6 Subtask 2.2

In this part I developed a custom MLP.

When initialising I pass the network shape (i.e. input layer size, hidden layer sizes and output layer size) to the custom class "my_MLP". In my implementation we have an input layer, two hidden layers and an output layer. Note that the class is designed such that it is possible to select the layers sizes.

Subsequently, the weight matrices (whose dimension depends on the network shape parameter) are initialized by sampling from a normal distribution centered in 0 and with standard deviation of 2/n (n is the number of instances). The weights are then stored in a python list.

The biases are first set as zero vectors and then they are stored in list.

The predict method loops through the weights' and biases' list to do a forward step. Finally, the method returns the prediction.

1.7 Subtask 2.3

In this task I implemented the approximated gradient (via finite difference) to approximate the gradient of the weights and the gradient of the biases. I used the mean squared error function as the cost function. I inserted a seed into the class to have consistent results. In this part I used the Adam Optimizer. More details are available in the comments inside the code.

The code for this part is present, but I did not have time to run the plot. The program should print them.