CS 6375.004: Machine Learning - Spring 2023

Project 2 Report

Kartikey Gupta (kartikey.gupta@utdallas.edu)

March 6, 2023

1 Collaborative Filtering

This part of the project tasked us to perform Predictive Collaboration Filtering on the "Netflix Prize" dataset. The dataset has three files - `movie_titles.txt`, `TrainingRatings.txt` and `TestingRatings.txt`. We disregard the `movie_titles.txt` file for the purpose of this project and focus solely on the `TrainingRatings.txt` and `TestingRatings.txt`. The training and testing dataset follow the same structure - each row denotes an example; a user rating for a movie. Each example has three attributes - ID of the user rating the movie, the ID of the movie that is being rated and the rating itself. The ratings are discrete values, on a scale of "1" to "5", "1" being lowest and "5" being highest.

The file `TrainingRatings.txt` contains upwards of 3.2 million training examples. The predictive collaborative filtering algorithm asks us to calculate Pearson Correlation Coefficient to calculate the similarity between two users. Due to the sheer size of the training set, calculating the correlation matrix on the all the users was simply not possible. As such, we needed a way to reduce the number of examples being considered to calculate the correlation coefficient in a way that minimizes the loss of information. In that regard, we perform a k-Nearest Neighbour Search on the training set to find users closest to our active user from the testing example.

For each testing example, we first find all the users that have rated the movie, and we find the 40 closest neighbors to our active user in that set. We then proceed to use the predictive algorithm as described in the paper. Since most of the users will have minimal impact on the final rating of the movie due to their low correlation with the active user, we lose nothing of value by not calculating their correlation coefficients in the first place.

This approach gives us a **Root Mean Squared Error of 0.933344**, with a **Mean Average Error of 0.742007**. The total runtime of the approach is approximately **15 minutes**.

2 Performance Analysis of Various Classifiers on the MNIST Dataset

To test the various classifiers, we first establish a baseline configuration and then change one hyperparameter at a time to isolate its effects. The best performing parameter is marked with a star symbol ($\star$).

2.1 MLPClassifier

The parameters being tested for `MLPClassifier` are: `hidden_layer_sizes`, `activation`, `alpha`, and `learning_rate_init`

The default values of the hyperparameters are as follows

```
{
        'hidden_layer_sizes': (50,),
        'activation': 'relu',
        'alpha': 0.0001,
        'learning_rate_init': 0.001
}
```

2.1.1 `hidden_layer_sizes`

`hidden_layer_sizes`	Accuracy
(50,)	96.94
$\star$ (100,)	**97.5**
(50, 50)	97.13

2.1.2 `activation`

`activation`	Accuracy
⋆ `relu`	**96.94**
`logistic`	95.5
`tanh`	95.42

2.1.3 `alpha` (Strength of L2 regularization term)

`alpha`	Accuracy
0.0001	96.94
0.00001	96.99
0.001	97.13
⋆ 0.01	**97.18**

2.1.4 `learning_rate_init`

`learning_rate_init`	Accuracy
⋆ 0.001	**96.94**
0.0001	96.7
0.001	97.66

2.2 KNeighborsClassifier

The parameters being tested for `KNeighborsClassifier` are: `n_neighbors`, `algorithm`, `weights`, and `p`
The default values of the hyperparameters are as follows

```
{
        'n_neighbors': 5,
        'algorithm': 'ball_tree',
        'weights': 'uniform',
        'p': 2
}
```

2.2.1 `n_neighbors`

`n_neighbors`	Accuracy
⋆ 5	**94.43**
1	94.34
50	92.91
100	90.77
500	86.4600
1000	82.8

2.2.2 `algorithm`

Apparently, the choice of the algorithm used has no effect on the accuracy of our kNN classifier.

`algorithm`	Accuracy
`ball_tree`	94.43
`kd_tree`	94.43
`brute`	94.43

2.2.3 `weights`

`weights`	Accuracy
`uniform`	94.43
⋆ `distance`	**94.5**

2.2.4 `p` (Power parameter for Minkowski metric)

`p`	Accuracy
2	94.43
⋆ 1	**95.73**

2.3 SVC

The parameters being tested for `SVC` are: `n_neighbors`, `C`, `kernel`, and `gamma`. There is an additional parameter called `degree`, which is only functional with the polynomial kernel. Furthermore, when testing the `gamma` parameter, we use the Radial Basis Function kernel.

The default values of the hyperparameters are as follows

```
{
        'C': 1,
        'kernel': 'sigmoid',
        'gamma': 'scale',
        'degree': 3
}
```

2.3.1 `C` (Regularization parameter penalty)

`C`	Accuracy
1	89.1
⋆ 0.1	**91.75**
10	86.74
0.01	89.64

2.3.2 `kernel`

`kernel`	Accuracy
`sigmoid`	89.1
⋆ `rbf`	**96.64**
`poly` (`degree`=3)	96.11
`linear`	92.93

2.3.3 `gamma` (Kernel Coeff. for `rbf`, `poly`, `sigmoid`)

`gamma` (`kernel`=`rbf`)	Accuracy
`scale`	89.1
⋆ `auto`	**96.66**

2.3.4 `degree` (Degree of the `poly` kernel)

`degree` (`kernel`=`poly`)	Accuracy
⋆ 3	**96.11**
2	97.14
4	89.62

⋆⋆ **END** ⋆⋆