Lab 5: Experimental Setup and Classification using NN

This lab's objective was to use a MLP NN classifier in order to predict sale opportunities. A dataset with about 78000 examples was given.

Data preprocessing

The dataset features multiple columns, these need to be analyzed:

COLUMN NAME	DATA FORMAT
OPPORTUNITY NUMBER	Integer
SUPPLIES SUBGROUP	String
SUPPLIES GROUP	String
REGION	String
ROUTE TO MARKET	String
ELAPSED DAYS IN SALES STAGE	Integer
OPPORTUNITY RESULT	String
SALES STAGE CHANGE COUNT	Integer
TOTAL DAYS IDENTIFIED THROUGH CLOSING	Integer
TOTAL DAYS IDENTIFIED THROUGH QUALIFIED	Integer
OPPORTUNITY AMOUNT USD	Integer
CLIENT SIZE BY REVENUE	Integer
CLIENT SIZE BY EMPLOYEE COUNT	Integer
REVENUE FROM CLIENT PAST TWO YEARS	Integer
COMPETITOR TYPE	String
RATIO DAYS IDENTIFIED TO TOTAL DAYS	Float
RATIO DAYS VALIDATED TO TOTAL DAYS	Float
RATIO DAYS QUALIFIED TO TOTAL DAYS	Float
DEAL SIZE CATEGORY	Integer

The first operation to be done to the dataset was the encoding of the string-type columns. This will result in integer data, for example, (0, 1) for the opportunity result column.

Next the dataset was split into "*data*" and "*target*" dataset. The data dataset contains all the useful information for the training of the NN. This includes all the columns except the "*OPPORTUNITY NUMBER*" and the "*OPPORTUNITY RESULT*". The "*target*" dataset only includes the "*OPPORTUNITY RESULT*" column.

After this, the non-encoded features were scaled using *sklearn's Standard scaler*. This concludes the data pre-processing.

Classification Task

In order to use the data on the NN, the dataset was split into a training, testing, and validation dataset (20, 50, 50).

Initial tests were made with the *MLPClassifier* function from *sklearn*, but in order to optimize the hyperparameters, the *GridSearchCV* was used in order to test multiple configurations and rank their performances. The test results are exported to .csv files and are included alongside this document.

For an initial test, a test grid was created with parameters for alfa and hidden layer size:

The initial set for testing alfa values was [0.001, 0.05, 0.5, 1, 10].

In order to test multiple combinations of hidden layer size values, a function that generates lists of permutations of constant values was used to create the set of testing hidden layer sizes. This resulted in 4050 tests and took about 15 hours to complete.

An excerpt of the head of the results is as follows:

mean_fit_time	alpha	hidden_layer_sizes	split0_test_score	split1_test_score	mean_test_score	std_test_score	rank_test_score
62.53273987	0.05	(39, 161)	0.868151	0.86546	0.866806	0.001346	1
29.89581108	0.05	(39, 11)	0.864691	0.867062	0.865876	0.001186	2
28.99324906	0.05	(30, 10)	0.866453	0.865236	0.865844	0.000609	3
56.81358862	0.05	(200,)	0.866677	0.864659	0.865668	0.001009	4
40.67831552	0.05	(26, 74)	0.866421	0.864467	0.865444	0.000977	5
26.75982773	0.05	(29, 6)	0.86687	0.863986	0.865428	0.001442	6
22.34275913	0.001	(20, 15)	0.867927	0.86264	0.865284	0.002643	7
31.38682926	0.05	(56, 44)	0.866645	0.86389	0.865268	0.001378	8
28.65921164	0.05	(28, 12)	0.86594	0.864242	0.865091	0.000849	9

Using the top performing configuration, we get the following results:

- Training set:
  - Accuracy: 0.86
  - Precision: 0.8
  - Recall: 0.65
  - Confusion Matrix:

    [46069, 2284]
    [4894,   9193]

- Testing set:
  - Accuracy: 0.87
  - Precision: 0.76
  - Recall: 0.62
  - Confusion Matrix:

    [11380,   685]
    [1334,   2206]

Next, in order to try to improve the results, an additional layer was added with the same method, starting with (39, 161) and with a fixed alfa of 0.05.

mean_fit_time	hidden_layer_sizes	split0_test_score	split1_test_score	mean_test_score	std_test_score	rank_test_score
266.3782916	(39, 161, 18)	0.85885934	0.855975649	0.857417494	0.001441846	1
432.7753334	(39, 161, 11)	0.858026274	0.85347645	0.855751362	0.002274912	2
449.0263267	(39, 161, 15)	0.856904838	0.853700737	0.855302788	0.001602051	3
324.8520572	(39, 161, 16)	0.858058315	0.852002563	0.855030439	0.003027876	4
512.0903938	(39, 161, 25)	0.858250561	0.850432554	0.854341557	0.003909004	5
444.1157403	(39, 161, 10)	0.854822172	0.850080103	0.852451137	0.002371035	6
417.0423012	(39, 161, 12)	0.847837232	0.855943608	0.85189042	0.004053188	7
985.8499718	(39, 161, 50)	0.852130727	0.851041333	0.85158603	0.000544697	8
776.2431267	(39, 161, 30)	0.851874399	0.850272349	0.851073374	0.000801025	9

Using the top performing configuration, we get the following results:

- Training set:
  - Accuracy: 0.90
  - Precision: 0.79
  - Recall: 0.77
  - Confusion Matrix:

    [45380, 2953]
    [3230, 10857]

- Testing set:
  - Accuracy: 0.87
  - Precision: 0. 70
  - Recall: 0. 70
  - Confusion Matrix:

    [11016, 1049]
    [1055,   2485]

Again, using the best configuration, an additional layer was added:

mean_fit_time	hidden_layer_sizes	split0_test_score	split1_test_score	mean_test_score	std_test_score	rank_test_score
203.4348381	(39, 161, 18, 2)	0.856673	0.846926	0.8518	0.004873	1
155.6683524	(39, 161, 18, 14)	0.842575	0.860811	0.851693	0.009118	2
195.6231918	(39, 161, 18, 5)	0.850179	0.850002	0.850091	8.86E-05	3
192.3084044	(39, 161, 18, 6)	0.85035	0.849276	0.849813	0.000537	4
110.7706622	(39, 161, 18, 8)	0.851077	0.846713	0.848895	0.002182	5
144.457392	(39, 161, 18, 15)	0.845523	0.849447	0.847485	0.001962	6
176.7698847	(39, 161, 18, 10)	0.846377	0.844064	0.845221	0.001157	7
260.8333316	(39, 161, 18, 11)	0.845993	0.844277	0.845135	0.000858	8
182.8678823	(39, 161, 18, 18)	0.845822	0.844192	0.845007	0.000815	9

Using the top performing configuration, we get the following results:

- Training set:
  - Accuracy: 0.91
  - Precision: 0.84
  - Recall: 0.71
  - Confusion Matrix:

    [34997, 1391]
    [3020, 7407]

- Testing set:
  - Accuracy: 0.86
  - Precision: 0.75
  - Recall: 0.61
  - Confusion Matrix:

    [11265, 745]
    [1386,   2209]

In this case, the recall score went down and the precision went up, since there was an increase in the number of layers and thus, neurons, this could be a sign of overfitting, so for the next optimization run, the previous 3-layer test was used but with more *hidden_layer* value options.

mean_fit_time	hidden_layer_sizes	split0_test_score	split1_test_score	mean_test_score	std_test_score	rank_test_score
196.8573	(39, 161, 6)	0.85317	0.855385	0.854277499	0.001108	1
192.8957	(39, 161, 2)	0.852914	0.852395	0.852654058	0.000259	2
183.0721	(39, 161, 150)	0.85458	0.847951	0.851265549	0.003314	3
132.7396	(39, 161, 19)	0.853213	0.846627	0.849919827	0.003293	4
187.9391	(39, 161, 20)	0.853042	0.846371	0.849706219	0.003335	5
215.8374	(39, 161, 18)	0.853426	0.845815	0.849620767	0.003805	6
202.5314	(39, 161, 9)	0.849966	0.848678	0.849321785	0.000644	7
158.4614	(39, 161, 8)	0.848684	0.847311	0.847997422	0.000687	8
135.5412	(39, 161, 4)	0.8439	0.851412	0.847655746	0.003756	9

Using the top performing configuration, we get the following results:

- Training set:
  - Accuracy: *lost values*
  - Precision: *lost values*
  - Recall: *lost values*
  - Confusion Matrix:

    [*lost values*]

- Testing set:
  - Accuracy: *lost values*
  - Precision: *lost values*
  - Recall: *lost values*
  - Confusion Matrix:

    [*lost values*]

This resulted in slightly better results, but unfortunately some of the score data was overwritten by a previous test.

The optimization procedure is very lengthy and time consuming, so I decided to use the best configuration of the last test as final and use it for the validation dataset. The validation prediction yielded the following results:

- Validation set:
    - Accuracy: 0.86
    - Precision: 0.70
    - Recall: 0.70
    - Confusion Matrix:

```
[5526,  527]
[530,  1220]
```

Conclusions

In the end, the program was able to predict sale opportunities approximately 70% of the time. Some aspects could be improved, for example, using another encoder like *"get_dummies"* for string columns.