Delft University of Technology

Data Science for Finance
WI3435TU

Assignment: Part II. Regression

Authors:
Dinu Gafton
Dan Sochirca

January 29, 2023

Question 1. Making the splitting plan

We decided to split the dataset into a training set (**70%** of initial dataset size) and a test set(**30%** of initial dataset size). We will train the models on the training set and then evaluate them on the test set. This allows us to get an estimate of how well the models will perform on new unseen data, and make adjustments if needed.

We will **use the same split of the dataset for the following assignments** as well, because we will be able to then compare performance of the models based on the same information/data that was given for training.

```
Training set:  X: (92722, 53);  Y: (92722,)
Test set:  X: (39738, 53);  Y: (39738,)
```

Figure 1: The size of the 2 datasets after the split

```
Original full set: unique vals:  [0, 1]  counts:  [30307, 102153]  proportions:  [0.22880114751623132, 0.7711988524837687]
Training set:      unique vals:  [0 1]   counts:  [21160 71562]    proportions:  [0.22820906 0.77179094]
Test set:          unique vals:  [0 1]   counts:  [ 9147 30591]    proportions:  [0.2301827 0.7698173]
```

Figure 2: Distribution of the 2 classes before and after the split

Question 2. Recoding the categorical variables

Categorical variables need to be encoded because many machine learning algorithms only accept numerical input data. Here are the categorical variables we need to deal with, and their unique values:

	name	description	number of unique values	unique values
1	term	The number of payments on the loan. Val...	2	[' 36 months' ' 60 months']
3	grade	LC assigned loan grade	7	['A' 'B' 'C' 'D' 'E' 'F' 'G']
4	sub_grade	LC assigned loan subgrade	35	['A1' 'A2' 'A3' 'A4' 'A5' 'B1' 'B2' 'B3' 'B4' 'B5' 'C1' 'C2' 'C3' 'C4' 'C5'...
5	emp_title	The job title supplied by the Borrower ...	49278	[' \tOFFICE MANAGER/MEDICAL ASSISTANT' ' Ag' ' Registered nurse ' ... 'z...
6	emp_length	Employment length in years. Possible va...	11	['1 year' '10+ years' '2 years' '3 years' '4 years' '5 years' '6 years' '7...
7	home_ownership	The home ownership status provided by t...	4	['ANY' 'MORTGAGE' 'OWN' 'RENT']
9	verification_status	Indicates if income was verified by LC,...	3	['Not Verified' 'Source Verified' 'Verified']
11	purpose	A category provided by the borrower for...	12	['car' 'credit_card' 'debt_consolidation' 'home_improvement' 'house' 'majo...
12	zip_code	The first 3 numbers of the zip code pro...	888	['007xx' '008xx' '009xx' '010xx' '011xx' '012xx' '013xx' '014xx' '015xx' '...
13	addr_state	The state provided by the borrower in t...	50	['AK' 'AL' 'AR' 'AZ' 'CA' 'CO' 'CT' 'DC' 'DE' 'FL' 'GA' 'HI' 'ID' 'IL' 'IN...

Figure 3: Categorical variables and their unique values

We decided to **encode all variables to integers rather than one-hot encoding**. As you can observe the number of unique values is quite big for most of the features, and one-hot encoding would lead to a significant increase in the dimensions of our dataset. And while we could have encoded some variables with a low number of unique values, we thought they're not worth the effort.

Nominal variables represent features that have no inherent order or ranking. We have 7 such variables: **term, emp_title, home_ownership, verification_status, purpose, zip_code, addr_state**.
Emp_title has *49278* unique values and **zip_code** has *888*, therefore these variables needed to be hashed, and we used *pandas.util.hash_array()* for this. For the rest, we encoded them using the method *pandas.factorize()*.

Ordinal variables represent data that can be ranked or ordered. We have 3 such variables: **grade, sub_grade** and **emp_length**:

- **grade** ranges from A to G, so we encoded it from 0 to 6 to eep it's order.
- **sub_grade** ranges from A1 to G5. We also encoded it from 0 to 6 but with decimals, so 0 represents A1, 0.2 represents A2, 0.4 represents A3, ..., 1 represents B1 and so on. G5 is encoded as 6.8.
- **emp_length** ranges from '<1 year' to '10+ years'. We kept the order and encoded it from 1 to 10.

Here are the categorical variables after the encoding process:

	name	description	number of unique values	unique values
1	term	The number of payments on the loan. Val...	2	[0 1]
3	grade	LC assigned loan grade	7	[0 1 2 3 4 5 6]
4	sub_grade	LC assigned loan subgrade	35	[0. 0.2 0.4 0.6 0.8 1. 1.2 1.4 1.6 1.8 2. 2.2 2.4 2.6 2.8 3...
5	emp_title	The job title supplied by the Borrower ...	49278	[889954848215734 1294130214511985 1426969337986 78...
6	emp_length	Employment length in years. Possible va...	11	[0 1 2 3 4 5 6 7 8 9 10]
7	home_ownership	The home ownership status provided by t...	4	[0 1 2 3]
9	verification_status	Indicates if income was verified by LC,...	3	[0 1 2]
11	purpose	A category provided by the borrower for...	12	[0 1 2 3 4 5 6 7 8 9 10 11]
12	zip_code	The first 3 numbers of the zip code pro...	888	[21790586110310201 34270912556226229 6492677679645515...
13	addr_state	The state provided by the borrower in t...	50	[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 2...

Figure 4: Categorical variables after the encoding

Question 3. Scaling the data

Scaling data is important for model training because many machine learning algorithms use some form of distance metric to compare observations. Features on a larger scale can disproportionately influence the distance metric, potentially distorting the model's ability to learn from other features.

For our case, we decided to choose **Z-score scaling** instead of Min-Max scaling, because there are multiple features in our set of 53 features used for training that do not have a "known" limit for their values (**i.e** loan_amnt). It is hard to assume if there can be bigger values of some features in the real world or test data than the max values found in the training set, and checks must be done on case-by-case basis. Thus, we chose to use the Z-score scaling for all the features, making the fit on the training set and then using the parameters to scale the test set.

To achieve it, we used the **StandardScaler** class from the *sklearn* library which fits and scales datasets using the Z-score.

Question 4. Building and fitting the regression models

To build all our Regression models, we have used the *LogisticRegression* class, as required, but changed the **penalty** parameter to make use of the different regularization techniques mentioned in the assignment.

To optimize the hyperparameter of each model, we generated a a list of **100 evenly spaced numbers between the interval of 10^{-10} to 10^{10}**, used as values for the hyperparameter. The models were trained using these values and then their performance were compared between each other to find the optimum value.

Ridge Regression

By training and evaluating a Ridge model with 100 different values for the hyperparameter (all the other parameters were left unchanged), as mentioned above, we were able to find the optimum value as shown in Figure 5. For **Lambda = 0.0018738174228603867** we have obtained the highest **Accuracy = 77.776%**. This is the optimum hyperparameter that we found and was used further in the assignment.

It should be mentioned that the plot shows the Lambda values in a **decreasing order** (from 10^{10} to 10^{-10}).

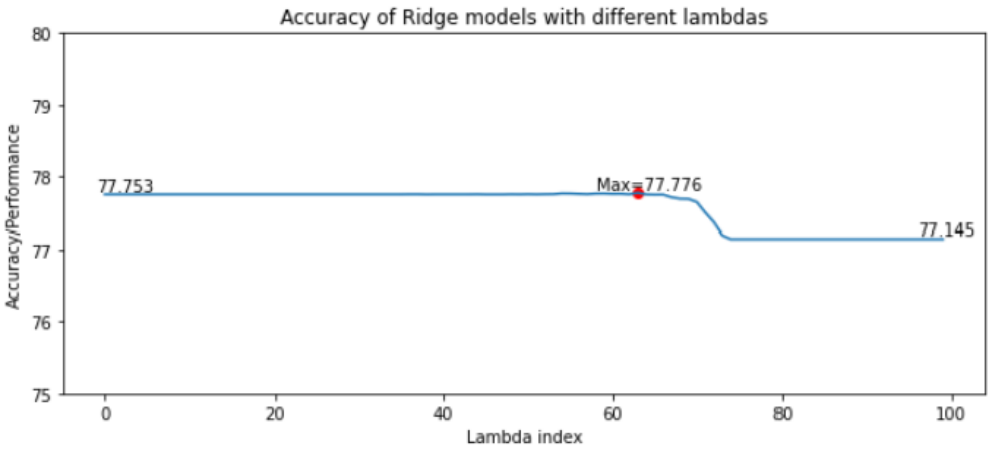

Figure 5: Performance of Ridge models using 100 different values for the hyperparameter

Lasso Regression

The training and evaluation of the Lasso models were conducted in the same fashion as the Ridge models. Following the execution of the program, Figure 6 was created to show the performance of the models. For **Lambda = 0.014563484775012445** we gained the highest **Accuracy of 77.772%**.

It is worth noting that for **Lambda** $< \mathbf{10^{-4}}$, the performance decreases to **22.855%** which is also almost equal to the proportion of Class 0 (**Not Paid**) instances in our dataset. We assume that the models are classifying correctly only the Class 0 instances.

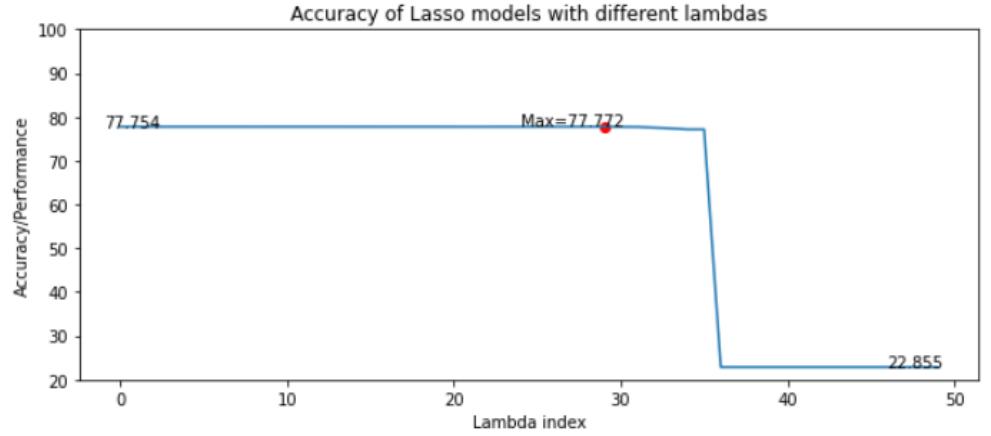

Figure 6: Performance of Lasso models using 100 different values for the hyperparameter

ElasticNet Regression

By optimising the model the same way as in the above-mentioned cases, we found **Lambda = 0.09540954763499924** to be the optimal value for the model with **Accuracy = 77.766%**, as it can be seen in Figure 7.

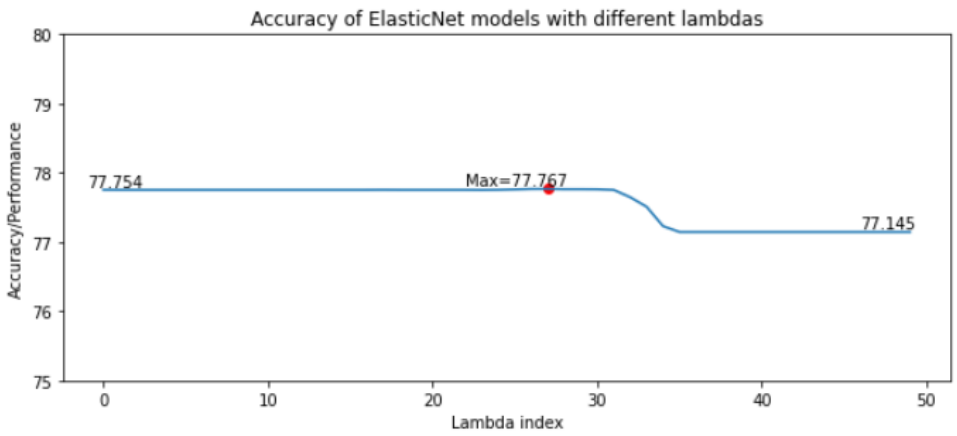

Figure 7: Performance of ElasticNet models using 100 different values for the hyperparameter

Most Important Features

For this task, we have used the optimum hyperparameters found by us to train and evaluate the Regression models. We chose the 5 most important features for each model, by selecting the features with the highest absolute weight values. Figure 8 shows the results. It is worth to notice that the majority of features shown in the table are common for all the models, but the order of their importance is different (though the order for **Lasso** and **ElasticNet** are the same).

Model	Feature nr.1	Feature nr.2	Feature nr.3	Feature nr.4	Feature nr.5
Ridge	term	sub_grade	int_rate	loan_amnt	dti
Lasso	sub_grade	term	loan_amnt	dti	fico_range_low
ElasticNet	sub_grade	term	loan_amnt	dti	fico_range_low

Figure 8: Most important features for decision-making of each model

Question 5. Model comparison

We applied the three logistic regression models to the **test dataset**. Since we have an imbalanced dataset (only 22% of it represents individuals who defaulted), we need to decide on a threshold of the probability that a default happens or not in order to improve our model.

Similiar to the jupyter notebook from the texbook, **we tried 6 thresholds for each model: [.5, .6, .7, .75, .80, .85]**, and computed the confusion matrix and the ROC curve for each. These results can be seen in the ***Appendix***. It can be observed from the figures that **all the three models are very similar**, and that **a threshold of 0.75-0.80 would be a good choice** because the false positive rate is smaller.

AUC comparison for different thresholds:

	0	1	2	3	4	5
Threshold	0.5	0.6	0.7	0.75	0.8	0.85
Ridge	0.553264	0.590326	0.63752	0.655498	0.659332	0.631634
Lasso	0.553079	0.591027	0.636593	0.656452	0.659394	0.62931
ElasticNet	0.553792	0.592658	0.638536	0.65623	0.659036	0.631989

Figure 9: Different models compared in terms of AUC. Highest AUC score is highlighted in yellow.

In **Figure 5** above we show different AUC (area under ROC curve) scores for each model and probability threshold. As it can be seen, **the models perform quite similarly for different thresholds**. The **lasso**

model with the threshold set to **0.8** has the highest AUC score.

The Best Model in terms of AUC

The best model in terms of the AUC score is **the lasso model with a probability threshold of 0.8**. This model yields an accuracy of **61.16%** on the test set.

Appendix

Results for Ridge model:

```
Confusion matrix for threshold = 0.5
 [[0.03115406 0.19822336]
 [0.02257285 0.74804973]]

Confusion matrix for threshold = 0.6
 [[0.05815592 0.1712215 ]
 [0.0561679  0.71445468]]

Confusion matrix for threshold = 0.7
 [[0.10423272 0.1251447 ]
 [0.13823041 0.63239217]]

Confusion matrix for threshold = 0.75
 [[0.13586492 0.09351251]
 [0.21679501 0.55382757]]

Confusion matrix for threshold = 0.8
 [[0.17076853 0.05860889]
 [0.32814938 0.4424732 ]]

Confusion matrix for threshold = 0.85
 [[0.19978358 0.02959384]
 [0.46831748 0.3023051 ]]
```

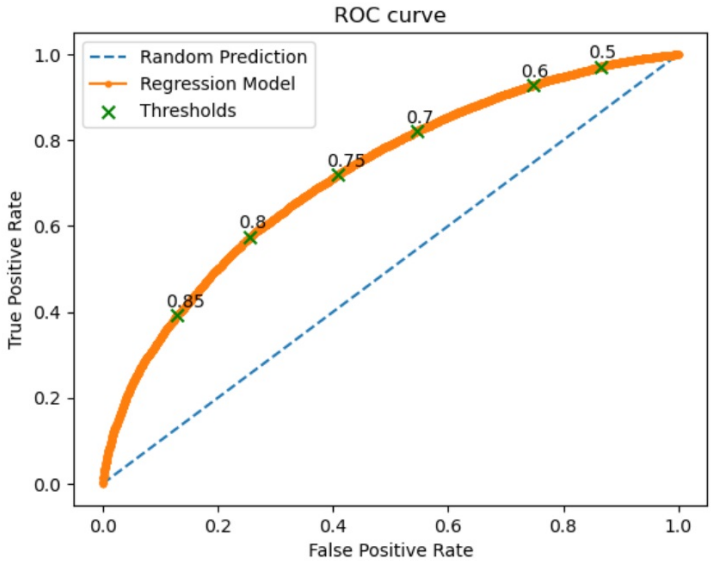

Figure 10: Different probability thresholds applied to the **ridge** model. Confusion matrices are given for each threshold as well as their False/True positive rates on the ROC curve.

Results for Lasso model:

```
Confusion matrix for threshold = 0.5
 [[0.03085208 0.19852534]
 [0.02184307 0.74877951]]

Confusion matrix for threshold = 0.6
 [[0.05843273 0.17094469]
 [0.05601691 0.71460567]]

Confusion matrix for threshold = 0.7
 [[0.1041069  0.12527052]
 [0.139237   0.63138558]]

Confusion matrix for threshold = 0.75
 [[0.1365947  0.09278273]
 [0.21777644 0.55284614]]

Confusion matrix for threshold = 0.8
 [[0.17152348 0.05785394]
 [0.33059037 0.44003221]]

Confusion matrix for threshold = 0.85
 [[0.20038754 0.02898988]
 [0.47392924 0.29669334]]
```

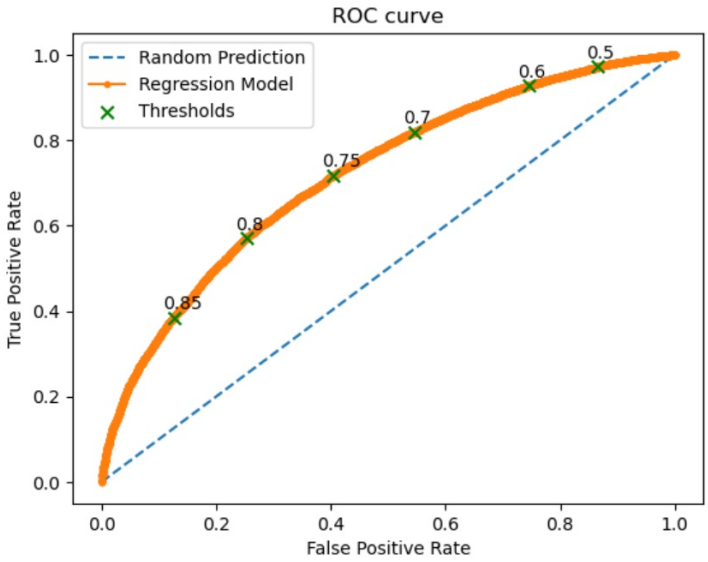

Figure 11: Different probability thresholds applied to the **lasso** model. Confusion matrices are given for each threshold as well as their False/True positive rates on the ROC curve.

Results for Elastic Net model:

```
Confusion matrix for threshold = 0.5
 [[0.03145604 0.19792139]
 [0.02277417 0.74784841]]

Confusion matrix for threshold = 0.6
 [[0.05961548 0.16976194]
 [0.05747647 0.71314611]]

Confusion matrix for threshold = 0.7
 [[0.10508833 0.12428909]
 [0.13953898 0.6310836 ]]

Confusion matrix for threshold = 0.75
 [[0.13634305 0.09303438]
 [0.21727314 0.55334944]]

Confusion matrix for threshold = 0.8
 [[0.17034073 0.05903669]
 [0.32716795 0.44345463]]

Confusion matrix for threshold = 0.85
 [[0.19885248 0.03052494]
 [0.46464341 0.30597916]]
```

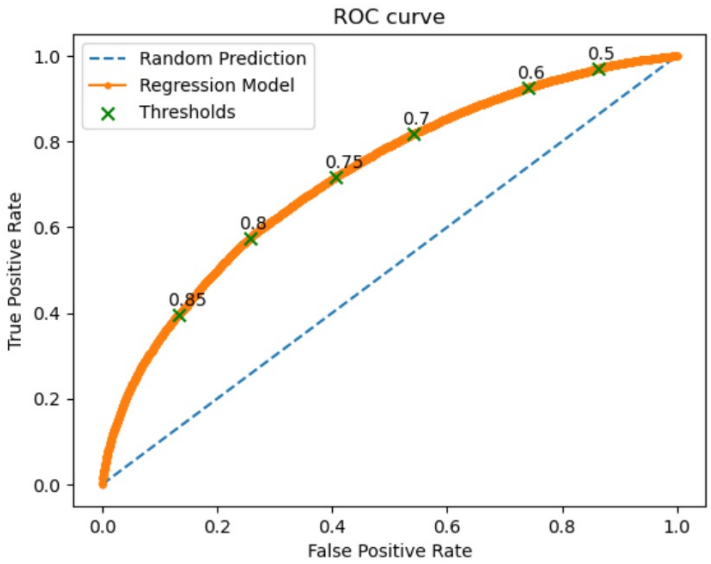

Figure 12: Different probability thresholds applied to the **elastic net** model. Confusion matrices are given for each threshold as well as their False/True positive rates on the ROC curve.