



Report

Assignment 2

Linear and Logistic Regression



Author:

Findlay Forsblom ff22ey

Termin: VT20

Ämne: Machine Learning

Kurskod: 2DV516



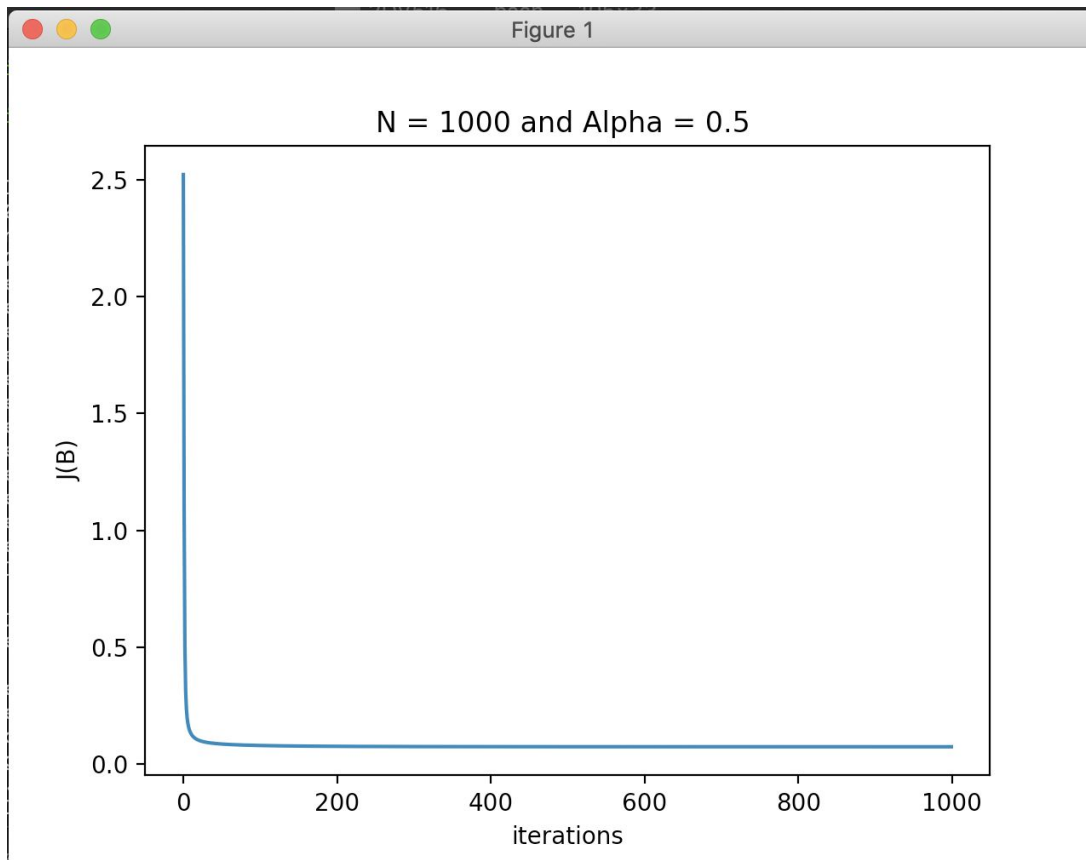
Table of contents

Table of contents	2
Exercise 3	3
Exercise 7 VG-exercise	5
Standard Linear Regression	6
Polynomial Linear Regression	7
Polynomial Linear Ridge regression, Degree 2	8
Polynomial Linear Lasso regression, Degree 2	10
Polynomial Linear elastic net regression, Degree 2	12
Model Selection and Assessment	13



Exercise 3

In this exercise we were asked to classify woman breast cancer tumours using logistic regression. We were first asked to shuffle the whole data and then divide the data into a training and test set size of our choice. Then using the training data normalize it and train a logistic model using gradient descent.



The picture above shows the cost function after 1000 iterations when using an alpha of 0.5

After training the model we were asked to use both the training set and test set to evaluate the model. But since the data was shuffled i always got different results, Down below are the results gotten where 3 tests were run using 80/20 % split.

On the first test i got the following results

```
Task 4
Traning error = 15, Traning accuracy = 97.25274725274726

Task 5
Test errors = 4, Test accuracy = 97.08029197080292
```

On the second try i got the following results



Task 4

Traning error = 17, Traning accuracy = 96.88644688644689

Task 5

Test errors = 3, Test accuracy = 97.81021897810218

and on the final try i got the following results

Task 4

Traning error = 19, Traning accuracy = 96.52014652014653

Task 5

Test errors = 2, Test accuracy = 98.54014598540147

As can be seen from the tests above both the there were little variance in both the Training accuracy and the test accuracy

I later then tried changing the ratio to like 60/40 50/50 40/50 but i did not notice a big change in the variance of both the test accuracy and the training accuracy



Exercise 7 VG-exercise

In this exercise we were asked to find a good linear model along with some regularization techniques that predicts the insurance charges given certain traits of the policyholders.

To begin since some of the feature contained categorical variables which were in text they had to be converted to numerical values since that's what most machine learning algorithms work with.

The dataset contained two features that contained categorical values, the first being the sex column which had two categories male and female and the other being the smokers column which also has two categories yes and no, and finally the last being the region column which contained more than two categories

Starting off with the sex and smoker columns, since these columns contain binary values such as male or female and yes or no, a simple labelencoder from the sklearn.preprocessing library was used.

	age	sex	bmi	children	smoker
0	19.00000	female	27.90000	0.00000	yes
1	18.00000	male	33.77000	1.00000	no
2	28.00000	male	33.00000	3.00000	no
3	33.00000	male	22.70500	0.00000	no
4	32.00000	male	28.88000	0.00000	no
5	31.00000	female	25.74000	0.00000	no
6	46.00000	female	33.44000	1.00000	no
7	37.00000	female	27.74000	3.00000	no
8	37.00000	male	29.83000	2.00000	no
9	60.00000	female	25.84000	0.00000	no
10	25.00000	male	26.22000	0.00000	no
11	62.00000	female	26.29000	0.00000	yes
12	23.00000	male	34.40000	0.00000	no
13	56.00000	female	39.82000	0.00000	no
14	27.00000	male	42.13000	0.00000	yes
15	19.00000	male	24.60000	1.00000	no
16	52.00000	female	30.78000	1.00000	no
17	23.00000	male	23.84500	0.00000	no
18	56.00000	male	40.30000	0.00000	no
19	30.00000	male	35.30000	0.00000	yes
20	60.00000	female	36.00500	0.00000	no

	0	1	2	3	4
0	19	0	27.9	0	1
1	18	1	33.77	1	0
2	28	1	33.0	3	0
3	33	1	22.705	0	0
4	32	1	28.88	0	0
5	31	0	25.74	0	0
6	46	0	33.44	1	0
7	37	0	27.74	3	0
8	37	1	29.83	2	0
9	60	0	25.84	0	0
10	25	1	26.22	0	0
11	62	0	26.29	0	1
12	23	1	34.4	0	0
13	56	0	39.82	0	0
14	27	1	42.13	0	1
15	19	1	24.6	1	0
16	52	0	30.78	1	0
17	23	1	23.845	0	0
18	56	1	40.3	0	0
19	30	1	35.3	0	1
20	60	0	36.005	0	0

The picture to the left depicts the original dataset with the sex and smoker columns as they were, whilst the second picture to the left shows how the sex and smoker has been encoded, in the sex column, female is a 0 and male is a 1, and in the smoker column, yes is a 1 and no is a 0

Moving on to the regions column, this one was a bit more tricky to encode since it contained more than two categories. Here, a simple label encoding would not work, since there are 4 different categories would then let's say give 1 to southeast and 3 to southwest then the machine learning algorithm might think that southwest is better than southeast since $3 > 1$ but that is not the case.

So to solve this problem so called dummy variables were created (also known as one hot encoding). This creates one binary attribute per category. This is depicted by the picture below



	smoker	shoesize	northeast	northwest	southeast	southwest
0	1	36	0.00000	0.00000	0.00000	1.00000
1	0	43	0.00000	0.00000	1.00000	0.00000
2	0	43	0.00000	0.00000	1.00000	0.00000
3	0	41	0.00000	1.00000	0.00000	0.00000
4	0	42	0.00000	1.00000	0.00000	0.00000
5	0	37	0.00000	0.00000	1.00000	0.00000
6	0	38	0.00000	0.00000	1.00000	0.00000
7	0	38	0.00000	1.00000	0.00000	0.00000
8	0	40	1.00000	0.00000	0.00000	0.00000
9	0	36	0.00000	1.00000	0.00000	0.00000
10	0	43	1.00000	0.00000	0.00000	0.00000
11	1	35	0.00000	0.00000	1.00000	0.00000
12	0	43	0.00000	0.00000	0.00000	1.00000
13	0	35	0.00000	0.00000	1.00000	0.00000
14	1	41	0.00000	0.00000	1.00000	0.00000
15	0	42	0.00000	0.00000	0.00000	1.00000
16	0	38	1.00000	0.00000	0.00000	0.00000
17	0	41	1.00000	0.00000	0.00000	0.00000
18	0	45	0.00000	0.00000	0.00000	1.00000
19	1	40	0.00000	0.00000	0.00000	1.00000
20	0	36	1.00000	0.00000	0.00000	0.00000

To avoid the dummy variable trap, one of the dummy columns was later removed. and now the dataset was ready to be used

Standard Linear Regression

Looking at the documentation for the sklearn linear regression class showed that if *fit_intercept* was set to true which was the default option then data would then be normalized. So the normalization was done by the library itself thus no need for normalization on my part.

After training a standard linear regression model, it was evaluated on both the training set and on a validation set, Down below are the results that were gotten

```
training error 38759995.574456125
validation error 39570881.52109474
```

As can be seen from the picture above it both the training error and the validation are as high, meaning that it suffers from high bias. This means that a simple linear regression is too simple for the problem and it underfits the data

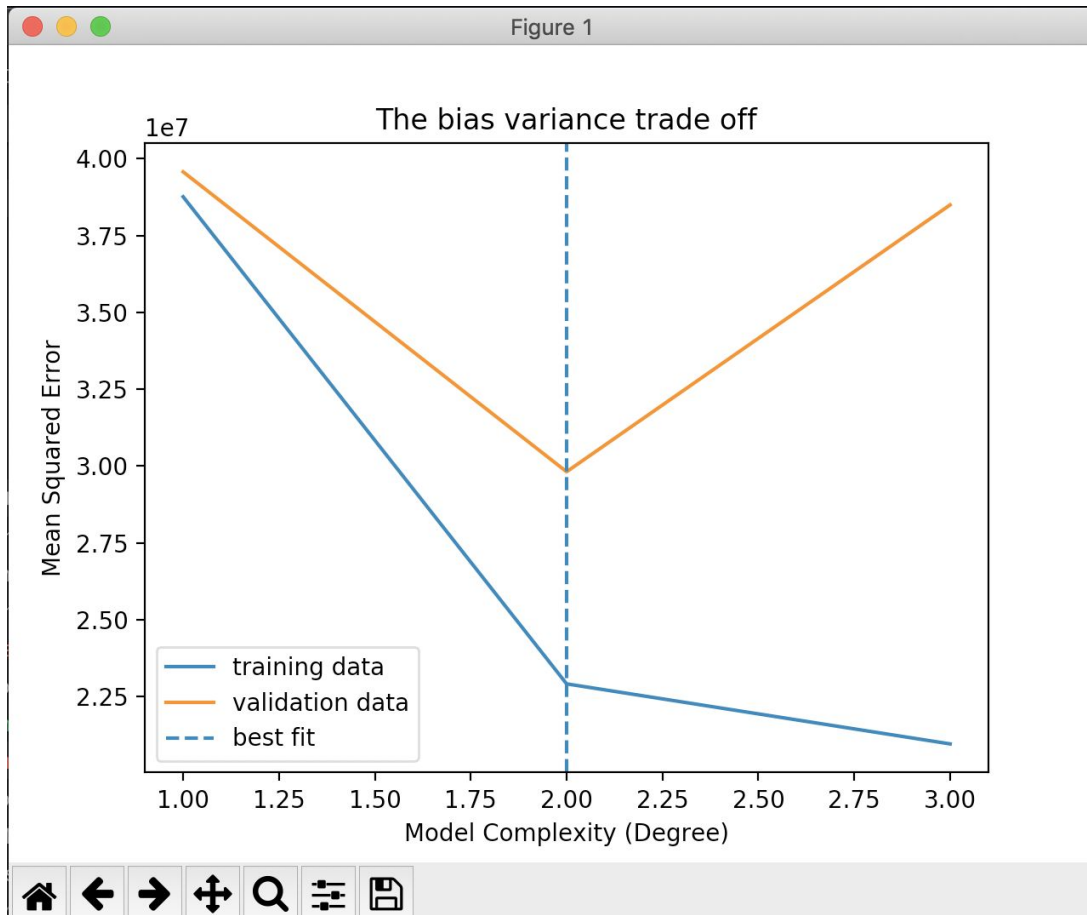
Polynomial Linear Regression

As constated above a simple linear regression model was to simple it made the model to suffer from high bias. A fix to this issue is to add more features, for example transformed features, and that was what was done

When creating polynomial regression the library `sklearn.preprocessing.Polynomialfeatures` was used to create a polynomial array from the X array.



The test was done using two extra degrees (degree 2 and 3). The picture below shows the result that was gotten



From the picture above we can come to conclusion that using degree 2 seems to be the sweetspot between model complexity and test errors. But taking a closer look at the picture below we can see the following

```
Degree 2
training error 22910568.283460163
validation error 29812457.880143624
```

We see that both the training error and validation error has been reduced by a long shot in comparison to when we used degree 1, but we can see that there is still a quite noticeable gap between the two errors, So the model is now suffering from high variance



Polynomial Linear Ridge regression, Degree 2

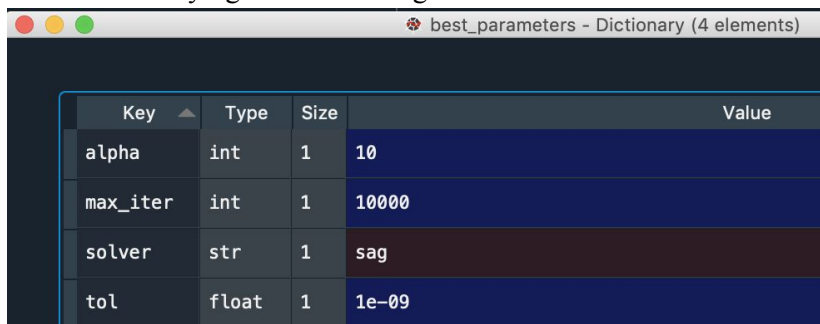
As mentioned earlier, the model seemed to be suffering from high variance. A common way fix is by using regularization techniques, this prevents the model from overfitting the training data.

The first regularization technique that was tested was ridge regression. In order to find the best hyper parameters to be used for ridge regression grid search was used.

Looking at the documentation for ridge regression, apart from the alpha parameter they were 3 other hyper-parameters which i thought could matter, They were *solver*, *max_iter* and *tol* parameters. Instead of writing all possible numbers alpha and max_iter could be, that would have taken forever to train, I instead chose few values and then changed those values depending on the results gotten. The picture below shows my first parameters that were searched

```
from sklearn.model_selection import GridSearchCV
parameters = [{'alpha': [1, 5, 10],
                  'max_iter': [1000, 2000, 5000, 10000],
                  'solver': ['auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga'],
                  'tol': [1e-5, 1e-1, 1e-9]}]
```

After the first try i got the following results



Key	Type	Size	Value
alpha	int	1	10
max_iter	int	1	10000
solver	str	1	sag
tol	float	1	1e-09

```
In [126]: best_mse
Out[126]: -24971492.236181326
```

As seen from the pictures above the best parameters was when alpha was set to 10 and solver set to sag, and max_iter value 10000. Those parameters gave a validation error of 24971492.236181326 which is already better than the standard polynomial regression.

But can we do better? looking at the pictures above, we see that the best alpha is said to be 10 and 10, and max_iter was set to 10000 and it used 10000, so maybe i did not even converge. Another test was made changing the alpha and max_iter values only as depicted in the picture below

```
from sklearn.model_selection import GridSearchCV
parameters = [{'alpha': [10, 20, 50],
                  'max_iter': [15000, 5000, 10000],
                  'solver': ['auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga'],
                  'tol': [1e-5, 1e-1, 1e-9]}]
```

The picture below shows the result gotten from the second run



best_parameters - Dictionary (4 elements)

Key	Type	Size	Value
alpha	int	1	50
max_iter	int	1	15000
solver	str	1	sag
tol	float	1	1e-05

```
In [130]: best_mse  
Out[130]: -24970446.95636032
```

As can be seen from the results above an alpha of 50 gave the best and a max_iter of 15000, but this time around the best tol was 1e-05. Since the solver stayed the same after 2 rounds i removed it in the third round, since it seemed to be the best. On the third and final round this were the parameters and their values used:

```
from sklearn.model_selection import GridSearchCV  
parameters = [{'alpha': [50, 75, 100],  
               'max_iter': [15000, 5000, 10000],  
               'tol': [1e-5, 1e-6, 1e-7, 1e-4, 1e-3, 1e-2, 1e-1, 1e-8]}]
```

and this were the results gotten

best_parameters - Dictionary (3 elements)

Key	Type	Size	Value
alpha	int	1	50
max_iter	int	1	15000
tol	float	1	1e-05

In conclusion the best hyper parameters for ridge regression seemed to be
alpha = 50
max_iter = 15000
tol = 1e-05
solver = sag



Polynomial Linear Lasso regression, Degree 2

The lasso regression was the second regularization tested. Lasso regression had a bit more parameters than i thought mattered. The picture below shows the parameters that were used on the first attempt

```
parameters = [{'alpha': [0.5, 1, 10],  
               'max_iter': [1000, 3000, 6000],  
               'tol': [1e-5, 1e-9, 1e-1],  
               'selection': ['random', 'cyclic'],  
               'random_state': [True, False],  
               'precompute': [True, False],  
               'positive': [True, False],  
               'warm_start': [True, False],}]
```

And this were the results gotten

Key	Type	Size	Value
alpha	int	1	1
max_iter	int	1	1000
positive	bool	1	False
precompute	bool	1	False
random_state	bool	1	True
selection	str	1	cyclic
tol	float	1	1e-05
warm_start	bool	1	True

```
In [149]: best_accuracy  
Out[149]: -24964041.804857507
```

As can be seen from the pictures above alpha was set surprisingly to 1. so i tried another test with values around 1 and i got the following results

Key	Type	Size	Value
alpha	float	1	0.9
max_iter	int	1	6000
positive	bool	1	False
precompute	bool	1	False
random_state	bool	1	True
selection	str	1	cyclic
tol	float	1	1e-06
warm_start	bool	1	True



All the other values stayed the same apart from the alpha, max_iter and tol values. I tried one more round where i only had to those values but with more values and this were the results gotten

Key	Type	Size	Value
alpha	int	1	1
max_iter	int	1	3000
tol	float	1	0.01

```
In [160]: best_accuracy  
Out[160]: -24908247.61597627
```

In conclusion the best hyper-parameters for the lasso regression were as follows

alpha = 1

max_iter = 3000

tol = 0.01

random_state = True

selection = cyclic

warm_start = True



Polynomial Linear elastic net regression, Degree 2

So far the Lasso regression seems to be the one that gives the accuracy, but not by much. But what happens if the 2 are combined using elastic net regression?

The picture below show the parametes that were started with

```
parameters = [{'alpha': [1, 0.5, 10],  
               'tol': [1e-5, 1e-1, 1e-8],  
               'positive': [True, False],  
               'random_state': [True, False],  
               'max_iter': [3000, 6000, 10000, 15000],  
               'selection': ['cyclic', 'random'],  
               'precompute': [True, False],  
               'l1_ratio': [0, 0.5, 0.75, 1]}]
```

and this were the results gotten from the elastic net

Key	Type	Size	Value
alpha	int	1	1
l1_ratio	int	1	1
max_iter	int	1	3000
positive	bool	1	False
precompute	bool	1	False
random_state	bool	1	True
selection	str	1	cyclic
tol	float	1	1e-05

```
In [168]: best_mse  
Out[168]: -24964041.804857507
```

Once again alpha seems to be centered around 1, so this was changed in the second and third try but with more values around 1 and the following were my best values

```
alpha = 1  
l1_ratio = 1  
max_iter = 3000  
tol = 0.01  
warm_start = True  
random_state = True  
precompute = False  
selection = cyclic
```

with the following mse

```
In [175]: best_mse  
Out[175]: -24908247.61597627
```



Model Selection and Assessment

As can be seen from the results gotten above, we can see that lasso and elastic net gave the lowest validation error and also the same validation error, meaning that only the betas from lasso was taken in elastic net. So it does not really matter which one goes with, but i decided to use lasso

When assessing the model using lasso assessed it with a test set that was set aside from the start. Using the score function it gave a score of 0.85 where 1 is the best with a generalization error of 21175081.726287294