FRTN50 hand in 2

Karl Tengelin & Daniel Jogstad

October 2019

1 Introduction

Task 1

> Use `ProximalOperators.jl` and the proximal gradient method to solve the problem for the data given in the `fileproblem.jl`, a suitable step-size is $\left\|XX^T\right\|^{-1}$. Use a pre-scaling so $r\left(x_i\right) \in (-1,1)$ for all i. Hint: the package `LinearAlgebra` contains the function `eigvals` for calculating the eigenvalues of XX^T.

For the first task, an optimal value of ω was calculated to $[-0.801, -1.529]$ by using the recommended step size.

See task1.jl

Task 2

> For $p = 1$ and $\lambda = 0$, plot and compare the resulting model $m_w(x)$ and the data (xi, yi). Increase p in steps up to $p = 10$ and study what happens. What happens with the model? In which ways is it better and/or worse? How many iterations does it take to find a solution?

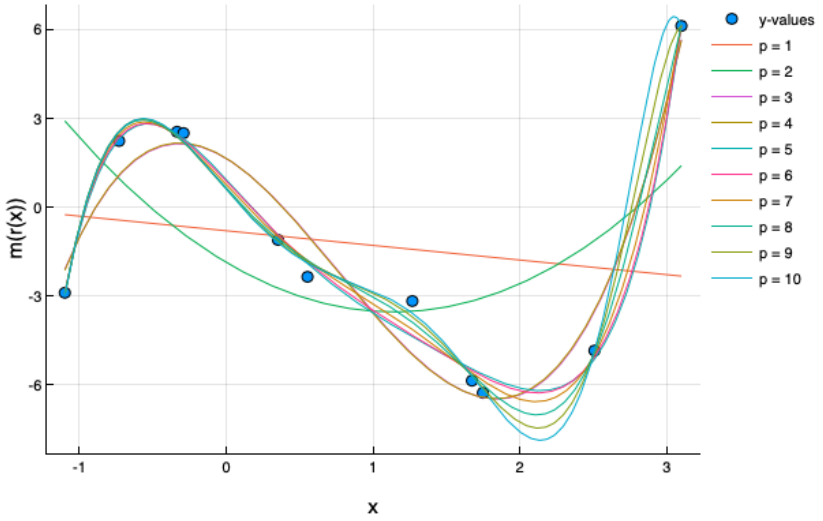


Figure 1: shows the original data points y, as well as the model of rescaled data (m(r(x)) for different p.

p:	1	2	3	4	5	6	7	8	9	10
Iterations:	207	2128	20903	245823	2605795	60949535	$>61e^6$	$>61e^6$	$>61e^6$	$>61e^6$

Table 1: Shows how many iterations needed for the model to converge for different p:s

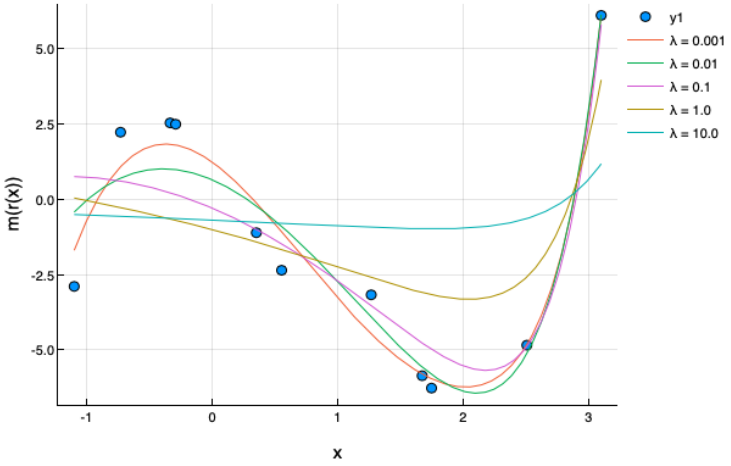
We can clearly see that for higher degree polynomials the model is more and more adjustable and can fit more and more of the data points "correctly". Initially one might think that since the highest degree polynomial (p = 10) fits the given data the best it is also the best model. However one can see that in order for the polynomial to fit all the data it has to do some peculiar twists and turns. Look for example around the last data point (3.1,6.11) in figure 1 where we can see that the p = 10 (turquoise plot) polynomial has done a "u-turn" at above 6.11 and come down to meet the data point. This is a typical example of overfitting and the essence is that the model is now probably not representable for a larger data set. On the other hand p = 1 also appears quite bad in fitting a larger data set so a reasonable guess would be that a polynomial of degree 5 or similar would give the better representation of a bigger data set.

Also a higher degree polynomial requires many more iterations for the proximal gradient method to converge, which can be seen in table 1. It appears that the number of iterations needed for convergence increase by roughly a factor 10 for each degree in the polynomial. This is also what is to be expected since larger polynomials mean more coefficients to tweak in order to give a good fit, the

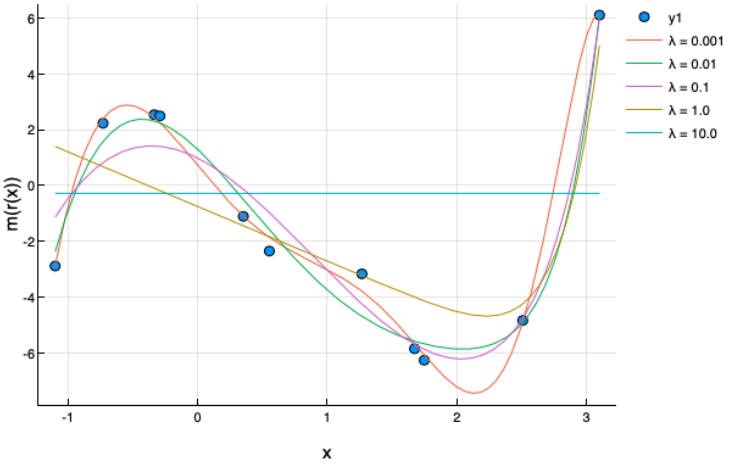
model needs to adjust more variables.

Task 3

Set $p = 10$ and introduce regularization with $q = 2$. Try different λ in the range [0.001,10] and study the resulting model $m_\omega(x)$, the solution ω^* and convergence rate. Repeat for $q = 1$. The regularization promotes ω^* to be small in $\|\cdot\|_q^q$-sense, how does that manifest in ω^* and $m_{\omega^*}(x)$? Do you find structure in any of the solutions?



(a) Model for different λ and $q = 2$



(b) Model for different λ and $q = 1$

If we look at $\min_\omega \frac{1}{2}\left\|X^T w - y\right\|_2^2 + \lambda\|w\|_q^q$

One can see that if we increase the value of λ the penalty for ω is increased, meaning that a high lambda would force the omega values to be small in order to minimize the function. This is also what we can see if we look in figure (b). Here small values of λ allows the ω-values to be big and big λ-values suppresses ω. One can also see in (a) that if λ is chosen to be small the model starts to follow the data points more closely, which indicates that a small λ (i.e a small regularising term) might lead to overfitting of the data. However in (a) it appears that the smallest lambda in our case gives quite a good representation without overfitting.

A similar conclusion can be drawn when looking at the effect of changing q. If we choose q as 2 instead of 1 the regularising term is more strict, i.e the penalising term grows quadratically with ω rather than linearly. This means that for q = 2 the model favours smaller values of w compared to q = 1. Also it means that the 2-norm regularization promotes more homogenous values of w. We can see this when looking at arrays 1 and 2 where quite a few values are zero in 2 whereas in 1 they might be small but never 0. And the other, non zero, values in 2 are quite a bit bigger than what we see in 1. This is due to the quadratic nature of the 2-norm. If the model would produce values as big as seen in 2 the penalising term would be huge if the penalty was quadratic. However when the penalty isn't quadratic, as is the case for q = 1, it pays of to set some variables to zero in order to focus more on variables that are more important (in the sense that they produce a model with lower penalty).

So in essence the 2-norm promotes equally sized parameters whereas the 1-norm is more flexible in a sense, it can choose to set a parameter to zero in order to increase another. It doesn't matter what the internal distance between the parameter values are but only the total "size" of the parameters matter. Hence we can observe zero values in 2 combined with large non-zero values but in 1 the internal distance between parameters are quite small in comparison.

One can also see in figure (b) that the 1-norm appears to be less regularized and more prone to overfitting. For small values of lambda one can see that the model shows tendencies of overshooting the data points (see for example $\lambda = 0.001$ in the lower part of the graph in figure (b)), which is a sign of overfitting.

The calculated w^* for the 2-norm:

$$\left[\begin{array}{ccccc} w^*_{0.001} & w^*_{0.01} & w^*_{0.1} & w^*_1 & w^*_{10} \\ \hline 1.236 & 0.647 & -0.291 & -1.004 & -0.689 \\ -9.251 & -5.567 & -5.461 & -3.439 & -0.577 \\ -29.248 & -19.617 & -6.77 & -1.407 & -0.143 \\ 50.177 & 12.711 & 0.889 & 0.004 & 0.089 \\ -16.887 & 1.186 & 0.643 & 0.571 & 0.209 \\ 1.732 & 5.617 & 2.265 & 1.063 & 0.293 \\ -4.582 & 3.286 & 2.5 & 1.332 & 0.344 \\ 1.012 & 3.189 & 2.867 & 1.537 & 0.381 \\ 2.191 & 2.203 & 2.98 & 1.675 & 0.407 \\ 4.309 & 1.571 & 3.069 & 1.778 & 0.425 \\ 5.41 & 0.902 & 3.104 & 1.855 & 0.44 \end{array}\right] \tag{1}$$

The calculated w^* for the 1-norm:

$$\left[\begin{array}{ccccc} w^*_{0.001} & w^*_{0.01} & w^*_{0.1} & w^*_1 & w^*_{10} \\ \hline 0.715 & 1.292 & 0.979 & -0.759 & -0.284 \\ -17.513 & -13.594 & -7.242 & -6.072 & 0.0 \\ 1.878 & -28.431 & -26.278 & 0.0 & 0.0 \\ 132.436 & 88.228 & 30.979 & 0.0 & 0.0 \\ -266.942 & -61.373 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 96.871 & 0.0 & 0.0 & 0.0 & 0.0 \\ 162.581 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 19.983 & 0.0 & 0.0 & 0.0 \\ -103.912 & 0.0 & 7.56 & 11.855 & 0.0 \end{array}\right] \tag{2}$$

An interesting remark one could do here when looking at 2 is that for higher lambda values the 1-norm strongly favours the 0-polynomial, i.e the constant.

The convergence rate for the 2-norm was calculated as:

$$\left[\begin{array}{cc} \lambda & \text{iter to conv} \\ 0.001 & 443048 \\ 0.01 & 47194 \\ 0.1 & 5088 \\ 1 & 565 \\ 10 & 76 \end{array}\right] \tag{3}$$

The convergence rate for the 1-norm was calculated as:

$$\begin{bmatrix} \lambda & \text{iter to conv} \\ 0.001 & > 10000000 \\ 0.01 & 2095846 \\ 0.1 & 96728 \\ 1 & 7911 \\ 10 & 36 \end{bmatrix} \tag{4}$$

An observation we can make here is that λ seems to affect the convergence rate more for the 1-norm compared to the 2-norm.

Task 4

Chose a configuration, q and λ, from Task 3 and remove the pre-scaling. What affect did it have? Was it necessary?

By choosing $q = 1$ and $\lambda = 0.01$ we got the following results by not using pre-scaling:

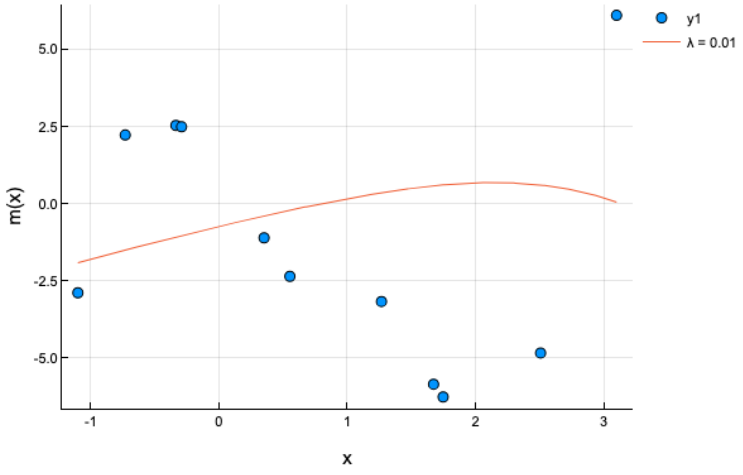


Figure 3: Shows the model with lambda = 0.01 and q = 1 after $1e^8$ iterations along with the data points y

The norm of the new solution vs the old solution ($w_old - w$) after $1e^8$ iterations was $1.31 \cdot 10^{-10}$, so the solution hadn't converged even after $1e^8$ iterations. This shows that more than anything the pre-scaling was important to assure convergence within a reasonable time frame. The resulting w-values are shown in the array below.

$$\begin{bmatrix} w^*_{0.01} \\ -0.652 \\ 2.992 \\ -0.615 \\ -0.387 \\ -1.342 \\ 0.451 \\ 0.069 \\ -0.94 \\ 0.898 \\ -0.286 \\ 0.03 \end{bmatrix} \tag{5}$$

As we can see, the model without pre-scaling does not do a good representation of the original data points for some coordinates i, which makes it clear that pre-scaling probably was necessary.

The prescaled and non-prescaled problem are equivalent apart from the regularization term. The regularization term does not take into consideration the pre-scaling and as a result when the pre-scaling is removed the regularization doesn't affect the higher order terms.

This might explain why we don't see zero values in 5 as we did for $w^*_{0.01}$ in 2. Since the higher (p = 4 to p = 9) terms are not as strongly regularized they can have a non zero value without the model being punished for it.

Task 5

Derive an expression for recovering the primal solution from the dual. Using this expression, show that it is possible to evaluate the model $mw(\hat{x}) = sign(w^T\Phi(\hat{x}))$ with only the kernel, data points (xi, yi), and dual solution.

Our dual problem is defined as:

$$\min_{\nu} h^*(\nu) + \frac{1}{2\lambda}\| - XY\nu\|_2^2 \tag{6}$$

Where we identify $g^*(z)$ as $\frac{1}{2\lambda}\|z\|_2^2$.

We insert the point we want to examine g^* at $(-XY\nu)$ and rewrite the 2-norm:

$$g^*(-XY\nu) = \frac{1}{2\lambda}\nu^T YX^T XY\nu \tag{7}$$

And since g^* is differentiable we recover a primal solution from the primal-dual optimal condition which is given by:

$$w = \delta g^*(-XY\nu) = -\frac{1}{\lambda}XY\nu \tag{8}$$

We insert the w-solution into our model expression ($m_w(\hat{x}) = sign(w^T\varphi(\hat{x}))$) to get the following:

$$m_w(\hat{x}) = sign(-\frac{1}{\lambda}\nu^T Y^T X^T \varphi(\hat{x})) \tag{9}$$

We know that the X matrix contains our feature mapping: $X = [\varphi(x_1), \varphi(x_2),..., \varphi(x_N)]$ so we can write $X^T\varphi(\hat{x})$ as:

$$\begin{bmatrix} \varphi(x_1)^T \\ \varphi(x_2)^T \\ \vdots \\ \varphi(x_N)^T \end{bmatrix} \varphi(\hat{x}) = \varphi(x)^T\varphi(\hat{x}) = K(x,\hat{x}) \tag{10}$$

Meaning that we can rewrite 9 as:

$$m_w(\hat{x}) = sign(-\frac{1}{\lambda}\nu^T Y^T \mathbb{K}) \tag{11}$$

Where $\mathbb{K}$ is the matrix that contains the kernels K.

Hence we can evaluate the model using only the kernels K, data points x_i and y_i, and dual solution ν.

Task 6

> Given the data from svm_train() in problem.jl, use the proximal gradient method to implement a solver for the SVM dual problem with a Gaussian kernel, $K(x_i, x_j) = e^{-\frac{1}{2\sigma^2}\|x_i - x_j\|_2^2}$. Implement the predictive model, $m_w(x)$, using the results from Task 5. Note that ProximalOperators.jl can perform conjugation for you and that the data is already appropriately scaled. A suitable step-size is $1/\|Q\|$.

See julia file.

Task 7

Try different hyper-parameters λ and σ over coarse grids, for instance $\lambda \in 0.1, 0.01, 0.001, 0.0001$ and $\sigma \in 1, 0.5, 0.25$. For each configuration, calculate the error rate on the validation data from svm_test_1() but also calculate the error rate on the training data. Are the error rates on the two sets the same? What is the lowest error rate you can achieve on each set? Can you identify overfitting? Based on your findings, select the λ and σ you feel is best.

Running the proximal gradient method for each λ and σ the following error rates were achieved for the test set and training set:

$$\text{test set: } \begin{bmatrix} & \sigma_1 & \sigma_2 & \sigma_3 \\ \lambda_1 & 0.31 & 0.15 & 0.09 \\ \lambda_2 & 0.21 & 0.09 & 0.09 \\ \lambda_3 & 0.11 & 0.08 & 0.11 \\ \lambda_4 & 0.09 & 0.07 & 0.09 \end{bmatrix} \quad \text{train set: } \begin{bmatrix} & \sigma_1 & \sigma_2 & \sigma_3 \\ \lambda_1 & 0.306 & 0.174 & 0.05 \\ \lambda_2 & 0.22 & 0.14 & 0.05 \\ \lambda_3 & 0.154 & 0.054 & 0.006 \\ \lambda_4 & 0.07 & 0.01 & 0.0 \end{bmatrix} \tag{12}$$

We can see in 12 that the optimal λ,σ-pair for the test set appears to be (0.0001,0.5) which gives an error rate of 0.07. For the training set the best pair seems to be (0.0001,0.25) which gives the error rate 0.0. We can also see that generally the error rates are quite similar for the two different sets (specifically the big errorrates are in the top left part of the matrix and the low error rates in the lower right), however they do differ slightly. They especially seem to differ for smaller values of σ, and it is also for smaller values of σ we can identify something that we believe to be overfitting. If one compares the matrices at element (3,3) and (4,3) in 12 we see that the error rate is very low for the training set but not particularly low for the test set. This indicates that the hyperparameters produce a model that is very good at classifying the training data but not the test data, and this is a indication that the model over fits.

Based on these findings we choose λ as 0.0001 and σ as 0.5.

Task 8

Four different validation sets are contained in problem.jl: svm_test_1(), svm_test_2(), svm_test_3(), and svm_test_4(). Does your selected model from Task 7 have the same error rate on all four validation sets? Try different hyperparameters, would you have made different hyper-parameter choices if you had been given a different validation set?

Bias have intentionally been introduced to two of the validation sets, making them unsuitable to validate the model on. Can you identify the biased sets? (Without looking at the code.) Can you explain what has been done to them and explain why it is bad? Hint: Look at the error rates on both the training and validation sets for different hyperparameter choices. For instance, (λ, σ) = (0.1, 2), (λ, σ) = (0.001, 0.5), and (λ, σ) = (0.00001, 0.25). It can also be a good idea to compare with a constant classifier (m(x) = 1 or m(x) = 1).

Using the model derived in task 7 we get the following error rates for each validation set:

val_1	val_2	val_3	val_4
0.07	0.02	0.11	0.07

(13)

We can also check the error rates for each validation set using the same λ values and σ values that were tested in task 7 (λ = [0.1 0.01 0.001 0.0001] and σ = [1 0.5 0.25]).

$$\text{val set 1:} \begin{bmatrix} 0.31 & 0.15 & 0.09 \\ 0.21 & 0.09 & 0.09 \\ 0.11 & 0.08 & 0.11 \\ 0.09 & 0.07 & 0.09 \end{bmatrix} \quad \text{val set 2:} \begin{bmatrix} 0.2 & 0.09 & 0.03 \\ 0.14 & 0.07 & 0.03 \\ 0.12 & 0.05 & 0.01 \\ 0.04 & 0.02 & 0.0 \end{bmatrix} \tag{14}$$

$$\text{val set 3:} \begin{bmatrix} 0.38 & 0.22 & 0.14 \\ 0.25 & 0.2 & 0.14 \\ 0.19 & 0.15 & 0.12 \\ 0.14 & 0.11 & 0.14 \end{bmatrix} \quad \text{val set 4:} \begin{bmatrix} 0.13 & 0.13 & 0.12 \\ 0.17 & 0.09 & 0.12 \\ 0.11 & 0.07 & 0.09 \\ 0.09 & 0.07 & 0.11 \end{bmatrix} \tag{15}$$

Here we can see that the best pair for each validation set would be:

validation set:	1	2	3	4
(λ,σ):	λ_4,σ_2	λ_4,σ_3	λ_4,σ_2	λ_4,σ_2 or λ_3,σ_2

Table 2: Shows the optimal λ-σ pair when using each of the 4 given validation data sets

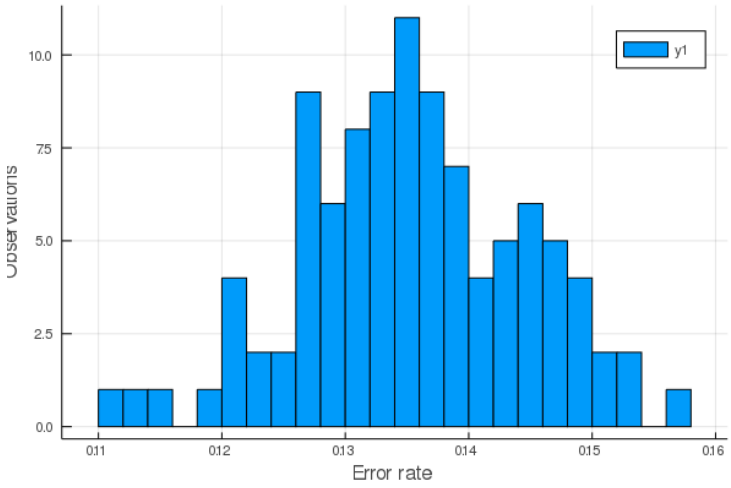
So if we were given validation set 2 we would have chosen a different λ,σ-pair and one could argue that we could have chosen a different λ,σ-pair if we were given validation set 4.

Two interesting remarks concerning validation set 2 and 4 is that for validation set 2 we get an error rate of 0 for our optimal λ,σ-pair and this brings us to believe that the validation set 2 is very similar to the training set, hence hyperparameters that would normally overfit the training data gives a good classifier for the validation data as well, which would suggest good parameter values when in reality the model is overfitted for the training data.

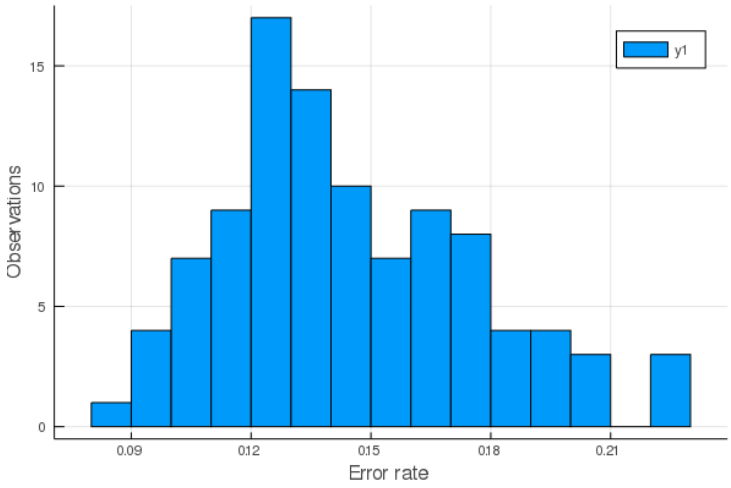
For validation set 4 we can observe that all error rates are quite similar and this is something that we don't observe for the other validation sets. Also if we check the labels for validation set 4 we can see that the majority of the elements have label -1 (91 of 100). This means that we test our model on a data set that more or less only contains one class of data. This is bad since we want a validation set that is representative for a large amount of data. If the data only contains one class more or less it isn't representative and thus introduced a bias.

Task 9

Using the λ and σ you chose from Task 7, compare the error rate estimates given by 10-fold cross-validation and holdout cross-validation. Use only the data from svm_training() and randomly set aside 100 data points for the holdout cross-validation. Do the two methods give similar error rate estimates? If you re-run the cross-validations with different random permutations of the training/validation splits, are the error rates the same? How much do they vary? Examine the variance and/or create a histogram of the estimated error rates. Which cross- validation method seem more reliable? Hint: To randomly permute the data, look at the function randperm in the standard library Random.



(a) Histogram of 100 error rates for 10-fold cross validation.



(b) Histogram of 100 error rates for holdout validation.

We can see in the histograms for 10-fold cross validation and holdout cross validation that the variance for the 10-fold cross validation is smaller than for holdout cross validation. This is also what is to be expected since 10-fold cross validation uses the data set several times and calculates an average error rate whereas the holdout cross validation does not calculate a mean average error but only produce a single value each run. However we can see that the average error rate over 100 iterations is quite similar for the both methods at around 0.135-0.145.

This leads us to believe that the 10-fold cross validation is more reliable than holdout cross validation, however the computational cost for the 10-fold cross validation is much larger than for holdout cross validation. Especially if one wants to check error rates for different hyperparameters the method would be very costly.