Classification Metrics

Classification on the spambase dataset

- Download the data from the Github-repo
- Load the data in python (note that the csv file has no header, see https://archive.ics.uci.edu/ml/datasets/spambase for a description of the columns)
- Most columns describe the frequency of certain words/elements appearing in the email
- Last column of data: e-mail was considered spam (1) or not (0)

	0	1	2	3	4	5	6	7	8	9	...	48	49	50	51	52	53	54	55	56	57
0	0.00	0.64	0.64	0.0	0.32	0.00	0.00	0.00	0.00	0.00	...	0.00	0.000	0.0	0.778	0.000	0.000	3.756	61	278	1
1	0.21	0.28	0.50	0.0	0.14	0.28	0.21	0.07	0.00	0.94	...	0.00	0.132	0.0	0.372	0.180	0.048	5.114	101	1028	1
2	0.06	0.00	0.71	0.0	1.23	0.19	0.19	0.12	0.64	0.25	...	0.01	0.143	0.0	0.276	0.184	0.010	9.821	485	2259	1
3	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.00	0.137	0.0	0.137	0.000	0.000	3.537	40	191	1
4	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.00	0.135	0.0	0.135	0.000	0.000	3.537	40	191	1

Fit a logistic regression

- You can use the sklearn.linear_model.LogisticRegression object
- Create your target vector *y* (last column of data table) and your design matrix *X* (note that you do NOT need to add the bias term in your design matrix, sklearn takes care of it for you)
- Fit your model on the data

```
1 from sklearn.linear_model import LogisticRegression
2 lr = LogisticRegression(solver='liblinear')
3 lr.fit(X,y)
```

How would you evaluate your model?

Popular evaluation metrics – Accuracy

(TP = True positive; FP = False positive; TN = True negative; FN = False negative)

Accuracy: percentage of samples classified correctly (sklearn.metrics.accuracy_score)

$$Acc = \frac{TP + TN}{TP + TN + FP + FN}$$

What is the accuracy of your model?

Based on your accuracy, would you say your model is good or bad?

Popular evaluation metrics – Confusion Matrix

(TP = True positive; FP = False positive; TN = True negative; FN = False negative)

Confusion Matrix: percentage of samples classified correctly (sklearn.metrics.confusion_matrix)

		Predicted	
		Class-0	Class-1
Actual	Class-0	TN	FP
	Class-1	FN	TP

		Predicted	
		Class-0	Class-1
Actual	Class-0	2779	8
	Class-1	38	12

What is the accuracy of positive and negative classes separately?

When calculating accuracy we lose information. The confusion matrix gives us a more thorough understanding of what is going on.

Precision/Recall/F1/ROC

		Predicted	
		Class-0	Class-1
Actual	Class-0	2779 (TN)	8 (FP)
	Class-1	38 (FN)	12 (TP)

F1 score (harmonic mean of pre and rec):

$$F1 = 2 \cdot \frac{pre * rec}{pre + rec}$$

$$Pre = \frac{TP}{TP + FP}$$

PRECISION: out of the emails that are predicted to be spam (class-1), how many are truly spam?

$$Rec = \frac{TP}{TP + FN}$$

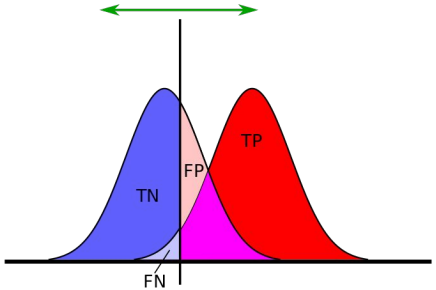
RECALL: out of the emails that are truly spam (class-1) how many are correctly classified by our model?

Precision/Recall/F1/ROC

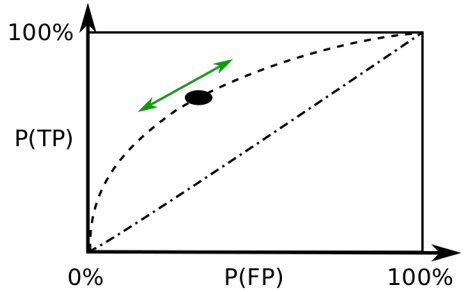
We can tune the decision threshold to maximize precision or recall, depending on the goal of our model.

What would you want to maximize/minimize in our case? For spam classification, would we minimize FP or FN? Why?

Receiver operating characteristic (ROC)



TP	FP
FN	TN



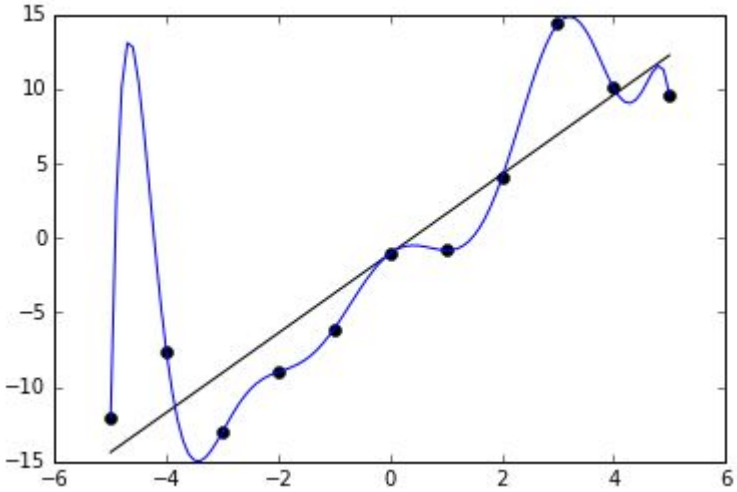
Precision/Recall/F1/ROC

		True condition				
	Total population	Condition positive	Condition negative	Prevalence $= \frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	Accuracy (ACC) $= \frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$	
Predicted condition	Predicted condition positive	**True positive**, Power	**False positive**, Type I error	Positive predictive value (PPV), Precision $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$	False discovery rate (FDR) $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$	
	Predicted condition negative	**False negative**, Type II error	**True negative**	False omission rate (FOR) $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$	Negative predictive value (NPV) $= \frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$	
		True positive rate (TPR), Recall, Sensitivity, probability of detection $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) $= \frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) $= \frac{\text{LR+}}{\text{LR−}}$	F_1 score $= \frac{1}{\frac{1}{\text{Recall}} + \frac{1}{\text{Precision}}}$
		False negative rate (FNR), Miss rate $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) $= \frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) $= \frac{\text{FNR}}{\text{TNR}}$		

Regularization

Bias, variance and model complexity

- Overfitting happens when a model is too complex for the data at hand, i.e. an overfitted model contains more parameters that can be justified by the data
- When a model overfits, we have very good performance on the training data, but poor generalization and predictive power for unseen data
- The opposite of overfitting is underfitting, i.e. the model is not complex enough to describe the data
- We usually say that complex model have variance (more flexibility), while simple models have bias (less flexibility).



Regularization

- Regularization is a way to reduce overfitting (variance)
- It can be “easily” implemented by adding a penalizing term to the loss function:

$$\mathcal{L}(\theta, X) = J(\theta, X) + \lambda R(\theta, X)$$

- The regularization term will constraint the parameters to usually small values (ridge) and sometimes will even drive some of these parameters to zero (lasso)

Ridge regression (L2 regularization)

- The most common regularizer in linear regression is the L2 norm, i.e. the loss function becomes:

$$\mathcal{L}(\theta, X) = \frac{1}{2}||y - X\theta||^2 + \lambda||\theta||^2$$

Exercise: derive the normal equations with the regularization term. How do they change?

(non-regularized normal equations: $\theta = (X^T X)^{-1} X^T y$)

Lasso regularization (L1 regularization)

- Another common regularizing function is obtaining by constraining on the L1 norm of the model parameters, i.e.

$$R(\theta, X) = ||\theta||_{L1} = |\theta_1| + |\theta_2| + |\theta_3| + \ldots$$

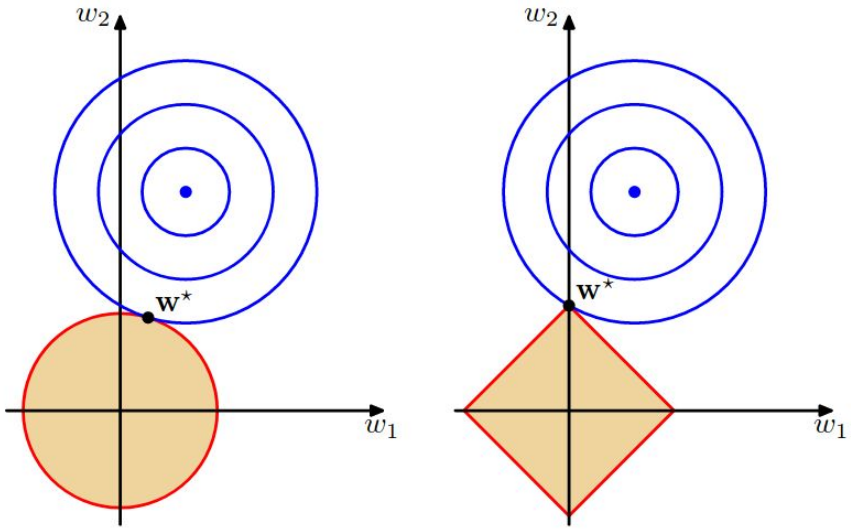
- L1 regularization can be used to do automatic feature selection (some of the parameters are driven to zero, thus practically dropping the contribution of the corresponding predictor to the response variable)

- We can also combine both L1 and L2 regularizers to get the best of the two worlds (in the context of regression, such a model is called elastic network)

L1 vs L2

Figure 3.4 Plot of the contours of the unregularized error function (blue) along with the constraint region (3.30) for the quadratic regularizer $q = 2$ on the left and the lasso regularizer $q = 1$ on the right, in which the optimum value for the parameter vector $\mathbf{w}$ is denoted by $\mathbf{w}^\star$. The lasso gives a sparse solution in which $w_1^\star = 0$.

(source: Pattern Recognition and Machine Learning by C.M. Bishop)

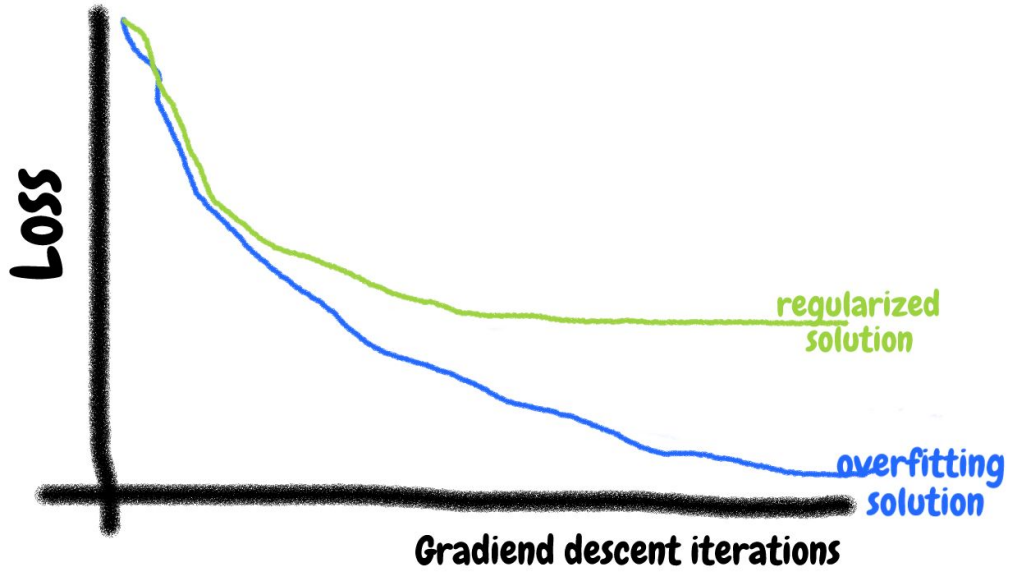


Bonus question:

L1 is less stable than L2, can you guess why?

(Hint: what happens to the gradient?)

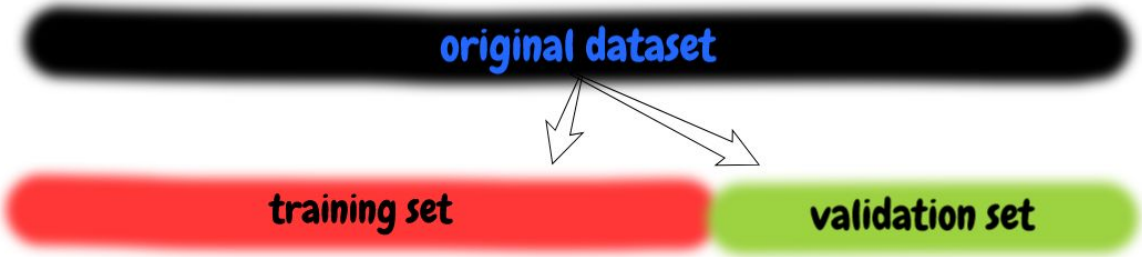
Effect on loss and convergence



Cross-validation

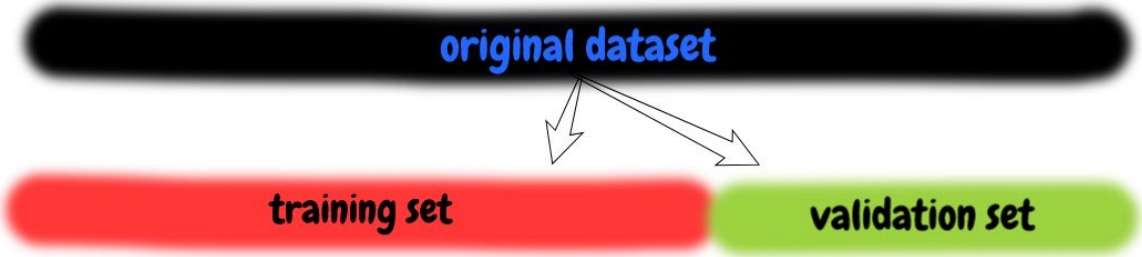
- We can fight overfitting by using regularization
- We hope that a regularized model generalizes better on new data, i.e., if the performance is good on the training data, then it will be as good on new observations
- There’s still a few questions: how do we set the regularization parameter? How do we test the generalization error?
- Cross-validation!

Cross-validation



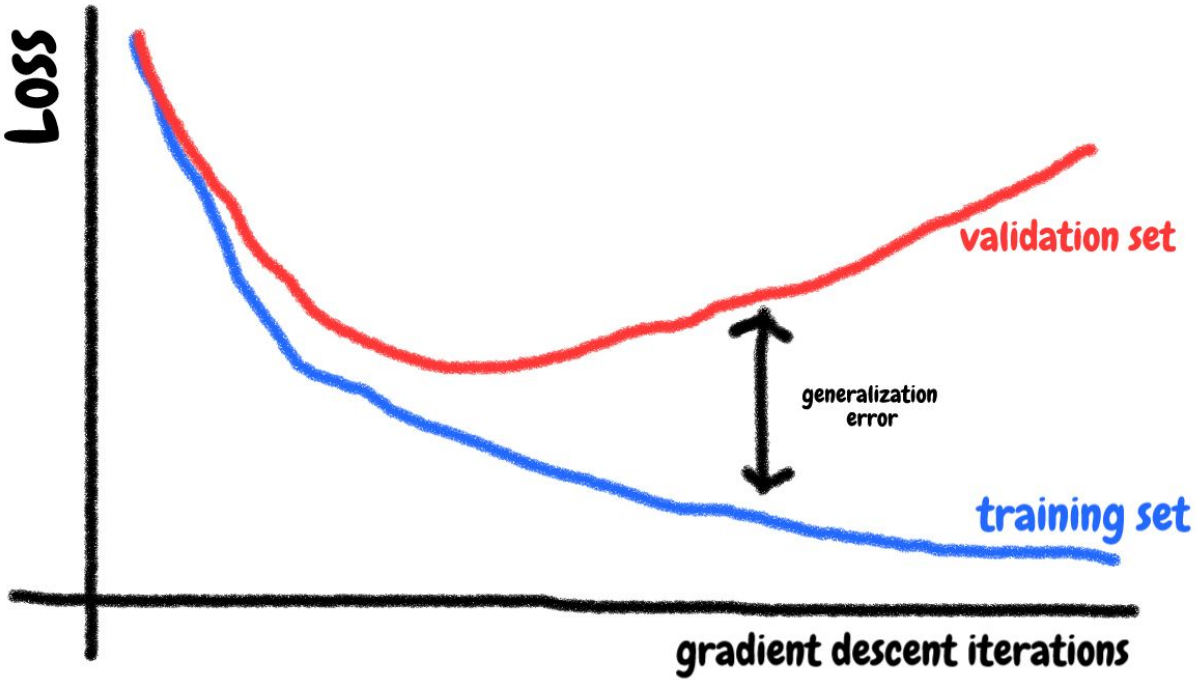
- Split the original data set into a training set and a validation set
- We adjust the model’s parameters on the training set
- We use the validation set as a control set

Cross-validation

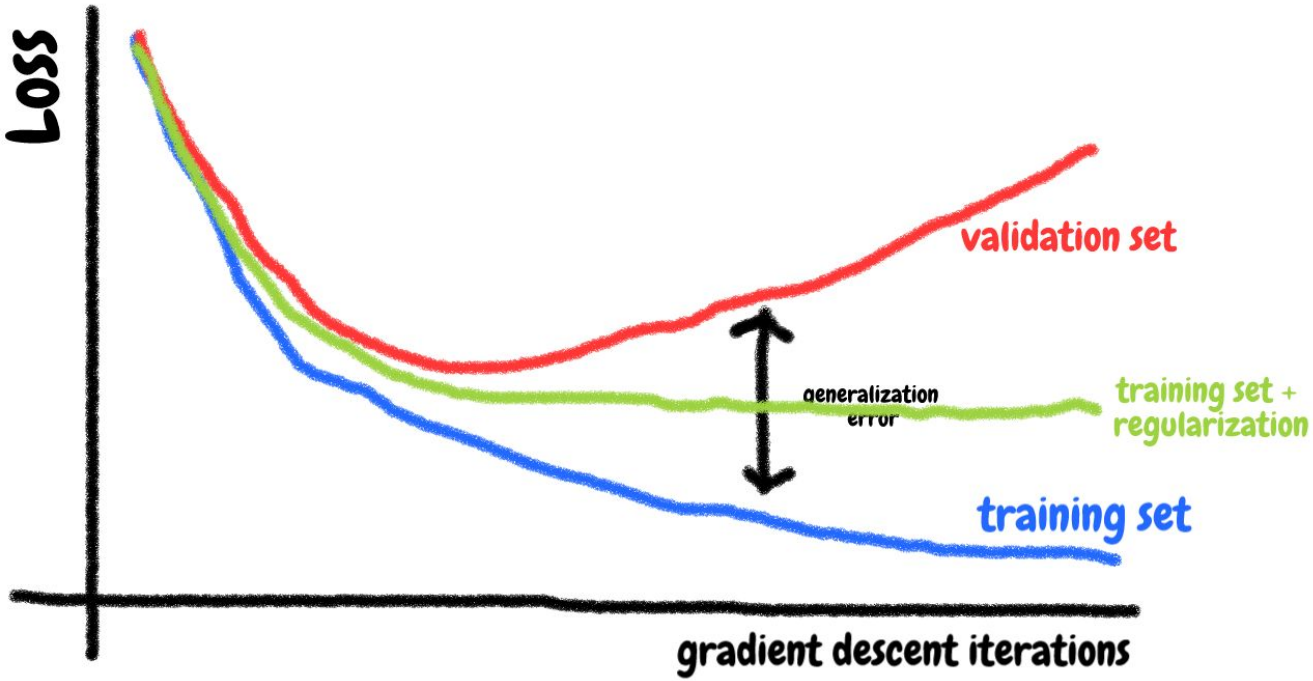


- Split the original data set into a training set and a validation set
- We adjust the model’s parameters on the training set
- We use the validation set as a control set

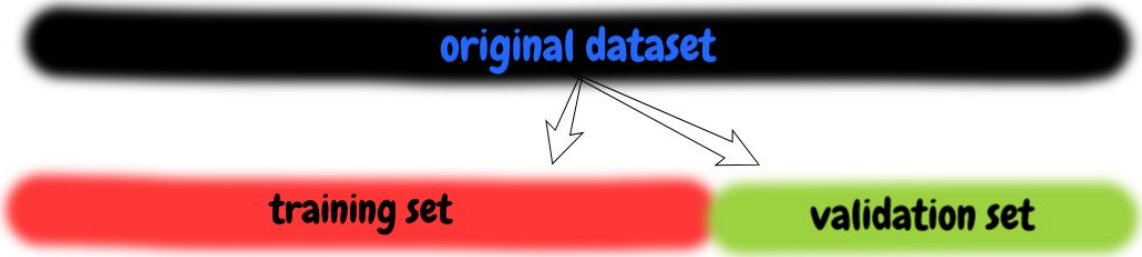
Cross-validation



Cross-validation



Cross-validation drawbacks



- Results may depend on the split (which specific samples we decide to put in the validation set)
- We are training on less data and validating on a small subsample, thus our generalization error estimation could be biased

LOOCV - (leave-one-out cross validation)

- LOOCV still splits the data in training/validation sets
- The validation set consists of a single sample
- The procedure is repeated as many times as the number of samples, leaving out of the training set each observation in turn
- Because of the high variability of the validation set, the validation error will be an unbiased estimator of the generalization error

K-fold CV

- LOOCV is computationally expensive for large data sets and has high variance
- LOOCV is a special case of K-fold cross-validation
- K-fold CV:
  - Split the data in k random partitions of equal size
  - Train the model on k-1 partitions
  - Test the model on the partition which was left out
  - Repeat for all k partitions (thus obtaining k estimates of the generalization error)

Practice exercise

- Download the full spambase dataset
- Implement a LogisticRegression class that minimizes the regularized cost function (with L2 regularization) using gradient descent
- Split your data in training/validation sets
- Train your model and plot the curves for the log-loss on the training/validation sets at each update of the parameters
- Now try to select the best value of the regularization parameter using K-fold CV (e.g. 5 folds) by varying the regularization parameter. For each trial value:
    - Train/validate the model on the K-folds
    - Obtain a score by averaging the scores on the different folds

  The best value of the regularization parameter is the one that has the best score

The cost function in logistic regression is given by:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}[-y^{(i)}log(h_\theta(x^{(i)})) - (1-y^{(i)})log(1-h_\theta(x^{(i)}))]$$

and the gradient of the cost is a vector of the same length as θ where the j^{th} element (for j = 0, 1, ..., n) is defined as follows:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

Note that while this gradient looks identical to the linear regression gradient, the formula is actually different because linear and logistic regression have different definitions of $h_\theta(x)$.