Machine Learning I

Chapter 4 - Evaluating a Model

Prof. Dr. Sandra Eisenreich

November 5/9 2023

Hochschule Landshut

Agenda

I. Data Preprocessing: Collect and clean up data.

II. Choosing/Training Model(s): see below.

III. Evaluating the Model(s): Check the performance on new, unseen data.

... Back to II., if III. is not good enough. If only some hyperparameters are changed, this is called finetuning the model.

II. The choice of ML model depends on the kind of

TASK,

DATA,

COMPLEXITY,

FOCUS,

Validation Scores

evaluate/validate a model = compute a performance measure on a set of unseen data (e.g. test set.) The result is often called validation score.

You can either

- "recycle" the performance measure used for training the model, or
- use additional performance measures (often derived from statistics)

The type of performance measure depends on the ML task. We will cover the two most frequent here, regression and classification.

Regression

Notation

- h = predictor
- $y^{(i)}$ = label for input $x^{(i)}$ (a real value or vector)
- $\hat{y}^{(i)} := h(x^i)$ = prediction for input $x^{(i)}$
- $\text{res}_i := y^{(i)} - \hat{y}^{(i)}$ is called residue (= the prediciton error).

Performance metrics for univariate regression

- MAE = Mean Absolute Error: $L_{\text{MAE}} = \frac{1}{m}\sum_{i=1}^{m} |\text{res}_i| = \frac{1}{m}\sum_{i=1}^{m} \left|y^{(i)} - \hat{y}^{(i)}\right|$
- MSE = Mean Squared Error: $L_{\text{MSE}} = \frac{1}{m}\sum_{i=1}^{m} \text{res}_i^2 = \frac{1}{m}\sum_{i=1}^{m} \left(y^{(i)} - \hat{y}^{(i)}\right)^2$
- RMSE = Root Mean Squared Error is the squareroot of MSE.
- R^2-score, see next slide: a number < 1 that describes how much of the labels' variance can be explained from the features' using the model.

With Scikit-Learn: (`y_true`=labels, `y_pred`=predicitons)

- MAE: `metrics.mean_absolute_error(y_true, y_pred)`
- MSE: `metrics.mean_squared_error(y_true, y_pred)`
- R^2: `metrics.r2_score(y_true, y_pred)`

R^2-score

- Denote the mean of all labels by $\bar{y} = \frac{1}{m} \sum_{i=1}^{m} y^{(i)}$
- residual sum of squares: $\mathrm{SS}_{\mathrm{res}} = \sum_{i=1}^{m} (y^{(i)} - \hat{y}^{(i)})^2 = m \cdot \mathrm{MSE}$, and the
- total sum of squares: $\mathrm{SS}_{\mathrm{tot}} = \sum_{i=1}^{m} (y^{(i)} - \bar{y})^2 = m \cdot \mathrm{Var}(\{y^{(i)}, i = 1, \ldots, m\})$.

Then the R^2-score is defined as

$$R^2 = 1 - \frac{\mathrm{SS}_{\mathrm{res}}}{\mathrm{SS}_{\mathrm{tot}}}$$

The R^2-score describes how much better the predictor is than the "dumb" one which predicts the mean value $\bar{y}$ for every input.

- $R^2 = 1 \Rightarrow$ the predictor fits the data perfectly
- $R^2 < 0 \Rightarrow$ the predictor fits the data even worse than the one equivalent to a horizontal hyperplane at a height equal to the mean of the observed data

Performance metrics for multivariate regression

- MAE:
$$L_{\text{MAE}} = \frac{1}{m}\sum_{i=1}^{m}\|\text{res}_i|\|_1 = \frac{1}{m}\sum_{i=1}^{m}\|y^{(i)} - \hat{y}^{(i)}\|_1$$
- MSE:
$$L_{\text{MSE}} = \frac{1}{m}\sum_{i=1}^{m}\|\text{res}_i\|_2^2 = \frac{1}{m}\sum_{i=1}^{m}\|y^{(i)} - \hat{y}^{(i)}\|_2^2.$$
- RMSE is the squareroot of MSE.
- R^2-score: same formula, where
$$\begin{aligned} \text{SS}_{\text{res}} &= \sum_{i=1}^{m}\|y^{(i)} - \hat{y}^{(i)}\|_2^2 = m \cdot \text{MSE} \\ \text{SS}_{\text{tot}} &= \sum_{i=1}^{m}\|y^{(i)} - \bar{y}\|_2^2 \end{aligned}$$

Example

Suppose the predictor is given by $h(x) = 2 \cdot x + 1$ and the data are

$$(1, 2), (2, 6), (3, 10)$$

.

- Compute MAE
- Compute MSE and RMSE
- Compute the R^2-score

Binary Classification

Notation

- h = predictor
- m instances
- $y^{(i)} \in \{0, 1\}$ = label for input $x^{(i)}$
- $\hat{y}^{(i)}$ = predicted class for input $x^{(i)} \in \{0, 1\}$
- $h(x^{(i)})$ = the output of the predictor. This is
  - in case of soft classification: the probability of the positive class

$$h(x^{(i)}) = P(\hat{y}^{(i)} = 1)$$

  - in case of hard classification: the predicted class $\in \{0, 1\}$

Accuracy

The most intuitive performance measure for classification is accuracy, i.e. the percentage of instance the model gets right:

$$\text{Accuracy} = \frac{\text{correct classifications}}{\text{all classifications}}$$

However, accuracy is not always a good measure, e.g. if there is a class imbalance in the data.

Example: if we want to identify if a MNIST image is a 5 or not, a dumb predictor which says that every single of the MNIST pictures is not 5 would still have high accuracy (since 90% of the MNIST images are not a 5).

Confusion Matrix

In binary classification, we always choose the positive class to be the minority class.

For binary classification we denote by

- TP = no. of true positives ($y = 1, \hat{y} = 1$),
- FP = no. of false positives ($y = 0, \hat{y} = 1$),
- TN = no. of true negatives ($y = 0, \hat{y} = 0$),
- FN = no. of false negatives ($y = 1, \hat{y} = 0$),

Then the confusion matrix is given by

$$\begin{pmatrix} TN & FP \\ FN & TP \end{pmatrix}$$

True class $0/1$ = rows; predicted class $0/1$ = columns

Binary Classification: Precision and Recall

With this notation, one can define performance measures for binary classification:

- Precision: if the prediction is positive, how likely is it correct?

$$\text{precision} = \frac{TP}{TP + FP},$$

- Recall (=true positive rate (TPR) = sensitivity): how likely will a true positive be predicted as positive?

$$\text{recall} = \frac{TP}{TP + FN},$$

Precision and Recall: Examples

Rule: If you don't want false positives: high precision!
If you don't want to miss a positive: high recall!

Questions: Precision or Recall?

- If your Corona test is positive, how likely are you actually ill? $\rightarrow$?
- If you are ill, how likely will the test be positive? $\rightarrow$?
- Healthcare: you dont want your model to predict an sick patient (1) to be a healthy one (0). $\rightarrow$ high ?
- In a zombie apocalypse, you try to accept as many as healthy people (1) you can into your safe zone, but you really don't want to mistakenly pass a zombie (0) into the safe zone. $\rightarrow$ high ?
- A long time ago in a galaxy far, far away... ML models were used to predict if someone is guilty of a crime. You'd rather let one guilty person (1) go free than imprison an innocent one (0). $\rightarrow$ high ?

Precision/recall tradeoff

How "cautious" should the model be in its predictions of a positive?

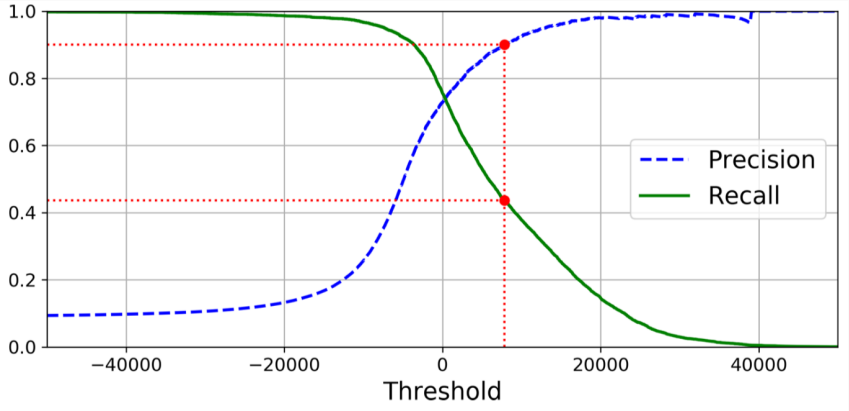
- Very cautious → high precision but some positives might be missed
- not very cautious → high recall but there might be false positives

→ precision-recall trade-off.

Question: How can you control how cautions a model is?

Answer:?

Precision and Recall for different thresholds



(Here, the threshold is not given as a probability but as an absolute value for some input.)

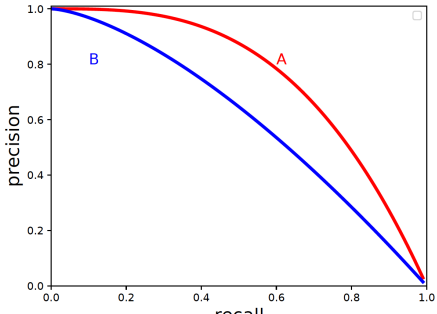
This diagram can be summarized in one performance measure called average precision.

Precision-Recall-Curve

Let $\tau \in (0, 1)$ be the threshold in a binary classification, i.e.

$$\hat{y} = \begin{cases} 1 & \text{for } h(x) \geq \tau \\ 0 & \text{for } h(x) < \tau \end{cases}$$

For varying τ from 0 to 1, as τ grows, precision grows and recall drops. The resulting curve of (precision, recall)-points for varying τ is called precision-recall-curve. **Rule:** Models with higher curve are better. The area under the curve is a performance measure for a model.



Here, model A obviously performs better than model B, because precision is higher for the same recall value. Therefore, one can use the Precision-Recall Curve as a performance measure.

Average Precision

Average Precision is the area under the Precision-Recall-Curve. To approximate it: Divide the range of thresholds into a grid of discrete values $\{1 > \tau_1 > \tau_2 > \ldots > \tau_n > 0\}$. Compute the precisions $\{P_1, \ldots, P_n\}$ and recalls $\{R_0 = 0, R_1, R_2, \ldots, R_n\}$ for these thresholds. Then the average precision is:

$$AP := \sum_{i=1}^{n} (R_i - R_{i-1}) \cdot \max(P_i, P_{i-1})$$

F_1-Score

Alternatively, you can combine precision and recall of your model in one single metric.

Question: Could you take the mean of precision and recall?

Suppose we take $\tau = 0.999$ (very bad idea) and have $TP = 1, FP = 0, FN = 99$. $\Rightarrow$ precision $=?$, recall $= ?$, but mean(precision, recall)$\simeq ?$.

The F_1-score$\in (0, 1)$ is the harmonic mean of precision and recall:

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{TP}{TP + \frac{FN+FP}{2}}$$

The F_1-score favours classifiers that have similar precision and recall.

Example above: $F1 = ?$.

ROC Curve

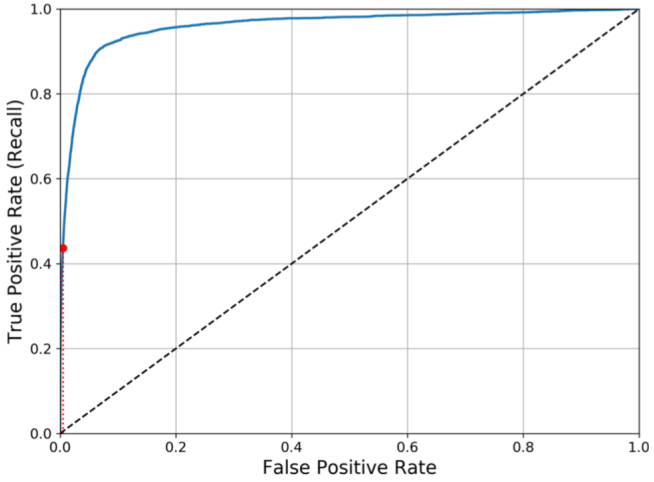
There is yet another curve that is used as a visual performance measure for binary classification models, which plots recall = true positive rate = $TPR = \frac{TP}{TP+FN}$ ="sensitivity" against false positive rate:

- False positive rate (="specificity") is the ratio of false positives among the ground-truth negatives, i.e.

$$FPR = \frac{FP}{FP + TN}$$

- The ROC-curve (receiver operating characteristic curve) is the plot of TPR vs. FPS for different thresholds (there is a trade-off like for precision and recall.) **Rule:** A bigger area under the curve is better.
- One defines the ROC-AUC-score (ROC-area-under-the-curve-score) as the area under the ROC curve. A perfect classifier would have ROC-AUC=1.

ROC Curve - Optimal Threshold



The classification threshold τ that returns the upper-left corner of the curve is optimal for a balanced model.

Binary Cross-Entropy: loss function for training

The above are used to interpret binary classification models. For training, you use NLL of the Bernoulli distribution. One can show that this is given by the binary cross-entropy loss.

Let $p^{(i)} := h(x^{(i)})$ denote the predicted probability of $x^{(i)}$ being in the positive class 1. Binary Cross-Entropy Loss = Log-Loss is given by

$$L_{\mathbb{H}} = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)} \cdot \log p^{(i)} + (1-y^{(i)}) \cdot \log\left(1-p^{(i)}\right)\right)$$

Note: Binary Cross-Entropy is the loss function used to train a binary classification model. It is derived from cross-entropy, see later.

Examples

- For a binary classification problem we have the following confusion matrix:

$$\begin{pmatrix} 10.000 & 1.000 \\ 3.000 & 9.000 \end{pmatrix}$$

  Compute precision, recall, F1-score.

- Suppose we have the following labels and prediction results of a binary classifier:

Label	Predicted prob. vector
1	(0.4, 0.6)
1	(0.1, 0.9)
0	(0.6, 0.4)
0	(0.4, 0.6)

  - Compute the accuracy of the model on these data
  - Compute the binary cross-entropy loss of the model on these data
  - Compute precision and recall for the thresholds $\tau_3 = 0.25, \tau_2 = 0.5, \tau_1 = 0.75$
  - Compute a crude approximate of the Average Precision with this grid of thresholds $1, 0.75, 0.5, 0.25, 0$

Performance Metrics for binary Classification with Scikit-Learn

- Confusion Matrix: `metrics.confusion_matrix(y_true, y_pred)`.
- Precision: `metrics.precision_score(y_true, y_pred)`
- Recall: `metrics.recall_score(y_true, y_pred)`
- Average Precision: `metrics.average_precision_score`
- Precision-recall curve: `metrics.precision_recall_curve(y_true)`
- F1-Score: `metrics.f1_score(y_true, y_pred)`
- ROC-AUC-score: compute `metrics.roc_auc_score(y_true, y_score)`, where `y_score` = predicted probability.
- ROC-curve: `metrics.roc_curve(y_true, y_score)` returns a tuple (fpr, tpr, thresholds) which you can plot with matplotlib, for example.

Multiclass classification

Notation

- h = predictor
- $y^{(i)}$ = label for input $x^{(i)}$
- $\hat{y}^{(i)}$ = predicted class for input $x^{(i)}$
- $h(x^{(i)})$ = the output of the predictor. This is
  - in case of soft classification: a C-dimensional probability vector with $C > 2$ classes

$$h(x^{(i)}) = \left(h_1(x^{(i)}), h_2(x^{(i)}), \ldots, h_C(x^{(i)})\right)$$

  - in case of hard classification: the predicted class

$$h(x^{(i)}) = \hat{y}^{(i)}.$$

Entropy, Cross-Entropy

- If p is the probability distribution of a discrete random variable X, its entropy is

$$\mathbb{H}(p) = -\sum_{x \in \Omega} p(x) \log p(x)$$

- If p, q are distributions of dicsrete random variables on Ω, their cross-entropy is

$$\mathbb{H}(p; q) = -\sum_{x \in \Omega} p(x) \log q(x)$$

Note: Information-theoretical interpretation: Cross-entropy describes the average number of bits needed to encode data coming from a source with distribution p when we use model q.

Cross-Entropy Loss = Log Loss

- h: predictor for a soft classification problem with C classes, i.e.
- $h(x^{(i)})$: vector of class probabilites for input $x^{(i)}$
- $\mathbf{y}^{(i)}$: the one-hot encoding of the true class $y^{(i)}$.

The cross-entropy loss or log-loss is given by

$$L_{\mathbb{H}} = -\frac{1}{m}\sum_{i=1}^{m}\sum_{c=1}^{C}\mathbf{y}_c^{(i)} \cdot \log h_c(x^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m}\log h_{y^{(i)}}(x^{(i)}).$$

It is the NLL for the multinomial distribution of a classification problem.

This is the loss function you optimize in the classification task (like MSE in the regression task).

Examples

- Suppose you have a classification task with 4 different classes 0,1,2,3 and the following labels/predicted probability vectors:

Label	Predicted prob. vector
0	(0.5, 0.2, 0,2, 0.1)
0	(0.4, 0.3, 0.2, 0.1)
1	(0.0, 0.7, 0.2, 0.1)
2	(0.1, 0.1, 0.8, 0.0)
4	(0.1, 0.2, 0.1, 0.6)

  Question: What is the cross-entropy loss?

- **Task:** Show that binary cross-entropy loss as defined before is indeed just a special case of the general cross-entropy loss above.

Other measures for multiclass and multilabel classification

Performance scores for multiclass classification apart from log-loss:

- Confusion matrix: How?
-
- Does accuracy work?

For multilabel classification, one can use the same performance measures as for simple classification: How?

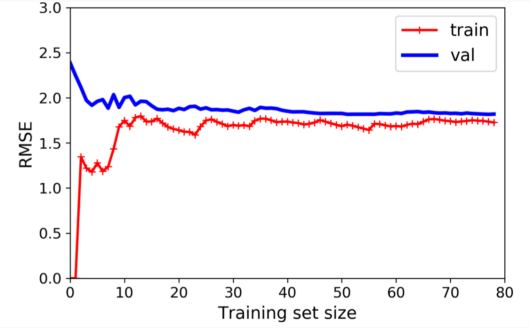
Performance Measures for classification with Scikit-Learn

- Cross-entropy: `metrics.log_loss(y_true, y_pred)`
- Accuracy: `metrics.accuracy_score(y_true, y_pred)`
- Confusion matrix: `metrics.confusion_matrix(y_true, y_pred)`.

Monitoring Performance

Monitoring Performance during Training: Learning Curves

A learning curve is a plot of the model's performance on the training set and the test/validation set as a function of the training set size. To generate the plots, train the model on growing subsets of the training set and evaluate after each subset on training and test set.

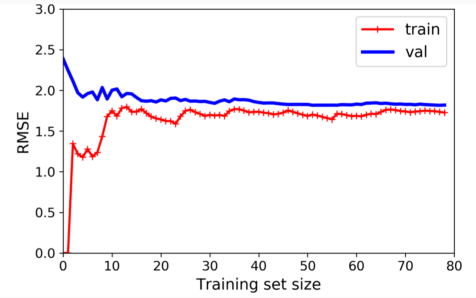


Quelle: Learning curves for underfitting linear model.

Learning Curves for under- and overfitting models

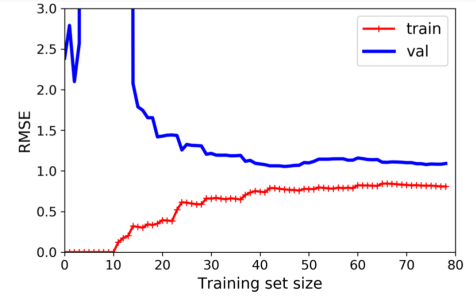
Learning curve for underfitting model (linear regression):

no learning over time, train and validation error are constant. more training data do not help.
→ more complex model or better features



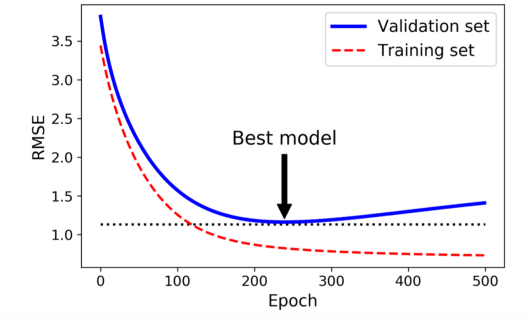
Learning curve for overfitting model (10th degree polynomial regression):

Constant gap between the train and validation curves. Bigger gap means more overfitting.
→ more training data, regularize



Another regularization method: Early stopping

If you track the validation error = performance measure on the test set during training and stop the training as soon as the validation error stops decreasing for a while (this while is a hyperparameter called "patience") even if training is not complete, this is called early stopping. It works as a regularization method.



Evaluating Performance over Time - Resilience

Resilience is a model's capability to work over a long period of time even under adversarial conditions.

There are two reasons for model decay = a model's decrease in performance over time:

- concept drift (/shift): the relationship betwen input and target changes over time (/suddenly) (i.e. the prediction function h doesn't work any more)
- covariate drift (/shift): the data distribution changes slowly (/suddenly)

How to test resilience when training a model: Small discrepancy between the errors for different test sets shows the model should be resilient wrt covariate shifts.

Examples of shifts and drifts

- **concept shift:** a model was trained to predict the number of train passengers. Then Covid-19 happened.
- **concept drift:** a model is trained to predict customer behaviour at one point in time. However, new products and technological advances change the customer's behaviour over time, so the old prediction function is not accurate any more.
- **coviariate shift**: a facial recognition model trained on the faces of people aged 20 to 30 is used on general population; a speech recognition model is trained on Irish people and used in New Zealand...
- **covariate drift:** a model was trained to predict the likelihood of a customer purchasing a product based on their age and income. If the distribution of ages and incomes of the customers changes significantly over time, the model may no longer be able to predict the likelihood of a purchase accurately.

Model Evaluation Process and Hyperparameter Tuning

Hyperparameters

Remember the regularization constant α (big $\alpha \rightarrow$ model might underfit; small $\alpha \rightarrow$ model might overfit). This is a so-called hyperparameter:

> A **hyperparameter** is a parameter which is not optimized during training (i.e. not part of θ), but whose value is **chosen** (e.g. to control the learning/optimization process, as part of the loss function, etc....).

How do you choose such hyperparameters?

Validation set - motivation

Idea: Try different values for the hyperparameter, i.e.:

(i) train the model several times with different values for the hyperparameter $\rightarrow$ different predictors

(ii) validate them and take the hyperparameter leading to the best performance

(iii) evaluate the resulting model on the test set to find out how it performs.

Question: should you validate (ii) on the test set?

Answer: ?

Testing and validation

- Training set: data the predictor is trained/built on
- Test set: data for testing the final model
- Validation set: data for validating intermediate models

Rule of thumb: train-validation-test split is 60-80% training data, 10-20% validation data, and 10-20% test data.

Drawback: ?

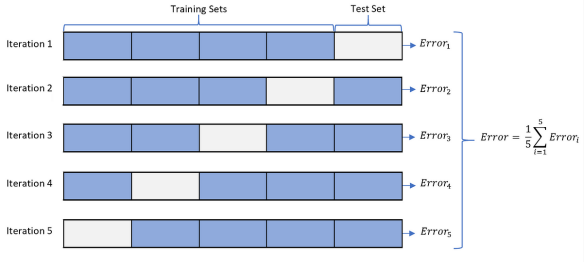
Better method: Cross-validation.

Cross-Validation

Instead of choosing three different datasets, one could only do a train-test-split (80-20) and choose hyperparameters with Cross-Validation on the training set:

K-fold Cross-Validation (CV)

- Divide the **training set** into K equally sized datasets D_i.
- For $i = 1, \ldots, K$, train the model on all data but D_k, and evaluate it on D_k. Denote this loss by L_k.
- Take the mean of all losses as the cross-validation error: $L^{cv} = \frac{1}{K} \sum_{i=k}^{K} L_k$



Drawback: training time is multiplied by K.

Finding the right hyperparameters

So to find the best hyperparameters, there are several methods:

- Grid Search with Cross-Validation: Just define a grid of hyperparameters you want to try out and train a model for each grid point (i.e. combination of hyperparameters). Then decide which one is the best using the performance measure on the validation set, or by performing Cross-Validation.
- Empirical Bayes: probabilistic method. Beyond the scope of this lecture.

Finding the right ML model

model, hyperparameters, training... training/test/validation set...

HOW?