8 Metrics Hyperparameter tuning

Date	@November 12, 2024
Topic	Theory

Performance metrics

Loss vs metric

Losses are made for machines to learn in order to minimize ERM and finding the parameters (**w**) of a model.

Metrics are for humans and are interpretable measures of quality. They are also used sometimes to choose the hyper-parameters of a model.

Parameters are automatically optimized while hyper-parameters are not, they should be checked in order to choose the one which is better for us.

Example:

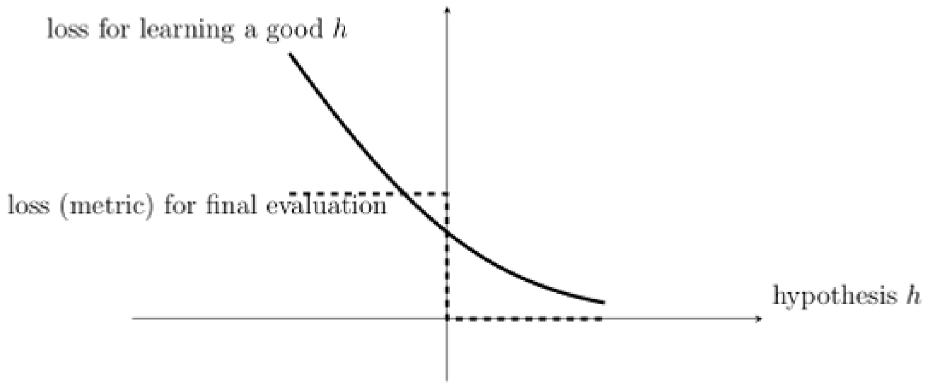


Figure 2.13: Two different loss functions for a given data point and varying hypothesis h. One of these loss functions (solid curve) is used to learn a good hypothesis by minimizing the loss. The other loss function (dashed curve) is used to evaluate the performance of the learnt hypothesis. The loss function used for this final performance evaluation is sometimes referred to as a metric.

Evaluation of supervised ML

There are several additional metrics (which are not used for ERM):

- efficiency
    - how much it takes to learn/perform ERM
    - how much it takes to predict the value of the hypothesis

- scalability: how the algorithm will perform in response to set size or attribute number changes
- robustness: how performances change when adding noise or removing data
- interpretability: explain why an algorithm is giving a certain output

Evaluation of regression techniques

For regression techniques usually losses are quite interpretable such as mean squared error or mean absolute error (which will tell you in a regression problem what is the average error that you will have on a random sample), and they allow to evaluate the quality of the prediction. So these kinds of losses can be metrics too.

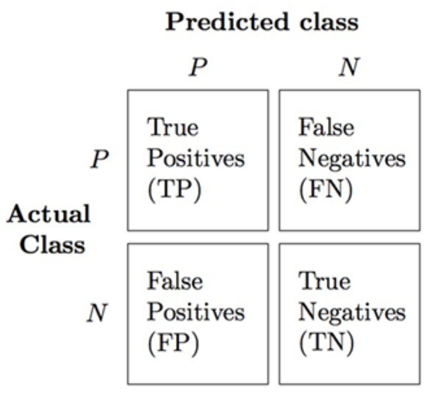
Evaluation of classification techniques

For classification techniques losses can be very different from metrics. These are confusion matrix, accuracy, precision, recall, F-measure (which can be computed per class or on average) and ROC curve.

Confusion matrix

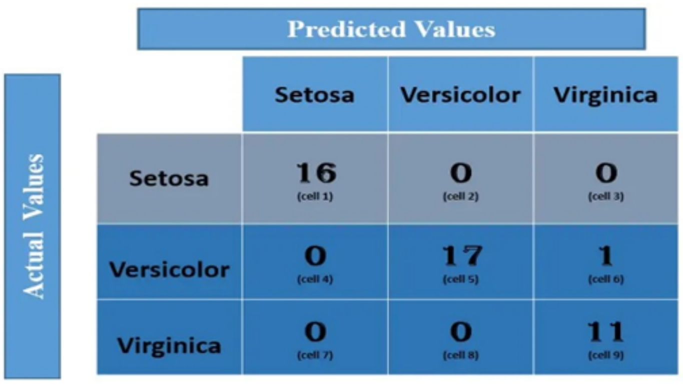
Given our n samples we are gonna predict the output $x \rightarrow h(x) = y$^ and comparing it with the true value.

For binary classifiers (so that y can be positive or negative or -1,+1, or 0,1, or true, false etc), we have to classes and we can draw a matrix 2×2 where the rows represent the actual class while the columns represent the predicted class.



		Predicted class	
		P	N
Actual Class	P	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

For multi-class classifiers



		Predicted Values		
		Setosa	Versicolor	Virginica
Actual Values	Setosa	16 (cell 1)	0 (cell 2)	0 (cell 3)
	Versicolor	0 (cell 4)	17 (cell 5)	1 (cell 6)
	Virginica	0 (cell 7)	0 (cell 8)	11 (cell 9)

In this way you can evaluate the quality of your prediction.

Accuracy

Accuracy is the most widely-used metric for model evaluation

$$\text{Accuracy} = \frac{\text{Number of correctly classified objects}}{\text{Number of classified objects}}$$

It is a number between 0 and 1.

For binary classifiers we have

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Accuracy is related to 0/1 loss, because when you have infinite samples, the accuracy is the probability of correctly classified objects, while the 0/1 loss we said to be the sum of the 0/1 losses, which is more or less the probability of miss-classified objects. So we can write the accuracy as $1 - ERM$(with 0/1 loss) and the 0/1 loss as $1 - accuracy$.

$$\text{Accuracy} \approx p(y = \widehat{y})$$

$$(1/m) \sum_{i=1}^{m} L\left((\mathbf{x}^{(i)}, y^{(i)}), h\right) \approx p(y \neq \hat{y})$$

$$\text{Accuracy} = 1 - (1/m) \sum_{i=1}^{m} L\left((\mathbf{x}^{(i)}, y^{(i)}), h\right).$$

Sometimes it can be misleading to use just the accuracy because we cannot really say if 0.99 is a good value or not. For example if we have a binary problem with classes P and N with very different cardinality (P = 9900, N = 100), and if we have a model with h(x) = P for each x, this hypothesis will have an accuracy of 99% because we'll have

9900/10000 = 99%. So even if the accuracy would be 0.99, it does not mean good performances.

Class specific measures

If the distribution of the samples over the labels is unbalanced then the accuracy might be misleading. Moreover, in some cases we have different classes and not all the classes are equally important for us, or making a mistake in one class is not as relevant as making it in another class.

For this reason we introduce some class specific measures to evaluate separately for each class. We introduce the precision and the recall

$$\text{Precision(c)} \quad p = \frac{\text{Number of objects correctly assigned to c}}{\text{Number of objects assigned to c}}$$

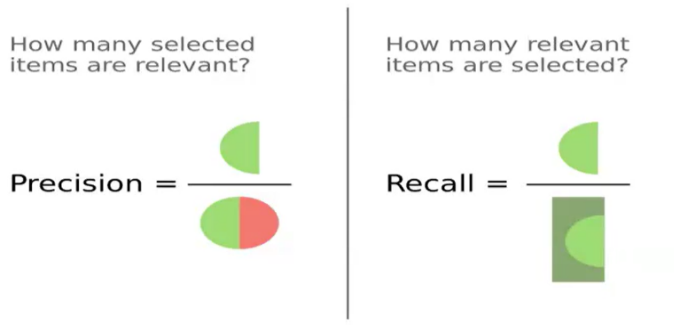
$$\text{Recall(c)} \quad r = \frac{\text{Number of objects correctly assigned to c}}{\text{Number of objects belonging to c}}$$

Since these are class specific measures, we will have distinct values of precision and the recall for the negative and the positive class.

For binary classifier, for the positive class we will have

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

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


How many selected items are relevant?
Precision =
How many relevant items are selected?
Recall =

We often average precision and recall to have a single metric for each class. One averaging function is known as F-measure or F1-score and it is an harmonic mean of the precision and recall.

F-measure(c) F = $\frac{2rp}{r+p}$

We use the harmonic mean rather than the arithmetic mean because the first is very restrict because if one of the two metrics is 0 the whole mean will be 0 meaning that our class is not really well found.

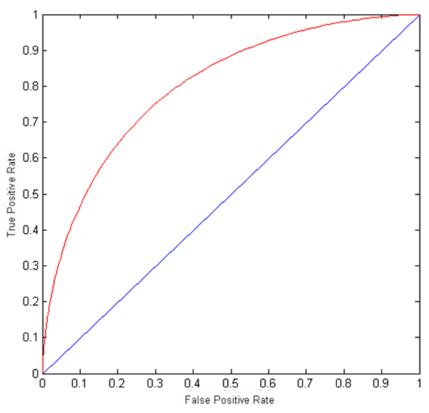
For binary classifier, for the positive class, we have

F = $\frac{2TP}{2TP+FP+FN}$

If we have multi-class classifiers, we need to use the standard definitions.

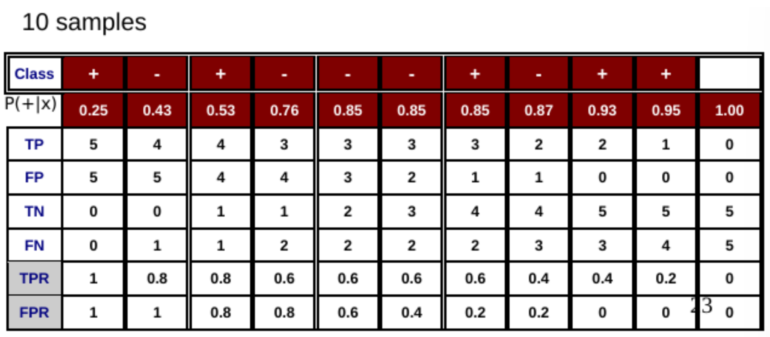
ROC curve

The ROC curve is Receiver Operating Characteristic and is developed for analyzing noisy signals and finding the trade-off between positive hits and false alarms. The ROC curve shows the TPR (true positive rate same as recall) on the y-axis and the FPR (false positive rate = $\frac{FP}{FP+TN}$) on the x-axis.

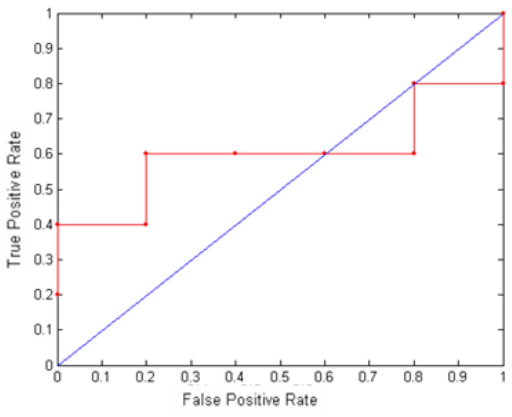


ROC curve is useful when you have a model that has different trades-off.

To build ROC curve you need to use a classifier that produces posterior probability $P(+|x)$ for each x sample, then you have to order the samples according to $P(+|x)$, apply the threshold at each unique value of $P(+|x)$ and compute TPR and FPR at each threshold.


10 samples

Class	+	-	+	-	-	-	+	-	+	+	
P(+|x)	0.25	0.43	0.53	0.76	0.85	0.85	0.85	0.87	0.93	0.95	1.00
TP	5	4	4	3	3	3	3	2	2	1	0
FP	5	5	4	4	3	2	1	1	0	0	0
TN	0	0	1	1	2	3	4	4	5	5	5
FN	0	1	1	2	2	2	2	3	3	4	5
TPR	1	0.8	0.8	0.6	0.6	0.6	0.6	0.4	0.4	0.2	0
FPR	1	1	0.8	0.8	0.6	0.4	0.2	0.2	0	0	0



Different metric for training and validation

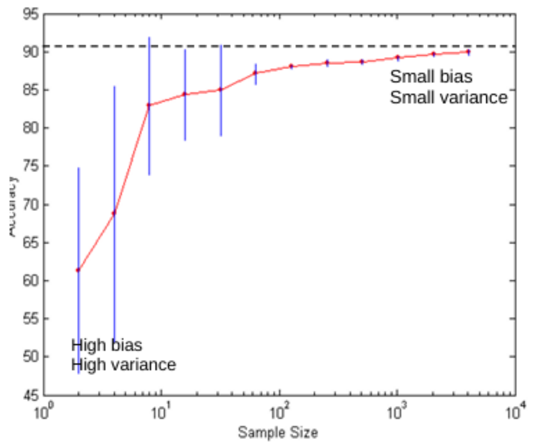
We can use different metrics for training and validation so we can perform the comparison of different ML methods.

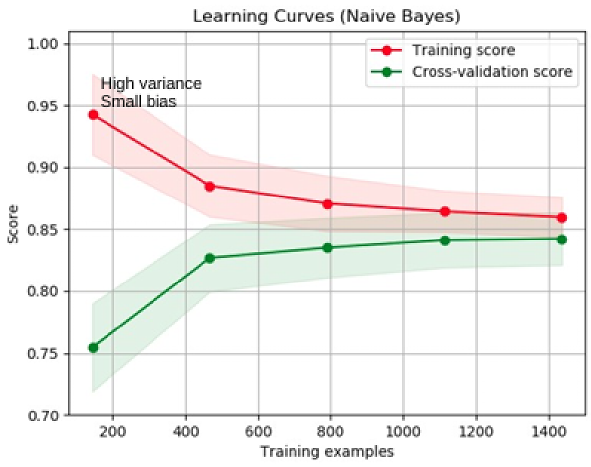
For example logistic regression uses log loss to learn hypothesis $h_1(x)$ while SVM uses hinge loss to learn hypothesis $h_2(x)$. Thus, we can compare $h_1(x)$ and $h_2(x)$ by their average 0/1 loss (accuracy) on validation set.

```python
# split dataset into training and validation set
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3, random_state=42)

# learn hypothesis h1(x) by minimizing its average logistic loss on the training set
logreg= LogisticRegression(random_state=0).fit(X_train, y_train)
# compute validation error of learnt hypothesis h1(x) using average 0/1 loss (accuracy)
val_error_logreg = logreg.score(X_val, y_val)

svmclf = svm.SVC()
#learn hypothesis h2(x) by minimizing its average hinge loss on the training set
svmclf.fit(X_train, y_train)
# compute validation error of learnt hypothesis h2(x) using average 0/1 loss (accuracy)
val_error_svm = svmclf.score(X_val, y_val)

print("accuracy of logistic regression on validation set:",val_error_logreg)
print("accuracy of logistic regression on validation set:",val_error_svm)
```

Learning curve for amount of data

We can use metrics to do a learning curve which shows how a metric (or loss) changes with varying training sample size. It is a tool to find out how much we benefit from adding more training data and it requires a sampling scheduling for creating learning curve.



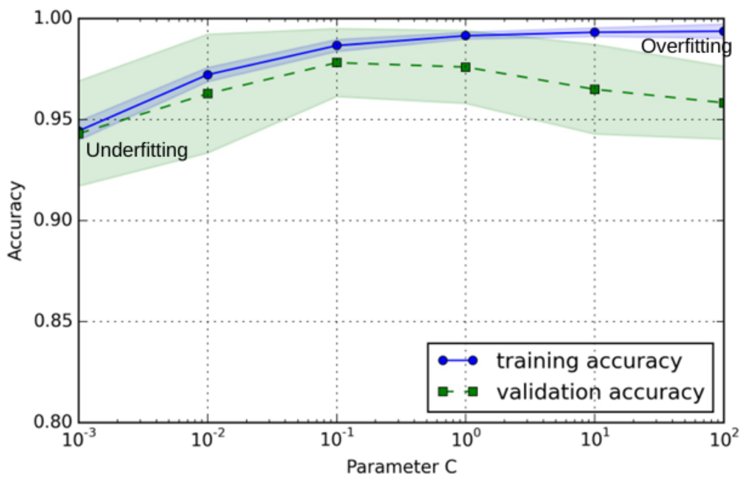


Hyper-parameter tuning

Hyper-parameters are parameters that are not directly learnt within estimators and the ones of the model define the model space H. However, these can still be parameters that can characterize your model. In scikit-learn they are passed as arguments to the constructor of the estimator classes. We have to tune the hyper-parameters in order to compute the best validation score.

Validation curve

The validation curve shows the influence of a single hyper-parameter on the training score and the validation score. The score can be a metric or a loss. Thanks to the validation curve we can find out whether the estimator is underfitting, if the training score and the validation score are both low, or overfitting, if the training score is high and the validation score is low.



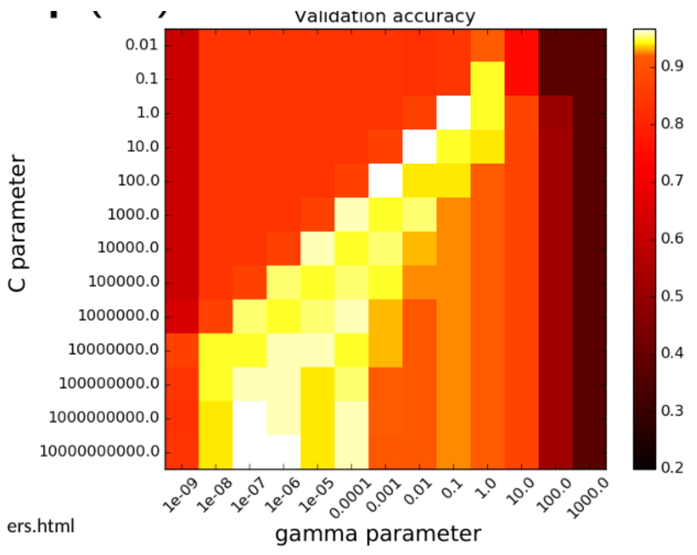
For each of the model spaces $H^{(i)}$ (on the x axis) we learn an hypothesis $h^{(i)}$. After you have the plot, you can choose the best parameters (the one associated to $H^{(2)} = 10^{-1}$ in this case), choosing the one which is the nearest to the training score (in blue) since it is more accurate. Note that the solid line is for training and the dashed one is for validation.

Grid search

Alternatively, you can try to optimize more hyper-parameters at the same time thanks to a grid of hyper-parameter values.

You compute all the possible combinations of hyper-parameters value, for each of them you define the model space, compute the ERM and find h.

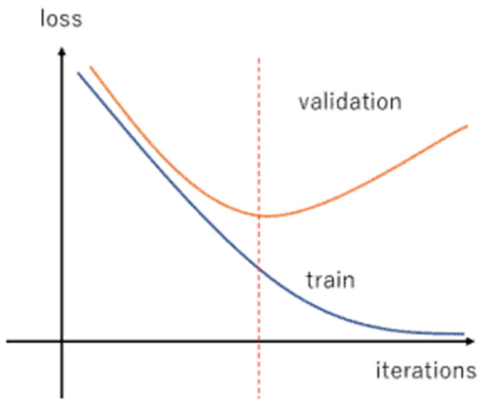
This can be done for many hyper-parameters (2 ore more). In case of 2 hyper-parameters, validation score can be represented as a heatmap:



Training/learning curve for number of iterations

Many algorithms/methods to solve ERM are based on iterative improvements of the empirical risk. At each iterations, a different hypothesis is proposed. These iterations are often called epochs and can be seen as an hyper-parameter of the algorithm/method and not of the model H itself. The score can be a metric or a loss.

Another hyper-parameter of the algorithm is the learning rate.



ML process summary

- Load/generate datasets
- Choose samples, features and labels
- Split the dataset in training, validation (or later cv) and testing
- Pre-processing techniques
- Define model and loss
- Perform ERM with an algorithm (learn)
- Evaluate the obtained performance (metrics)
- Tune the hyper-parameters (and check learning/training curves)
- Test the final hypothesis on test data
- Use it in the wild for inference