Introduction to Machine Learning

Lecture 5: Model Selection and Validation

Alexis Zubiolo
alexis.zubiolo@gmail.com

Data Science Team Lead @ Adcash

December 1, 2016

Shameless advertisement: There will be a more advanced course starting in January 2017!

More info:

http://itstep.bg/news-bg/kurs-machine-learning-from-scratch/

Introduction

Model evaluation is a key component of a machine learning pipeline.

Introduction

Model evaluation is a key component of a machine learning pipeline.

It makes it possible to chose a set of hyper-parameters so that

- The model is accurate enough
- The model generalizes well (*i.e.* does not over-fit)

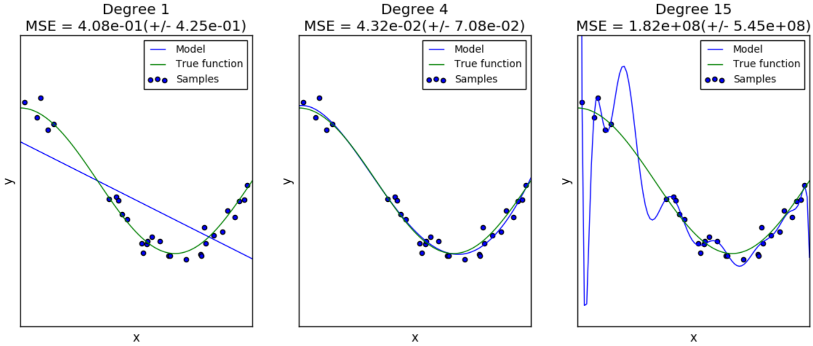
Introduction

Model evaluation is a key component of a machine learning pipeline.

It makes it possible to chose a set of hyper-parameters so that

- The model is accurate enough
- The model generalizes well (*i.e.* does not over-fit)

Recall from lecture 3:



Course outline:

- Evaluation metrics, what they mean
- How/when/why yo apply them

Evaluation metrics

Accuracy

Accuracy is the most natural classification evaluation:

$$\text{accuracy} = \frac{\#\text{good classifications}}{\#\text{instances}}$$

However, it has many limitations:

- Misleading when classes are imbalanced
- ...

Confusion matrix

Confusion matrix sums up all the $(y, \hat{y})$ possibilities in a matrix.
Example for binary classification:

		Prediction	
		0	1
Actual	0	TN	FP
	1	FN	TP

Machine learning losses

0-1 loss:

$$\ell\left(\hat{y}, y\right) = \mathbb{1}\left[\hat{y} = y\right]$$

Machine learning losses

0-1 loss:

$$\ell(\hat{y}, y) = \mathbb{1}[\hat{y} = y]$$

Squared loss:

$$\ell(\hat{y}, y) = (\hat{y} - y)^2$$

Machine learning losses

0-1 loss:

$$\ell(\hat{y}, y) = \mathbb{1}[\hat{y} = y]$$

Squared loss:

$$\ell(\hat{y}, y) = (\hat{y} - y)^2$$

Hinge loss:

$$\ell(\hat{y}, y) = \max(0, 1 - \hat{y}y)$$

Machine learning losses

0-1 loss:

$$\ell\left(\hat{y}, y\right) = \mathbb{1}\left[\hat{y} = y\right]$$

Squared loss:

$$\ell\left(\hat{y}, y\right) = \left(\hat{y} - y\right)^2$$

Hinge loss:

$$\ell\left(\hat{y}, y\right) = \max\left(0, 1 - \hat{y}y\right)$$

Squared hinge loss:

$$\ell\left(\hat{y}, y\right) = \max\left(0, 1 - \hat{y}y\right)^2$$

Machine learning losses

0-1 loss:

$$\ell\left(\hat{y}, y\right) = \mathbb{1}\left[\hat{y} = y\right]$$

Squared loss:

$$\ell\left(\hat{y}, y\right) = \left(\hat{y} - y\right)^2$$

Hinge loss:

$$\ell\left(\hat{y}, y\right) = \max\left(0, 1 - \hat{y}y\right)$$

Squared hinge loss:

$$\ell\left(\hat{y}, y\right) = \max\left(0, 1 - \hat{y}y\right)^2$$

Log-loss:

$$\ell\left(\hat{y}, y\right) = \log\left(1 + \exp(-\hat{y}y)\right)$$

Machine learning losses

0-1 loss:

$$\ell(\hat{y}, y) = \mathbb{1}[\hat{y} = y]$$

Squared loss:

$$\ell(\hat{y}, y) = (\hat{y} - y)^2$$

Hinge loss:

$$\ell(\hat{y}, y) = \max(0, 1 - \hat{y}y)$$

Squared hinge loss:

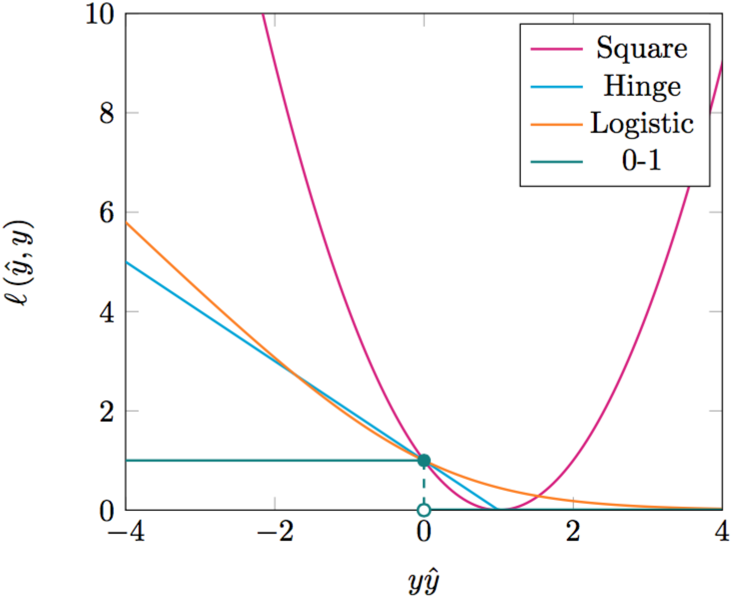
$$\ell(\hat{y}, y) = \max(0, 1 - \hat{y}y)^2$$

Log-loss:

$$\ell(\hat{y}, y) = \log(1 + \exp(-\hat{y}y))$$

Possibility to add a weight to the loss!

Machine learning losses



Precision, recall

ROC curve

Applying evaluation metrics

Train-test split

Reminder: ML algorithms (classification/regression) often rely on **many parameters**. How to **tune** them properly given a data set?

Train-test split

Reminder: ML algorithms (classification/regression) often rely on **many parameters**. How to **tune** them properly given a data set?

The most commonly used principle is the train-test split:

- **Split the data** into a training set and a test set
- **Train** on the training set
- **Test** on the test set

Train-test split

Reminder: ML algorithms (classification/regression) often rely on **many parameters**. How to **tune** them properly given a data set?

The most commonly used principle is the train-test split:

- **Split the data** into a training set and a test set
- **Train** on the training set
- **Test** on the test set

This is often referred to as **cross-validation**.

Cross-validation

Standard technique: k-fold cross-validation

- Split the data into k equally sized folds
- Remove 1 fold (= test fold)
- Train on the other folds
- Test on the removed fold
- Do it for all the folds

Cross-validation

Standard technique: k-fold cross-validation

- Split the data into k equally sized folds
- Remove 1 fold (= test fold)
- Train on the other folds
- Test on the removed fold
- Do it for all the folds

Note: It is often advised to perform a **stratified** cross-validation, *i.e.* each fold contains approximately the **same percentage** of samples of each target class **as the complete set**.

Small data set

Suppose you have **a small data set**. How to evaluate a classifier in this case?

Small data set

Suppose you have **a small data set**. How to evaluate a classifier in this case?

k-fold cross-validation? Even with $k = 2$, it would make the training set even smaller and make it hard to fit a proper model.

Small data set

Suppose you have **a small data set**. How to evaluate a classifier in this case?

k-fold cross-validation? Even with $k = 2$, it would make the training set even smaller and make it hard to fit a proper model.

Other option: **Leave-one-out** (LOO) cross-validation:

Small data set

Suppose you have **a small data set**. How to evaluate a classifier in this case?

k-fold cross-validation? Even with $k = 2$, it would make the training set even smaller and make it hard to fit a proper model.

Other option: **Leave-one-out** (LOO) cross-validation:

- **Remove 1 sample** from the data set
- Train on **all the other samples**
- Test on the sample you've removed
- **Evaluate** the prediction
- Do it **for each sample of the data set**
- **Aggregate** the evaluations

Small data set

Suppose you have **a small data set**. How to evaluate a classifier in this case?

k-fold cross-validation? Even with $k = 2$, it would make the training set even smaller and make it hard to fit a proper model.

Other option: **Leave-one-out** (LOO) cross-validation:

- **Remove 1 sample** from the data set
- Train on **all the other samples**
- Test on the sample you've removed
- **Evaluate** the prediction
- Do it **for each sample of the data set**
- **Aggregate** the evaluations

Remark: This could lead to many iterations even if the data set is small.

Small data set

Suppose you have **a small data set**. How to evaluate a classifier in this case?

k-fold cross-validation? Even with $k = 2$, it would make the training set even smaller and make it hard to fit a proper model.

Other option: **Leave-one-out** (LOO) cross-validation:

- **Remove 1 sample** from the data set
- Train on **all the other samples**
- Test on the sample you've removed
- **Evaluate** the prediction
- Do it **for each sample of the data set**
- **Aggregate** the evaluations

Remark: This could lead to many iterations even if the data set is small.

Alternative: Leave-p-out (LPO)

Parameter selection

One of the goals of model evaluation is to select a **good model**.

Parameter selection

One of the goals of model evaluation is to select a **good model**.

Hence we want to choose one (or several) hyper-parameters. How do we do?

Parameter selection

One of the goals of model evaluation is to select a **good model**.

Hence we want to choose one (or several) hyper-parameters. How do we do?

Most classic way: a **grid-search**

- For each parameter, define a set of possible values
- For each parameter combination, train/test the model
- Pick the parameter combination which gives best results

Parameter selection

One of the goals of model evaluation is to select a **good model**.

Hence we want to choose one (or several) hyper-parameters. How do we do?

Most classic way: a **grid-search**

- For each parameter, define a set of possible values
- For each parameter combination, train/test the model
- Pick the parameter combination which gives best results

Example: SVM with kernel degree $\in \{1, 2, 3, 4, 5\}$ and soft-margin parameter $\in \{0.01, 0.1, 1, 10\}$.

Parameter selection

One of the goals of model evaluation is to select a **good model**.

Hence we want to choose one (or several) hyper-parameters. How do we do?

Most classic way: a **grid-search**

- For each parameter, define a set of possible values
- For each parameter combination, train/test the model
- Pick the parameter combination which gives best results

Example: SVM with kernel degree $\in \{1, 2, 3, 4, 5\}$ and soft-margin parameter $\in \{0.01, 0.1, 1, 10\}$.

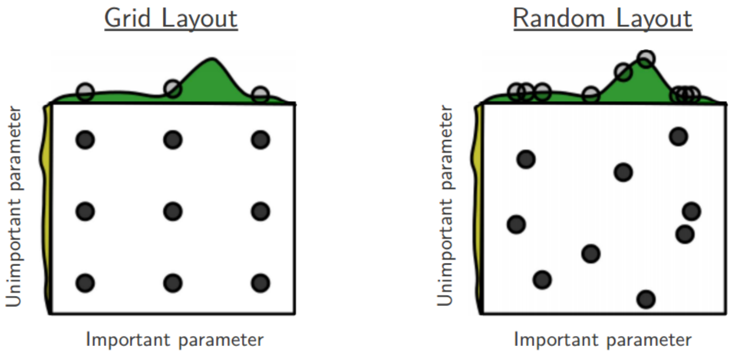
Important note: Hyper-parameter ranges vary a lot from an application to another. It is **data-dependent**.

Grid search vs random search

Random search is a more and more popular alternative to grid search, especially in this case:

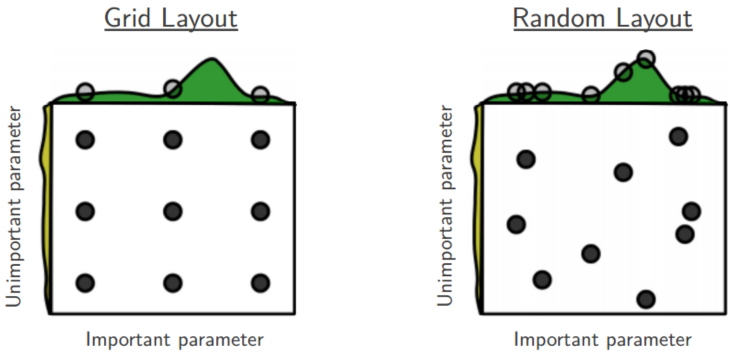
Grid search vs random search

Random search is a more and more popular alternative to grid search, especially in this case:



Grid search vs random search

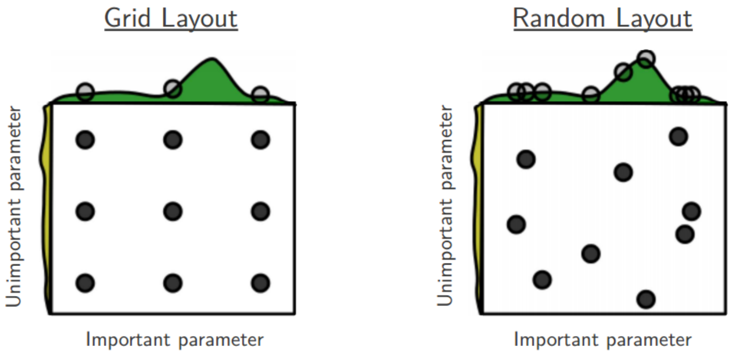
Random search is a more and more popular alternative to grid search, especially in this case:



In any case, you need to guess upper/lower bounds on the parameters.

Grid search vs random search

Random search is a more and more popular alternative to grid search, especially in this case:



In any case, you need to guess upper/lower bounds on the parameters.

Practical note: Each parameter combination can be trained/tested separately => possibility to distribute the tasks

Conclusion

There are several ways to evaluate a model depending on

- **What you value** in your application
  - Proper metric & loss function choices
- **Your data**
  - Proper evaluation framework choice

Conclusion

There are several ways to evaluate a model depending on

- **What you value** in your application
  - Proper metric & loss function choices
- **Your data**
  - Proper evaluation framework choice

Think about this before applying a **random algorithm** and evaluating it with a **random metric**!

Thank you! Questions?