# B(E)3M33UI — Exercise E: Model evaluation and diagnostics.

Petr Pošík, Jiří Spilka

March 21, 2017

The goals of this exercise:

- learn about different performance evaluation metrics
- evaluate quality of models using different methods (holdout, cross-validation)
- gain insights into models using diagnostic tools (validation and learning curves)

## 1 Performance evaluation metrics

We are already familiar with two different metrics: **mean square error** for regression task and **zero-one error** for **classification** task. In this exercise we further focus on the binary classification only, i.e. $y \in \{0, 1\}$. A detailed information about classification performance provides the so called **confusion matrix** that describes relationship between true class, $y$, and predicted class by a model, $\hat{y}$.

| | | **predicted class** | |
|---|---|---|---|
| | | $\hat{y} = 1$ | $\hat{y} = 0$ |
| **actual class** | $y = 1$ | True Positive (TP) | False Negative (FN) |
| | $y = 0$ | False Positive (FP) | True Negative (TN) |

The **prediction accuracy** is then given by:

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

Further, two additional metrics are widely used, **true positive rate** (TPR) and **false positive rate** (FPR):

$$TPR = \frac{TP}{TP + FN} \qquad FPR = \frac{TN}{TN + FP}$$

In the previous exercises, we measured the model error by our own implementation (`model_evaluation.py` module). Now, we will use the built-in facility of scikit-learn.

**Task 1**: In exE-1.py use SVM classifier and evaluate its performance using confusion matrix, accuracy, true positive rate, and false positive rate. Use the `sklearn.metrics` module and print the results. Note that we still use the training data for evaluation!

**Hints:**

- Experiment with number of features included in the model. Which model is better, using only few features or all available features?

Clearly. the confusion matrix depends on a **single threshold** used to decide if the example is positive or negative. Receiver operating characteristic (ROC) is one of the graphical methods to compare binary classifiers for **different values of threshold** and shows how performance change with a **change of threshold** that is used to divide classes. ROC plots true positive rate (TPR) on $y$ axis and false positive rate (FPR) on $x$ axis. An ideal classifier corresponds to a point (0,1), i.e. $TPR = 1, FPR = 0$.

**Task 2**: Study the ROC and the documentation for `metrics.sklearn.roc_curve`. Plot a ROC curve for SVM model.

**Task 3**: Experiment with the number of features included in the model. Setup several models and compare them using ROC curve.

**Hints:**

- Study the function `plot_roc()` in `plotting.py`.
- You can compare several models of different kind, e.g. logistic regression, SVM, Linear Discriminant Analysis etc., or you can compare the same type of model with different settings, e.g. SVM with different kernels (linear, polynomial, rbf) and with different $C$.
- Can you train a "perfect" model that has Area under ROC curve: AUC = 1? Again, try to answer the question: Which model is better? Which one you would use?

# 2 Model evaluation

In the model evaluation we want to **estimate predictive performance of model** given new, previously unseen, data (i.e. we want to estimate performance when a model will be deployed in practice). So far we have evaluated models on training data only and we have observed that by tuning model parameters we can obtain correct classification without errors. However, this does not provide information about predictive performance of the model.

## 2.1 Training / test split (hold out)

To get some insight, how well the model will perform on new data, let's split the data into training and test sets. Then train (learn) models on training set and test it on the both.

**Task 4**: In `exE.py`, fill in the code to split the data into training and testing data sets. Use the function `cross_validation.train_test_split`.

**Hints:**

- In the output of the scripts, check shapes of resulting Numpy arrays. Are they compatible?

**Task 5**: In `exE.py`, fill in the code to train the SVM on training data, and compute the accuracy on both, training and testing. How do they compare? Do you like what you see?

### 2.2 $k$-fold cross-validation (CV)

Simple train/test split is used for initial analysis or for a large datasets where data are abundant. However, when data are scarce or when we need to tune model parameter we resort to $k$-fold cross-validation technique, where training dataset is randomly split into $k$ fold without replacement. Then $k-1$folds are used for training and one fold is used for testing. The procedure is repeated $k$ times.

**Task 6**: In `exE.py`, use the function `sklearn.cross_validation.cross_val_score` to get the CV estimate of SVM performance. You should get a list of the accuracies, one for each fold.

**Hints:**

- Look at the documentation for crossvalidation.

**Task 7**: Run the script several times. Do you see any fluctuations in the accuracy estimates for train/test split and cross-validation?

**Task 8**: How do you judge the SVM model? Is it properly set?

## 3 Model tuning

SVM models have several parameters that are used to tune them. We have already mentioned them briefly above. You should also have knowledge from the SVM lecture.

### 3.1 Manual tuning

**Task 9**: If you need to, check the different available kernels and meaning of `C` and `gamma` parameters: `sklearn.svm.SVC`

**Task 10**: Try to find a better setting for the `C` and `gamma` parameters of SVM by hand. What does "a better setting" actually mean?

### 3.2 Automatic tuning via grid search

Clearly, to guess the best parameters manually is impractical. Let's try to use one of the automatic techniques, **grid search** to search for optimal settings for SVM.

**Task 11**: Learn about the `GridSearchCV()` function in the documentation.

**Task 12**: Use `GridSearchCV()` to find near-optimal values of `C` and `gamma` for SVM with RBF kernel. *Use only the training part of data to search for the parameter values!*

**Task 13**: Print out the scores of the final classifier on training and testing data.

**Dealing the datasets correctly with respect to the learning algorithms is a *crucial* thing in data analysis. Think about it thoroughly!**

# 4 Model diagnostics

For educational purposes we will use a synthetic Ripley's dataset here instead of the auto-mpg dataset. The dataset is composed of two classes, where each comes from bimodal normal distribution with the same variance of 0.04. Positive cases are generated from: $\mu_1^+ = [0.4, 0.7]$, $\mu_2^+ = [-0.3, 0.7]$ and negative cases from: $\mu_3^- = [-0.7, 0.3]$, $\mu_4^- = [0.3, 0.3]$. The dataset is represent by features $X_1$ and $X_2$.

**Task 14**: Run the script exE-2.py to have an idea how the dataset looks like.

## 4.1 Learning curves

The learning curve is a technique that help us to diagnose if a learning algorithm has a problem with underfitting (high bias) or overfitting (high variance). Further, it can be used to investigate if obtaining more data would help to reach better performance.

**Task 15**: The script exE-2.py only plots a learning curve but its computation is not implemented yet. Implement function `compute_learning_curve()` that takes following arguments:

- selected `model` and `tr_sizes` (the sizes of training data that are used to train model)
- `Xtr,ytr,Xtst,ytst` training/test data and labels

  The function outputs:

- `train_errors` an array of training errors corresponding to the number of training examples
- `test_errors` an array of testing errors corresponding to the number of training examples

**Hints:**

- To simulate the increasing training dataset size, the function shall iteratively use increasingly large part of the training data.
- To compute the error, you can use the function 1 - metrics.accuracy().

**Task 16**: Try to display the learning curve several times, it is a stochastic process. Try to display learning curves for different classifiers, e.g. use SVM with $C = 1$ and change the parameter `gamma`. Can you provide examples of underfitting, overfitting, and optimal classifier? For which classifier it would be helpful to get more data?

# 5 Validation curves

Validation curves are very closely related to learning curves but instead of plotting the error as function of training sizes, the error is plotted as a function of model parameters (e.g. $\gamma$ for SVM). The validation curve thus provided information about underfitting and overfitting with respect to model parameters.

**Task 17**: Implement computation of validation curve (`compute_validation_curve()`) that takes the following arguments:

- selected `model`, parameter name `param_name` and parameter values `param_range`
- `Xtr,ytr,Xtst,ytst` training/test data and labels

The function outputs:

- `train_errors` an array of training errors corresponding to the number of training examples
- `test_errors` an array of testing errors corresponding to the number of training examples

**Task 18**: Try to play with the validation curves, e.g. use SVM and vary parameter `gamma`. Try to interpret the results.

# 6 Have fun!

**Complete the exercise as a homework, ask questions on the forum, and upload the solution via Upload system!**