DATA MINING AND MACHINE LEARNING

Lecture 6: Data preprocessing and model selection

Lecturer: Simone Scardapane

Academic Year 2016/2017
Table of contents

Data preprocessing
  Feature normalization
  Missing data imputation
  Outlier detection

Model selection, evaluation and fine-tuning
  K-fold cross-validation
  Model fine-tuning
  Other topics in model selection
  Advanced model evaluation techniques
Suppose you are a bank, and you wish to draw inferences from your clients, who are described by only two attributes:

- **Sex**, encoded as [0, 1] (male), or [1, 0] (female).
- **Income** [$/month], going from 0 to 10,000.

Now consider a generic client $\mathbf{x} \in \mathbb{R}^3$, and two other clients very similar to $\mathbf{x}$:

- $\mathbf{x}'$ is the same as $\mathbf{x}$, but of opposite sex. We have that $\|\mathbf{x}' - \mathbf{x}\|_2 = \sqrt{2}$.
- $\mathbf{x}''$ is the same as $\mathbf{x}$, but his income per month is 100$ higher. We have that $\|\mathbf{x}'' - \mathbf{x}\|_2 = 100$. 

The importance of feature normalization (1)
Any algorithm based on the Euclidean distance between patterns (e.g., $k$-NN) will give 10 times more importance to the income with respect to the sex, \textit{irrespective of its actual importance for the discriminative task}.

In practice, almost any machine learning algorithm will not work properly if the features are measured in units with vastly different scales.

The process of transforming data so that all features lie in the same range is called \textbf{feature normalization}.
Min-max scaling

Feature normalization is generally applied feature-wise, although a few techniques scale each pattern separately.

Denote by $\mathbf{x}$ a generic pattern. A common normalization step, **min-max scaling**, scales each feature to the range $[0, 1]$ by applying the following transformation:

$$x_i = \frac{x_i - x_{i,\text{min}}}{x_{i,\text{max}} - x_{i,\text{min}}}.$$

where $x_{i,\text{min}}$ ($x_{i,\text{max}}$) are the minimum (maximum) value for the $i$th feature in the dataset. We can successively scale in a generic range $[\alpha, \beta]$ by applying a second transformation:

$$x_i = x_i(\beta - \alpha) + \alpha.$$
Another popular technique is **standard scaling**, where:

\[ x_i = \frac{x_i - \mu_i}{\sigma_i^2} . \]

where \( \mu_i \) is the empirical mean and \( \sigma_i^2 \) the empirical variance computed on the dataset.

Most feature normalization techniques can be applied by excluding ‘extreme’ data from the computation of the statistics (e.g., using only data between the 15th and the 85th percentile) in order to provide more robustness to outliers. Sometimes, this is called **robust scaling**.
An example of feature normalization

Figure 1: A comparison of several feature normalization techniques on the third feature of the ‘Boston’ dataset. Note how the distribution of the data is never affected. Robust scaling is computed on the inter-quantile range (i.e., between the 25th and 75th percentile).
Many features in the real world exhibit highly skewed distributions with a very long tail. A classic example is the income of a person.

A common normalization in this case is log normalization:

\[ x_i = \log(x_i) . \]

Log normalization is important when we only care about relative changes in a quantity. For example, the relative increase of \( a + 1 \) with respect to \( a \) is always \( \log(1) \) using log normalization, irrespective of the value of \( a \).
Figure 2: Log normalization applied to the first feature of the ‘Boston’ dataset (crime rate).
Table of contents

Data preprocessing
  Feature normalization
  Missing data imputation
  Outlier detection

Model selection, evaluation and fine-tuning
  K-fold cross-validation
  Model fine-tuning
  Other topics in model selection
  Advanced model evaluation techniques
Another common problem in real-world data is missing data, wherein several entries of the input matrix $X$ are missing.

Most learning methods are not able to cope with missing data without some form of human preprocessing.

A baseline strategy is to simply remove any pattern with one or more missing features. This approach can be sub-optimal for two reasons:

1. If many features have missing values, we can end up with an extremely small dataset.
2. If the lack of data is correlated with some underlying factor (e.g., male users are more reluctant to share some information), then the resulting dataset will be biased.
An effective approach in practice is to replace missing data with the mean (or median) computed with respect to the corresponding column.

For classification problems, we can also do **conditional data imputation**, where the mean (or median) is computed only using data of the corresponding class.

If we have prior knowledge on the missing data process, we can do conditional imputation with respect to some other input features, e.g., compute the mean with respect to the user’s nationality.
If a categorical variable is missing values, an alternative approach is to add a new category corresponding to the missing case. For example, considering a feature $x = \{\text{male, female}\}$, we can construct a dummy encoding as:

\[
\text{male} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \text{female} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \text{missing value} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.
\]

The same can be done with a continuous variable; in that case, we use a default value for the feature when missing, and we add a bitwise feature which is 1 if the feature is missing, 0 otherwise.
Suppose that only the $k$th feature has missing values. We can model data imputation as a supervised learning problem, where we look for a function:

\[
f(x_{-k}, y) = x_k,
\]

where $x_{-k}$ denotes the set of all features excluding the $k$th one. Mean and median imputation are a special case of this setup, where the predicted value is constant (or piecewise constant for conditional imputation).

It is rare to obtain good results with overly complicated models for $f$. Linear regression (or logistic regression) are common choices using this setup.
Table of contents

Data preprocessing
  Feature normalization
  Missing data imputation
  Outlier detection

Model selection, evaluation and fine-tuning
  K-fold cross-validation
  Model fine-tuning
  Other topics in model selection
  Advanced model evaluation techniques
A third common problem is given by outliers, i.e., points that are not ‘consistent’ (in some sense) with respect to the rest of the dataset. Outliers can be due to a wide range of reasons:

- Noise in the data collection process.
- Real anomalous patterns, e.g., frauds in a credit card dataset.

If we are interested in detecting these anomalous patterns (e.g., for fraud detection), the problem is called anomaly detection or novelty detection. If we only wish to remove outliers in order to improve the accuracy we have outliers detection.
For low-dimensional problems, a common solution for outlier detection is to fit a known parametric distribution (typically a multivariate Gaussian), and remove points whose Mahalanobis distance is greater than a given threshold.

The covariance of the Gaussian is generally computed using techniques that are robust to outliers (e.g., minimum covariance estimation).

Automatic techniques for outlier detection can severely hinder the dataset if used improperly. If outliers are only due to a single feature, it is more common to visually inspect the dataset using box-plots.
Figure 3: Example of outlier detection using elliptical envelopes. (http://scikit-learn.org/stable/auto_examples/covariance/plot_mahalanobis_distances.html).
Example of box-plot

Figure 4: An example of box-plot on three features of the ‘Boston’ dataset. The red line is the median; the box covers the interquantile range (between the 25th percentile and the 75th percentile). Points outside the ‘whiskers’ are considered outliers.
There are countless other operations that can be applied to data before learning, at the discretion of the specialist and depending on the domain knowledge. Just to give a few examples:

- **Generalization**: replacing a categorical value with a coarser one (e.g., region instead of postcode).

- **Aggregation**: replacing a continuous variable by binning its value in a set of possible ranges (this can also be done on the output to transform a regression problem into a classification one).

- Constructing **polynomial features** from the original ones.
Table of contents

Data preprocessing
  Feature normalization
  Missing data imputation
  Outlier detection

Model selection, evaluation and fine-tuning
  K-fold cross-validation
  Model fine-tuning
  Other topics in model selection
  Advanced model evaluation techniques
K-fold cross-validation

Up to now, we used the holdout procedure to estimate the (expected) test error of a function. A single holdout can be misleading, particularly for estimators having a large variance.

A more principled (and common way) to assess a model is to run a K-fold cross-validation:

1. Split the dataset into K equally sized portions $S_1, \ldots, S_K$.
2. Train on $S_2, \ldots, S_K$, and test on $S_1$. Get error $E_1$.
3. Train on $S_1, S_3, \ldots, S_K$, and test on $S_2$. Get error $E_2$.
4. ...
5. Train on $S_1, \ldots, S_{K-1}$, and test on $S_K$. Get error $E_K$.
6. Compute final accuracy as $E = \frac{1}{K} \sum_{i=1}^{K} E_i$. 
Visualization of K-fold cross-validation

Figure 5: https://sebastianraschka.com/faq/docs/evaluate-a-model.html.
A third possibility is the **bootstrap method**:

1. Construct a bootstrap dataset by drawing $N$ examples with replacement from the original dataset.
2. Build $B$ different bootstraps (e.g., $B = 100$), and use them to train $B$ different functions.
3. For each point in the training set, compute the average error of all functions trained from bootstrap datasets not containing that point.
4. Obtain final accuracy by averaging the error over the entire training dataset.

This procedure can be extended in a large number of ways, see Section 7.11 in the book.
Table of contents

Data preprocessing
  Feature normalization
  Missing data imputation
  Outlier detection

Model selection, evaluation and fine-tuning
  K-fold cross-validation
  Model fine-tuning
  Other topics in model selection
  Advanced model evaluation techniques
Finding the hyper-parameters

Most learning methods require the user to select one or more hyper-parameters, most notably complexity-based parameters:

- The $C$ parameter in ridge regression and LASSO.
- The $k$ parameter in $k$-NN.
- The $\varepsilon$ parameter in the $\varepsilon$-insensitive loss function.

Suppose we try several possibilities for each hyper-parameter. How do we select which one is the best? We already know that training error is a poor proxy for the generalization capability of a learning model. As a matter of fact, these two problems are almost equivalent:

- **Model assessment**: evaluate the expected error of a model.
- **Model selection**: evaluate the expected error of two models to select the best one.
We can solve the model selection problem equivalently to the model assessment problem, e.g., using holdout:

1. Keep a portion of the training dataset apart (called the validation dataset).
2. Train several models using the remaining portion, each one with a different selection of hyper-parameters.
3. Select the hyper-parameter with the lowest error on the validation dataset, and retrain the model using the entire dataset.

One can equivalently use holdout, K-fold cross-validation, or bootstrap, either for testing or for validation.
Visualizing nested cross-validation

Figure 6: https://sebastianraschka.com/faq/docs/evaluate-a-model.html.
The proper way of doing model selection

Figure 7: https://sebastianraschka.com/faq/docs/evaluate-a-model.html.
Grid search for model fine-tuning

For models with one/two hyper-parameters, the most common way to do fine-tuning is a **grid search**, where we specify a list of possible configurations, and we exhaustively search for the best one.

For example, we can optimize $k$-NN by searching for $k$ in 5, 10, \ldots, 50. Some parameters are commonly optimized in *log space*, since we assume that small changes are not significant, e.g., it is common to optimize $C$ by searching in an interval such as:

$$2^{-10}, 2^{-9}, \ldots, 2^9, 2^{10}.$$

We can also repeat several grid search procedures at different levels of resolution, by focusing around the optimum of the coarser grid search.
Another popular choice is **random search**:

1. We specify a sampling distribution for each parameter, e.g. a uniform distribution in \([-10, 10]\) for \(\log(C)\).
2. At each iteration, we randomly sample one combination of parameters, and optimize a model up to some budget of computational time.
3. We conclude the search after a given number of iterations.

Note how both grid search and random search can be trivially parallelized by training and evaluating several models at once in a parallel architecture.
Comparing grid search and random search

Hyper-parameter selection as an optimization problem

Optimizing parameters can be seen as an optimization problem; denoting by $\Theta$ the set of hyper-parameters, we want to optimize:

$$\Theta^* = \arg\min_{\Theta} \text{Err}(\Theta),$$

where $\text{Err}(\Theta)$ can be, for example, a 5-fold cross-validated error over the training dataset.

The problems are that (i) we generally cannot compute any derivative information on $\text{Err}(\cdot)$; (ii) even a single evaluation of the objective function can be expensive. **Bayesian optimization** can be used as a black-box technique to this end.
Other forms of fine-tuning

Model selection is not limited to selecting hyper-parameters related to complexity:

- We can use cross-validated accuracy as a proxy measure inside wrapper methods for feature selection (see later), e.g., to recursively remove one feature at a time.
- In an iterative optimization algorithm, we can use a validation set to choose when to stop the optimization process (early stopping). Early stopping can be as effective as regularization, particularly for neural networks models.
Table of contents

Data preprocessing
- Feature normalization
- Missing data imputation
- Outlier detection

Model selection, evaluation and fine-tuning
- K-fold cross-validation
- Model fine-tuning
- Other topics in model selection
- Advanced model evaluation techniques
Feature selection is the task of selecting a subset of features which is ‘optimal’ (in some sense) for our learning task.

Feature selection methods belong to three distinct categories:

- **Filter methods** select features by computing some ‘measure of interest’, such as the Pearson correlation coefficient between the feature and the output. They are fast, but their accuracy in general is not optimal.

- **Wrapper methods** iteratively train several models by removing or adding features, and use their validated accuracy to choose the final subset.

- **Embedded methods** are learning methods which automatically select features (e.g., the LASSO algorithm).
We already saw in the linear case that selecting an optimal subset of features is a very difficult problem. This is a *combinatorial optimization* problem, which can (in principle) be solved using techniques such as genetic algorithms or particle swarm optimization, which are outside the scope of this course.

Sometimes, we are interested in reducing the dimensionality of the input, but we do not care to keep the original features. In this case, we can apply *dimensionality reduction* techniques, which will be introduced in a future lecture.
Table of contents

Data preprocessing
   Feature normalization
   Missing data imputation
   Outlier detection

Model selection, evaluation and fine-tuning
   K-fold cross-validation
   Model fine-tuning
   Other topics in model selection
   Advanced model evaluation techniques
Once a model is trained and fine-tuned, we need to evaluate its performance. Reporting the MSE or the classification accuracy on the test set is a good measure, but it might not provide all the necessary information.

As a concrete example, consider a binary classification problem, where 99% of the data is of class 0. In this case, a classifier that always predicts 0 as output would have 99% accuracy! This is an example of unbalanced dataset.
For binary classification, the outputs of the classifier can be grouped in the so-called **confusion matrix**:

<table>
<thead>
<tr>
<th>Prediction is 0</th>
<th>Real class is 0</th>
<th>Real class is 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TN</strong>: True negative</td>
<td><strong>FN</strong>: False negative (Type II error)</td>
<td></td>
</tr>
<tr>
<td><strong>FP</strong>: False positive (Type I error)</td>
<td><strong>TP</strong>: True positive</td>
<td></td>
</tr>
</tbody>
</table>

This is easily extended to a multiclass case.
Precision and recall

From the values of the confusion matrix, we can define several new metrics of accuracy:

\[ \text{Precision} = \frac{TP}{FP + TP} \] (1)

Recall (or true precision rate, TPR) = \[\frac{TP}{FN + TP}\] (2)

False positive rate (FPR) = \[\frac{FP}{FP + TN}\]. (3)

The F1 score is defined as the harmonic mean of precision and recall, and it is especially useful in unbalanced scenarios:

\[ F_1 = 2 \cdot \frac{1}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \] (4)
In practice, we can obtain insights into the classification process at an even lower level of granularity, by remembering that the outputs of a classifier are almost always real numbers, that we binarize in order to obtain the final classes.

Up to now, we considered the simple case where we round these outputs to the nearest integer (when $y \in \{0, 1\}$), or we take the sign (when $y \in \{-1, 1\}$). By actually varying the threshold for binarization, we can obtain different values of recall and FPR:

- For a very low threshold, we obtain perfect recall (all patterns are classified as positive), but very high FPR.
- On the opposite, for a very high threshold, we obtain a poor recall with an extremely small FPR.
By plotting the recall against the FPR for several choices of the threshold, we obtain the receiver operating characteristic (ROC) curve. The area under the ROC curve (AUC) is another very common measure of performance for a binary classifier.

We can compare two classifiers by plotting their ROC curves to get a wider set of insights into their behavior. Another related curve plots the precision against the recall (PR curve).
Plotting the ROC curve

Figure 9: If we only consider a single threshold, each classifier is a point in the ROC space [Wikipedia, en: Receiver Operating Characteristic].
Figure 10: More in general, we can plot an entire curve in the ROC space [Wikipedia, en: Receiver Operating Characteristic].
Some material in this lecture is taken from the following Sections: 3.3 (subset selection), 9.6 (missing data), 7 (model assessment).

The following is a classic introduction to feature selection:


An accessible entry point for Bayesian optimization (outside the scope of the course) can be found in the following article: