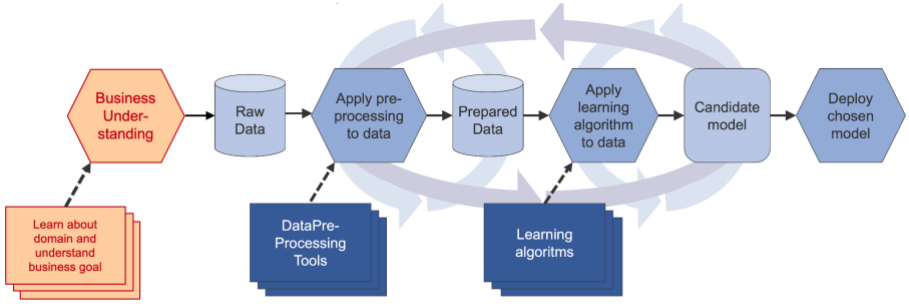
Practicing Learning from Data

Assignment 1 - Regression

The Data Science Process - Step by Step

1. Business Understanding

In the introductory lecture, we introduced the complete data science process. Here we assumed that data scientists are familiar with the domain at hand. Indeed, data scientists are sometimes specialized on a certain domain (like medical doemain, finance, etc.). Yet, this is not always the case. Therefore an additional, preliminary step may be necessary before they can even start with their project: the "Business Understanding" step[1].



Business understanding means understanding the domain and the business goals of your client. Understanding the domain is a precondition to understanding the feature semantics of your data set, as well as the specific business goals of your client. Understanding the business goals in turn enables you to translate them into analytics goals in a correct way.

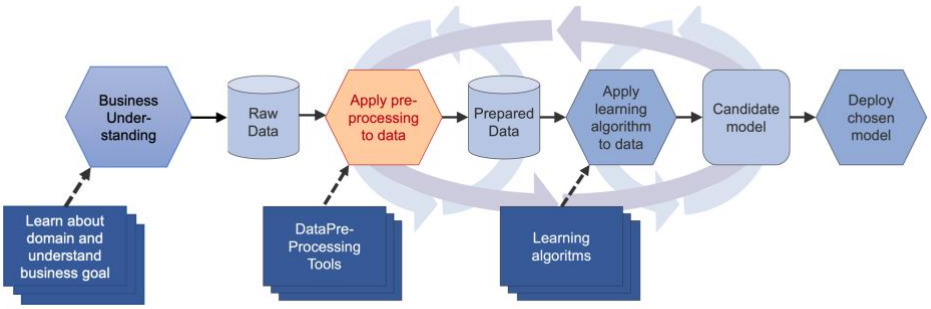
In the Business Understanding step, data scientists need to research about the domain and interview the stakeholders, usually repeatedly. Together with data preparation, this step often takes up a huge portion of the project time. Yet, without proper understanding of business needs and feature semantics, a data science project is doomed to failiure.

Here are some links that you may find helpful in this step of your project:

- About LC
- LC website (with generic FAQs)
- Personal loans FAQs
- How a personal loan works
- Personal loan eligibility criteria
- Whats a good personal loan interest rate?
- What is a credit score?
- What is a dept-to-income-ratio (DTI-ratio)?
- What is an annual percentage rate (APR)?
- Understanding revolving credit
- What is installment credit?
- What is an APR?

[1] Cf., e.g., the CRISP-DM process model.

2. Data Preprocessing



Descriptive data summarization

After you gained sufficient *Business Understanding*, *Desciptive Data Summarization* helps you to gain *Data Understanding*: Refer to the data summarization tools discussed in the second lecture on data preparation. E.g.: What are the scales of measurement of each attribute (categorical or numerical)? How many levels (different values) does a categorical variable have? What are the levels? What is the range of a numerical variable? How are they distributed? Are they skewed? Biased? Are there paired correlations? Etc.

Data Cleaning, Integration, Transformation and Imputation

Refer to the steps and tools discussed in the second lecture on data preparation. E.g.: Are there data quality problems such as outliers, redundancies, inconsistencies, missing values, obviously erroneous values? Sparse attributes or sparse data objects? Where do you need to *impute* (fill in missing values), and how? Where do you need to transform your data, e.g. discretize or normalize numerical features? Are there data objects or features who's quality is too bad to fix them (cf. also "*Data Reduction*" below)? Etc.

- *Remark:* You will notice that some learning algorithms require pre-preocessing steps (e.g., transformations to ensure normality or homoscedasticity) that others don't. So, to decide if you want to carry out a certain pre-processing step (that might be a lot of work), you need to know which algorithm you will apply. But: Which algorithm you will apply depends on its perfromance, which depends on the features you select. The chicken egg problem.
- *Advice:* Keep calm and carry on! The data science process is an iterative process. Approach this issue pragmatically: Start with the simplest learning algorithm (linear regression) and include all features that are low-hanging fruits (i.e., not too much work in pre-processing). Only after having tried out different more sophisticated algorithms with these features, consider jumping back to this point to include some more features (cf. also '*3. Applying a Model to Data'*).

Data Reduction

- *General Approaches to Data Reduction*

  To reduce the size of your data set, you can make it *less deep* (reduce the number of data objects) or *less wide* (reduce the number of features).

  - *Decreasing Depth:* Reducing the number of data objects (e.g. by random subsampling) can be helpful to reduce calculation time when you are familiarizing yourself with the code or when trying to get some methods to work for the first time. Once you figured it out, you can work with a bigger sample again to achieve more reliable evaluation results. Remark: Be aware that some methods (e.g., some feature selection methods) may take long to finish on your machine when run on the full data set (e.g., several days if you are unlucky).
  - *Decreasing Width*: Reducing the number of features may decrease noise in the data, thereby potentially improving prediction results. Some algorithms also perform better with a smaller feature space. And, of course, reducing the number of dimensions also can speed up the process in most cases.

For more sophisticated methods of data reduction, please refer to the video lectures. In the following we only give some hints regarding *manual data reduction*.

- *Manual deletion of data points*

  As one of the first steps in data pre-processing, check if your data set includes sparse data points (rows). Set a threshold for sparsity, and kick them out if applicable.

- *Manual feature selection*

  Before you even start playing around with the automated feature selection methods or dimensionality reduction methods discussed in the lecture (such as best subset selection or PCA), the first step is to go through all the features manually. You can use the following criteria to decide if you wanna keep a feature or kick it out:

  - *Data quality*

    If the data quality of a feature is too bad to fix it, you will have to remove the feature right away. This happens, e.g. if the feature is sparse, or contains too many obvious errors.

  - *Pre-processing effort*

    Some features might seem promising for prediction (judging from feature semantics), so that you don't want to exclude them right away. Yet, some of them may require a high pre-processing effort to become applicable with your model in the first place (e.g., categorical variables with a lot of levels are difficult for regression tasks). In this case, it is sensible to persue an iterative approach: try out the low hanging fruits first, and add such a feature only if the model performance is really bad, or if project time admits it easily. Good guiding princples are Ocam's Razor and the Pareto Principle.

  - *Feature semantics*

    Sometimes feature semantics demands or suggests that you don't use a certain feature for prediction. To determine if this is the case, you need to have sufficient "Business Understanding" (see step 1 above.) Examples for our data set are as follows:

    - The feature is *obviously* not related to the target variable, i.e., it is obviously not informative for prediction (E.g., `member_id`).

      Such features add additional noise that can decrease prediction performance, because the algorithm tries to find a connection that's not there.

    - The feature is not available for new incoming data, but only for historic data.

      E.g., the feature is not available for loan applications, but only for running loans (e.g., `issue_d`, `mths_since_last_delinq`). Attention: If a feature not available for loan applications, it will not be present in the new incoming data (the "secret data set")! You need to make sure that your model excludes them, otherwise it will run into an error in the "Reality Check".

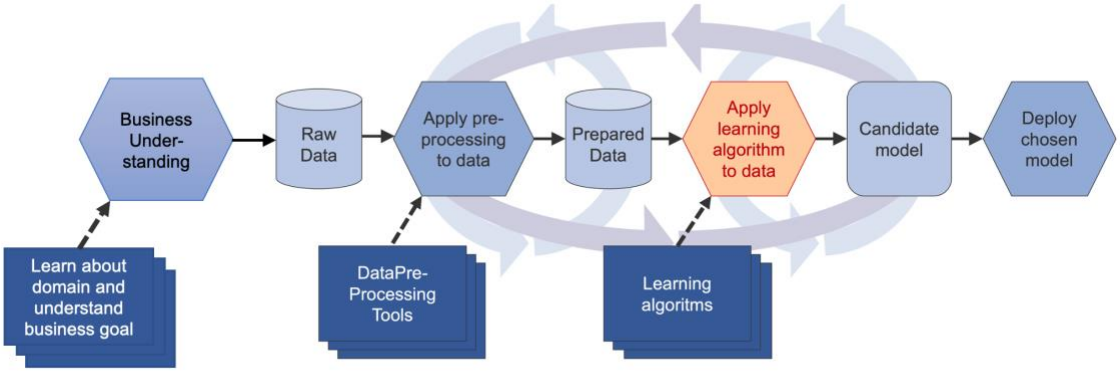
    - The feature is only available for a certain type of new incoming data.

      E.g., the feature is only available for loan applicants who have a credit history with LC (e.g., `mths_since_last_major_derog`); or the feature is only available for loan applicants, who apply for a "joint credit" (e.g., `verified_status_joint`).

      In such cases, you need to decide how to deal with it: you might exclude certain customer types right away, or might you train different models for each of the types. Again, there is a trade-off between implementation effort and potential preformance gain. Again, an iterative approach guided by Occam's razor is your friend. Make sure you don't loose sight of project management and the minimal deliverables needed to pass the assignment.

  Remark: Notice that the choice of features also may depend on the learning algorithm that you plan to employ. E.g., if you are employing multiple linear regression, you need to make sure that your set of input features do not contain collinear variables.

3. Applying a Model to Data



Choose a Model Family

Now is the time to choose the the type of learning algorithm (the model family) that you want to try first. Since this task is a *regression task*, you can choose between regression analysis, regression trees and ensemble trees. A good approach here is to start with a simple type of algorithm (e.g., linear regression) that you use as a benchmark for your project. After that – if project time permits - try a more sophisticated model (if sensible), and try to outperform the benchmark results. This way you make sure that you have a running model early in the process, that at least fulfills the minimal requirements needed to pass the assignement.

Configure, Train and Evaluate your Benchmark Model

Most learning algorithms need *configuration*, i.e., you need to *choose the hyperparameter(s)*. These are the model parameters that cannot be learned by the algorithm automatically. E.g., in polynomial regression, you need to choose the degree of the polynomial that you want to use.

After that, you *train and evaluate* your model: You split the data set into training and test data. You let the algorithm learn the parameters from the training data (e.g. slope and intercept in linear regression). After that, you evaluate the learnt model on the test data set (e.g. using MSE). Now your benchmark model is finished.

Remark 1: If you now want to deploy the model directly, train it again on the whole data set (training + test data) to get the best performance for real world use. The resulting model will deviate a bit from the evaluated model, but since you use more data points for learning the parameters, it it usually performs a bit better.

Remark 2: The above approach corresponds to the validation set approach. To get a more reliable evaluation result, you can instead apply *k-fold cross validation* (on the whole data set, i.e., training + test data). To deploy it, you again use the whole data set to learn the final parameters.

Remark 3: Instead of “training the algorithm” some people use the term “learning the model parameters” or “model fitting”.

What next?

Most likely, you are not entirely happy with the evaluation results, and you want to improve your model. In this case, you may want to *jump back* to one of the previous steps:

Business Understanding

- If your results are either extremely bad or "too good to be true", you may have misinterpreted a feature and used it the wrong way. Double-check its meaning.

Data Preprocessing

- If your model performs extremely bad, you might wanna check if you made a grave mistake in one of the pre-processing steps. Double-check the descriptive data summaries of the raw and pre-processed data to identify potential problems.
- If you have an acceptable model that you want to tweak even further, consider including one of the features that you kicked out during manual feature selection, or explore more sophisticated methods for imputation or outlier detection. (Notice: if this is a lot of work, jump to hyperparameter tuning first – see below – and save this iteration loop for later).

Alternatively, you may want to *jump forward* to the next step, hyperparameter tuning.

Hyperparameter tuning

After having trained your model with a certain hyperparameter configuration (e.g., choose plynomial degree of 2 for regression analysis), you might wonder if another choice of hyperparameter (e.g. polynomial of degree 3) would have given better evaluation results. You can now try out different hyperparameter settings, and compare the resulting models.

The easiest way to do this is the *3-way split holdout method*:

- Instead of splitting your data set in training and test data, you split it in training, test and validation data.
- For each choice of hyperparameter values (e.g. degree 2 and degree 3), you train and evaluate the corresponding model on the training and test data. You compare the evaluation results, and choose the best one (e.g. the model with degree 2 may win).
- To avoid overfitting on the test set, you evaluate the winning model again on the validation data set. This is the performance value that you can communicate to you stakeholders.

Remark 1: To get more reliable (robust) evaluation results for comparing the different hyperparameter settings, employ *k-fold cross validation* (on training + test data) instead of the validation set approach described above.

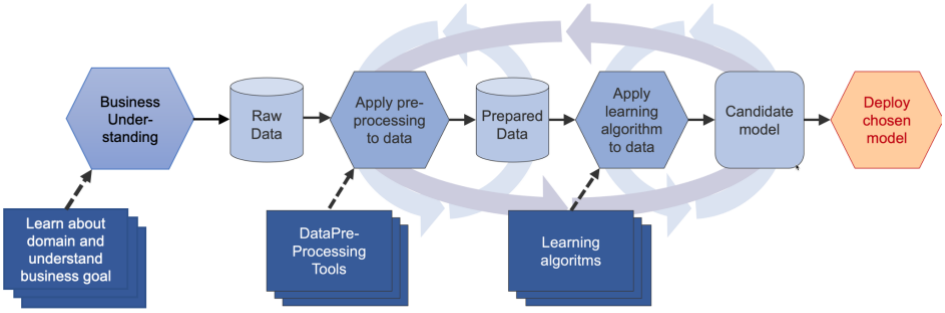
Remark 2: Before you deploy the winning model, train it again on the whole data set (training + test + validation data) to learn the final parameters for best performance.

Remark 3: Instead of "hyperparameter tuning" some people use the term "model selection".

What next?

If you want to teak your model further, you can try out additional algorithm types (e.g. spline regression or ensemble trees). Apply the 3-way split holdout method with k-fold cross validation to compare all algorithm/hyperparameter-pairs, and select the best one. Also consider to jump further back into another iteration cycle (Business Understanding or Data Pre-processing).

4. Deploying the final Model



When you have your winning model ready, deploy it (see also remarks above) by preparing the files for the "Reality Check". ☺