Project Machine Learning
Task 2: Data Preparation

Philipp Liznerski[1], Saurabh Varshneya[2], and Tobias Michels[3]

[1]liznerski@cs.uni-kl.de, `@liznerski`
[2]varshneya@cs.uni-kl.de, `@varshneya`
[3]t_michels15@cs.uni-kl.de, `@tmichels`

November 24, 2021
Deadline: 10/01/2022

Introduction

In contrast to the previous task, we now consider the case where our label space $\mathcal{Y}$ is no longer finite. Instead we assume that $\mathcal{Y} = \mathbb{R}$ and we call such a problem *regression* instead of classification. Other than that we still want to find a function $\hat{f}$ such that $\hat{f}(\mathbf{x}) \approx y$.

Perhaps a bit surprisingly, problems of this kind are often easier to solve. Assume that we restrict the function to be linear, i.e., $\mathcal{X} = \mathbb{R}^d$ and $\hat{f}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ for some $\mathbf{w} \in \mathbb{R}^d$. If we are given exactly n datapoints and the input matrix $X = \begin{bmatrix}\mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_n\end{bmatrix}^\mathsf{T}$ is of full rank, we can simply find $\mathbf{w}$ as the unique solution of the linear system $X\mathbf{w} = \mathbf{y}$. If, however, $n > d$, this direct approach won't yield a solution and therefore we need to change our approach. For example, we could try to find a $\mathbf{w}$ that minimizes the squared difference between the true labels and our function output while at the same time having a small norm:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}\in\mathbb{R}^d} \|\mathbf{w}\|^2 + C\sum_{i=1}^{n}(\mathbf{w}^T\mathbf{x}_i - y_i)^2, \tag{1}$$

where $C > 0$ is called *regularization constant* and controls the amount of regularization that is applied to $\mathbf{w}$. This method is called *ridge regression* and its solution can be found by solving another linear system $(X^TX + \frac{1}{C}I_d)\mathbf{w}^* = X^T\mathbf{y}$ instead, which is guaranteed to have a unique solution.

1 Data Pre-processing

In theory, you can now throw ridge regression at a problem and solve it, but, as usual, it is not quite that easy in practice. Note that we assumed $\mathcal{X} = \mathbb{R}^d$, i.e., inputs are available as real-valued vectors, but often this is not the case. So essentially, we are looking for a function $t\colon \mathcal{X} \to \mathbb{R}^d$ that *transforms* other types of data into a format usable by ML methods such as ridge regression.

Word embedding The goal of this task is to give you an idea how to transform text into real-valued vectors, but before we can possibly know how to transform the data, we need to take a closer look at it. We recommend you to carry out tasks a), b), c), and d) first.

As you would have noticed, many columns of the wine reviews dataset contain *text* data, which is particularly hard to transform into a format that can be used by ML algorithms. Traditionally, one would choose certain words to form a vocabulary $V = (v_1, \ldots, v_l)$ and each text or *document* is represented by a vector $\mathbf{a} \in \mathbb{N}^l$, where $a_i = n_i$ means that the word v_i occurs n_i times in the text. To reduce the impact of frequent words, each entry is usually multiplied by a factor that is inversely proportional to the number of documents in your data that the word appears in. This method is known as *TF*IDF* in the literature and there exist several variations of it.

TF*IDF is a simple concept and therefore relatively easy to implement, but it also has some significant drawbacks. Most often, $l = |V|$ will be quite large, meaning that each text is represented by a vector of many elements, most of which will be zero. In other words, TF*IDF induces a *sparse* representation. Recently, another approach based on *word embeddings* became quite popular, since it produces lower dimensional, *dense* vector representations instead. Some of the most widely known methods based on this approach are *word2vec* and *doc2vec*. In general, we cannot say that one approach is strictly superior compared to the other, however, implementations and trained models for both can be found on the internet, which means you can experiment with both approaches if you want. You now have the adequate information to carry out tasks e) and f) listed below.

Tasks

a) Install and familiarize yourselves with the *pandas* [1] library. It will be very helpful for handling and transforming all kinds of data types. Furthermore, it can import and export data from/to a variety of data formats.

[1] `https://pandas.pydata.org/`

b) The code template in your repository contains a function that downloads the *Wine-Reviews dataset* [2] and returns it as a pandas `DataFrame`. Write a function that extracts the wine's year or vintage from the `title` column. Add that as a new column to your data.

c) Plot a histogram for each column that visualizes the distribution of its values. If the column contains numbers, you can group them into bins as usual, for other types of data count the occurrences of each unique value and order them on the x-axis by descending count. You should also mention how many of the values are missing for each column.

d) Compute some statistics for the columns that contain numerical values. These should include the minimal, maximal and average values. You can also include other useful statistics such as standard deviation and median if you feel that they provide some valuable insight.

e) Implement a transformation that takes a raw data point from the wine reviews dataset and outputs a vector of real numbers. Include a paragraph in the report that explains how each column is transformed and why you chose to implement it that way. Think thoroughly about how you want to handle missing values in any of the columns.

f) Assume you have pre-processed your data according to your transformation. What happens when you are supposed to predict a new data point and the type of wine is new, i.e., it was not contained within your training set? Explain how your transformation handles this case. Can you think of a better solution?

2 Regression

Since we now have a function that transforms a data point to a real-valued vector, we can finally attempt to apply ridge regression on the wine reviews dataset. In fact, our ultimate goal will be to predict the rating of a wine, i.e., the *points* column, from the rest of each data point. We recommend you to try out tasks a) and b) first.

Regression with bias The version of ridge regression defined in eq. (1) will always produce a line or, more generally, a hyperplane that contains the point $\mathbf{0} \in \mathbb{R}^{d+1}$, since the point $\mathbf{x} = \mathbf{0} \in \mathbb{R}^d$ is always mapped to $y = 0$.

If we also want to learn other hyperplanes that don't necessarily contain the origin, we can modify our model of the relationship between input and output to include a learnable

[2] `https://www.kaggle.com/zynicide/wine-reviews`

bias term b: $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$. Now the input $\mathbf{0}$ will be mapped to b instead. Based on that, we can formulate a new version of ridge regression that includes the bias term:

$$\mathbf{w}^*, b^* = \underset{\mathbf{w}\in\mathbb{R}^d, b\in\mathbb{R}}{\arg\min} \|\mathbf{w}\|^2 + b^2 + C\sum_{i=1}^{n}(\mathbf{w}^T\mathbf{x}_i + b - y_i)^2, \tag{2}$$

You now have adequate information to carry out tasks c), d) and e).

Interpretation via subset selection In linear regression, we can obtain a direct interpretation of the model by looking at the coefficients w_j for all features in $\mathbf{x}$. The larger the value w_j, the more impact the feature j is having on the prediction. More precisely, the coefficient of a feature tells you how much the predicted value is expected to increase when the feature increases, keeping all other features as constant. It is important to note that features can also be related to each other in a regression model. Thus, the coefficients will change when a related feature is removed or added to the regression model.

Another way to determine a subset of features that exhibits the strongest effects is *subset selection.* With subset selection we retain only a subset of the features—using the ridge regression estimates—and eliminate the rest of the features from the model. One instance of subset selection is *Forward-Stepwise Selection.* Here, we start with a bias equal to the mean of the labels, and then sequentially add a feature into the model that decreases the regression loss the most (see task f)).

Ridge regression on non-linear data Ridge regression does not really work well in the case when the relation between input and target variables is not linear for a dataset (see task g)). However, it is possible to circumvent this problem by pre-processing the input data in a clever way. Formally, we apply a function $\phi\colon \mathbb{R} \to \mathbb{R}^d$, where $d > 1$, to each input point before we then apply ridge regression on the transformed data. Thus, the prediction function then becomes

$$f(x) = \mathbf{w}^T\phi(x) + b = \sum_{j=1}^{d} w_j\phi(x)_j + b. \tag{3}$$

If we choose $\phi(x)$ to be non-linear in x, f will also be non-linear in x, meaning that we can also learn non-linear prediction functions with this trick.

2.1 Tasks

a) The code template includes a class `RidgeRegression`. Please modify this class (especially the `fit` and `predict` methods) such that it actually performs ridge regression as defined in eq. (1). Don't use any other libraries except for Numpy in your implementation.

b) Now we want to investigate the relationship between columns in our dataset. More specifically, we are interested in the "points" column. Therefore, you should execute the following steps for each of the other columns *after* it has been transformed into a vector of real numbers:

   1. If the column is represented by a d-dimensional vector with $d > 1$, you'll need to come up with a projection $p\colon \mathbb{R}^d \to \mathbb{R}$ first. A simple instance of such a projection would be to simply take the j-th element of the vector, i.e., $p(\mathbf{v}) = v_j$ or, alternatively, you can also use a more sophisticated dimensionality reduction method like *PCA* [3].

   2. Draw a scatter plot that shows the projected value of the column, i.e., $p(\mathbf{v})$, on the x-axis and the corresponding value of the points column on the y-axis.

   3. After that, use the projected data as the input and the points column as the target for your implementation of ridge regression from item a). This will produce a linear function in two dimensions, i.e., a line, that you should also include in each of the scatter plots.

c) Create a new class `RidgeRegressionBias` that uses the **unchanged** class from item a) as a black box to implement ridge regression with bias, as defined in eq. (2). Repeat the plots from item b) with the new implementation.

d) Now apply the improved ridge regression on the non-projected version of the entire wine reviews data. Your goal is to predict the *points* column from the values of the other columns. Estimate the performance of your regressor using 5-fold cross validation and the *mean squared error* (MSE) as the performance metric. On a dataset D it is defined as

$$\mathrm{MSE}(f, D) := \frac{1}{|D|} \sum_{(\mathbf{x},y)\in D} (f(x) - y)^2.$$

   You should achieve an MSE that is smaller than 6.3. If that is not the case, you need to adapt your pre-processing routines and/or the parameter C of the ridge regression algorithm.

e) Regularized methods like ridge regression exhibit a rather peculiar behaviour: Each component w_j of the weight vector is forced to be small in magnitude by the regularization term. If the corresponding input feature x_j varies only marginally compared to the other features, it can never contribute much to the final prediction, even if it would be important for the problem at hand. On the other hand, if the corresponding feature takes values in a rather big interval, its effect on the final prediction will be disproportionately large.

   Think of a way to solve this problem, apply it to your algorithm from item d) and compare the performance.

[3] `https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html`

f) Use the *Forward-Stepwise Selection* method—as described above—to obtain the best 5-subset of features that exhibits the strongest effect on MSE score over the entire dataset. Report the MSE score for each of the $k \in 0, \ldots, 5$ selected features in each iteration of the process.

g) Your git repository contains a simple dataset in the files `task2/data/data_2g.csv` and `task2/data/data_2g_labels.csv` consisting of scalar inputs $\{x_1, \ldots, x_n\} \subset \mathbb{R}$ and regression targets $\{y_1, \ldots, y_n\} \subset \mathbb{R}$

1. Plot the dataset and think about how it might have been generated.
2. Apply your implementation of ridge regression from item c) on this data, report the MSE and draw the resulting line in the same plot as the data.
3. As per eq. (3), find a suitable ϕ s.t. ridge regression on the dataset transformed with this function will yield close to zero MSE (smaller than **0.0001**) on the dataset mentioned above.
4. Give an explicit formula for the prediction function f in eq. (3) after inserting your transformation ϕ, i.e., it should only depend on x and the parameters $\mathbf{w}$ and b. Draw the resulting function in the same plot from before.

2.2 Competition (15 Points)

This competition will be about minimizing the MSE for task d). Adapt your transformations from ex. 1 and the hyperparameters for the regression algorithm to get the best possible cross validation MSE score. If you are unsure whether your implementation of ridge regression is correct or fast enough, you can also use the classes `sklearn.linear_model.Ridge` or `sklearn.svm.LinearSVR` from the *scikit-learn* [4] library for this competition. `Ridge` is just their implementation of ridge regression and should correspond more or less to what you have implemented yourselves. `LinearSVR` on the other hand solves the following optimisation problem:

$$\mathbf{w}^*, b^* = \underset{\mathbf{w}\in\mathbb{R}^d, b\in\mathbb{R}}{\arg\min} \|\mathbf{w}\|^2 + b^2 + C\sum_{i=1}^{n} \max(|\mathbf{w}^T\mathbf{x}_i + b - y_i| - \epsilon, 0). \tag{4}$$

It has a few more parameters that you can play around with, for example the ϵ in eq. (4), but unlike for ridge regression, a closed form solution does not exist. In any case, please mention the class and values for all parameters that were used in your regression algorithm and your pre-processing in the report.

We will instantiate the model with those parameters, transform the data, run 5-fold cross validation 10 times and average the result. To make things as fair as possible, we will determine the random folds for each of the 10 runs first and then perform the actual

[4] `https://scikit-learn.org/stable/index.html`

training and testing for all submissions on the exact same folds. The team that achieves the smallest MSE according to this procedure wins the competition.

To make our life easier, you need to implement the functions in `competition.py` as detailed in the comments. We will use those to evaluate your model and pre-processing routine.

Note: *Remember that you are not allowed to use any test data during training. If you perform cross validation, make sure that the left-out data is not used for any part of your algorithm, including your pre-processing, data transformations, and model. Also, you are not allowed to use the test labels for anything apart from comparing them with your prediction to evaluate the model's performance.*

2.3 Competition (10 Points)

This competition will be very similar to task 2 g). Indeed, the folder `task2/data` contains a dataset in the files `data_competition.csv` and `data_competition_labels.csv`, which is a little more complex than the dataset from task g). Your goal, however, remains the same: find a good mapping ϕ that will yield the lowest possible MSE on this dataset. Please implement your transformation in the function `phi_competition` in the file `competition.py`. Groups will be ranked according to the MSE that their transformation achieves when we run ridge regression on the entire transformed dataset (we will set $C = 10^9$).