Introduction to Machine Learning (67577)

Exercise 1
Mathematical Background & Linear Regression

Second Semester, 2024

Contents

1 Submission Instructions

Please make sure to follow the general submission instructions available on the course website. In addition, for the following assignment, submit a single `ex1_ID.zip` file containing:
- An `Answers.pdf` file with the answers for all theoretical questions, as well as the plotted graphs and explanations you are asked in the practical part.
- The following python files:
    1. `linear_regression.py`
    2. `polynomial_fitting.py`
    3. `house_price_prediction.py`
    4. `city_temperature_prediction.py`
- A single requirements.txt file, containing all the additional packages you import in your implementation. Each package needs to be written in a separate line.

Additional guidelines:
- You can add helper functions to any of the above files, but do not submit any additional files.
- You can assume that the input is in the expected format as described in the function documentation.
- You can import any additional packages to support your implementation, but you cannot import functions that implement exactly what we ask you. For example, if we ask you to implement a function that calculates the mean square error loss, you *cannot* import the `sklearn.metrics.mean_squared_error` function.
- The `ex1_ID.zip` file must be submitted in the designated Moodle activity prior to the date specified *in the activity*.
    - Plots and explanations included as separate files will be considered as not provided. You need to provide it as part of the `Answers.pdf` file.

2 Theoretical Part

2.1 Mathematical Background

2.1.1 Linear Algebra

Based on Recitation 1

1. Calculate the SVD of the following matrix A. That is, find the matrices $U, \Sigma, V^\top$ where U, V are orthogonal matrices and Σ diagonal.

$$A = \begin{bmatrix} 1 & 1 & 0 \\ 1 & -1 & 2 \end{bmatrix}$$

   Recall, that to find the SVD of A we can calculate $A^\top A$ to deduce V, Σ and then calculate $AA^\top$ to deduce U. Equivalently, once we deduced V, Σ we can fine U using the equality $AV = U\Sigma$.

2. Show that the outer product of two vectors $\mathbf{v} \in \mathbb{R}^n, \mathbf{u} \in \mathbb{R}^m$, which is denoted by $\mathbf{v} \otimes \mathbf{u}$ or $\mathbf{v} \cdot \mathbf{u}^\top$ is a matrix $A \in \mathbb{R}^{n \times m}$ with $rank(A) = 1$. That is, show that all rows (or columns) in A are linearly dependent.

3. Show that for any otrthonormal basis $(\mathbf{u}_1, ..., \mathbf{u}_n)$ and any aribtrary vector $\mathbf{x} \in \mathbb{R}_n$ such that $\mathbf{x} = \sum_{i=1}^n a_i \cdot \mathbf{u}_i$, it holds that $a_i = \langle \mathbf{x}, \mathbf{u}_i \rangle$ for any $i \in [1, n]$. That is, show that the i'th coefficient of representing $\mathbf{x}$ in the basis $(\mathbf{u}_1, ..., \mathbf{u}_n)$, is the inner product between $\mathbf{x}$ and $\mathbf{u}_i$.

4. In (a-e) you will prove some properties of orthogonal projection matrices seen in recitation 1. Let $V \subseteq \mathbb{R}^d$, $dim(V) = k$ and let $\mathbf{v}_1, \ldots, \mathbf{v}_k$ be an orthonormal basis of V. So the orthogonal projection matrix onto V is defined as $P = \sum_{i=1}^{k} \mathbf{v}_i \mathbf{v}_i^{\top}$ (notice this is an outer product).
   (a) Show that P is symmetric.
   (b) Prove that 0 and 1 are the eigenvalues of P and show that $\mathbf{v}_1, \ldots, \mathbf{v}_k$ are the eigenvectors corresponding the eigenvalue 1.
   (c) Show that $\forall \mathbf{v} \in V \ P\mathbf{v} = \mathbf{v}$.
   (d) Prove that $P^2 = P$.
   (e) Prove that $(I - P)P = 0$.

2.1.2   Multivariate Calculus

Based on Recitation 2

5. Use the chain rule to calculate the gradient of $h(\sigma) = \frac{1}{2}\|f(\sigma) - y\|^2$, where $\sigma \in \mathbb{R}^d$ and f is some arbitrary function from $\mathbb{R}^d$ to $\mathbb{R}^d$.
6. In recitation 2 we saw the softmax function $S : \mathbb{R}^k \to [0,1]^k$, which is defined as follows:

$$S(\mathbf{x})_j = \frac{e^{x_j}}{\sum_{l=1}^{k} e^{x_l}}$$

This function takes an input vector $x \in \mathbb{R}^d$ and outputs a probability vector (non-negative entries that sum up to 1), corresponding to the weight of original entries of x.
Question: Calculate the Jacobian of the softmax function S.

2.2   Linear Regression

Based on Lecture 1 and Recitation 3

Let $\mathbf{X}$ be the input matrix of a linear regression problem with m rows (samples) and d columns (variables/features). Let $\mathbf{y} \in \mathbb{R}^m$ be the response vector corresponding the samples in $\mathbf{X}$.

2.2.1   Solutions of The Normal Equations

1. In (a-d) you will incrementally prove several important properties regarding the solutions of the normal equations.
   (a) Prove that: $Ker(\mathbf{X}) = Ker(\mathbf{X}^{\top}\mathbf{X})$
   (b) Prove that for a square matrix A: $Im(A^{\top}) = Ker(A)^{\perp}$
   (c) Let $\mathbf{y} = \mathbf{X}\mathbf{w}$ be a non-homogeneous system of linear equations. Assume that $\mathbf{X}$ is square and not invertible. Show that the system has ∞ solutions $\Leftrightarrow y \perp Ker(\mathbf{X}^{\top})$.
   (d) Consider the (normal) linear system $\mathbf{X}^{\top}\mathbf{X}\mathbf{w} = \mathbf{X}^{\top}\mathbf{y}$. Using what you have proved above prove that the normal equations can only have a unique solution (if $\mathbf{X}^{\top}\mathbf{X}$ is invertible) or infinitely many solutions (otherwise).

2.2.2   Least Squares

Given a sample $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$, the ERM rule for linear regression w.r.t. the squared loss is

$$\hat{\mathbf{w}} \in \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^d} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$$

where $\mathbf{X}$ is the input matrix of the linear regression with rows as samples and y the vector of responses. Let $\mathbf{X} = U\Sigma V^{\top}$ be the SVD of $\mathbf{X}$, where U is a $m \times m$ orthonormal matrix, Σ is a $m \times d$ diagonal matrix, and V is an $d \times d$ orthonormal matrix. Let $\sigma_i = \Sigma_{i,i}$ and note that only the non-zero σ_i-s are

singular values of $\mathbf{X}$. Recall that the pseudoinverse of $\mathbf{X}$ is defined by $\mathbf{X}^\dagger = V\Sigma^\dagger U^\top$ where $\Sigma^\dagger$ is an $d \times m$ diagonal matrix, such that

$$\Sigma^\dagger_{i,i} = \begin{cases} \sigma_i^{-1} & \sigma_i \neq 0 \\ 0 & \sigma_i = 0 \end{cases}$$

2. Assuming that $\mathbf{X}^\top\mathbf{X}$ is invertible, show that the general solution we derived in recitation 3 $\left(\mathbf{X}^\dagger\mathbf{y}\right)$ equals to the solution you have seen in lecture 1 $\left(\left[\mathbf{X}^\top\mathbf{X}\right]^{-1}\mathbf{X}^\top\mathbf{y}\right)$.

3 Practical Part

(R) In this section, you'll work on implementing code that may be new to many of you. It's not expected that you know these concepts beforehand; part of this exercise is learning as you go.

3.1 Fitting A Linear Regression Model

In this question you will have to deal with a real-world dataset and fit a linear regression model to it. As data is noisy, messy and difficult, take the time to "play" and get familiar with it. The data is provided in `house_prices.csv`.

Implement the following:
1. In the `linear_regression.py` file, we provide a template for the `LinearRegression` class. Follow the provided documentation, and implement `LinearRegression` class API according to the class and methods documentation.

Then, implement code of following questions in the `exercise2/house_price_prediction.py` file:
2. Randomly split the data frame to a training set (75%) and test set (25%). To make sure your training data does not change every time you re-run the code, set a `random_seed` or `random_state` parameter.
3. Implement the `preprocess_train` function. The function receives a loaded `pandas.DataFrame` object of the features matrix, and a `pandas.Series` object of the respone vector, and returns them after preprocessing. Make sure that if you remove a sample, you remove it properly from both the observation matrix as well as from the response vector.
   Explore the data (some information can be found on Kaggle) and perform any necessary preprocessing (such as but not limited to):
   - What sort of values are valid for different types of features? Can the year that the house was built be later than the year that the house was renovated? Can a living room size be too small?
   - Are there any additional features that might be beneficial for predicting the house price and that can be derived from existing features?

   In the `Answers.pdf` file, describe in details the analysis process that lead you to the decisions of:
   - Which features to keep and which not?
   - Which features are categorical how how did you treat them?

- What other features did you design and what is the logic behind creating them?
- How did you treat invalid/missing values?
- Explain any additional processing performed on the data.

(R) Why do we want the split before preprocessing? We aim to construct a model capable of predicting on new samples it has not encountered previously, a concept known as generalization. To evaluate the success of our model, we require the test set to remain concealed and independent from our design choices. If we were to include the test set in preprocessing tasks, such as exploring feature correlations with the response, we would risk contaminating our training process. Consequently, our assessments of our model's generalization performance would be compromised, leading to inaccurate conclusions.

4. Basics of feature selection - implement the `feature_evaluation` as specified in the documentation. This function will compute the Pearson Correlation between each of the features and the response:

$$\text{Pearson Correlation: } \rho := \frac{COV(X,Y)}{\sigma_X \sigma_Y}$$

for X being one of the features, and Y the response.

(R) Pearson Correlation: This measure captures the strength and direction of the *linear* relation between the two random variables A and B. Its values range between $[-1,1]$ where a value of 1 indicates a perfect linear relation between A and B, a value of 0 indicates no linear relation between A and B, and a value of -1 indicates a perfect linear anti-relation between A and B (i.e. $A = -B$).

Implementation clarifications:
- Run this function on the *training set* only.
- You are allowed to use functions that calculate the standard deviation and covariance, but not functions that calculate the Pearson correlation itself.
- The function does not need to display the plots on the screen, but saves them (that is, use `im.save()` and not `im.show()`)

For the `Answers.pdf`, choose two features: one that seems to be beneficial for the model and one that does not. Include in `Answers.pdf` the plots of the chosen two features, and explain how do you conclude if they are beneficial or not.

5. Implement the `preprocess_test` function. This function receives as an input only the features matrix of the *test set*, and outputs a pre-processed features matrix that corresponds with what you implemented in the `preprocess_train`. For example, if you added an additional column that indicates if the house is recently renovated, you need to add this column also in the test set, or the coefficients that you fit will not match. Note: You are not allowed to remove samples (i.e., entire rows) from the test set, but only add/remove columns or edit values in specific cells.

(R) Why are we not allowed to remove rows from the test set? Because doing so would invalidate the reported results as they would not accurately reflect the performance on the

test set. For instance, if you were asked to predict on 100 samples and chose to remove 50 of them for any reason, the accuracy reported would not provide a comprehensive understanding of how your model operates on real-world, non-hand-crafted data.

6. Fit a linear regression model over increasing percentages of the *training set*, and measure the loss over the *test set*:
   - Iterate for every percentage $p = 10\%, 11\%, ..., 100\%$ of the training set.
   - Sample $p\%$ of the train set. You can use the `pandas.DataFrame.sample` function.
   - Repeat sampling, fitting and evaluating 10 times for each value of p.

   Plot the mean loss as a function of $p\%$, as well as a confidence interval of $mean(loss) \pm 2 * std(loss)$. If implementing using the `Plotly` library, see how to create the confidence interval in Chapter 2 - Linear Regression code examples.

   Add the plot to the `Answers.pdf` file and explain what is seen. Address both trends in loss and in confidence interval as function of training size.

   R In this question, our objective is to assess the impact of enlarging the training set on the accuracy of the test set. By iteratively sampling the data 10 times, we aim to enhance the reliability of our conclusions. A single sampling instance may inadequately capture the data's variability, potentially leading to skewed results. However, by conducting the sampling procedure repeatedly, we mitigate the influence of outlier samples and ensure a more reliable conclusions regarding our model.

   Implementation clarification:
   (a) Notice that when predicting on the training set, you need to be consistent with some of the preprocessing

3.2   Polynomial Fitting

In this question you are going to use linear regression model, for a problem that at first sight does not seem like linear regression. But, after input transformation, linear regression becomes applicable.

Polynomial fitting is a specialized application of linear regression, aimed at solving the task of identifying a polynomial of degree k that accurately models a given set of data.
In mathematical terms, this involves finding a function $f(x) = y$, where both x and y are real numbers $(x, y \in \mathbb{R})$, and f is a polynomial of degree k.
Here comes the "trick": This problem can also be reformulated in terms of vectors. Instead of our input to be $x \in \mathbb{R}$, we represent our input by the following vector $\hat{x} = [x^0, x^1, ..., x^k]$, so that the $i'th$ entry is x to the power of i. So now our input $\hat{x}$ is in $\mathbb{R}^k$, and polynomial fitting then becomes the task of finding the coefficients $w = [w_0, w_1, ..., w_k]$ so that the dot product $w^T \cdot \hat{x} = y$ is a different representation for the original $f(x) = y$.
This reformulation transforms the problem into a classic linear regression challenge, where the goal is to find the weight vector w that best maps the transformed input vector to the output y.

Class implementation:
   1. Implement the `PolynomialFitting` class in the `polynomial_fitting.py` file as specified

in class documentation. Avoid repeating code from the `LinearRegression` class and instead use inheritance or composition patterns. You are allowed to use the `np.vander` function.

In the following questions you will use the Daily Temperature of Major Cities dataset. Notice, that the supplied file `city_temperature.csv` is a modified subset of the dataset found on Kaggle containing only 4 countries. You will fit and analyse performance of a polynomial model over the dataset.

2. Implement the `load_data` function in the `city_tempretaure_prediction.py` file.
   - When loading the dataset remember to deal with invalid data.
   - Use the `parse_dates` argument of the `pandas.read_csv` to set the type of the 'Date' column.
   - Add a 'DayOfYear' column based on the 'Date' column. This column will be the feature to be used for the polynomial fitting.

3. Filter the dataset to contain samples only from the country of Israel. Investigate how the average daily temperature ('Temp' column) change as a function of the 'DayOfYear'.
   - Plot a scatter plot showing this relation, and color code the dots by the different years (make sure color scale is discrete and not continuous). What polynomial degree might be suitable for this data?
   - Group the samples by 'Month' (have a look at the pandas 'groupby' and 'agg' functions) and plot a bar plot showing for each month the standard deviation of the daily temperatures. Suppose you fit a polynomial model (with the correct degree) over data sampled uniformly at random from this dataset, and then use it to predict temperatures from random days across the year. Based on this graph, do you expect a model to succeed equally over all months or are there times of the year where it will perform better than on others? Explain your answer.

   Add plots and answers to the `Answers.pdf` file.

4. Returning to the full dataset (including all 4 original countries), group the samples according to 'Country' and 'Month' and calculate the average and standard deviation of the temperature. Plot a line plot of the average monthly temperature, with error bars (using the standard deviation) color coded by the country. If using `plotly.express.line` have a look at the `error_y` argument.

   Based on this graph, do all countries share a similar pattern? For which other countries is the model fitted for Israel likely to work well and for which not? Explain your answers.

5. Over the subset containing observations (i.e., samples) only from Israel perform the following:
   - Randomly split the dataset into a training set (75%) and test set (25%).
   - For every value $k \in [1, 10]$, fit a polynomial model of degree k using the training set.
   - Record the loss of the model over the test set, rounded to 2 decimal places.

   Print the test error recorded for each value of k. In addition plot a bar plot showing the test error recorded for each value of k. Based on these which value of k best fits the data? In the

case of multiple values of k achieving the same loss select the simplest model of them. Are there any other values that could be considered?

6. Fit a model over the entire subset of records from Israel using the k chosen above. Plot a bar plot showing the model's error over each of the other countries. Explain your results based on this plot and the results seen in question 3.