HW2 - ML in Healthcare

Submission Guidelines: Your GitHub must contain at least the following files:

- Dry_part.pdf (that you create)- Containing your answers to the dry questions across all of the assignment. This file should have images from your notebook if you need to refer to them or at least to have a clear reference to the image inside the notebook. When answering a question, **mention specifically the part and the section of the question you are answering.**
- Part1_and_2_2.ipynb - A Jupyter notebook that we provide for both part I and section (2.2) with your implementation.
- Part2_3.ipynb - A Jupyter notebook that we provide containing our template with your implementation for (2.3).
- manual_log_reg.py - A python file we provide containing our template with your implementation for (2.3).
- Any other files you must have to run the notebooks correctly.

In theoretical questions in all of the assignments' parts, it is better if you typed the calculations and\or your verbal explanations but you may also write them by hand (in English or Hebrew). Either way, make sure that your pdf file for submission can be easily read. An unreadable answer will not be marked (0 pt).

1 Part I: Loss minimization with gradient descent

Let $L(w_1, w_2)$ be a loss function defined as follows:

$$L(w_1, w_2) = -10(0.4cos(w_1) - w_1^2 - 0.2w_2^7 + sin(w_2))e^{(-w_1^2 - w_2^2)}$$

Assume the features and the labels (x and y) are embedded within the coefficients so you may ignore them. Notice that this loss function can have negative values. The error surface created by the loss is shown below.

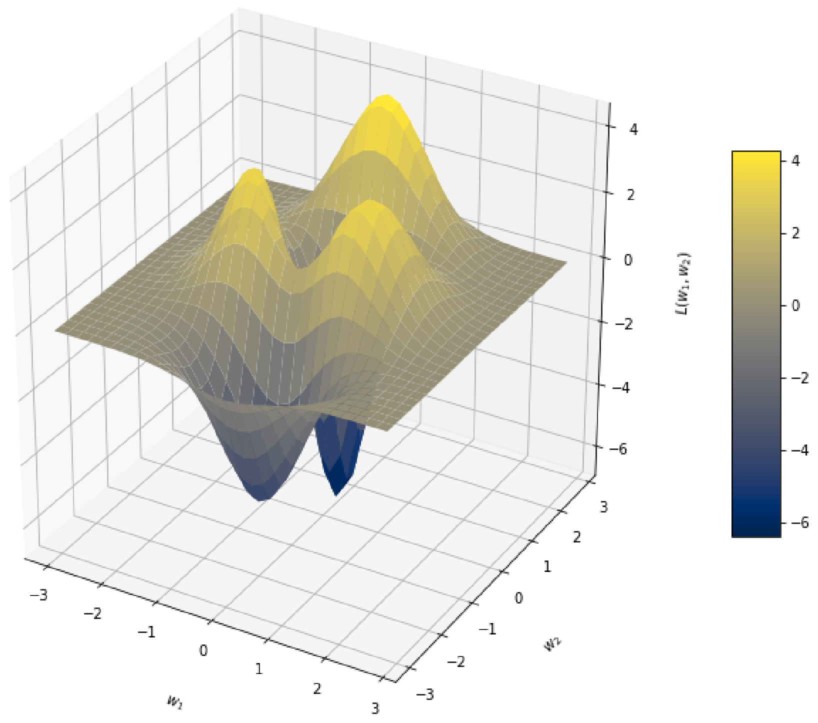

Figure 1: The loss as a function of the weights.

1. Calculate by hand $\nabla L_{w_1,w_2}$. Show the derivation and mention any derivatives' rules if used.

2. In the following subsections you should apply gradient descent update of the weights according to the instructions. Implement a function called *grad_desc* that has three positional arguments (w_{01}, w_{02}, η) which are the initial weights and the learning rate. The algorithm should stop either when the number of iterations is over 300 or when $||\overrightarrow{\mathrm{w}}_{n+1} - \overrightarrow{\mathrm{w}}_n||_2 < 10^{-4}$. Implement the algorithm so it will stop at whatever condition **it reaches first**. The function should also plot the loss as a function of iterations, print the "optimal" $\overrightarrow{\mathrm{w}}$ that you got and the final value of the loss at the optimal weight. Run your function with the following:

   - Initialize $[w_1, w_2]$ to $[1, 1]$ and set η to 0.001.
   - Initialize $[w_1, w_2]$ to $[0, -2.2]$ and set η to 0.001.
   - Initialize $[w_1, w_2]$ to $[1, 1]$ and set η to 0.1.
   - Initialize $[w_1, w_2]$ to $[2, 2]$ and set η to 0.001.

3. Explain the differences between the four runs and their causes. Address issues such as convergence and convergence rate, local minima vs. global minima etc.

4. Add an L_2 regularization term to the loss and recalculate by hand $\nabla L_{w_1,w_2}$. You can use the result you got in section (1) but mention a justification for that.

5. Modify your function by using a keyword argument named *lmbda* (not *lambda*) that has the default value of 0. Initialize again $[w_1, w_2]$ to $[1, 1]$ and η to 0.001. Run your function with $\lambda = 0.01$ and $\lambda = 100$. Compare the results with the results for the same initialized weights in section (2). Did you expect to have these final weights when using regularization?

2 Part II: Binary classifiers

Retinopathy is an eye condition that causes changes to the blood vessels in the part of the eye called the "retina". This often leads to vision deterioration or blindness. Diabetic patients are known to be at high risk for a special type of retinopathy which is called "Diabetic Retinopahty" (DR). In this assignment, you will receive a dataset of 6,000 diabetic patients with four risk factor features and an outcome - high risk of DR (whether or not a patient is at high risk of DR) which is based on a recent risk model. Your mission will be to predict whether or not a patient has high risk for developing DR using binary classification methods.

2.1 Dataset

The features (X) include the following fields:

- Age: [years]
- Systolic_BP: Systolic blood pressure [mmHg]
- Diastolic_BP: Diastolic blood pressure [mmHg]
- Cholesterol: [mg/DL]
- The target (y) is an indicator of whether or not a patient has high risk for developing DR (y = 1 the patient has DR).

2.2 Linear vs. nonlinear classifiers

In this part we provide you with a dataset ("X_data.csv", "y_data.csv") You are required to implement and present your results in Part1_and_2_2.ipynb in the adequate part. Mark the begining of the cells with the adequate section. In this assignment, you will do the following:

- Explore the data provided.
- Train linear and non-linear classifiers.
- Optimize a model with k-fold cross validation.
- Evaluate your model performance with appropriate metrics.
- Present a 2D visualization of multi-featured data.
- Use feature selection tools.

You may only use your bm-336546 environment. Packages outside of it are not allowed!

2.2.1 Instructions

1. Perform a test-train split of 20% test and preprocess the data. You should use preprocessing methods that you saw in the tutorial. Add at least one new method from the literature that you think is relevant.
2. Provide detailed visualization and exploration of the data. You should at least include:
   (a) An analysis to show that the distribution of the features is similar between test and train. Show also the ratio of labels in train and test.
      i. What issues could an imbalance of labels **in the training set** rise? Search in the scientific literature at least one possible solution. Explain how this solution should help.
      ii. What labels imbalance **between train and test** cause? What is the solution we gave in the tutorial?
   (b) Plots to show the relationship between feature and label and additional plots that help to explore the data (similar to the ones done in the tutorials).
   (c) State any insights you have:
      i. Was there anything unexpected?
      ii. Are there any features that you feel will be particularly important to your model? Explain why. Base your answer on a valid reference.
3. Choose, build and optimize Machine Learning Models:

(a) Use 5k cross fold validation and tune the hyperparameters to get the highest AUROC (aka AUC). For doing so, apply *GridSearchCV* and *PipeLine*. Set *random_state* of all classifiers to 336546. The hyperparmeter grid should not have more than 5 values for every hyperparmeter. The hyperparameters to tune are:
   - $[C, penalty]$ for logistic regression. Set the solver to be *'saga'*.
   - $[C, kernel]$ for SVM. Set *probability=True* so you can have AUC.
   - $[criterion, max\_depth, max\_features]$ for Random Forest.

(b) Train all of the models with the chosen hyperparmeters on the full training set. You may skip this and move on to (c) by using correctly *GridSearchCV* from the previous section.

(c) Report the evaluation metrics of AUC, F1, ACC on the testset for the three models.

(d) Which performed the best on the testset? Linear or nonlinear models? Compare all of them to a naive classifier. *Notice that the difference in performance between the models can be subtle.

4. Feature Selection:

(a) As seen previously, a Random Forest classifier can be used to explore feature importance. Train a Random Forest on your data.
   i. Report AUC, F1, ACC for both train and test set for the Random Forest model.
   ii. Is there a performance difference between train and test? Why? If there is a difference, what can you do to reduce it?
   iii. What is the advantage of Random Forest model compared to SVM model?
   iv. What are the 2 most important features according to the random forest? What are the two most important according to logistic regression? Does this match up with the feature exploration you did?

2.3 Logistic regression from scratch

In this section, you will implement a class which is a basic form of sklearn *LogisticRegression* and apply its methods on the same data as before. You should use this class to run Part2_3.ipynb provided to you. Implement your code wherever it is asked to in the manual_log_reg.py file. Notice the following:

- In the *log_loss* implementation you should add ϵ to the probabilities p and $1-p$ for stability. Set $\epsilon = 1e-5$. The function should return the mean cross-entropy loss according to the inputs.

- In *fit* implementation you should use the closed form of the gradient of binary cross-entropy shown in both lecture and tutorial. Do not build any numerical derrivation function.

- *predict_proba* should return a single probability for every example and not a two element vector as in *sklearn*.

- Don't forget to implement the *sigmoid* function which is out of the class.

- In the *conf_matrix* function, the top left matrix calculation was already given to you. By understanding its action, you can relatively easily implement the remaining three values of the confusion matrix. You do not have to use it if you don't want to but do not delete this code line. The confusion matrix rows are interpreted as: "True label is 0", "True labels is 1". The columns are interpreted as: "Predicted label is 0", "Predicted label is 1".

- In order to call a method (a function in the class) from another method you will need to use "self". Look at the implemented method of "score" for example to see how we call the "predict" method.

- All of the methods can be implemented with 1-3 short line code. Efficiency won't be marked but please make sure you understand exactly what you are asked to implement.

Follow the rest of the instructions in the notebook and implement accordingly.

GOOD LUCK!