EMORY UNIVERSITY
Department of Computer Science
CS 334 Section 2 — Machine Learning
Spring 2025

Homework 3 - Predicting Survival of ICU Patients

Issued: 02/09 - Due: 02/23 at 11:59pm

With the rapid advancement of technology, hospitals are creating and collecting vast amounts of data about their patients. These data, stored in the electronic health record, have the potential to impact clinical care in a variety of ways, e.g., in identifying patients at greatest risk of an adverse outcome, in predicting the efficacy of a particular drug, or in matching patients with the best treatments.

You will be working with a subset of data collected from patients admitted to an intensive care unit at the Beth Israel Deaconess Medical Center in Boston. You will develop classification models to identify patients at high risk of in-hospital mortality. These tools could help clinicians target interventions, facilitate better management, and improve patient outcomes. **We emphasize that these data represent real patients**; they have been made available for research and educational purposes by PhysioNet.org.

Submission Instructions: The homework is due on Gradescope in **two** parts.

- **Upload PDF to HW3-Written**: Your submission may be typed or handwritten, and pages must be tagged with relevant parts on Gradescope. To help you navigate different questions of this homework, we've highlighted question parts that should be included in the write-up.
- **Submit `challenge.csv` and your code to HW3-Code&Challenge**: You may submit multiple times; only the last submission will be considered. Code is not auto-graded for this assignment. Please also include the following ***SIGNED*** *honor code statement*:

```
THIS HOMEWORK IS MY OWN WORK, WRITTEN WITHOUT COPYING FROM OTHER STUDENTS
OR DIRECTLY FROM LARGE LANGUAGE MODELS SUCH AS CHATGPT.
Any collaboration or external resources have been properly acknowledged.
<add details>
/* Your_Name_Here */
```

Getting Started

Please download `HW3_SkeletonCode.zip` from Canvas. This zip file contains the full project dataset, and may take several minutes to download. It contains the following files:

Data:

- `data/files/`
- `data/labels.csv`
- `config.yaml`

Scripts:

- `hw3_main.py`
- `hw3_challenge.py`
- `helper.py`

We recommend using a local python setup for this homework.

If you prefer to use Google Colab, use the notebook here: Open in Colab

Just a heads-up: the challenge question may take >5 min to run on Google Colab.

Dataset Description

This dataset contains a total of 12,000 patient admissions. Each admission is associated with a `RecordID` and a separate csv file under `data/files/`. These csv files contain timestamped observations for several variables. Outcomes for each patient admission are listed in `data/labels.csv`, in which the first column is `RecordID`. The second column `In-hospital_mortality` contains binary labels: 1 if the patient died in the hospital, and -1 if the patient survived and was discharged. The third column `30-day_survival` also contains binary labels: 1 if the patient died within 30 days, and -1 if the patient survived past 30 days.

For questions 1-2 you will use data from the first 2,500 examples to explore modeling and hyperparameter selection techniques. For the final challenge, you may use data from all 12,000 examples. Note that <u>both</u> outcome <u>labels</u> for the last 2,000 examples have been <u>removed</u>, and we will use these examples as the held-out set for evaluating your challenge predictions.

1 Feature Extraction [30 pts]

For each patient admission, you are given data representing the first 48 hours of the ICU stay, as a csv file with three columns: `Time`, `Variable` and `Value`. Each row in the csv file represents a single observation. Each observation has an associated timestamp indicating the time of the observation relative to the start of the ICU admission, in hours and minutes. For example, a timestamp of 35:19 means that the associated observation was made 35 hours and 19 minutes after the patient was admitted to the ICU.

There are two types of variables: time-invariant and time-varying - as specified in `config.yaml`. Time-invariant variables are collected at the time the patient is admitted to the ICU. Their associated timestamps are set to 00:00 (thus they appear at the beginning of each patient's record). Unknown values are explicitly encoded as -1. On the other hand, time-series variables (e.g., heart rate) for a patient might be measured one time, many times or not at all. The value of a variable could be either numeric or categorical. See the documentation for more details regarding the meaning of each variable.

The goal of feature extraction is to transform each patient's raw data into a d-dimensional feature vector, so that we can learn an ML model. Here, we will summarize each time-varying variable by taking the mean. See Figure 1 for an illustration.

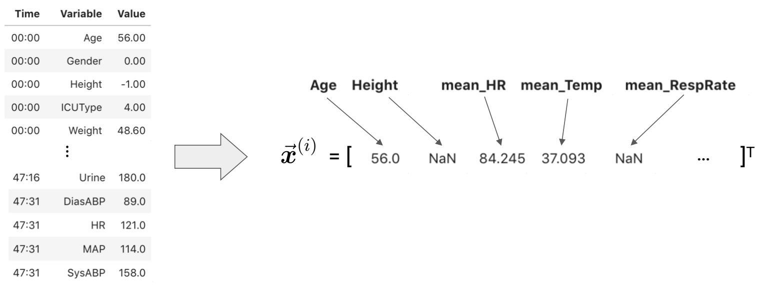

Figure 1: Transforming EHR data into a feature vector. Age and Height are time-invariant variables, each of which is encoded as a separate feature. For this patient, the feature `Age` has its original value, while the feature `Height` is `np.nan` since it is unknown (-1). HR, Temp and RespRate are time-varying variables. Here, we encode each variable by its mean. The feature `mean_HR` contains the mean heart rate, whereas `mean_RespeRate` is `np.nan` because no respiratory rate was recorded for this patient.

(a) Implement `generate_feature_vector(df)` [4 pts]. The input to this function is a `pd.DataFrame` containing the data for a single patient admission. The output is a python dictionary object, corresponding to a d-dimensional feature vector for this patient (with missing values). The keys of this dictionary are feature names, and the values are the corresponding feature values.

For the time-invariant variables, use the raw values. Replace unknown observations (-1) with undefined (use `np.nan`), and name these features with the original variable names. For each time-varying variable, compute the mean of all measurements for that variable. If no measurement exists for a variable, the mean is also undefined (use `np.nan`). Name these features as `mean_{Variable}` for each variable. For example, the variable HR would correspond to the feature with name `mean_HR`.

Also answer the following questions (no implementation required):

i) [2 pts] Read the documentation on the variable `ICUType`, and reflect on the current feature representation of this variable. What does such a representation imply, when using a linear classifier? How else might you represent this variable (as possibly more than one feature)?

ii) [2 pts] Here we only consider the mean of the numerical variables. What limitations are associated with this representation? What other summary statistics could be useful?

(b) Implement `impute_missing_values(X)` [2 pts]. Given a feature matrix `X` (where each row corresponds to a patient admission and each column a feature) with missing values, we consider each feature column independently. For each column, we impute the missing values by replacing it with the mean value of the observed values in that column. Hint: use `np.nanmean()` to compute the mean of an `np.array` with `np.nan` values. Answer the following question (no implementation required):

i) [2 pts] What assumptions does this approach make? How else might you handle missing data?

(c) Notice that many of these feature values lie on very different scales. Here, we will address this issue. Implement `normalize_feature_matrix(X)` [2 pts]. Given a feature matrix `X` (where each row corresponds to a patient admission and each column a feature) now without any missing values, we use the following formula to normalize each feature column x_d to have range between 0 and 1:

$$\tilde{x}_d = \frac{(x_d - \min)}{(\max - \min)}, \text{ where } \min = \min_{i=1\ldots n} x_d^{(i)}, \text{ and } \max = \max_{i=1\ldots n} x_d^{(i)}$$

i) [2 pts] Why might it be useful to scale the features in this way?

(d) Review the implementation of `get_train_test_split()`. This helper function uses the three functions you implemented. First, it generates a feature vector for each patient, then aggregates them into a feature matrix (features are sorted alphabetically by name), and lastly performs imputation and normalization with respect to the population. After all these steps, it splits the data into 80% train and 20% test. Report

i) [2 pts] the dimensionality d of feature vector

ii) [12 pts] the *average* feature vector, and corresponding feature names of the training set `X_train`

2 Hyperparameter and Model Selection [50 pts]

The `get_train_test_split()` function calls what you implemented from part 1 and transforms the raw data into a feature matrix and label vector, and then splits the data to training and test sets. Training data `X_train, y_train` and test data `X_test, y_test` have already been loaded in for you. **You will use these data for all of parts of question 2.** You should only have 2,000 training examples and 500 test.

Let's learn a classifier to separate the *training* data into two classes, with binary labels in $\{-1, 1\}$, where 1 means that the patient died in hospital, and -1 means the patient recovered and survived. We will be using the `sklearn.linear_model.LogisticRegression` class and its three methods: `fit(X,y)`, `predict(X)` and `decision_function(X)`. For consistency with our results, please use the `get_classifier()` helper function to instantiate your classifier. As discussed in lecture, logistic regression has hyperparameters that must be set by the user. We will select hyperparameters that lead to the best mean performance in a 5-fold cross-validation (CV) on training data. The result of hyperparameter selection often depends on the choice of performance measure.

2.1 Hyperparameter Selection for L2-Regularized Logistic Regression [40 pts]

(a) Implement helper function `performance(clf_trained, X_test, y_test, metric='accuracy')` to calculate performance metrics given a trained classifier and a test set. We'll consider the following metrics: Accuracy, Precision, Sensitivity, Specificity, F1-Score, AUROC, AUPRC. Some of them can be found under `sklearn.metrics`, feel free to use them directly. For example, you may use `sklearn.metrics.roc_auc_score` to calculate AUROC. For others, you may calculate them using the output of `sklearn.metrics.confusion_matrix`. We recommend setting `labels=[-1,1]` for consistent ordering of the confusion matrix output. Note that for F1-scores, a divide-by-zero warning be raised. Consider how this metric is calculated by reviewing the relevant documentation.

(b) Implement `cv_performance(clf, X, y, metric='accuracy', k=5)` following the specification. The function returns the mean k-fold CV performance for the performance metric passed into the function. The default metric is `'accuracy'`, however your function should work for all metrics listed above (by calling `performance()` function). You may use `sklearn.model_selection.StratifiedKFold` to create folds, and `fit(X,y)` and `decision_function(X)` methods to train models and generate predictions. For this HW, do not shuffle the points (i.e., do not set `shuffle=True` for `StratifiedKFold`).

[2 pts] Stratified folds help to keep the class proportions (ratio of positive to negative labels) roughly the same across folds. Briefly explain why this might be beneficial.

(c) Review the implementation of `select_C(X, y, penalty, metric, k, C_range=[])`.
Given a range of C values, this function sweeps over each C in `C_range`, calculates the k-fold CV performance (on the given dataset `X,y`) for the given `metric` of a logistic regression classifier with each C value, and returns the best C. This is known as a "grid search".

[12 pts] Then, using the training data from part 1, call `select_C()` to find the best C of a L2-regularized logistic regression based on a 5-fold cross-validation, with respect to each performance measure. Consider `C_range` in powers of 10 between 10^{-3} and 10^{3}. Report your results in the table below.

[4 pts] Describe how the 5-fold CV performance varies with C for each performance measure. To train a final model, which performance measure would you optimize for when choosing C? Explain.

part (c) table

Performance Measures	Best C	CV Performance
Accuracy		
Precision		
Sensitivity		
Specificity		
F1-Score		
AUROC		
AUPRC		

part (d) table

Chosen $C = ?$	
Performance Measures	**Test Performance**
Accuracy	
Precision	
Sensitvity	
Specificity	
F1-Score	
AUROC	
AUPRC	

(d) [8 pts] Take your chosen performance measure, select the best C value that maximizes that measure, and then create a L2-regularized logistic regression classifier with this best C. Re-train this classifier on the entirety of training data `X_train, y_train`. Report the performance of this classifier on the test data `X_test, y_test` in the table above.

(e) [6 pts] Finish the implementation of `plot_coefficients(X, y, penalty, C_range)` that finds the L0-norm of $\vec{\theta}$ for each value of C. The L0-norm counts the number of nonzero entries of a vector: for $\vec{\theta} \in \mathbb{R}^d$, $\|\vec{\theta}\|_0 = \sum_{j=1}^{d} \mathbb{1}(\theta_j \neq 0)$. Here, $\mathbb{1}(a)$ is the indicator function that returns 1 if a is true and 0 otherwise. You may use `np.linalg.norm` or `scipy.linalg.norm` to calculate the L0-norm, or you can implement it yourself. You may wish to dig into the documentation page of `LogisticRegression` to figure out how to extract the coefficient vector $\vec{\theta}$.

Once you complete this function, calling it should generate a plot of L0-norms $\|\vec{\theta}\|_0$ against C values and save it to a file. Include the plot and describe any interesting trends you observe.

(f) [8 pts] Recall that each coefficient θ_j in $\vec{\theta}$ is associated with a feature. The more positive a coefficient is, the more that feature contributes to a positive label. Similarly, the more negative a coefficient is, the more that feature contributes to a negative label. By examining these coefficients, we can start to generate hypotheses for which features are associated with higher (or lower) risk of in-hospital mortality.

Using $C = 1.0$ (for consistency with our results), train L2-penalized logistic regression on `X_train, y_train` and find the 4 most positive coefficients and the 4 most negative coefficients of $\bar{\theta}$ and report the corresponding feature names. Think: do these features make sense (no need to answer this)?

Positive Coefficient	Feature Name

Negative Coefficient	Feature Name

2.2 Hyperparameter Selection for L1-Regularized Logistic Regression [10 pts]

You will now explore the use of a different penalty (i.e., regularization term). In particular, we will consider logistic regression with L1-penalty which corresponds to the following optimization problem. You should be able use the functions you implemented above but passing in `penalty='l1'` instead.

$$\min_{\bar{\theta},b} \|\vec{\theta}\|_1 + C \sum_{i=1}^{N} \text{loss}_{\text{log}} \left(y^{(i)} (\vec{\theta} \cdot \vec{x}^{(i)} + b) \right)$$

(a) [5 pts] Follow what you did in 2.1(c) and 2.1(d), repeat this for L1-regularized logistic regression using AUROC as the metric. Report the best C, CV performance, and test performance.

(b) [5 pts] Similar to 2.1(e), plot the L0-norm of the parameter vector $\vec{\theta}$ against C. Include the plot and describe any interesting trends you observe. Compared to Similar to 2.1(e), besides any changes in model performance, what effect does the L1 penalty have on the classifier's coefficients?

3 Challenge [20 pts]

In the previous problems, we explored a basic pipeline for extracting features and training a binary classifier. Now, a challenge: you will use all 10,000 patients to train a binary classifier to predict ***30-day mortality***, i.e., whether a patient will survive for at least 30 days post admission to the ICU. You can call function `get_challenge_data()` to obtain the training set `X_challenge, y_challenge` and a test set `X_heldout` with no labels. Note that the class balance in this training set approximately matches the class balance in the held-out set. Also note that, given the size of the dataset, training may take several minutes.

Your grade for this question will be assessed by the following two components:

1. Effort [10 pts]: We will evaluate how much effort you have put in to attempt this challenge based on your write-up and code. **Ensure that both are present.** Your write-up should include your methodology as well as your classifier's 2×2 confusion matrix on the training data `X_challenge, y_challenge`. Specifically, you should discuss the choices you made when designing the following components of your classifier:
   - Feature engineering
   - Hyperparameter selection
   - Model selection
   - Any techniques you used that go above and beyond current course material
2. Prediction Quality [10 pts]: We will evaluate your predictions using AUROC and F1-score. You'll need to save the output of your classifier into a `csv` file using the provided helper function `generate_challenge_labels(y_label, y_score)`. Include both the binary predictions (output of `clf.predict`) and the predicted probabilities (output of `clf.predict_proba(X)[:,1]`). You may use the function `test_challenge_output` to check your output format.

While attempting this challenge, we encourage you to apply what you have learned about linear classifiers, regularization, and hyperparameter selection. You may consider the following extensions:

1. **Try different feature engineering methods**. The features we have used so far are rather simplistic. To extract more features from the raw data, you can:
   - Treat numerical and categorical variables differently
   - Calculate other summary statistics for numerical variables
   - Break up the 48 hours into sub-periods (e.g., two 24-hour windows)
   - Use only a subset of the observed data
2. **Incorporate additional preprocessing steps**. In particular, you can:
   - Handle missing values differently
   - Other feature scaling and transformation techniques
3. **Use different loss functions or regularizations**. You may use any class in `sklearn.linear_model` and `sklearn.svm`. For example, `SGDClassifier(loss='hinge', penalty='elasticnet')`. You are encouraged to consult the documentation for class definitions and possible input arguments.