Faculty of Sciences

Faculty of Mathematics, Statistics, and Computer Science

NLP CA3

Part 1: Gamma Regression

Generalized Linear Models (GLMs) are powerful tools for statistical modeling. They extend linear regression to response variables that follow non-normal distributions by assuming the response variable belongs to the exponential family of distributions. In this framework, the expected value of the response variable is related to a linear combination of input covariates through a suitable link function.

In other words, GLMs allow us to model diverse types of data—such as binary outcomes, count data, or positive continuous variables—by selecting appropriate distributions (e.g., binomial, Poisson, or gamma) and link functions (e.g., logit, log, identity). This flexibility makes GLMs highly useful for applied modeling tasks where assumptions of ordinary linear regression do not hold.

Logistic Regression: A Simple Case of GLM

Suppose we are dealing with a binary classification problem. In this case, the response variable $Y \in \{0, 1\}$ is assumed to follow a Bernoulli distribution:

$$Y \sim \text{Bernoulli}(p)$$

In logistic regression, which is a special case of GLM, we model the probability p as a function of the input features $\mathbf{x} \in \mathbb{R}^p$ using the sigmoid (logistic) function:

$$p = \sigma(\mathbf{x}^\top \beta) = \frac{1}{1 + e^{-\mathbf{x}^\top \beta}}$$

This means the log-odds of the outcome is modeled linearly:

$$\log\left(\frac{p}{1-p}\right) = \mathbf{x}^\top\beta$$

Let us now write the likelihood function for a dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$. The likelihood for one observation is:

$$\mathcal{L}_i = p_i^{y_i}(1-p_i)^{1-y_i}$$

So the full likelihood is:

$$\mathcal{L}(\beta) = \prod_{i=1}^n p_i^{y_i}(1-p_i)^{1-y_i}$$

Taking the negative log-likelihood (NLL), we get:

$$\text{NLL}(\beta) = -\sum_{i=1}^n \left[y_i \log p_i + (1-y_i)\log(1-p_i)\right]$$

Substituting $p_i = \sigma(\mathbf{x}_i^\top\beta)$, this becomes:

$$\text{NLL}(\beta) = \sum_{i=1}^n \log\left(1 + e^{-\mathbf{x}_i^\top\beta}\right) - y_i\mathbf{x}_i^\top\beta$$

As we can see, this is a **convex function** in β, which means we can efficiently find the global minimum using convex optimization techniques.

Gamma Regression: Modeling Right-Skewed Continuous Data

Suppose we want to model a continuous response variable Y that is strictly positive and **right-skewed**. In such cases, the assumption of **constant variance**—central to **Ordinary Least Squares (OLS)** or classical linear regression—does not hold. In fact, we often observe that the **variance increases with the mean**, which violates key assumptions of OLS.

So, what should we do?

We need to consider a continuous distribution that is defined only on $Y > 0$, allows for **skewness**, and can capture the **increasing variance**. A natural candidate is the **Gamma distribution**.

Gamma regression allows us to model these types of data properly by assuming that:

$$Y \sim \text{Gamma}(\mu, \sigma^2\mu^2)$$

where the mean μ is linked to predictors $\mathbf{x}$ through a **log link**:

$$\log(\mu_i) = \mathbf{x}_i^\top\beta$$

We need a distribution for $Y_i \geq 0$ with $\mathbb{E}(Y_i) = \mu_i$ and $\mathrm{Var}(Y_i) = \sigma^2\mu_i^2$. Such a distribution is the **Gamma distribution**, i.e.,

$$Y \sim \mathrm{Gamma}(\nu, \lambda) \quad \text{with} \quad f_Y(y) = \frac{\lambda}{\Gamma(\nu)}(\lambda y)^{\nu-1}e^{-\lambda y}, \quad y \geq 0$$

$$\Rightarrow \mathbb{E}(Y) = \frac{\nu}{\lambda} = \mu, \quad \mathrm{Var}(Y) = \frac{\nu}{\lambda^2} = \left(\frac{\nu}{\lambda}\right)^2 \cdot \frac{1}{\nu} = \mu^2 \cdot \underbrace{\frac{1}{\nu}}_{=\sigma^2}$$

In this section, we aim to learn how to model such data using **Gamma regression**, derive the **objective function**, and explore how to solve it using **optimization tools**.

Derive the Objective Function

Using the formulas provided earlier for the Gamma distribution and the log-link function, you are now asked to derive the **Negative Log-Likelihood (NLL)** for the Gamma regression model.

- Assume you have n i.i.d. samples $(x_1, y_1), \ldots, (x_n, y_n)$, where each $Y_i \sim \mathrm{Gamma}(\nu, \lambda_i)$ and $\mu_i = \mathbb{E}[Y_i] = \frac{\nu}{\lambda_i}$.
- The link function is given as:

$$\log(\mu_i) = x_i^\top \beta$$

- Your task is to:
  1. Write the full likelihood function for the observed data.
  2. Take the log of the likelihood to obtain the log-likelihood.
  3. Drop terms that do not depend on β, and write down the expression for the **Negative Log-Likelihood (NLL)** as a function of β.
- After obtaining the NLL expression, **prove that it is convex** in β. This can be done by using known convexity properties of exponential and linear functions.

Implementation

Before setting up your model, begin with an exploratory data analysis. Use visualizations to assess the structure of the dataset and determine whether linear regression is a suitable modeling approach.

- Plot a histogram or kernel density plot (KDE) of the response variable Y.

- Plot scatter plots of Y against each predictor variable in X.

Through these visualizations, look for the following:

- Is Y strictly positive and right-skewed?
- Does the variance of Y appear to increase with the mean?
- Are the relationships between Y and the predictors nonlinear or multiplicative?

Explain why Ordinary Least Squares (OLS) may not be appropriate in this context

In this part, you will implement and compare different optimization approaches to solve the Gamma regression problem using the dataset `gamma.csv` provided.

Your tasks are as follows:

1. **Model Setup**:
   - Load the dataset `gamma.csv` and define the covariate matrix X and response variable Y.
   - Use the derived Negative Log-Likelihood (NLL) function as the objective function to minimize.
2. **Implement three different optimization methods to minimize the NLL:**
   (a) **Convex Optimization with `cvxpy`** Solve the problem using the convex optimization library `cvxpy`.
   (b) **Gradient Descent (Manual Implementation)** Implement your own gradient descent optimizer to minimize the NLL. Use appropriate learning rate scheduling and stopping criteria.
   (c) **Newton-Raphson Method (Manual Implementation)** Implement Newton-Raphson optimization by computing the gradient and Hessian of the NLL function.
3. **Built-in Library Comparison**:
   - Fit a Gamma regression model using `statsmodels` with a log link function:

```
import statsmodels.api as sm
model = sm.GLM(y, X, family=sm.families.Gamma(link=sm.families.links.log()))
result = model.fit()
```

- Compare the estimated coefficients with your own implementations.

4. **Evaluation and Comparison:**

   - Use **10-fold cross-validation** to evaluate each method.
   - Compare the methods in terms of:
     - **Adjusted** R^2 (custom-computed since GLM does not directly output this)
     - **Root Mean Squared Error (RMSE)**
     - **Computation Time**

Note: Both the Gradient Descent and Newton-Raphson algorithms must be implemented manually. Do not use pre-built solvers. You may, however, use NumPy for matrix operations.

Part 2: Support Vector Machines

Support Vector Machines (SVMs) are powerful non-parametric methods widely used for classification tasks across various applications, including text categorization, image recognition, and bioinformatics. One of the key strengths of SVMs is their ability to perform well in sparse and high-dimensional feature spaces.

This robustness in high-dimensional settings is due to the fact that the SVM decision boundary depends only on a subset of the training data — the **support vectors** — rather than all the data points. As a result, SVMs are less prone to overfitting in these spaces and are effective at finding large-margin classifiers even when the number of features exceeds the number of samples.

As discussed in the lectures, training an SVM involves solving a constrained convex optimization problem. The objective in the linearly separable case is given by:

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|^2$$
$$\text{subject to} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 \quad \forall i$$

This optimization formulation seeks the hyperplane with the maximum margin that perfectly separates the two classes. The convexity of the objective ensures that any local minimum is also a global minimum, making it highly amenable to efficient optimization algorithms.

This formulation is referred to as the **Hard-Margin SVM**, because it assumes that the data is perfectly linearly separable. In this part, we will explore two other variants — the **Soft-Margin SVM** and the **Kernel SVM** — both with and without regularization.

Soft-Margin SVM

In practice, data is not always linearly separable. Go and find out why the original hard-margin formulation is not sufficient in this case. What modification is introduced in the SVM objective to allow for some misclassification?

- Find and write the optimization problem of the Soft-Margin SVM.
- What are the slack variables and what role do they play?
- What is the role of the regularization parameter C?

Kernel SVM

Now suppose the data is not linearly separable in the input space. Why might a linear hyperplane fail to classify such data? What can we do in this case?

- Go and find what is meant by the **kernel trick**.
- Derive the dual form of the SVM objective that uses a kernel function.
- What are some common kernels used in practice?
- When and why would you prefer a kernelized SVM over a linear one?

Regularized SVMs

All of the above SVM variants can include regularization to prevent overfitting. Your task:

- Investigate how the regularization term appears in each formulation.
- How does changing the regularization parameter affect the solution?
- (optional) Compare how regularization is handled in the primal and dual forms.

Implementation

In the file `svm.csv`, you are given a binary classification problem. Your task is to implement all three types of SVMs discussed earlier:

- **Hard-Margin SVM**
- **Soft-Margin SVM**
- **Kernel SVM**

For each model, you must:

1. Implement it **manually** using `cvxpy`, by formulating and solving the appropriate convex optimization problem.
2. Implement it using **built-in libraries** such as `scikit-learn`.

Once you have trained all models, compare their performance using the following metrics:

- Precision
- Recall
- F1-Score
- Accuracy

You should also train a **Logistic Regression model** on the same dataset and report its performance as a baseline for comparison.

Bonus Task: Tune the regularization parameters (e.g., C) for each SVM variant — both in your manual implementation and with the built-in tools — and re-evaluate using the same metrics above. Discuss how regularization affects performance and what values give the best results.