CSCI 6960/4140: Homework 1

Assigned Friday January 23, 2026. Due by 11:59pm ET Friday February 6, 2026.

Objective of this assignment. Modern machine learning is not just about fitting models—it is about understanding data, making realistic assumptions, and validating that your tools actually work. In this assignment you will use `pandas`, `scikit-learn`, and `numpy` to tackle a real prediction task involving count data. Along the way you will:

- See how Poisson regression is used to model practical quantities like bike rentals, web traffic, or disease cases.
- Practice the full ML pipeline: exploring data, creating meaningful features, training and tuning models, and reporting results that someone else could trust.
- Learn how to use finite difference approximations to verify your symbolic computations of gradients, a key skill when implementing custom models used in research and industry.

Instructions. Create a Jupyter notebook for this assignment, and use Python 3. Write documented, well-structured, readable and clear code (e.g. use reasonable variable names, put all imports at the top, use markdown cells appropriately and split your code across cells in a well-structured manner). Your code should run on the TA's computer, but *do not expect him to run it to produce the output*; instead, submit it with the outputs shown. Any textual answers should be clearly visible and plots should be appropriately labeled: legends and axes (with units if appropriate) should be labeled, differentiating styles should be used for different data series in the same plot, scales and limits should be chosen appropriately, and a title should be provided.

(This next part feels strict and not especially fun; however, it is necessary for fairness and for the learning outcomes of the course. AI– just like older modes of cheating– hamstrings development of the skills, patterns of thought, and intuitions this assignment is meant to encourage, so I will strictly enforce the following policies.)

Academic Honesty and AI Usage. *Your submitted work must reflect your own understanding and effort.* Use of AI systems to generate or debug code, derivations, or analysis defeats the learning objectives of this assignment, undermines assessment integrity, is unfair to other students, and harms your learning.

Authorship Statement (Required). At the top of your notebook, you must include the following statement verbatim:

> "I completed this work without assistance from AI tools. All code, derivations, analysis, and discussion are my own."

Submissions that omit this statement *will be treated as late submissions*.

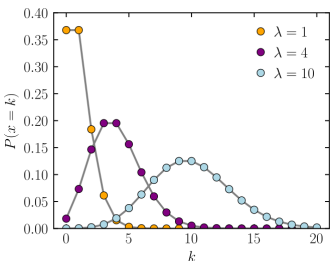
Verification of Authorship. The instructor and TA may conduct oral spot-checks or individual explanations to confirm authorship. These may be random, non-random, or triggered by discrepancies between in-class performance and submitted work. If significant gaps in understanding are observed, the assignment grade may be reduced and, in severe cases, treated as an academic honesty violation.

1 Poisson Regression for Count Data [50 pts total]

In class we covered noise models that help us to model regression, binary classification, and multiclass classification models.

In this problem, we introduce the Poisson noise model that is commonly used for counting problems (where the count can be arbitrarily large, so we cannot use multiclass classification). Example applications include:

- Modeling the number of traffic accidents occurring at a specific intersection or along a stretch of road in a given period.
- Analyzing the number of new cases of a particular disease in different regions or over time, where the rate of occurrence might vary by location or demographic factors.
- Predicting the number of attendees at events or meetings based on factors like day of the week, weather conditions, or location.



Poisson pmfs for different parameters. Taken from Wikipedia.

A discrete nonnegative random variable z follows the Poisson distribution with parameter $\lambda \geq 0$, written $z \sim \text{Poisson}(\lambda)$, if its pmf is given by

$$p_z(k) = \mathrm{e}^{-\lambda}\frac{\lambda^k}{k!}, \quad \text{where} \quad k \in \{0, 1, 2, \ldots\}.$$

The mean of this distribution is λ, and the variance is also[1] λ. See the figure for a visualization of three Poisson pmfs.

Consider the problem of predicting the number of students present in lecture. We can predict this quantity more accurately if we take into account the time of year (are we before or after the drop date?), the weather, and so on. We use Poisson regression to model this conditional dependence.

In Poisson regression, the target y is a nonnegative integer, and we model its dependence on the predictors $\mathbf{x}$ using a Poisson distribution:

$$y \mid \mathbf{x} \sim \text{Poisson}(\exp(\mathbf{w}^T\mathbf{x} + b)).$$

Here, the rate function $\lambda(\mathbf{x}) = \exp(\mathbf{w}^T\mathbf{x} + b)$ models the dependence of y on the covariates $\mathbf{x}$. As mentioned above, $\lambda(\mathbf{x}) = \mathbb{E}[y \mid \mathbf{x}] = \exp(\mathbf{w}^T\mathbf{x} + b)$ models the expected value of y as well as the variance of y, given the predictors $\mathbf{x}$.

(a) **[2 pts] Why the log-link?** In a sentence, explain why $\lambda(\mathbf{x}) = \exp(b + \mathbf{w}^\top\mathbf{x})$ is more appropriate than a linear model $\lambda(\mathbf{x}) = b + \mathbf{w}^\top\mathbf{x}$ for Poisson regression.

(b) **[4 pts] Probability of an observation:** Give the expression for $p_{\boldsymbol{\theta}}(y_i \mid \mathbf{x}_i)$.

(c) **[5 pts] MLE derivation:** Given training data $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, derive the negative log-likelihood loss function. Derive the MLE problem used for determining the parameter estimates $\hat{b}, \hat{\mathbf{w}}$. Similar to what we did when deriving the MLE problem for regression, drop constants that do not depend on the parameter.

(d) **[5 pts] Using a Poisson regression model:** Given a fitted Poisson regression model $\boldsymbol{\theta}$ and a new data point $\mathbf{x}_{\text{new}}$, how would you predict the corresponding target y? Explain your answer.

[1] There are other models for count data (e.g. negative binomial) that can model over- and under-dispersion, where the variance is smaller or greater than the mean.

(e) **[4 pts] Role of the Bias Term:** Consider the Poisson rate expression $\lambda(\mathbf{x}) = \exp(b + \mathbf{w}^\top \mathbf{x}) = e^b \cdot \exp(\mathbf{w}^\top \mathbf{x})$. Explain why if we were using MAP estimation instead of MLE, regularizing b (i.e., adding a regularization term proportional to b^2) would be counterproductive. Hint: how can we interpret $\mathbf{w}$ as capturing how the covariates $\mathbf{x}$ modifies a baseline prediction?

(f) **[20 pts] Computing the gradient and Hessian:** Let $J(\mathbf{w}, b)$ be the objective corresponding to ℓ_2-regularized Poisson regression, where we use regularization coefficient $\mu > 0$ to penalize the norm of $\mathbf{w}$. Note that the bias term is *not* regularized. For convenience, let $\boldsymbol{\theta} = [\mathbf{w}; b]$. Show the computation of $\nabla_{\boldsymbol{\theta}} J$ and $\nabla^2_{\boldsymbol{\theta}} J$ as concisely and neatly as you can.

(g) **[10 pts] Numerical verification of your gradient.** To build confidence in your analytic gradient, we will check it using the *directional derivative test*. This test is what ML researchers/developers use to test the accuracy of their gradient expressions (or of their implementation of automatic differentiation).

For a differentiable function $J(\boldsymbol{\theta})$ and any unit direction $\mathbf{u}$, the directional derivative satisfies

$$\nabla J(\boldsymbol{\theta})^\top \mathbf{u} \;=\; \lim_{h \to 0} \frac{J(\boldsymbol{\theta} + h\,\mathbf{u}) - J(\boldsymbol{\theta})}{h}.$$

For sufficiently small h, the finite-difference approximation above converges to the true directional derivative, and it can be shown by Taylor series arguments that the approximation error scales like $O(h)$.[2]

Experiment setup. For this part, do *not* use real data. Instead:

(a) Let $d = 20$ be the feature dimension, and let $n = 2000$ be the number of samples.

(b) Sample a "ground truth" parameter $(\mathbf{w}, b)$ where $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I_d)$ and $b \sim \mathcal{N}(0, \sigma^2)$. Use $\sigma = 1/\sqrt{d}$ (this scaling avoids unrealistically large Poisson rates).

(c) Sample covariates using `X = np.random.normal(0, 1, size=(n, d))`.

(d) Generate Poisson responses using `y = np.random.poisson(np.exp(X @ w + b))`. This corresponds to the Poisson regression model seen in lecture.

(e) Define the regularization coefficient $\mu = 1$ and construct the ℓ_2-regularized Poisson regression objective $J(\mathbf{w}, b)$ that you derived above.

Choosing a direction u and a random parameter $\boldsymbol{\theta}$. Pick a random direction by drawing a Gaussian vector and normalizing it:

```
u = np.random.normal(0, 1, size=d+1)
u = u / np.linalg.norm(u)
```

This produces a unit-length direction with no preferred bias; any direction works for this test.

Pick a random parameter $\boldsymbol{\theta}$ that isn't too large, to prevent numerical issues:

```
theta = np.random.normal(0, 0.1, size=d+1)
```

Finite-difference check. Evaluate the finite-difference approximation to the directional derivative $\nabla J(\boldsymbol{\theta})^\top \mathbf{u}$ for a geometric range of step sizes $h \in \{10^{-1}, 10^{-2}, \ldots, 10^{-10}\}$. For each h, compute the absolute error

$$\varepsilon(h) = \left| \frac{J(\boldsymbol{\theta} + h\,\mathbf{u}) - J(\boldsymbol{\theta})}{h} - \nabla J(\boldsymbol{\theta})^\top \mathbf{u} \right|.$$

[2] You're not required to prove this; ask in office hours if you're curious.

Plot. Make a log–log plot of $\varepsilon(h)$ versus h and include a reference line proportional to h on the same axes. If your gradient is correct, the finite-difference error should decrease toward zero at roughly the same rate as h until numerical rounding errors dominate for very small h. Briefly describe what you observe.

2 Building a Linear Poisson Regression Model [30 pts]

You will now build a complete Poisson regression model for predicting hourly bike rentals in Washington D.C. This part emphasizes the full ML pipeline: data loading, exploration, preprocessing, model training, validation, and evaluation.

[5 pts] Loading, exploration, and pre-processing.

(a) **Load the dataset:**

```
from ucimlrepo import fetch_ucirepo
bike_sharing = fetch_ucirepo(id=275)
X = bike_sharing.data.features
y = bike_sharing.data.targets['cnt']
```

Install `ucimlrepo` via: `pip install ucimlrepo`

(b) **Data exploration:**

- Print the shape of X and list the feature names
- Drop the `dteday` column (it's redundant with `yr`, `mnth`, etc.)
- Print summary statistics: $\min(y)$, $\max(y)$, $\text{mean}(y)$, $\text{median}(y)$
- Create a histogram of the target variable y (bike counts)

(c) **Identify feature types:** Categorical features: `['season', 'yr', 'mnth', 'hr', 'holiday', 'weekday', 'workingday', 'weathersit']`
Continuous features: `['temp', 'atemp', 'hum', 'windspeed']`
In 1–2 sentences, explain why `hr` (hour of day, values 0–23) should be treated as categorical (discrete and non-ordinal) rather than continuous.

[5pts] One-hot encoding categorical features.

(d) **One-hot encoding:** To handle categorical covariates, we apply one-hot encoding:

```
import pandas as pd
X_encoded = pd.get_dummies(X, columns=categorical_cols,
                           drop_first=False)
```

Print the number of features before and after encoding.

(e) **Sanity check one-hot features:** Count how many times each one-hot feature appears in the dataset:

```
occurrence_counts = X_encoded[one_hot_cols].sum(axis=0)
```

where `one_hot_cols` excludes the continuous features.
Report: minimum, maximum, and mean occurrence counts.
Question: If a one-hot feature appears only 5 times in 17,000 samples, what concern does this raise about model training?

(f) **Train/validation/test split:** Create a 60%/20%/20% split:

```
from sklearn.model_selection import train_test_split
X_temp, X_test, y_temp, y_test = train_test_split(
    X_encoded, y, test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(
    X_temp, y_temp, test_size=0.25, random_state=42)
```

[10 pts] Fitting the model with various regularizations.

(g) **Regularization path:** Fit Poisson regression models with different regularization strengths $\alpha \in \{10^{-4}, 10^{-3.5}, 10^{-3}, \ldots, 10^{1.5}, 10^{2}\}$ (use 30 values via `np.logspace(-4, 2, 30)`):

```
from sklearn.linear_model import PoissonRegressor
for alpha in alpha_values:
    model = PoissonRegressor(alpha=alpha, max_iter=1000)
    model.fit(X_train, y_train)
    # Compute train, validation, test errors (Poisson deviance)
```

For each α, compute the Poisson deviance on train, validation, and test sets using `sklearn.metrics.mean_poisson_deviance`.

(h) **Select best α:** Choose the α that minimizes **validation error**. Report:

- Best α value
- Training, validation, and test deviance at best α

(i) **Visualize regularization path:** Create a plot with α on the x-axis (log scale) and Poisson deviance on the y-axis, showing three curves: train, validation, and test error. Note that the test error is a horizontal line, as it is the test error of the model fit using the optimal α. Mark the optimal α with a vertical line.

Explain why the train and validation error curves look as they do.

[10 pts] Meaningfully reporting the performance of the model.

(j) **The problem with training loss as a metric:** Poisson deviance is the loss function we optimize, but it's hard to interpret. For bike-sharing, we care about: "How many bikes will we be short or over?"

Compute these interpretable metrics on the test set:

```
from sklearn.metrics import mean_absolute_error, r2_score
test_mae = mean_absolute_error(y_test, y_pred)
test_mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100
test_r2 = r2_score(y_test, y_pred)
```

Report all three metrics and explain what each one tells you (2–3 sentences total):

- **MAE (Mean Absolute Error):** What does "MAE = 45 bikes" mean in practical terms?
- **MAPE (Mean Absolute Percentage Error):** Why might MAPE be misleading for this dataset? Hint: consider hours with very low demand (2–5 bikes).
- **R^2 (R-squared):** What does "R^2 = 0.70" tell you about model fit?

(k) **Baseline comparison:** Fit a trivial baseline model that always predicts $\hat{y} = \text{mean}(y_{\text{train}})$. Compute its test MAE and compare to your Poisson model.

Report the % improvement: $100 \times \frac{\text{MAE}_{\text{baseline}} - \text{MAE}_{\text{model}}}{\text{MAE}_{\text{baseline}}}$.

Sanity check: Your model should significantly outperform the baseline. If not, something is wrong!

(l) **Effect of removing the bias term from our model:** Just for this question, use `drop_first=True` when encoding your categorical features[3]

Train two models on the training data:

- `PoissonRegressor(alpha=best_alpha, fit_intercept=True)`
- `PoissonRegressor(alpha=best_alpha, fit_intercept=False)`

Report test MAE for both. Explain briefly how these results relate to the observation made in question 1(e).

Remember to do the rest of the homework with features encoded with `drop_first=False`!

3 Feature Engineering with Polynomial Features [25 pts]

Linear features assume relationships like "demand increases linearly with temperature." But real-world relationships are often nonlinear: bike demand might peak at moderate temperatures (78°F) and decrease when too cold (60°F) or too hot (88°F). You will now engineer polynomial features to *attempt* to capture these nonlinear effects.

[8 pts] Motivation and Feature Engineering.

(a) **Interaction example:** Consider temperature and humidity. Linear features assume their effects are independent. But in reality: high temperature + high humidity = very uncomfortable → fewer bikes.

Explain in 1–2 sentences how a feature `temp` × `humidity` could capture this interaction effect.

(b) **Create polynomial features:** Apply degree-2 polynomial expansion to *continuous features only*:

```
from sklearn.preprocessing import PolynomialFeatures
continuous_data = X_encoded[continuous_cols]
poly = PolynomialFeatures(degree=2, include_bias=False)
continuous_poly = poly.fit_transform(continuous_data)
```

Report: How many features result from the 4 continuous features after polynomial expansion? List 5 example features that were created (e.g., `temp^2`, `temp*hum`).

(c) **Why only continuous features?**

- Explain in 1–2 sentences why we don't apply polynomial expansion to one-hot encoded features. Hint: What is $\texttt{hr_17}^2$ if $\texttt{hr_17} \in \{0, 1\}$?
- Also note: including all polynomial combinations of 59 one-hot features would create thousands of features, making the problem computationally expensive.

(d) **Why `include_bias=False`?** The `PolynomialFeatures` class has a parameter `include_bias`. Explain in 1–2 sentences why we set it to `False`. Hint: look at the documentation for `PoissonRegressor`!

[10 pts] The Importance of Feature Scaling.

[3]You can look up exactly how the one-hot encodings differ when you set this option differently. The consequence of the differnce in the encodings is that with full one-hot encoding (`drop_first=False`), the one-hot columns already span the constant vector. So omitting the intercept and using a full one-hot encoding yields the same model class as including an intercept and using drop-first encoding — the difference is just in how coefficients are parameterized and interpreted. Since we want to see the effect of NOT having a constant offset (b) in our model, we have to simultaneously omit the intercept AND use drop-first encoding.

(e) **Motivation for scaling:** When fitting models with gradient-based optimization (like L-BFGS, as is used by `PoissonRegressor` by default), we want all features to be on similar scales. A common heuristic is to *standardize features to look like standard Gaussians* (mean 0, variance 1):

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
continuous_scaled = scaler.fit_transform(continuous_data)
```

Explain what `StandardScaler` does, precisely, and why you think standardization may help with optimization. Hint: consider if the features were independent, but have very different variances (e.g. one feature has variance .001 and another has variance 10,000) vs if they are independent and have the same variance; revisit the lecture 1 notes.

(f) **Experiment - WITHOUT scaling:** Create polynomial features without standardization first:

```
poly = PolynomialFeatures(degree=2, include_bias=False)
continuous_poly_unscaled = poly.fit_transform(continuous_data)
```

Combine with one-hot features, split into train/val/test, and fit Poisson regression with your best α from Part B.

Observe and report: Do you see any warnings from the optimizer? If so, what do they say?

(g) **Experiment - WITH scaling:** Now standardize before polynomial expansion:

```
scaler = StandardScaler()
continuous_scaled = scaler.fit_transform(continuous_data)
poly = PolynomialFeatures(degree=2, include_bias=False)
continuous_poly_scaled = poly.fit_transform(continuous_scaled)
```

Fit the same model. Report: how do the results change?

(h) **Explain:** Why do we standardize *before* the polynomial expansion instead of after?

[7 pts] Comparing Linear vs Polynomial Models.

(i) **Find optimal α for polynomial model:** Repeat the regularization path from Part B.3, but now using polynomial features (with proper scaling). Select the best α based on validation error.

(j) **Compare performance:** Create a comparison table:

Metric	Linear Model	Polynomial Model
Number of features	?	?
Best α	?	?
Test MAE (bikes)	?	?
Test MAPE (%)	?	?
Test R^2	?	?

(k) **Interpret results:**

- Does the polynomial model perform better? By how much (in terms of MAE or R^2)?
- Compare the optimal α values. Which model requires more regularization? Explain why.

(l) **Feature importance:** We've added many features. Which ones are actually important? Research *permutation importance* (see `sklearn.inspection.permutation_importance`). Compute permutation importance for your polynomial model and report the top 5 most important features. Are these reasonable to you: why or why not?

4 Graduate Students Only: Why KL Divergence? [20 pts]

In class we claimed that KL divergence (equivalently, cross-entropy) is a natural choice for measuring how similar two probability distributions are. A reasonable question is: why not just use the ℓ_2 distance between the parameters of distributions in the same family?

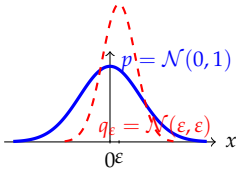
There are several issues with this approach:

- **No notion across families:** When fitting empirical data (which has no parametric form) to a parametric family, there's no meaningful way to compute an ℓ_2 distance between parameters. This alone disqualifies parameter-based ℓ_2 distance for most practical applications.
- **Lack of invariance:** The ℓ_2 distance between parameters depends on the parameterization. For example, we could parameterize a Gaussian by (μ, σ^2) or by $(\mu, \log \sigma^2)$, and these give different ℓ_2 distances.
- **Insensitivity to distributional differences:** Even within a family, ℓ_2 parameter distance may not reflect how different the distributions actually are in a probabilistic sense.

We'll explore the last two points through a concrete example. Consider two one-dimensional Gaussian distributions:

$$p = \mathcal{N}(0, 1)$$
$$q_\varepsilon = \mathcal{N}(\varepsilon, \varepsilon)$$

where $\varepsilon > 0$ is a small parameter.



Plot of the densities of p and q_ε

(a) **[2 pts] KL divergence formula.** The KL divergence from q to p for two one-dimensional Gaussians $p = \mathcal{N}(\mu_1, \sigma_1^2)$ and $q = \mathcal{N}(\mu_2, \sigma_2^2)$ is given by:[4]

$$D_{\mathrm{KL}}(q \| p) = \log \frac{\sigma_1}{\sigma_2} + \frac{\sigma_2^2 + (\mu_2 - \mu_1)^2}{2\sigma_1^2} - \frac{1}{2}.$$

Write down the expression for $D_{\mathrm{KL}}(q_\varepsilon \| p)$ by substituting $\mu_1 = 0$, $\sigma_1^2 = 1$, $\mu_2 = \varepsilon$, and $\sigma_2^2 = \varepsilon$.

(b) **[4 pts] Behavior as $\varepsilon \to 0$.** Compute:

$$\lim_{\varepsilon \to 0^+} D_{\mathrm{KL}}(q_\varepsilon \| p) \text{ and } \lim_{\varepsilon \to 0^+} D_{\mathrm{KL}}(p \| q_\varepsilon)$$

Show your work.

(c) **[4 pts] Parameter ℓ_2 distance.** The ℓ_2 distance between the parameters of p and q_ε (using the parameterization (μ, σ^2)) is:

$$d_{\ell_2}(p, q_\varepsilon) = \sqrt{(\mu_2 - \mu_1)^2 + (\sigma_2^2 - \sigma_1^2)^2}.$$

Compute $\lim_{\varepsilon \to 0^+} d_{\ell_2}(p, q_\varepsilon)$. Show your work.

[4] You can derive this yourself if you're curious, but it's not required for this problem.

(d) **[4 pts] Alternative parameterization.** Now consider parameterizing the Gaussians by $(\mu, \log \sigma^2)$ instead. Compute the ℓ_2 distance:

$$\tilde{d}_{\ell_2}(p, q_\varepsilon) = \sqrt{(\mu_2 - \mu_1)^2 + (\log \sigma_2^2 - \log \sigma_1^2)^2}.$$

What is $\lim_{\varepsilon \to 0^+} \tilde{d}_{\ell_2}(p, q_\varepsilon)$? How does this compare to the limit you found in part (c)?

(e) **[6 pts] Interpretation.** Based on your answers above, explain in a few sentences why KL divergence is a more reasonable measure of distributional similarity than ℓ_2 parameter distance. Your answer should address:

- What you would expect to happen as ε approaches 0, and why.
- How well each metric (KL divergence vs ℓ_2 parameter distance) captures this expected behavior.
- Why it is reasonable that the two KL divergences have different values for nonzero ε.
- The issue of parameterization invariance.