midterm

October 31, 2024

1 Midterm - Ishaan Sathaye

1.1 Part I

A researcher wishes to fit a linear regression model with several predictors. She is considering three loss functions:

1. $loss = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda^2 \sum_{j=1}^{p} \beta_j^2 + 5\lambda$
2. $loss = \sum_{i=1}^{n} |y_i - \hat{y}_i| + \sum_{j=1}^{p} |\beta_j|$
3. $loss = \prod_{i=1}^{n} \frac{(y_i - \hat{y}_i)^2}{y_i^2}$

For each of these options:

1. Give an intuitive explanation for this choice of loss function. How does it express the desire for accurate predictions of a quantitative variable?
   - For loss function (1), the first term is just the sum of squared residuals (SSR), which is a very common loss function for linear regression. The second term is a penalty for large coefficients, which helps prevent overfitting. The third term which is just a constant really does not depend on the data, so it does not really affect the whole fitting process. It expresses the desire for accurate predictions by minimizing the SSR and the penalty term.
   - For loss function (2), the first term is the sum of absolute residuals and the second term is the sum of absolute coefficients. This loss function would be used when she wants to prevent overfitting and have a simpler model. The absolute values in the loss function would potentially make the model more robust to outliers since the loss function is not as sensitive to outliers as the squared loss function. In terms of the desire for accurate predictions, this loss function aims to minimize the sum of absolute residuals and the sum of absolute coefficients.
   - For loss function (3), the loss function is the product of the squared residuals divided by the true value squared. This loss function would be used when she wants to penalize the model more for larger residuals when the true value is larger. Also, this model would be perfect if she wants to have something where it is more accurate for larger true y values. The loss function is trying to minimize the product of the squared residuals divided by the true value squared.
2. Do you see any possible issues with this loss function? What assumptions have to be true about the data for this loss function to be viable?
   - For loss function (1), the issue with this loss function is that the third term is just a constant, so again it does not really affect the fitting process. The regular assumptions

would have to be true about the data for this loss function to be viable that the data has to be linear and the residuals have to be normally distributed.
    - For loss function (2), the issue with this loss function is that it is not differentiable at 0, so it would be difficult to optimize. There would be no closed form solution for this loss function. So to continue with this loss function she would need to use gradient descent. The same assumptions would have to be true about the data for this loss function to be viable that the data has to be linear and the residuals have to be normally distributed.
    - For loss function (3), the issue with this loss function is that it is not differentiable at 0 and this is because of the division by the true value squared. The same assumptions as the first 2 would have to be true for the data for this loss function to be viable.
3. Find an equation for the *gradient* of the loss function. (A general equation for the partial derivative at β_j will suffice.)
    - For loss function (1), the gradient of the loss function would be:
        - $\frac{\partial loss}{\partial \beta_j} = -2\sum_{i=1}^{n}(y_i - \hat{y}_i)x_{ij} + 2\lambda^2\beta_j$
    - For loss function (2), the gradient of the loss function would be:
        - $\frac{\partial loss}{\partial \beta_j} = -\sum_{i=1}^{n} sign(y_i - \hat{y}_i)x_{ij} + sign(\beta_j)$
    - For loss function (3), the gradient of the loss function would be:
        - $\frac{\partial loss}{\partial \beta_j} = -2\sum_{i=1}^{n} \frac{(y_i - \hat{y}_i)x_{ij}}{y_i^2}$
        - x_{ij} term comes from the chain rule: $\frac{\partial loss}{\partial \beta_j} = \frac{\partial loss}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial \beta_j}$
4. Give a brief code outline (psuedocode) to show the procedure you would use to calculate the "best" β's according to this loss function. You code to does not need to run; however, it **does** need to be specific about the inputs as well as the equations. For example:

Sufficient:

```
def ols(x, y):
    sx = std dev of x
    sy = std dev of y

    mx = mean of x
    my = mean of y

    rxy = correlation of x and y

    beta_1 = sy/sx * rxy
    beta_0 = my - beta_1 * mx

    return beta_0, beta_1
```

- For loss function (1), we can use the closed form solution for the gradient of the loss function to find the best betas. We can use the following code:

```
def loss1(x, y, beta, lambda):
    n, p = x.shape
    x = np.c_[np.ones(n), x] # intercept column of 1s
    I = np.eye(p+1)
    betas = np.linalg.inv(x.T @ x + lambda**2 * I) @ x.T @ y
    return betas
```

- For loss function (2), we can use gradient descent to find the best betas. We can use the following psuedocodish python code:

```
def loss2(x, y, beta, eta):
    n, p = x.shape
    x = np.c_[np.ones(n), x] # intercept column of 1s
    beta = np.zeros(p+1)
    while True:
        gradient = compute_gradient(x, y, beta)
        beta = beta - eta * gradient
        if stopping_condition(gradient):
            break
    return beta

def compute_gradient(x, y, beta):
    n, p = x.shape
    return -np.sign(y - x @ beta) @ x + np.sign(beta)

def stopping_condition(gradient):
    # check if norm of gradient is less than 1e-6
    return np.linalg.norm(gradient) < 1e-6
```

- For loss function (3), we can use gradient descent to find the best betas. We can use the following psuedocodish python code:

```
def loss3(x, y, beta, eta):
    n, p = x.shape
    x = np.c_[np.ones(n), x] # intercept column of 1s
    beta = np.zeros(p+1)
    while True:
        gradient = compute_gradient(x, y, beta)
        beta = beta - eta * gradient
        if stopping_condition(gradient):
            break
    return beta

def compute_gradient(x, y, beta):
    n, p = x.shape
    return -2 * np.sum((y - x @ beta) * x / y**2)

def stopping_condition(gradient):
    # check if norm of gradient is less than 1e-6
    return np.linalg.norm(gradient) < 1e-6
```

1.2 Part II

1.2.1 Question A

A researcher is trying to fit a linear regression model using a LASSO penalty. To choose a good value of the penalty parameters, λ, she decides to take the following approach:

- For each of the many possible λ values:
  - Fit the model using that λ
  - Find the predicted values from the model, $\hat{y}_1 ... \hat{y}_n$
  - Calculate the residuals, $r_i = y_i - \hat{y}_i$
  - Calculate the sum of the squared residuals
- Then we choose the value of λ that resulted in the model with the smallest sum of squared residuals.

Discuss this strategy: Do you think it is a good idea? Why or why not? Do you have any suggestions to make it more efficient, more justifiable, or more correct?

- I believe this strategy is not a good idea because it is solely relying on the training data to choose the best penalty parameter. This definitely would lead to overfitting since the model would be biased towards the training data and therefore would not generalize to any new data that it will see. As we have done in homeworks, cross-validation or LOO is a better approach to choose the penalty parameter. Now the model will be evaluated on data that was not used in the training and from there the penalty parameter will be chosen based on model's success on unseen data.
- Another suggestion would be maybe to use a grid search to find the best penalty parameter. This would be more efficient than just trying every possible penalty parameter.

1.2.2 Question B

Your fame as a data scientist has spread far and wide, and university hires you to investigate faculty happiness. They supply you with two datasets:

1. 1000 emails sent from faculty accounts in January 2022 all of which have been analyzed by language experts and given a "happiness score" on a scale of 0 to 100.
2. 10,000 emails sent from faculty accounts in February to December 2022, which have not been analyzed.

The university cannot afford to hire their language analysts for all 10,000 emails, but they can bring them back for another batch of around 1000 if you ask them to.

Your client would like you to create a predictive model from the 1000 email dataset, that can be used to assess the happiness of the 10,000 email dataset. They would then like you to tell them if faculty happiness was increasing, decreasing, of staying the same across 2022.

Propose a modeling process to address this question

You should include:

- A (very brief) description of how you might **pre-process** the data
- A **model specification**, and **why** you think that model would be a good choice for this task. This can be a model we studied, an existing model we haven't studied, or you can "invent" something–but you must include some discussion of why you think that choice is good/reasonable for this scenario.
- A **loss function** that you will use to fit the model, and **why** this loss function correctly expresses your desires for your "best" model.
- A **metric** you will use to report your model's abilities and/or to tune hyperparameters, and **why** this metric is a good measure of "model success" in this case. This metric should not

be R-squared, MAE, or MSE. (Those would be reasonable in this case, but I want you to find/invent a different one and justify it.)

Note: You do *not* need to concern yourself in this question with the computational feasibility of your model, loss, or metric. I am only looking for your to translate the "real world" needs of the scenario into mathematical decisions.

Modeling Process:

- Pre-processing: I would first check for missing values and outliers in the data. I would then check for multicollinearity between the predictors, which if they were would lead to needing to remove some predictors. The predictors in this case would be the words in the emails, and I would use a count vectorizer to convert the words into a matrix of token counts and continue with TF-IDF. I would then split the data into a training and testing set. I would then use the training set to fit the model and the testing set to evaluate the model.
- Model Specification: The model that would be a good choice for this task would be a linear regression model with a LASSO penalty. This model would be a good choice because it would help prevent overfitting and would help with feature selection. The penalty ensures that the model is not too complex and that it generalizes well to new data. The model would be able to predict the happiness score of the emails in the 10,000 email dataset, and after doing that I can see if the faculty happiness was increasing, decreasing, or staying the same across 2022.
- Loss Function: The loss function that I would use to fit the model would be the sum of squared residuals with a L1 penalty. This loss function is perfect since it would help decide which features are the most important.This function correctly expresses my desires for the "best" model because it prevents overfitting.
  - potential $loss = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$
- Metric: The metric I would use to report the model's abilities and to tune my λ would be the mean absolute percentage error (MAPE). This is different from the MAE because I am getting back a percentage error. In a nutshell, the lower this metric is the better the model. This metric is a good measure of "model success" in this case because it would give me the percentage of how far off the model is from the true value. This is a really good value in this case since it allows me to get a good idea of how well this model is doing in predicting the happiness score of the emails in the 10,000 email dataset. In the train/test split, I would use the MAPE to tune the λ value to get the best model. Obviously, I would need to call the language experts back to analyze another 1000 emails in the 10,000 email dataset to get more true values to calculate the MAPE, since I do not want to extrapolate the MAPE from the 1000 email dataset to the 10,000 email dataset.