Problem 1

i)

i) Yes, we can estimate the parameters β analytically as we did for linear regression.

The given loss function is:

$$L(\beta) = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{d} \beta_j x_{ij}\right)^2 + \lambda \sum_{j=1}^{d} \beta_j^2$$

where the loss function represents a regularized linear regression model (also known as Ridge Regression). The first term is the sum of squared residuals and the second term is the regularization term which includes λ which is a regularization parameter which controls the amount of shrinkage applied to the coefficients β.

First, we need to take the derivative of $L(\beta)$ with respect to β_j for the first term:

$$\frac{\partial}{\partial \beta_j} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{k=1}^{d} \beta_k x_{ik}\right)^2 = -2 \sum_{i=1}^{n} x_{ij} \left(y_i - \beta_0 - \sum_{k=1}^{d} \beta_k x_{ik}\right)$$

Then, we will do the same with respect to β_0:

$$\frac{\partial}{\partial \beta_0} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{k=1}^{d} \beta_k x_{ik}\right)^2 = -2 \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{k=1}^{d} \beta_k x_{ik}\right)$$

The differentiation of the regularization term with respect to β_j is:

$$\frac{\partial}{\partial \beta_j} \lambda \sum_{k=1}^{d} \beta_k^2 = 2\lambda \beta_j$$

(For β_0, this term is considered to be 0 because we do not regularize the intercept)

Combining the above derivatives for both parts & the loss function, we get:

$$-2 \sum_{i=1}^{n} x_{ij} \left(y_i - \beta_0 - \sum_{k=1}^{d} \beta_k x_{ik}\right) + 2\lambda \beta_j = 0,$$

and

$$-2\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{k=1}^{d}\beta_k x_{ik}\right)=0$$

(for β_0, the regularization term is not added).

Now we will combine the above two equations in matrix notation, where X is the design matrix (including a column of ones for the intercept) and y is the vector of outcomes. So, the derivative of the loss function $L(\beta)$ with respect to β can be expressed as:

$$\frac{\partial L}{\partial \beta} = -2X^T(y-X\beta)+2\lambda I'\beta=0$$

where I' is a modified identity matrix that has a zero for the intercept term β_0 to account for the lack of regularization on the intercept.

We can rearrange the above expression to solve for β:

$$X^T(y-X\beta)=\lambda I'\beta$$

$$(X^TX+\lambda I')\beta = X^Ty$$

$$\beta=(X^TX+\lambda I')^{-1}X^Ty$$

This equation finds the β values which minimize the loss function.

ii)

ii) To find the gradient of $L(\beta)$, we need to compute the partial derivatives of $L(\beta)$, with respect to each β_j, $j=0,1,\ldots,d$.
So, the gradient vector can be written as:

$$\frac{\partial L}{\partial \beta_0}, \frac{\partial L}{\partial \beta_1}, \ldots, \frac{\partial L}{\partial \beta_d}$$

Since the partial derivatives for $L(\beta)$ with respect to β_0 and β_j were calculated in part (i), we will rewrite them here:

$$\frac{\partial L}{\partial \beta_j} = -2\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{k=1}^{d}\beta_k x_{ik}\right)x_{ij}+2\lambda\beta_j$$

$$\frac{\partial L}{\partial \beta_0} = -2\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{k=1}^{d}\beta_k x_{ik}\right)$$

Again, similarly to what was done in part (i), we can rewrite these two expressions in matrix notation as follows:

$$\frac{\partial L}{\partial \beta} = -2X^T(y - X\beta) + 2\lambda I'\beta = \nabla L(\beta).$$

where again, I' is a modified identity matrix that has a zero for the intercept term β_0 to account for the lack of regularization on the intercept.

iii)

iii) The general formula for the update step for each parameter β_j in gradient descent is:

$$\beta_j := \beta_j - \alpha \frac{\partial L}{\partial \beta_j} \quad \text{(where } \alpha \text{ is the learning rate)}$$

Since we found $\frac{\partial L}{\partial \beta_j}$ for our loss function $L(\beta)$ in part (ii) to be $-2\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{k=1}^{d}\beta_k x_{ik}\right)x_{ij} + 2\lambda\beta_j$, we can apply this to the general formula outlined above to get:

$$\beta_j := \beta_j - \alpha\left(-2\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{k=1}^{d}\beta_k x_{ik}\right)x_{ij} + 2\lambda\beta_j\right)$$

which can be rewritten as:

$$\beta_j := \beta_j - \alpha\left(-2X^T(y - X\beta) + 2\lambda I'\beta\right)$$

(using the reformulation from parts (i) and (ii))

which can finally be rewritten as:

$$\beta_j := \beta_j - \alpha \nabla L(\beta).$$

iv) (Please see attached Python file)

Problem 2

i) For this problem, I will be using the twitter_samples dataset from the nltk corpus. The features I will be using in my logistic model include bag of words (which calculates the frequency count of each word) and TF-IDF. Bag of words is a straightforward way to transform text into a fixed length set of features, where each feature represents the occurrence or frequency of a word in the text corpus. This feature also provides information about the classification of tweets as the presence of certain words can strongly indicate the category of the tweet (in this case, whether it has positive or negative sentiment). TF-IDF adjusts the counts of each word in the tweet by how common they are across all tweets in the corpus. This helps to highlight words that are more unique to a tweet, providing insight into its content. This feature also penalizes common words that appear in many tweets (such as “the”, “is”, etc.) which reduces their impact on the model’s decision-making process. This allows the model to focus on more meaningful words for classification. Additionally, TF-IDF scales down the effects of words that occur very frequently and might overshadow the presence of less frequent, potentially more informative words. The combination of bag of words and TF-IDF benefits the model by implementing both the straightforward representation of text data and the more nuanced, importance-weighted representation. The bag of words feature ensures that frequency signals are captured, while TF-IDF adjustments ensure that the features are weighted by their relevance and uniqueness across the corpus.

Now we can write the mathematical model as follows:

Given a tweet T, X is the feature vector of the features mentioned above.

We can model the probability that tweet T expresses a positive sentiment ($Y=1$) as:

$$P(Y=1|X) = \frac{1}{1+\exp\{-(\beta_0+\beta_1 X_1+\dots+\beta_n X_n)\}}$$

where $X_1, \dots, X_n$ are the features extracted from T, β_0 is the intercept, and $\beta_1, \dots, \beta_n$ are the coefficients for each feature.

ii)

ii) Given the expression in part (i), the likelihood function can be expressed in its general form as:

$$L(\beta) = \prod_{i=1}^{N} P(Y=y_i \mid X=x_i)$$

(where N is the set of training observations (x_i, y_i) — # of tweets)

Since y_i can take on two values (1 for positive sentiment, 0 for negative sentiment), the likelihood can be expressed as:

$$L(\beta) = \prod_{i=1}^{N} \left(\frac{1}{1+e^{-\beta^T x_i}}\right)^{y_i} \left(1-\frac{1}{1+e^{-\beta^T x_i}}\right)^{1-y_i}$$

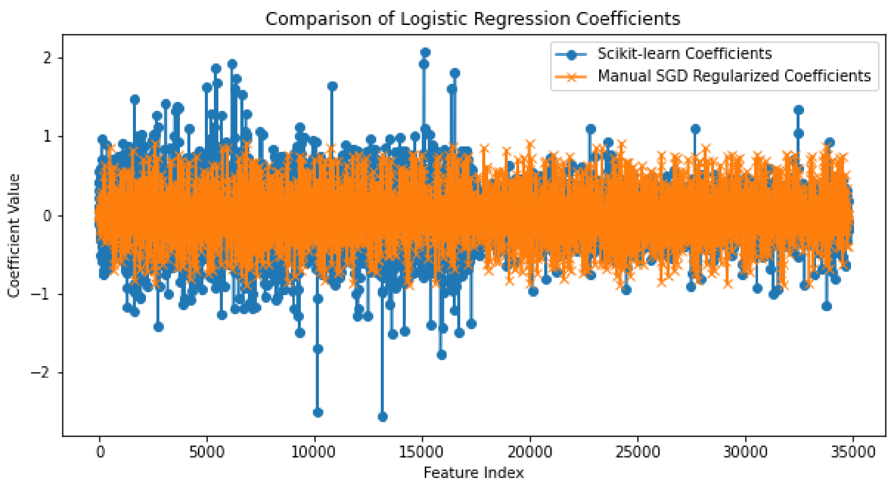
where $\beta^T x_i = \beta_0 + \beta_1 x_{i1} + \dots + \beta_n x_{in}$ is the linear combination of features for the i^{th} tweet.

Finally, since its more common to work with the log-likelihood, we can write:

$$\log L(\beta) = \sum_{i=1}^{N} \left[y_i \log\left(\frac{1}{1+e^{-\beta^T x_i}} \right) + (1-y_i) \log\left(1 - \frac{1}{1+e^{-\beta^T x_i}} \right) \right]$$

iii) (Please see attached Python file for code)
After training the logistic regression classifier using a black-box implementation, an accuracy score of 0.7625 was obtained.

iv) (Please see attached Python file for code)
Next, the logistic regression classifier was trained by minimizing the negative log-likelihood function using a numerical optimization procedure, more specifically, stochastic gradient. To compare the coefficients obtained in step (iii) with the coefficients obtained here, I decided to create a plot:



From the plot above, we can see that the scikit-learn model's coefficients display a broader spread, with some coefficients having relatively high absolute values. This indicates that the scikit-learn model may be giving more weight to certain features. On the other hand, the manual SGD coefficients are more tightly clustered around zero, suggesting a stronger regularization effect which reduces the magnitude of the coefficients. Although there is a difference in magnitude of the coefficients between the two models, there appears to be similarity in the influence of features between both models (coefficients that are positive in one model tend to be positive in the other and vice versa). The coefficients of the scikit-learn model are slightly larger (even more so for feature index values from 0 to 17500) which indicates that the model is capturing more of the variance in the training data (could be a sign of overfitting) while on the other hand, the manual SGD coefficients are smaller, indicating potential underfitting.

Additionally, since the intercepts of the two models are not included in the plot above, we can find these manually to compare them. The intercept for the scikit-learn model is -0.335303467968054 which the intercept for the manual SGD model is -0.15482978702500036.