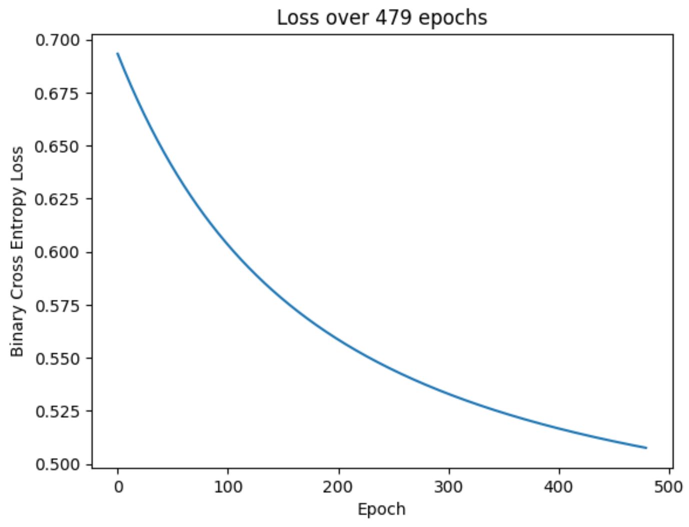
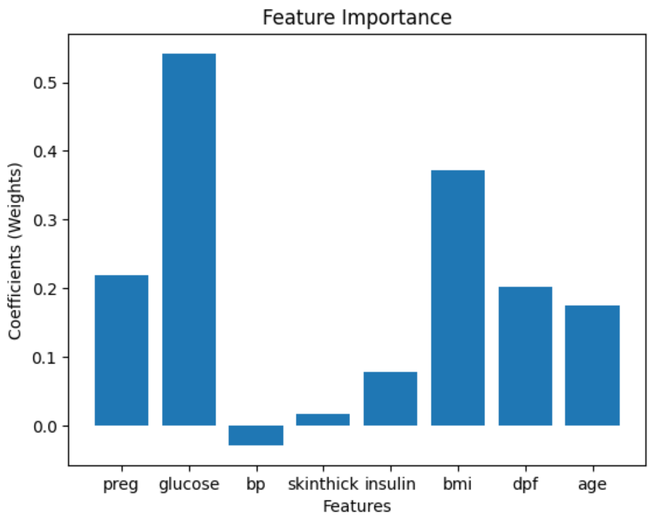
Question 1

This question involved creating a binary classifier to predict whether someone was positive for diabetes. The methodology for this was the same as the gradient descent algorithms from the past problems. After calculating the predictions (values between 1 and 0), we simply call predictions below 0.5 negative and above 0.5 positive for diabetes. If this diabetes dataset is normalized, it makes the model freak out and produces unreliable results. But using standardization effectively solves this issue and produces a good model.





Regularization was also applied to try and remove the parameters that don't have as much of an effect on diabetes, such as blood pressure and skin thickness. It emphasizes glucose levels and BMI, which are historically the most important factors in predicting diabetes.

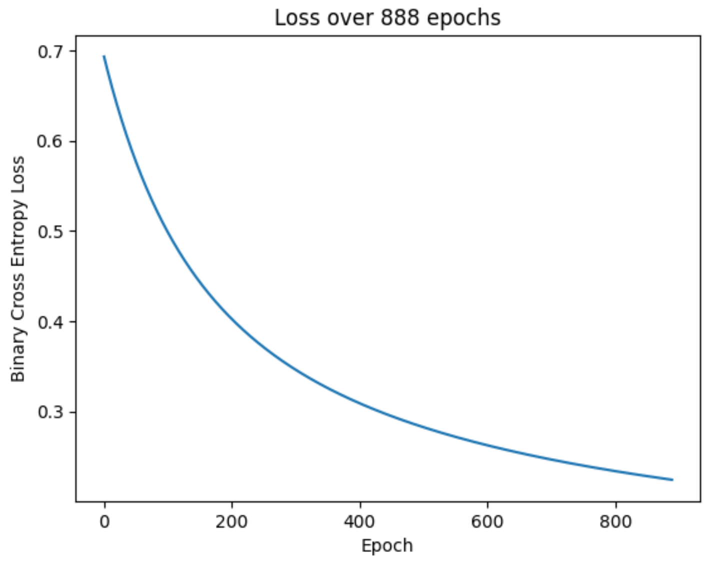
Confusion Matrix	*Positive (1)*	*Negative (0)*
Positive (1)	(true positive) **122**	(false positive) **33**
Negative (0)	(false negative) **11**	(true negative) **22**

Accuracy: 78.7097% | Precision: 91.73% | Recall: 84.72% | F1: 0.8809

These are pretty good results for a classifier model. The model gets about 80% of predictions correct, and about 92% of the cases that we predicted as positive were positive, which is an encouraging result. The F1 score combines the previous scores (recall, precision) to give us a view into how well the model is performing. Generally, this model is really good, but it could use improvement with its recall score.

Problem 2

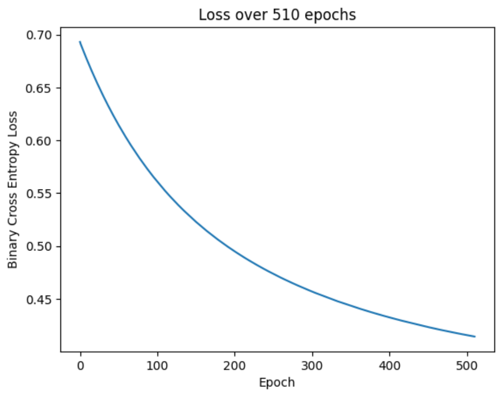
This problem is a repeat of the previous problem. Without L1 regularization, the model performs quite well.



Confusion Matrix	*Positive (1)*	*Negative (0)*
Positive (1)	(true positive) **112**	(false positive) **4**
Negative (0)	(false negative) **2**	(true negative) **2**

Accuracy: 96.5517% | Precision: 98.25% | Recall: 98.25% | F1: 0.9825

This is an extraordinarily well-performing model, and oddly enough most of the data predicts that 0th option (malignant). Adding L1 regularization bumps up all the scores a little bit and improves the training speed.



Confusion Matrix:

[113, 3]

[2, 1]

Accuracy: 97.4138% | Precision: 98.26%

Recall: 99.12%

F1: 0.9869

Problem 3

Instead of using the gradient descent algorithm from last time, we now switch to a naïve Bayes classifier. This algorithm works on the principle of Bayes' theorem and is considered naïve because it makes an assumption that all the input features are independent of each other. We calculate the frequency of each class and then calculate the conditional probability by counting the frequency of that feature within each class.

Confusion Matrix	*Positive (1)*	*Negative (0)*
Positive (1)	(true positive) **108**	(false positive) 8
Negative (0)	(false negative) 1	(true negative) 7

Accuracy: 93.1034% | Precision: 99.08% | Recall: 93.91% | F1: 0.9643

This model is still a pretty good model for the task, but it falls short of the previous logistic regression model. It predicts positive cases incorrectly but cuts the number of false negatives in half. Based on the F1 score alone, this model is inferior. It is important to note, however, that this model is a lot less compute-intensive to train and is a much more streamlined process than the logistic regression, which takes 500+ iterations to train in most cases.

Problem 4

This section introduces us to a new method of preprocessing data (PCA) and evaluating which features have the least effect on the outcome. By removing these features, we can get a more precise model with more control, unlike the approach used in L1 regularization. Our model actually degrades in almost every single metric after incorporating PCA, regardless of how many input features are chosen to remain (the k value). In some cases, the PCA model achieves a better precision value, but loses by around .1 (on average) in recall, which is considered the most important parameter.

K = 1: Confusion Matrix: [101, 15] [2, 13] Accuracy: 87.069% Precision: 98.06% Recall: 88.6% F1: 0.9309	**K = 6** Confusion Matrix: [104, 12] [1, 11] Accuracy: 89.6552% Precision: 99.05000000000001% Recall: 90.42999999999999% F1: 0.9454	**K = 11** Confusion Matrix: [103, 13] [1, 12] Accuracy: 88.7931% Precision: 99.03999999999999% Recall: 89.57000000000001% F1: 0.9407
K = 16 [103, 13] [1, 12] Accuracy: 88.7931% Precision: 99.03999999999999% Recall: 89.57000000000001% F1: 0.9407	**K = 20** [103, 13] [1, 12] Accuracy: 88.7931% Precision: 99.03999999999999% Recall: 89.57000000000001% F1: 0.9407	…

You will notice that at some point the model scores the exact same no matter how many extra parameters we add. The difference in performance between the PCA model and the logistic regression model is because as the number of parameters increase, PCA will simply return a linear transformation of the original features, which in this case is not beneficial to the performance of the model.

Switching to the naïve Bayes classifier instead of logistic regression severely downgrades the performance of the model. F1 scores range from 0.2 to 0.7, which is a far cry from the performance of the previous models we tested. This is a suboptimal strategy for multiple reasons: PCA reduces the amount of information that the bayes model receives, and information is crucial because the model relies on the individual features' probability distributions. The distribution of the data is also not Gaussian anymore, which was one of the key reasons the Bayes classifier worked so well.

Link to all the code and the Jupyter notebook file: Anu78/intro-to-ml-hw: homework for ECGR-4105 @ uncc (github.com)