CS5014 Credit Approval Report

170011474

February 2021

1 Part 1

1.1 (a)

To load the data from `crx.data`, I used panda's embeded `read_csv` to load. Since the each instance/row uses ',' to separate cell values, the parameter `sep` should be set to ','. As the data does not use its first row as its column names, the parameter `header` is set to `None`. **To better refer the dataframe**, I gave each column a name, corresponding to the `crx.names`, so the name is like "A12" and so on.

Secondly, we need to handle missing values. By using `unique()`, we can know there are many `?`s inside cell values, which represent for missing values. The solution to this problem is to drop rows where there are `?`. **The reason to have this step is that training a model with a data set that has a lot of missing values can drastically impact the quality of machine learning model.** Other approaches to handle missing values, such as filling in with average values, would also bring dependencies and outliers will dramatically affect it. Therefore, simply deleting the row where missing values existing would be intuitive and straightforward.

1.2 (b)

When to split the dataset into training set and testing set, **to prevent data dependency existing between rows**, I shuffled the whole dataset. The way to split it is $\frac{2}{3}$ for training and $\frac{1}{3}$ for testing, provided by sklearn packages. 5 fold cross validation

is also implemented. **Since the current dataset is relatively small and there is no sufficient data to train a very sound model,** 5 fold cross validation generally results in a less biased model compare to other methods. It ensures that every observation from the original dataset has the chance of appearing in training and testing set.

1.3 (c)

As we know from the data, there are some attributes which hold categorical values, for example, `t` and `f`. Those values cannot properly fed into machine learning models, so one hot encoder is needed. I implemented it by my own rather than directly using the prepared toolkit. Another approach, such as using 1, 2 to represent different categories, would bring implicit similarities and further affect the training. **Therefore, the one hot encoding ensures each category has a distinct but identical distance in the vector space.**

The standard scaler was adopted to normalise numerical values. This step would result in convergable iterations. Otherwise, a warning would show: `STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.` **The most important reason to do this is that if one of the features has a broad range of values, the distance governs this particular feature**. Besides, when to do gradient descent, there were be very large steps in one dimension but minor steps in other dimensions. **The reason to choose the Standard Scaler rather than others is that it is best for classification problem**, stated by this article.

1.4 (d)

To ensure data leakage is not involved, corresponding to the lecture slides, handling missing values is settled before training and testing set split. Afterward, separately do feature scaling for training set and testing set. If we calculate mean and variance in the feature scaling **before** splitting, it have knowledge of the full distribution of data in the whole dataset.

2 Part 2

2.1 (a)

The worst accuracy is 357 / (357 + 296) = 0.55. This is the baseline for the current classifier since the classifier can simply guess all instances as the majority class and it does not care about whether the dataset is balanced or not, so there is still 357 instances are guessed correctly.

Theoretically the best accuracy it can achieve is to predict all instances correctly so the best accuracy is 1.0.

2.2 (b)

penalty In training process, regulation is a technique to solve over-fitting problems, and adding a penalty to the cost function is one way to do regulation. There are multiple available penalty strategies: l1 and l2. The general idea here is in the traditional gradient method, it would give us an optimal converged point where the error value is minimal. However, it would predict horribly in the testing set if it overfits. Therefore, we add a penalty so the final converged point is not the very optimal position, but is a position near the optimal point. l1 is the sum of absolute values of weights, while l2 is the sum of squared values of weights.

tol It is a tolenrance value for stopping criteria. It decides to stop searching for a minimum (or maximum) once it is met (error is less than one specific value). If it is too big, then the algorithm stops before it can converge.

max_iter It means the maximum number of iterations taken for the solvers to converge. If it is set to 100, then the algorithm is stopped after 100 iterations. If it is too small, then it would result in under-fitting.

2.3 (c)

Since the distribution of different class samples is very different, the algorithms tend to get biased towards the majority classes present and don't perform well on the mi-

nority ones. Consequently, class's frequency is highly imbalanced. By introducing `class_weight`, it penalizes the misclassification made by the minority class by setting a higher class weight while reducing weight for the majority class. Large weights result in a large penalty and a large update to the model coefficients.

The "balanced" mode automatically adjust weights inversely proportional to class frequencies in the input data. This considers the frequencies of class samples so the minority classes are able to achieve better training.

2.4 (d)

Take the current case as an example, in `predict()`, the model would return 1 or 0 for each testing instance, where 1 is for "+" and 0 is for "-". Assume it returns 0, in `predict_proba()`, it would return a vector with a shape 1 * 2: [0.99, 0.01] where 0.99 is the probability of being 0, and 0.01 is the probability of being 1. Therefore, it returns a vector containing probabilities for every class.

In `decision_function()`, it returns array of shape (n_samples, n_classes) with filled confidence score, which is basically a signed distance of sample from hyperplane of your model. The hyperplane can be seen as a decision boundary of positive and negative space. It would return one value for one sample with positive confidence score and negative score for negative predictions.

Overall, the relationship among the three is that the `decision_function()` would tell how confident it is according to its distance to the hyperplane, and `predict_proba()`'s maximum number's indice indicates the `predict()`'s class.

3 Part 3

3.1 (a)

The classification accuracy is 0.853.

In multilabel classification, the function returns the subset accuracy. If the entire

set of predicted labels for a sample strictly match with the true set of labels, then the subset accuracy is 1.0; otherwise it is 0.0. The accuracy formula is defined as:

$$accuracy(\hat{y}_i, y_i) = \frac{1}{N} \sum_{i=0}^{N-1} 1(\hat{y}_i = y_i)$$

where N is the number of samples. $\hat{y}_i$ is the predicted value of the i-th sample and y_i is the corresponding true value. $1(\hat{y}_i = y_i)$ is the indicator function, which can be viewed as the number of correct matches.

3.2 (b)

The balanced classification accuracy is 0.867.

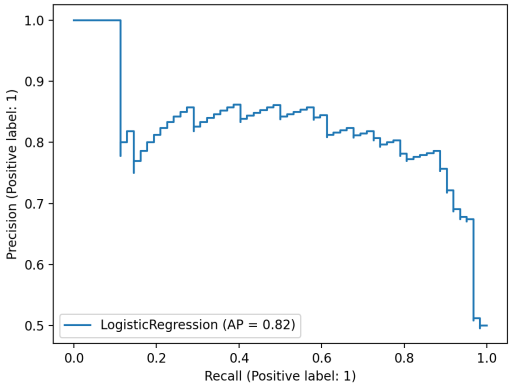
The balanced accuracy in binary and multiclass classification problems to deal with imbalanced datasets. The reason is similar to the one mentioned in `class_weight`. For example, if one problem is to predict if one person is male or female, and the whole dataset consists of 90 men and 10 women. If a model simply predicts all instances to be men, it still has 90% accuracy. However, this does not reflect the model's genuine performance. In order to solve this problem, the balanced accuracy would take care of this. It is defined as the arithmetic mean of sensitivity (true positive rate) and specificity (true negative rate),

$$balanced\_accuracy(\hat{y}_i, y_i) = \frac{1}{2}(\frac{TN}{TN + FP} + \frac{TP}{TP + FN})$$

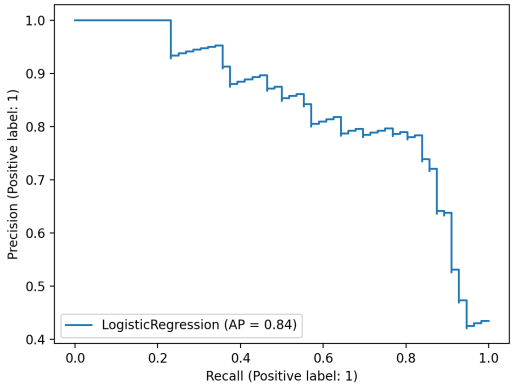
3.3 (c)

When the logistic regression model is unbalanced, the confusion matrix is: [[303 54][51 245]]. When it is balanced, the confusion matrix is: [[307 50][41 255]]. The reason for this is that the class distribution in the training set is not balanced: after handling missing data, the '+' class has 296 samples but the '-' class has 357 samples. Therefore, it would focus on reducing '+' class's cost more, which results in better accuracy in class '+'. From those matrices, we can see that TP is improved (from 303 to 307) and TN is also improved (from 245 to 255).

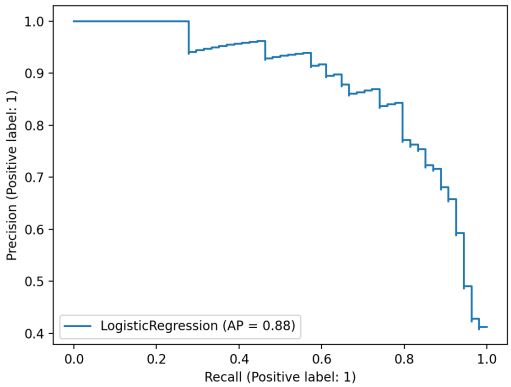
3.4 (d)



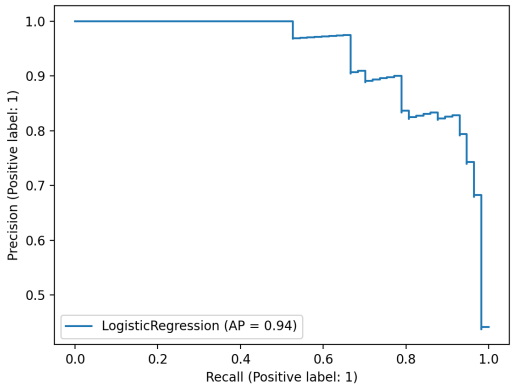
(a) AP = 0.82



(b) AP = 0.84



(a) AP = 0.88



(b) AP = 0.94

Figure 1: AP with Precision Recall Curve

Average Precision is the area under the precision recall curve. As the area goes larger, the average precision goes larger.

4 Part 4

4.1 (a)

The original cost function for logistic regression is:

$$L(\theta) = \sum_{i=1}^{m} y^{(i)} log\sigma^{(i)} + (1 - y^{(i)}) log(1 - \sigma^{(i)})$$

Notes that $\sigma^i = \sigma(\theta^T x^{(i)})$, and for each instance $I^{(i)}$, its gradient can be calculated as:

$$\bigtriangledown_\theta I^{(i)} = (y^{(i)} - \sigma^{(i)})(X^{(i)})^T$$

Therefore, the gradient is:

$$\bigtriangledown_\theta L = \sum_{i=1}^{m} (y^{(i)} - \sigma^{(i)})(X^{(i)})^T = (y - \sigma(X\theta))^T X$$

Since the l2 factor is $\lambda\beta^T\beta$, so add it to the cost function:

$$\bigtriangledown_\theta L = (y - \sigma(X\theta))^T X + 2\lambda\beta^T$$

4.2 (b)

Since the current dataframe has 46 attributes, so the output dimensions is (1 + 46 + 46 + 46 * 45 / 2 = 1128). 1 for 1, 46 for x_i, 46 for x_i^2, and 46 * 45 / 2 means a product of any two attributes.

4.3 (c)

The unregularised classification accuracy is 0.802 and regularised one is 0.83. We can see regularisation will actually improve its performance.

Unregularised confusion matrix is [[304 53][76 220]] and regularised confusion matrix is [[304 53][58 238]]. From here we can see TN has improved: from 220 to 238.

Compared to the original training set's classification accuracy without polynomial expansion, its performance reduced. The reasons could be: many features might not be useful enough and combinations of them will bring difficulty to converge. The default `max_iter` would not be able to converge and the parameter needs to be set manually.

4.4 (d)

Any algorithm that requires at least one first-derivative/gradient is a first order algorithm. Similarly, any algorithm that requires at least one second-derivative/gradient is a second order optimisation algorithm.

The `lbfgs` is a second-order optimisation since it uses Hessian matrix to iterate until converging. `newton-cg` is also a second-order optimisation, belonging to Newton's Method Family. `sag` is a first-order optimisation.

`sag` is better for large datasets. The reason for that is it uses a batch when updating weights rather than all instances. If the dataset is small, then using a batch would bring high variance and under-fitting problems. Only when the dataset is large, one batch can easily represent the whole dataset and the computing complexity is reduced. It can then fulfill its strengths.