CS5014 Machine Learning: Practical 1
Methodology: Credit Approval

170004680

University of St. Andrews

March 2021

Contents

1 Part 1: Data Processing **1**
- A How did you load and clean the data, and why? 1
- B How did you split the data into test/train sets, and why? 1
- C How did you process the data including encoding, conversion, and scaling, and why? . . . 1
- D How did you ensure that there is no data leakage? 2

2 Part 2: Training **2**
- A Train using `penalty='none'` and `class_weight='none'`. What is the best and worst classification accuracy you can expect and why? 2
- B Explain each of the following parameters used by `LogisticRegression` in your own words: `penalty`, `tol`, `max_iter`. 2
- C Train using balanced class weights (setting `class_weight='balanced'`). What does this do and why is it useful? 3
- D `LogisticRegression` provides three functions to obtain classification results. Explain the relationship between the vectors returned by `predict()`, `decision_function()`, and `predict_proba()`. 3

3 Part 3: Evaluation **3**
- A What is the *classification accuracy* of your model and how is it calculated? Give the formula. 3
- B What is the *balanced accuracy* of your model and how is it calculated? Give the formula. 4
- C Show the *confusion matrix* for your classifier for both unbalanced (2a) and balanced (2b) cases. Discuss any differences. 4
- D Plot the *precision-recall curve* and report the *Average Precision* (AP) for your algorithm. What is the relationship between AP and the PR curve? 5

4 Part 4: Advanced Tasks **6**
- A Set the `penalty` parameter in `LogisticRegression` to 'l2'. Give the equation of the cost function used by `LogisticRegression` as the result. Derive the gradient of this l2-regularised cost. 6
- B Implement a 2nd degree *polynomial expansion* on the data-set. Explain how many dimensions this produces and why. 7
- C Compare the results of regularised and unregularised classifiers on the expanded data and explain any differences. 8
- D Read the relevant references and answer: 8
  - D.1 Which of the three methods are first-order optimisation and which are second-order optimisation methods? 8
  - D.2 What are the differences between these two methods (first-order and second-order optimisation)? 8
  - D.3 Why does the package suggests that "sag" is faster for large data-sets? 8

1 Part 1: Data Processing

A How did you load and clean the data, and why?

The original data set was loaded using `read_csv` from the `pandas` package as the data set was provided in a comma-separated value (CSV) format. The use of this method allowed for missing values within the data set to be parsed as null (`NaN`) values that can be handled. Meaningful column names were provided to this method as the column indices. These column names mimic those described in '*crx.names*' to improve comprehension. The data set is retrieved as a `DataFrame`, which allows for elegant manipulation of the data set for pre-processing.

To clean the data, any rows containing null values were dropped; suitable values for null entries cannot be extrapolated from the data set to give to the learning algorithm. Similarly, infinite and invalid values (values not permitted for a non-contiguous domain) were checked for to be dropped; there were none. Duplicate rows in the data set were also checked for to be removed; there were also none. Duplicate rows in the data set are redundant for learning and could risk equivalent data being split into both the training and testing set (data leakage).

Further cleaning of the data set was not required as knowledge of the feature domains was limited. For example, it is not declared in '*crx.names*' whether contiguous domains have to be positive or within a given range.

B How did you split the data into test/train sets, and why?

The '*crx.data*' data set was split into a training and testing set using the `train_test_split` method from `sklearn`. Using this method, the proportion of the original data set that is used for the training and testing sets could be specified. The training set uses 80% of the original data set, with the remaining 20% being allocated to the testing set. This 4:1 split is common in practice and provides sufficient data for both training and evaluating the model. A seed was specified to randomly sample the original data set when splitting, as the original data set contains oscillating blocks of positive and negative cases (i.e., somewhat ordered). The seed was initially chosen at random but is kept the same in the program for reproducibility purposes. Furthermore, the split into the training and testing sets is stratified by the output ('A16'), which ensures that both sets contain approximately the same percentage of samples for each target class ('+' and '-') as the original (complete) data set. There is not a significant imbalance in the distribution of the target classes ('+' is 44.5%, '-' is 55.5%), so the improvement given by stratification could be minor.

C How did you process the data including encoding, conversion, and scaling, and why?

The logistic regression algorithm requires numerical inputs, thus, all categorical features ('A1', 'A4', 'A5', 'A6', 'A7', 'A9', 'A10', 'A12', and 'A13') were encoded to dummy variables. For each categorical feature, dummy columns for the feature were appended and then the original categorical feature column was removed. The dummy columns were created using the `get_dummies` method provided with the `pandas` package. This method creates a dummy column for each value in a categorical feature and assigns ones within the dummy column only where the original column contained the value. Many of the categorical features have equivalent domains, so the feature name that each dummy column originated from was appended as a prefix to the dummy column name for comprehension.

The output column ('A16') was converted such that positive cases are represented by '1' instead of '+', and negative cases are represented as '0' instead of '-'. Whilst this is not necessary for the logistic regression (can handle a categorical output), this enabled feature selection via correlation to occur.

Feature selection is performed on the training input set to make learning more tractable and efficient by removing redundancy. Correlation is used for feature selection, which retains features that are 'strongly' correlated with the output and 'weakly' correlated with all other features. As a result, the selected features inform the output and have a higher degree of linear independence. A threshold value of 0.3 was used for the minimum correlation that features must have with the output. Higher threshold values reduce the selected features to one or two, which is unlikely to perform well in the general case (no field knowledge to be certain). A threshold value of 0.9 was used for the maximum correlation that features must have with each other. Features above 0.9 have significant linear dependence, which introduces redundancy. This threshold is also used to retain one dummy variable from pairs of dummy variables corresponding to features with Boolean domains. This form of feature selection would not be performed in practice without strong field knowledge; however, it was carried out in this practical as an exercise. The feature selection also assumes that the original data set is representative of the general case, to justify the removal of correlated features.

The training/testing data sets contained feature values that were not well-spaced out and did not have well-defined limits, so scaling of the data was required. The existence of outliers was investigated to determine the type of scaling needed. To identify outliers, the Z-score of the selected features ('A8', 'A9_t', 'A10_t', and 'A11') was calculated. The Z-score of the features is the signed number of standard deviations that values are from the mean. Values that were more than 3 standard deviations from the mean were considered as outliers. There were 16 outliers for 'A8' and 6 outliers for 'A11' within the feature reduced training set. This meant that scaling via normalisation was unsuitable, so standardisation was used instead (`StandardScaler`).

D How did you ensure that there is no data leakage?

Data leakage has been prevented as no data dependant actions are completed before the data set is split into the training and testing set. Creating dummy variables for the features and changing the output encoding are fixed transformations (no data dependency). The removal of duplicate rows before splitting, if they exist, ensures that equivalent data can not accidentally end up in both the training and testing sets. Thus, none of the data in the training set has 'knowledge' of the data in the test set.

2 Part 2: Training

A Train using `penalty='none'` and `class_weight='none'`. What is the best and worst classification accuracy you can expect and why?

A classifier has been trained using `LogisticRegression` with `penalty='none'` and `class_weight='none'`. The pre-processed training input set and training output set were used for the fit of the model. The classification accuracy of the model for the training set is 0.874.

Classification data sets can be very imbalanced, so a classifier can get a good training accuracy by simply guessing all samples as the majority class, regardless of feature differences. In this instance, the majority negative class is 55.5% of the distribution. Therefore, the worst baseline training accuracy we can expect is 0.555.

The classifier was trained with no penalty (i.e., no regularisation), so the model is likely to have been over-fit to the training data. Over-fitting can be severe enough as to achieve the highest training accuracy of 1.0. This is not ideal as the model likely does not capture general trends in the data. Therefore, the best accuracy expected is between 0.555 and 1.0 (not inclusive).

B Explain each of the following parameters used by `LogisticRegression` in your own words: `penalty`, `tol`, `max_iter`.

- `penalty`: This parameter specifies the norm used in penalisation for a regularised classifier. The loss function is modified by adding a penalty term (determined by `penalty`). The penalty term effectively shrinks the coefficients in the regression towards zero; the only way the optimisation keeps the overall cost function minimum is to assign smaller values to the coefficients. This limits the flexibility of the model to avoid over-fitting, which can improve performance for new, unseen data sets.

- `tol`: This parameter specifies the tolerance for the stopping criteria of the model. If the stopping criteria has been met within the tolerance range for an iteration, then the learning algorithm is considered to have *converged*.

- `max_iter`: This parameter specifies the maximum number of iterations the logistic regression algorithm can perform to reach *convergence*. The algorithm stops when `max_iter` iterations is reached, regardless of whether the stopping condition is met and the algorithm has *converged*.

C Train using balanced class weights (setting `class_weight='balanced'`). What does this do and why is it useful?

A `LogisticRegression` classifier has been trained with: `penalty='none'`, `class_weight='balanced'`. The pre-processed training input set and training output set were used for the fit of the model. The classification accuracy of the model for the training set is 0.874.

Class imbalance refers to the data set being unevenly distributed in terms of positive ('+') and negative ('-') output cases. This is seen in the '*crx*' data as there is a majority negative class with 55.5% of the distribution and a minority positive class with 44.5% of the distribution. This skewed distribution in the output classes can lead to biases in the learning algorithm. Here, the majority negative class will be favoured, so reducing the miss-classification errors for the minority positive class is unfocused.

However, the training algorithm can be modified to account for the class imbalance. This is achieved by assigning weights to both the majority and minority output classes, which influences the classification when training to remove the bias. Miss-classification of the minority class is penalised more by setting a higher class weight whilst simultaneously reducing the class weight of the majority class. Here, 'balancing' the class weights means the model assigns class weights inversely proportional to their respective frequencies, via the following formula:

$$weight_j = \frac{n_{samples}}{(n_{classes} * n_{jsamples})}$$

, where w_j is the weight of each output class j, $n_{samples}$ is the total number of samples/rows in the data set, $n_{classes}$ is the number of unique classes in the output (2 for this binary classifier), and $n_{jsamples}$ is the total number of samples/rows in the data set of class j.

D `LogisticRegression` provides three functions to obtain classification results. Explain the relationship between the vectors returned by `predict()`, `decision_function()`, and `predict_proba()`.

- `decision_function()`: Given a set of samples (i.e., rows) with corresponding features, this function provides the predicted confidence score for each of the samples. The confidence score for each sample is proportional to the signed distance from the sample to the decision boundary (i.e., hyper-plane).
- `predict_proba()`: Given a set of samples (i.e., rows) with corresponding features, this function provides the probability estimates that each sample belongs to each output class. In this case, there is a single binary output, so the probabilities of both output classes (positive and negative) sums to one for each sample.
- `predict()`: Given a set of samples (i.e., rows) with corresponding features, this function provides the predicted output class for each sample.

For each sample, a positive confidence score (given by `decision_function()`) means that the `predict_proba()` result for the sample will estimate a greater percentage for the positive class than for the negative class. The larger the magnitude of a sample's confidence score, the greater the difference between the predicted probabilities of the output classes. The `predict()` result for the sample will be a positive classification as the estimated probability for the positive class is greatest.

3 Part 3: Evaluation

A What is the *classification accuracy* of your model and how is it calculated? Give the formula.

- Testing Set Classification Accuracy (`penalty='none'`, `class_weight='none'`) = 0.824

- Testing Set Classification Accuracy (`penalty='none'`, `class_weight='balanced'`) = 0.824

The classification accuracy (i.e. accuracy score) of the model is calculated using the following formula:

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

, where y is the true output set, $\hat{y}$ is the predicted output set, $\hat{y}_i$ is the predicted value at sample i, y_i is the corresponding true value at sample i, n is the total number of samples, and $1(x)$ is the indicator function (which is 1 when $\hat{y}_i = y_i$ and 0 otherwise).

Alternatively, we can calculate accuracy as the proportion of correct predictions against the total number of predictions:

$$accuracy = \frac{(TN + TP)}{(TN + TP + FN + FP)}$$

, where TP is the true positive count, TN is the true negative count, FP is the false positive count, and FN is the false negative count.

B What is the *balanced accuracy* of your model and how is it calculated? Give the formula.

- Testing Set Balanced Accuracy (`penalty='none'`, `class_weight='none'`) = 0.834
- Testing Set Balanced Accuracy (`penalty='none'`, `class_weight='balanced'`) = 0.834

The balanced accuracy (i.e., `balanced_accuracy_score`) of the model is calculated using the following formula:

$$balanced\_accuracy = \frac{1}{2}(\frac{TP}{TP + FN} + \frac{TN}{TN + FP})$$

, where TP is the true positive count, TN is the true negative count, FP is the false positive count, and FN is the false negative count.

C Show the *confusion matrix* for your classifier for both unbalanced (2a) and balanced (2b) cases. Discuss any differences.

The confusion matrix for the unbalanced classifier is as follows: $C_u = \begin{bmatrix} 53 & 19 \\ 4 & 55 \end{bmatrix}$

The confusion matrix for the balanced classifier is as follows: $C_b = \begin{bmatrix} 53 & 19 \\ 4 & 55 \end{bmatrix}$

Note that $C_{(0,0)}$ = TN, $C_{(0,1)}$ = FP, $C_{(1,0)}$ = FN, and $C_{(1,1)}$ = TP.

There are no differences in the confusion matrices between the unbalanced and balanced classifiers. This is likely because the original data set that the training and testing sets were formed from was only moderately imbalanced; the positive class is 44.5% of the distribution, and the negative class is 55.5% of the distribution. A more pronounced imbalance between the majority negative and minority positive classes would likely have shown a larger false negative count for the unbalanced classifier compared to the balanced classifier; the negative output class would have a bias leading to more positive cases being incorrectly identified as negative cases.

D Plot the *precision-recall curve* and report the *Average Precision* (AP) for your algorithm. What is the relationship between AP and the PR curve?

The precision-recall curve and average precision (AP) for both the unbalanced and balanced classifiers are shown in figures 1 and 2 respectively. In the figures, precision (p) is plotted as a function of recall (r), giving $p(r)$. Average precision is the average value of precision over the interval from $r = 0$ to $r = 1$:

$$AP = \int_0^1 p(r)dr$$

Thus, average precision is the area under the precision-recall curve. In `sklearn`, this integral is replaced with a finite sum: the weighted mean of precision achieved at each threshold, with the increase in recall from the previous threshold used as the weight:

$$AP = \sum_n (R_n - R_{n-1})P_n$$

, where P_n and R_n are the precision and recall at the nth threshold. Note that this is not interpolated, which explains why the use of `auc` (from `sklearn.metrics`) to get the area under the precision-recall curve is not equivalent to this result.

Figure 1: The following is the precision-recall curve for the unbalanced classifier with no penalty. The average precision of the classifier is also provided.

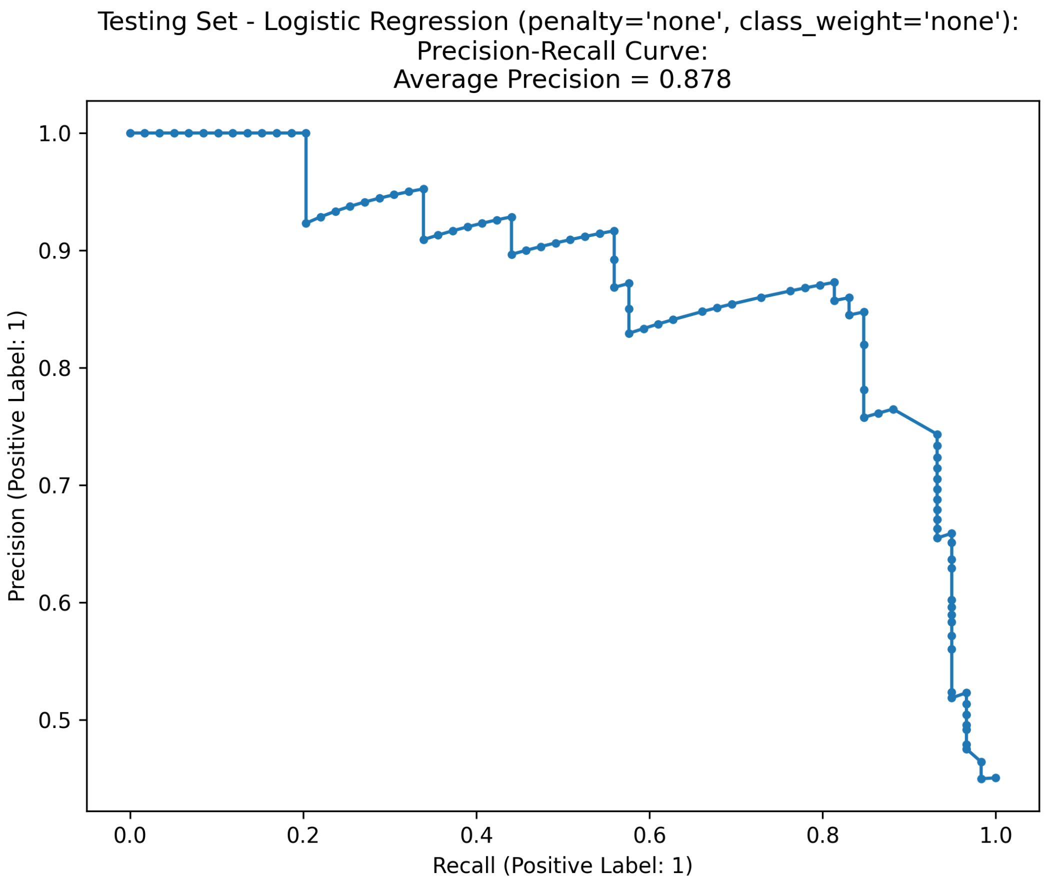
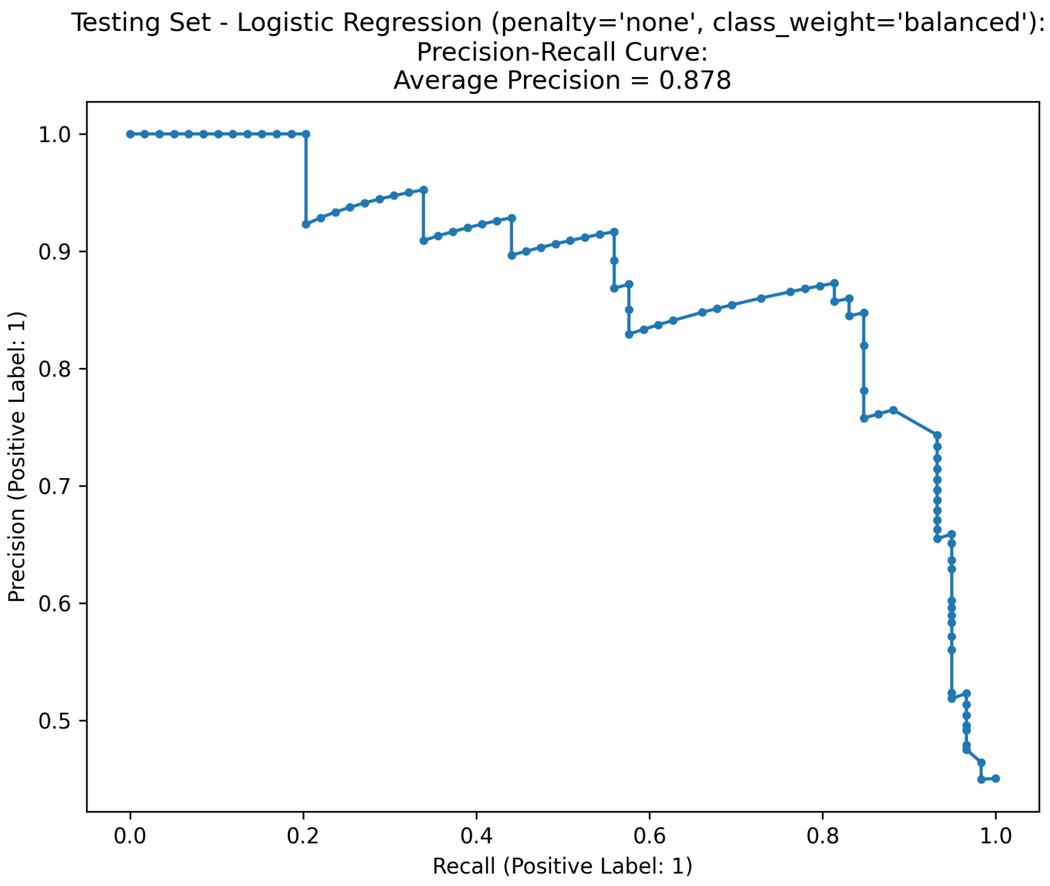


Figure 2: The following is the precision-recall curve for the balanced classifier with no penalty. The average precision of the classifier is also provided. The precision-recall curve is identical for the balanced and unbalanced classifiers.



4 Part 4: Advanced Tasks

A Set the `penalty` parameter in `LogisticRegression` to '12'. Give the equation of the cost function used by `LogisticRegression` as the result. Derive the gradient of this 12-regularised cost.

The following is the equation of the cost function used by `LogisticRegression` with L2 penalty by `sklearn`:

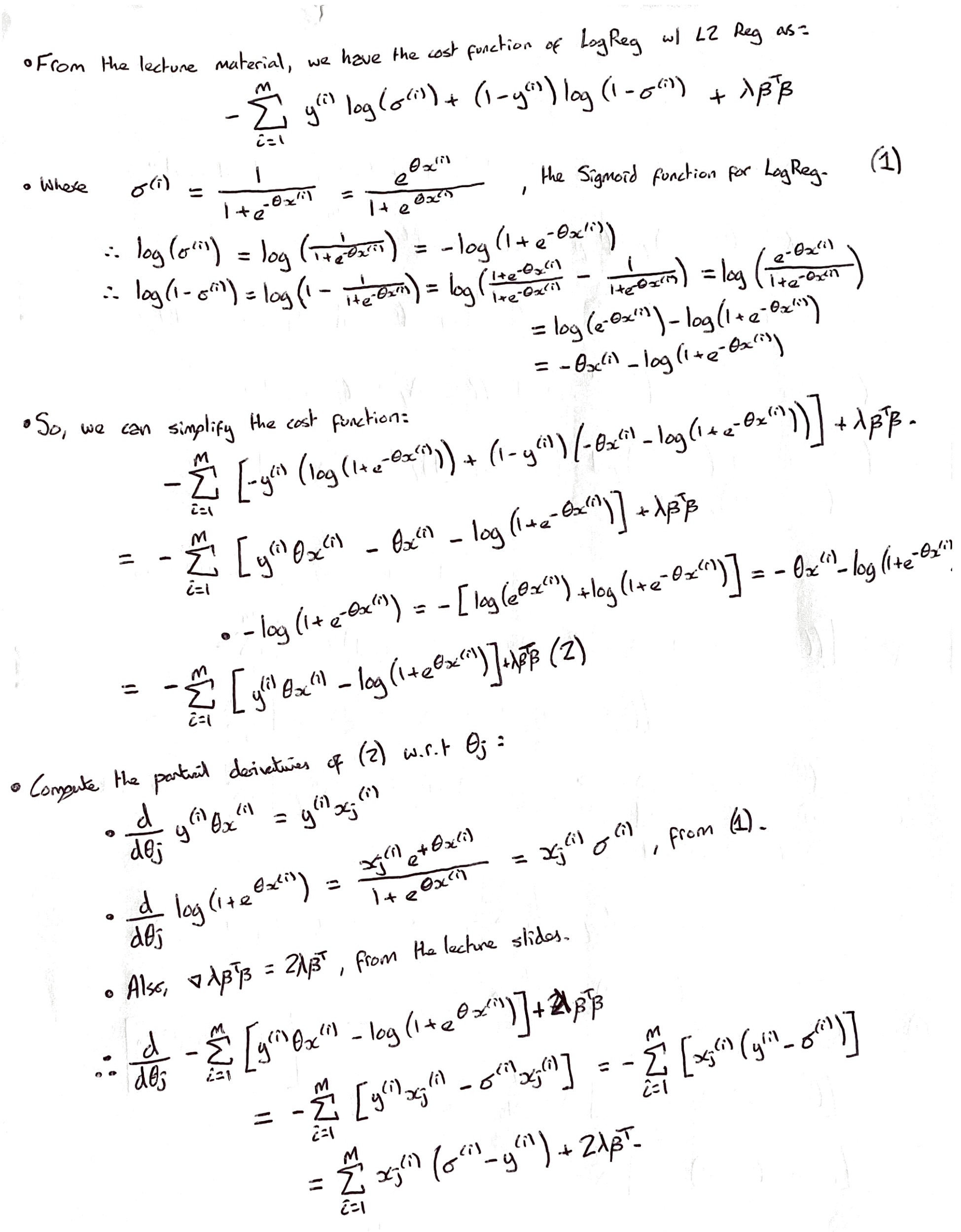
$$C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1) + \frac{1}{2} w^T w$$

The following is the cost function for L2 Logistic Regression, which is provided in the lecture material:

$$-\sum_{i=1}^{m} y^{(i)} \log(\sigma^{(i)}) + (1 - y^{(i)}) \log(1 - \sigma^{(i)}) + \lambda \beta^T \beta$$

An attempt at the gradient of the cost function (from the lecture material) is provided in figure 3, which has been derived with help from the 'Regularisation' lecture content and this resource.

Figure 3: Derivation of the gradient for the L2 Logistic Regression cost function.



- From the lecture material, we have the cost function of LogReg w/ L2 Reg as:

$$-\sum_{i=1}^{M} y^{(i)}\log(\sigma^{(i)}) + (1-y^{(i)})\log(1-\sigma^{(i)}) + \lambda\beta^T\beta$$

- Where $\sigma^{(i)} = \frac{1}{1+e^{-\theta x^{(i)}}} = \frac{e^{\theta x^{(i)}}}{1+e^{\theta x^{(i)}}}$, the Sigmoid function for LogReg. (1)

$$\therefore \log(\sigma^{(i)}) = \log\left(\frac{1}{1+e^{-\theta x^{(i)}}}\right) = -\log(1+e^{-\theta x^{(i)}})$$

$$\therefore \log(1-\sigma^{(i)}) = \log\left(1-\frac{1}{1+e^{-\theta x^{(i)}}}\right) = \log\left(\frac{1+e^{-\theta x^{(i)}}}{1+e^{-\theta x^{(i)}}} - \frac{1}{1+e^{-\theta x^{(i)}}}\right) = \log\left(\frac{e^{-\theta x^{(i)}}}{1+e^{-\theta x^{(i)}}}\right)$$

$$= \log(e^{-\theta x^{(i)}}) - \log(1+e^{-\theta x^{(i)}})$$

$$= -\theta x^{(i)} - \log(1+e^{-\theta x^{(i)}})$$

- So, we can simplify the cost function:

$$-\sum_{i=1}^{M}\left[-y^{(i)}\left(\log(1+e^{-\theta x^{(i)}})\right) + (1-y^{(i)})\left(-\theta x^{(i)} - \log(1+e^{-\theta x^{(i)}})\right)\right] + \lambda\beta^T\beta.$$

$$= -\sum_{i=1}^{M}\left[y^{(i)}\theta x^{(i)} - \theta x^{(i)} - \log(1+e^{-\theta x^{(i)}})\right] + \lambda\beta^T\beta$$

- $-\log(1+e^{-\theta x^{(i)}}) = -\left[\log(e^{\theta x^{(i)}}) + \log(1+e^{-\theta x^{(i)}})\right] = -\theta x^{(i)} - \log(1+e^{-\theta x^{(i)}})$

$$= -\sum_{i=1}^{M}\left[y^{(i)}\theta x^{(i)} - \log(1+e^{\theta x^{(i)}})\right] + \lambda\beta^T\beta \quad (2)$$

- Compute the partial derivatives of (2) w.r.t θ_j:
  - $\frac{d}{d\theta_j} y^{(i)}\theta x^{(i)} = y^{(i)}x_j^{(i)}$
  - $\frac{d}{d\theta_j}\log(1+e^{\theta x^{(i)}}) = \frac{x_j^{(i)}e^{+\theta x^{(i)}}}{1+e^{\theta x^{(i)}}} = x_j^{(i)}\sigma^{(i)}$, from (1).
  - Also, $\nabla\lambda\beta^T\beta = 2\lambda\beta^T$, from the lecture slides.

$$\therefore \frac{d}{d\theta_j} -\sum_{i=1}^{M}\left[y^{(i)}\theta x^{(i)} - \log(1+e^{\theta x^{(i)}})\right] + 2\lambda\beta^T\beta$$

$$= -\sum_{i=1}^{M}\left[y^{(i)}x_j^{(i)} - \sigma^{(i)}x_j^{(i)}\right] = -\sum_{i=1}^{M}\left[x_j^{(i)}(y^{(i)}-\sigma^{(i)})\right]$$

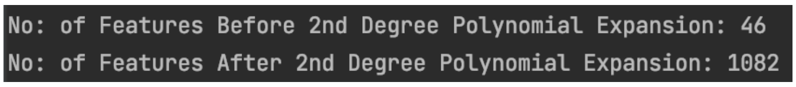
$$= \sum_{i=1}^{M} x_j^{(i)}(\sigma^{(i)} - y^{(i)}) + 2\lambda\beta^T.$$

B Implement a 2nd degree *polynomial expansion* on the data-set. Explain how many dimensions this produces and why.

A 2^{nd} degree polynomial expansion has been implemented over the features of the '*crx*' data set using `PolynomialFeatures` from `sklearn.preprocessing`.

The '*crx*' data set has 15 features, which is extended to 46 features when the dummy variable conversion occurs. The 2^{nd} degree polynomial expansion creates: a bias, features raised to the power of each degree (i.e., $A1^1$, $A1^2$, $A2^1$, $A2^2$, ..., $A15^2$), and interactions between all unique pairs of features (i.e., $A1 * A2$, $A1 * A3$, ..., $A14 * A15$). This expansion has $1 + 2(46) + \binom{46}{2} = 1128$ dimensions. However, for the polynomial expansion implemented, only the interaction features are produced; features that are products of at most 2 distinct input features (i.e., no $A1^2$, $A2^2$, ..., $A15^2$). As a result, the dimensions of the implemented expansion is $1 + 46 + \binom{46}{2} = 1082$. This is seen to be the case with figure 4.

Figure 4: Program output before and after expanding the '*crx*' data set (with dummy variables).



```
No: of Features Before 2nd Degree Polynomial Expansion: 46
No: of Features After 2nd Degree Polynomial Expansion: 1082
```

C Compare the results of regularised and unregularised classifiers on the expanded data and explain any differences.

Table 1: Table comparing classifiers using none versus L2 regularisation for the polynomial expanded testing set.

	Expanded Testing Set	
	No Regularisation	**L2 Regularisation**
Classification Accuracy	0.779	0.809
Balanced Accuracy	0.771	0.806
Confusion Matrix	$\begin{bmatrix} 61 & 11 \\ 18 & 41 \end{bmatrix}$	$\begin{bmatrix} 60 & 12 \\ 13 & 46 \end{bmatrix}$
Average Precision	0.723	0.821

The L2 regularisation classifier gives better performance for the expanded testing set; classification accuracy, balanced accuracy, and average precision are higher when using L2 regularisation versus no regularisation. This result was expected; regularisation reduces the flexibility of the model during training to prevent over-fitting on the training data. The L2 regularisation model generalises better, so performance will be improved for new, unseen data on average.

D Read the relevant references and answer:

Note that 'newton-cg' refers to 'Newton Conjugate Gradient', 'lbfgs' refers to 'Limited Memory Broyden Fletcher Goldfarb Shanno', and 'sag' refers to 'Stochastic Average Gradient'.

D.1 Which of the three methods are first-order optimisation and which are second-order optimisation methods?

- 'newton-cg': second-order optimisation method.
- 'lbfgs': second-order optimisation method.
- 'sag': first-order optimisation method.

D.2 What are the differences between these two methods (first-order and second-order optimisation)?

First-order optimisation methods use the gradient of the cost function (first derivative hence first-order) to iterate towards the global minimum. Second-order methods use the gradient and Hessian (matrix of second-order partial derivatives hence second-order) to step more directly towards the global minimum.

As first-order methods rely only on gradient information, whilst second-order methods consider both gradient and curvature, first-order methods do not perform comparably to second-order methods on average. To achieve this, second-order methods trade-off iterative cost for much fewer iterations in the optimisation.

D.3 Why does the package suggests that "sag" is faster for large data-sets?

For full gradient optimisation methods, the cost per iteration scales with the number of data points, n. However, stochastic gradient methods (e.g., 'sag') instead approximate the gradient using a random subset of the input, so the iterative cost is independent of n. As the size of n increases, there is no proportional increase in time taken for each iteration by 'sag', but this occurs with full gradient methods. As a result, 'sag' is faster for large data sets.