COMP9417: Homework Set #2

z5113817

University of New South Wales — July 18, 2021

All code for this homework set is available here.

Question 1

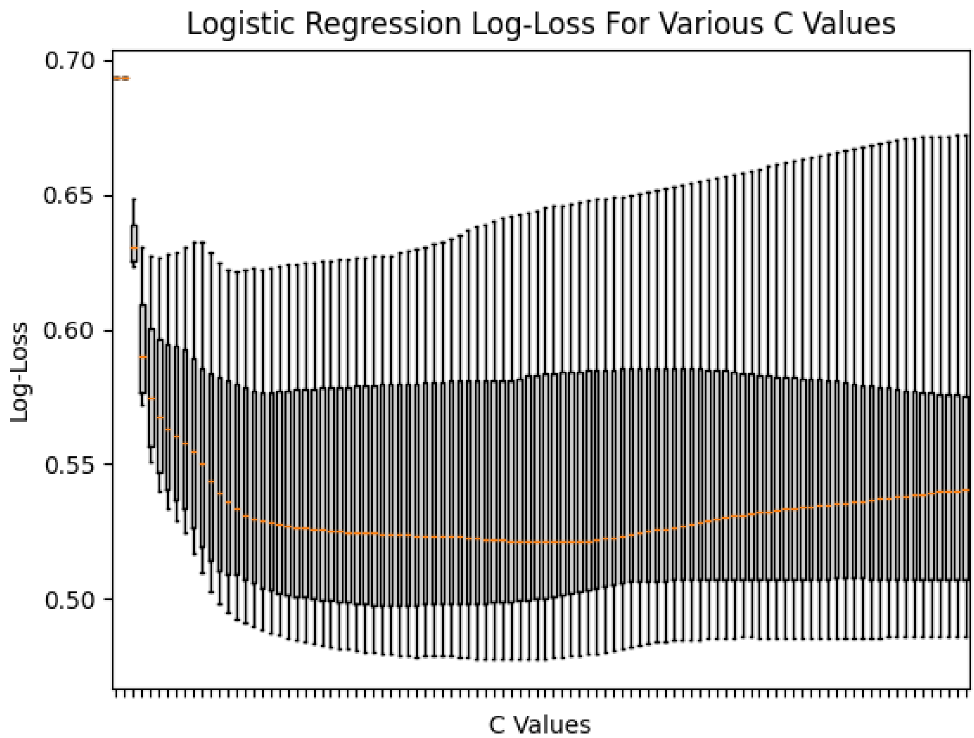
a

The possible values for both y_i and $\tilde{y}_i$ are binary. Even though they have different values ($y_i \in \{0, 1\}$ and $\tilde{y}_i \in \{-1, 1\}$), the objective of each logistic regression implementation is to divide the dataset into 2 classifications. Because of this, the actual value that each classification has will not affect the parameters that the regression is attempting to optimise ($(\hat{\beta}_0, \hat{\beta})$ and $(\hat{w}, \hat{c})$). Therefore the solutions for the parameters being minimised by each regression will be the same.

C is a hyper-parameter that adjusts the sensitivity that the model has to its coefficients. Compared with the standard LASSO parameter λ, C is a multiple of the Loss function whereas λ is a multiple of the Penalty.

b

Boxplot of testing accuracy for each value of C:



The value of C returning best results: **0.18794747474747472**
The testing accuracy of this model: **76%**

c

From GridSearchCV:

The value of C returning best results: **0.012219191919191918**

The testing accuracy of this model: **75.2%**

In our answer for b , we determined the "best" value of C as the value which corresponded to the average lowest *log-loss* value across all folds. The value from the *GridSearchCV* are different because, by default, it determines the "best" value of C as the one which corresponds to the average highest *score* across all folds [1].

We can modify the *GridSearchCV* class by providing our own metric for *scoring*. The following code is a scorer that uses the smallest *log-loss* value as its scoring metric.

```
scoring = make_scorer(
    log_loss,                # The sklearn implementation of log_loss
    greater_is_better=False, # A smaller log_loss is a better value
    needs_proba=True         # Calculating log_loss needs the probability predictions
)
```

This yields the following results:

*grid_lr.best_estimator_*: LogisticRegression(C=0.18188787878787877, penalty='l1', solver='liblinear')
*grid_lr.best_score_* (lowest log-loss): -0.5374067706086373

This value of *C* matches my value in b .

[1] See *scoring* parameter in documentation. If the estimator provided exposes a *score* method and a value for *scoring* is not provided, then *score* is used to determine the "best" value of C

Code For Questions

q1b

```python
def q1b():
    # Import data and take first 500 rows
    data = import_data(Q1_DATA_DIR)
    data = data.head(500)

    # Create folds
    folds = []
    fold_size = 50
    for i in range(0,10):
        folds.append(data[i*fold_size:i*fold_size+fold_size])

    # Create grid of 100 C values
    C_grid = linspace(0.0001, 0.6, 100)

    # Iterate over C values and train Logistic Model
    scores = []
    for C in C_grid.tolist():
        # Create model
        model = LogisticRegression(penalty="l1", solver="liblinear", C=C)

        # Iterate over each fold to train and test on each
        fold_scores = []
        for i in range(0, len(folds)):
            # Set up training and test data
            folds_train = folds.copy()  # Create a temp so that we don't mutate original 'folds'
            test_df = folds_train.pop(i)
            train_df = concat(folds_train)

            # Fit model to data
            clf = model.fit(train_df.drop("Y", axis=1), train_df["Y"])

            # Score data
            clf_probs = clf.predict_proba(test_df.drop("Y", axis=1))
            fold_score = log_loss(test_df["Y"], clf_probs)
            fold_scores.append(fold_score)

        scores.append(fold_scores)
```

```
    # Save boxplot
    fig, ax = plt.subplots()
    ax.boxplot(scores)
    ax.set_xticklabels("")      You, 21 hours ago • Finish q1c
    ax.set_title("Logistic Regression Log-Loss For Various C Values")
    ax.set_xlabel("C Values")
    ax.set_ylabel("Log-Loss")
    plt.savefig("./outputs/q1b_boxplot.png")

    # Record the value of C that returns the "best" CV score
    #
    # "Best" is taken to be the lowest average log-loss
    averages = list(map(lambda x : mean(x), scores))

    # Get index of lowest average
    i = averages.index(min(averages))

    # Map this to a C value
    C_best = C_grid[i]
    print(C_best)

    # Retrain this model and report its accuracy
    model = LogisticRegression(penalty="l1", solver="liblinear", C=C_best)

    # Let's test on the first fold and train on the remainder
    test_df = folds.pop(0)
    train_df = concat(folds)
    clf = model.fit(train_df.drop("Y", axis=1), train_df["Y"])

    print(clf.score(test_df.drop("Y", axis=1), test_df["Y"]))
```

q1c

```
def q1c():
    # Import data
    data = import_data(Q1_DATA_DIR)
    data = data.head(500)
    Xtrain = data.drop("Y", axis=1)
    Ytrain = data["Y"]

    # Create grid of 100 C values
    C_grid = linspace(0.0001, 0.6, 100)
    param_grid = { "C": C_grid }

    # Set our own scoring metric
    scoring = make_scorer(
        log_loss,
        greater_is_better=False,
        needs_proba=True
    )

    # Assignment code
    grid_lr = GridSearchCV(
        estimator=LogisticRegression(penalty='l1', solver='liblinear'),
        cv=10,
        param_grid=param_grid,
        scoring=scoring
    )
    grid_lr.fit(Xtrain, Ytrain)
    print(grid_lr.best_estimator_)
    print(grid_lr.best_score_)
```