# btt004_week4_assignment-solution

May 1, 2024

## 1 Assignment 4: Optimizing Logistic Regression

[1]:
```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import log_loss
from sklearn.metrics import accuracy_score
```

In this assignment, you will continue practicing the fourth step of the machine learning life cycle and train logistic regression models that will be used to solve a classification problem. You will build many variants, each one with a different value of the $C$ hyperparameter, which governs the amount of regularization used. Regularization is a process where we add a "penalty" to the original log loss function. This penalty is a function of the magnitudes of the weights learned in the Logistic Regression. The following shows the regularized log loss using what is called "L2" regularization.

$$Regularized\ LogLoss = -\frac{1}{N}\sum_{i=1}^{N}(y_i log(P_i) + (1-y_i) log(1-P_i)) + \frac{1}{C}\sum_{j=1}^{m} w_j^2$$

With L2 regularization, the penalty is the sum of the squares of the weights scaled by a constant $1/C$. When the hyperparameter $C$ is large, we reduce the weight of the penalty, which results in less regularization. You will build Logistic regressions with different values of $C$ and will check how this impacts the log loss.

You will complete the following tasks:

1. Build your DataFrame and define your ML problem:
    - Load the "cell2cell" data set into a DataFrame
    - Define the label - what are you predicting?
    - Identify features
2. Create labeled examples from the data set
3. Split the data into training and test data sets
4. Train logistic regression classifiers and evaluate their performances:

- Fit logistic regression models to the training data using different hyperparameter values per classifier
- Evaluate the accuracy of each model's predictions
- Plot and analyze the resulting log loss and accuracy scores

### 1.1 Part 1. Build Your DataFrame and Define Your ML Problem

**Load a Data Set and Save it as a Pandas DataFrame** We will work with the "cell2celltrain" data set. This version of the data set has been preprocessed and is ready for modeling.

[2]:
```
# Do not remove or edit the line below:
filename = os.path.join(os.getcwd(), "data", "cell2celltrain.csv")
```

**Task**: Load the data and save it to DataFrame `df`.

[3]:
```
# YOUR CODE HERE

#solution
df = pd.read_csv(filename, header=0)
```

**Define the Label** This is a binary classification problem in which we will predict customer churn. The label is the `Churn` column.

**Identify Features** To implement a Logistic Regression model, we must use only the numeric columns.

Task: Use the Pandas DataFrame select_dtypes() method to obtain all of names of columns that have a dtype of "float64." Save the result to a list named `feature_list`.

[4]:
```
# YOUR CODE HERE

### Solution:
feature_list = list(df.select_dtypes(include=['float64']).columns)
```

### 1.2 Part 2: Create Labeled Examples from the Data Set

Our data is fully prepared for modeling. We can now create labeled examples from DataFrame `df`.

Task: Obtain the feature columns from DataFrame `df` and assign to `X`. Obtain the label column from DataFrame `df` and assign to `y`.

You should have 51047 labeled examples. Each example contains 35 features and one label.

[5]:
```
# YOUR CODE HERE


#solution
y = df['Churn']
X = df[feature_list]


print("Number of examples: " + str(X.shape[0]))
print("\nNumber of Features:" + str(X.shape[1]))
```

```
print(str(list(X.columns)))
```

```
Number of examples: 51047

Number of Features:35
['MonthlyRevenue', 'MonthlyMinutes', 'TotalRecurringCharge',
'DirectorAssistedCalls', 'OverageMinutes', 'RoamingCalls', 'PercChangeMinutes',
'PercChangeRevenues', 'DroppedCalls', 'BlockedCalls', 'UnansweredCalls',
'CustomerCareCalls', 'ThreewayCalls', 'ReceivedCalls', 'OutboundCalls',
'InboundCalls', 'PeakCallsInOut', 'OffPeakCallsInOut', 'DroppedBlockedCalls',
'CallForwardingCalls', 'CallWaitingCalls', 'MonthsInService', 'UniqueSubs',
'ActiveSubs', 'Handsets', 'HandsetModels', 'CurrentEquipmentDays', 'AgeHH1',
'AgeHH2', 'RetentionCalls', 'RetentionOffersAccepted',
'ReferralsMadeBySubscriber', 'IncomeGroup', 'AdjustmentsToCreditRating',
'HandsetPrice']
```

### 1.3 Part 3: Create Training and Test Data Sets

Task: Create training and test data sets out of the labeled examples. Save the results to variables `X_train, X_test, y_train, y_test`.

[6]:
```
# YOUR CODE HERE

#solution
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,␣
 ↪random_state=1234)
```

Task: Check the dimensions of the training and test data sets.

[7]:
```
# YOUR CODE HERE

#solution
print(X_train.shape)
print(X_test.shape)
```

```
(34201, 35)
(16846, 35)
```

### 1.4 Part 4. Train a Logistic Regression Classifier and Evaluate the Model

The code cell below contains a function definition `train_test_LR()`.

Inspect the function definition `train_test_LR(X_train, X_test, y_train, y_test, c=1)`. The function expects the training and test data sets, as well as a value for hyperparameter *C*. Note that we supplied the value of 1 for *C* by default.

Task: Complete the function to make it work.

This function should: 1. train a Logistic Regression model on the training data 2. test the resulting model on the test data 3. compute and return two items: * the log loss of the resulting probability predictions on the test data * the accuracy score of the resulting predicted class labels on the test data

You will use the scikit-learn `LogisticRegression` class and will provide the arguments `C=c` when creating the model object.

[8]:
```
def train_test_LR(X_train, y_train, X_test, y_test, c=1):
    '''
    Fit a Linear Regression classifier to the training data X_train, y_train.
    Return the loss and accuracy of resulting predictions on the test set.
    Parameters:
        C = Factor that controls how much regularization is applied to the␣
↪model.
    '''

    # YOUR CODE HERE


    # Solution
     # 1. Create the  scikit-learn LogisticRegression model object below and␣
↪assign to variable 'model'
    model = LogisticRegression(C=c)


        # 2. Fit the model to the training data below
    model.fit(X_train, y_train)


        # 3. Make predictions on the test data using the predict_proba() method␣
↪and assign the result to the
        # variable 'probability_predictions' below
    probability_predictions = model.predict_proba(X_test)


        # 4. Compute the log loss on 'probability_predictions' and save the␣
↪result to the variable 'l_loss' below
    l_loss = log_loss(y_test, probability_predictions)


        # 5. Make predictions on the test data using the predict() method and␣
↪assign the result to the
        # variable 'class_label_predictions' below
    class_label_predictions = model.predict(X_test)


        # 6. Compute the accuracy score on 'class_label_predictions' and save␣
↪the result to the variable 'acc_score' below
    acc_score = accuracy_score(y_test, class_label_predictions)

    return l_loss, acc_score
```

### 1.4.1 Train a Model and Analyze the Results

Task: Use your function `train_test_LR()` to train one Logistic Regression classifier with the default value of hyperparameter C (`c=1`). Print the resulting log loss and accuracy score.

[9]:
```
# YOUR CODE HERE

# solution
loss, acc = train_test_LR(X_train, y_train, X_test, y_test)
print('Log loss: ' + str(loss))
print('Accuracy: ' + str(acc))
```

```
Log loss: 0.5878612157234173
Accuracy: 0.7097827377418972
```

## 1.5 Part 5. Train on Different Hyperparameter Values and Analyze the Results

Now we will adjust the $C$ regularization hyperparameter to check its impact on the model's log loss and accuracy. Hyperparameter `C` stands for the inverse of regularization strength. Smaller values specify stronger regularization and a simpler model. Larger values specify weaker regularization and a more complex model.

The code cell below creates a list `cs` of twenty values of $C$. Every item in the list has a value $10^i$ for every integer $i$ in the output of `range(-10,10)`. Run the code cell below and inspect the different values of $C$.

[10]:
```
cs = [10**i for i in range(-10,10)]
cs
```

[10]:
```
[1e-10,
 1e-09,
 1e-08,
 1e-07,
 1e-06,
 1e-05,
 0.0001,
 0.001,
 0.01,
 0.1,
 1,
 10,
 100,
 1000,
 10000,
 100000,
 1000000,
 10000000,
 100000000,
 1000000000]
```

Task: In the code cell below, loop over list `cs` and train and evaluate a different Logistic Regression model for every value of $C$. Use your function `train_test_LR()`. Print the resulting log

loss and accuracy scores per model.

We will want to create visualizations that plot the resulting log loss and accuracy score for every value of hyperparameter *C*. Considering this, save the resulting log loss values and accuracy scores that your function returns to two different lists. You will use these lists to create plots later.

[19]:
```
# YOUR CODE HERE


#solution (solutions may vary slightly)
ll_cs = []
acc_cs = []

for c in cs:
    ll, acc = train_test_LR(X_train, y_train, X_test, y_test, c)
    print('c of {0}:\nlog loss of {1}\naccuracy score of {2}'.format(c, ll,␣
 ↪acc))
    ll_cs.append(ll)
    acc_cs.append(acc)
```

```
c of 1e-10:
log loss of 0.6019882218839937
accuracy score of 0.710198266650837
c of 1e-09:
log loss of 0.6019879879688643
accuracy score of 0.710198266650837
c of 1e-08:
log loss of 0.6019856457586286
accuracy score of 0.710198266650837
c of 1e-07:
log loss of 0.6019623116656803
accuracy score of 0.710198266650837
c of 1e-06:
log loss of 0.6017368944992653
accuracy score of 0.710198266650837
c of 1e-05:
log loss of 0.6000102566181061
accuracy score of 0.710198266650837
c of 0.0001:
log loss of 0.5939550491932645
accuracy score of 0.710198266650837
c of 0.001:
log loss of 0.5882530046237049
accuracy score of 0.7104950730143654
c of 0.01:
log loss of 0.5876588226394373
accuracy score of 0.7099014602873086
c of 0.1:
log loss of 0.587835892808505
```

```
accuracy score of 0.7099014602873086
c of 1:
log loss of 0.5878612157234173
accuracy score of 0.7097827377418972
c of 10:
log loss of 0.5878648343540094
accuracy score of 0.7098420990146028
c of 100:
log loss of 0.5878651012583729
accuracy score of 0.7098420990146028
c of 1000:
log loss of 0.5878651279496574
accuracy score of 0.7098420990146028
c of 10000:
log loss of 0.5878651306188116
accuracy score of 0.7098420990146028
c of 100000:
log loss of 0.587865130885716
accuracy score of 0.7098420990146028
c of 1000000:
log loss of 0.5878651309124132
accuracy score of 0.7098420990146028
c of 10000000:
log loss of 0.5878651309150807
accuracy score of 0.7098420990146028
c of 100000000:
log loss of 0.587865130915354
accuracy score of 0.7098420990146028
c of 1000000000:
log loss of 0.5878651309153761
accuracy score of 0.7098420990146028
```

Now let's visualize the results.

Before we create plots, let's reformat the hyperparameter values in list `cs` so that they can be easily visualized in our plots. We will take the log 10 of the hyperparameter values and save it to a new list called `cs_log10`. Let's take a look at the original values and transformed values:

[12]:
```
cs_log10 = np.log10(cs)

print(cs)
print(cs_log10)
```

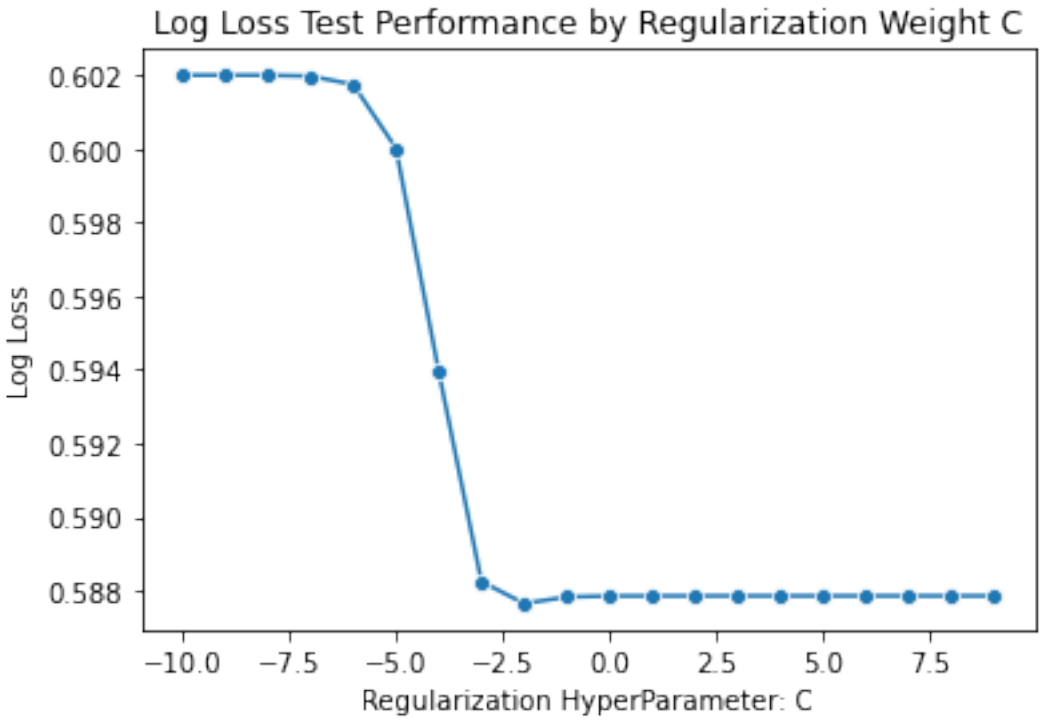
```
[1e-10, 1e-09, 1e-08, 1e-07, 1e-06, 1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100,
1000, 10000, 100000, 1000000, 10000000, 100000000, 1000000000]
[-10.  -9.  -8.  -7.  -6.  -5.  -4.  -3.  -2.  -1.   0.   1.   2.   3.
   4.   5.   6.   7.   8.   9.]
```

**Plot Log Loss** Task: Create a `seaborn` lineplot to plot the resulting log loss for every value of hyperparameter *C*. The hyperparameter *C* should be plotted on the x axis and the log loss should be plotted on the y axis. Label the x and y axes accordingly. Use the transformed values of hyperparameter *C* contained in the list `cs_log10`.

[14]:
```
# YOUR CODE HERE

# Solution (solutions may vary slightly)
fig = plt.figure()
ax = fig.add_subplot(111)
sns.lineplot(x=cs_log10, y=ll_cs, marker='o')


plt.title('Log Loss Test Performance by Regularization Weight C')
ax.set_xlabel('Regularization HyperParameter: C')
ax.set_ylabel('Log Loss')
plt.show()
```



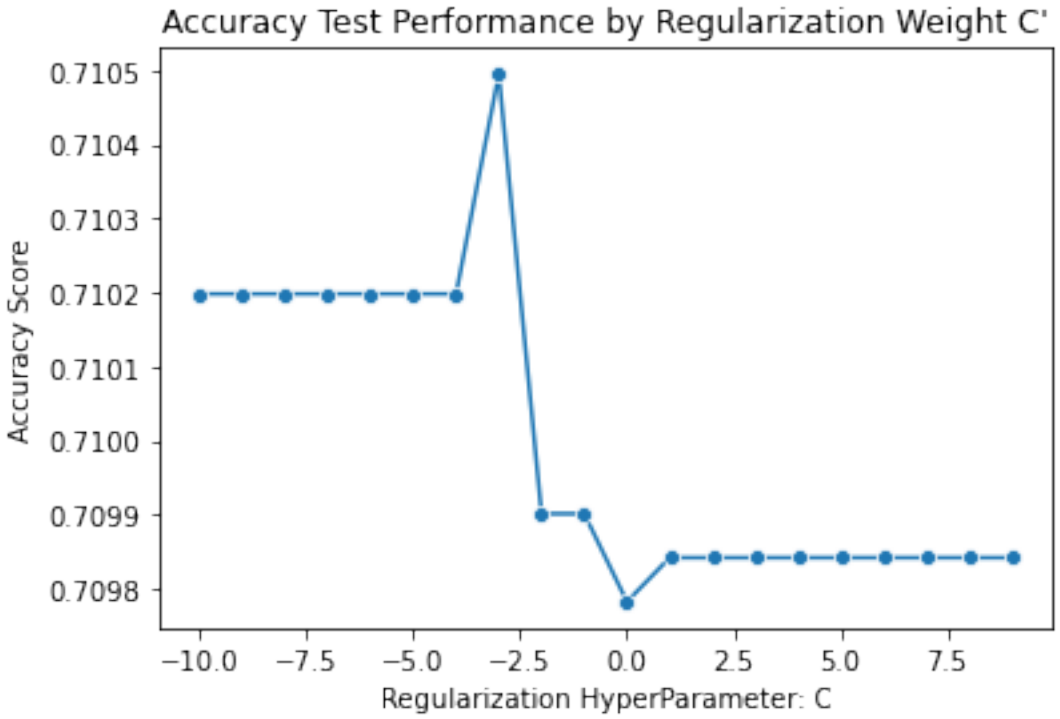

Analysis: Which value of *C* yields the best results, in terms of loss?
Solution: 0.01 (from the original cs list)

**Plot Accuracy** Task: Create a `seaborn` lineplot to plot the resulting accuracy score for every value of hyperparameter *C*. The hyperparameter *C* should be plotted on the x axis and the accuracy score should be plotted on the y axis. Label the x and y axes accordingly. Use the transformed values of hyperparameter *C* contained in the list `cs_log10`.

[15]:
```
# YOUR CODE HERE

# Solution (solutions may vary slighlty)
fig = plt.figure()
ax = fig.add_subplot(111)
sns.lineplot(x=cs_log10, y=acc_cs, marker='o')

plt.title("Accuracy Test Performance by Regularization Weight C'")
ax.set_xlabel('Regularization HyperParameter: C')
ax.set_ylabel('Accuracy Score')
plt.show()
```




Analysis: Which value of $C$ yields the best results, in terms of accuracy?
Solution: 0.001 (from the original cs list)