3. Diabetes Prediction

In this section, we will train a logistic regression model using stochastic gradient descent on the diabetes dataset.

The example assumes that a CSV copy of the dataset is in the current working directory with the filename **pima-indians-diabetes.csv**.

The dataset is first loaded, the string values converted to numeric and each column is normalized to values in the range of 0 to 1. This is achieved with the helper functions **load_csv()** and **str_column_to_float()** to load and prepare the dataset and **dataset_minmax()** and **normalize_dataset()** to normalize it.

We will use k-fold cross validation to estimate the performance of the learned model on unseen data. This means that we will construct and evaluate k models and estimate the performance as the mean model performance. Classification accuracy will be used to evaluate each model. These behaviors are provided in the **cross_validation_split()**, **accuracy_metric()** and **evaluate_algorithm()** helper functions.

We will use the **predict()**, **coefficients_sgd()** functions created above and a new **logistic_regression()** function to train the model.

Below is the complete example.

```
# Logistic Regression on Diabetes Dataset
from random import seed
from random import randrange
from csv import reader
from math import exp

# Load a CSV file
def load_csv(filename):
    dataset = list()
    with open(filename, 'r') as file:
        csv_reader = reader(file)
        for row in csv_reader:
            if not row:
                continue
            dataset.append(row)
    return dataset

# Convert string column to float
def str_column_to_float(dataset, column):
    for row in dataset:
        row[column] = float(row[column].strip())

# Find the min and max values for each column
def dataset_minmax(dataset):
    minmax = list()
    for i in range(len(dataset[0])):
        col_values = [row[i] for row in dataset]
        value_min = min(col_values)
        value_max = max(col_values)
        minmax.append([value_min, value_max])
    return minmax

# Rescale dataset columns to the range 0-1
def normalize_dataset(dataset, minmax):
    for row in dataset:
        for i in range(len(row)):
            row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] - minmax[i][0])
```

```

# Split a dataset into k folds
def cross_validation_split(dataset, n_folds):
    dataset_split = list()
    dataset_copy = list(dataset)
    fold_size = int(len(dataset) / n_folds)
    for i in range(n_folds):
        fold = list()
        while len(fold) < fold_size:
            index = randrange(len(dataset_copy))
            fold.append(dataset_copy.pop(index))
        dataset_split.append(fold)
    return dataset_split

# Calculate accuracy percentage
def accuracy_metric(actual, predicted):
    correct = 0
    for i in range(len(actual)):
        if actual[i] == predicted[i]:
            correct += 1
    return correct / float(len(actual)) * 100.0

# Evaluate an algorithm using a cross validation split
def evaluate_algorithm(dataset, algorithm, n_folds, *args):
    folds = cross_validation_split(dataset, n_folds)
    scores = list()
    for fold in folds:
        train_set = list(folds)
        train_set.remove(fold)
        train_set = sum(train_set, [])
        test_set = list()
        for row in fold:
            row_copy = list(row)
            test_set.append(row_copy)
            row_copy[-1] = None
        predicted = algorithm(train_set, test_set, *args)
        actual = [row[-1] for row in fold]
        accuracy = accuracy_metric(actual, predicted)
        scores.append(accuracy)
    return scores

# Make a prediction with coefficients
def predict(row, coefficients):
    yhat = coefficients[0]
    for i in range(len(row)-1):
        yhat += coefficients[i + 1] * row[i]
    return 1.0 / (1.0 + exp(-yhat))

# Estimate logistic regression coefficients using stochastic gradient descent
def coefficients_sgd(train, l_rate, n_epoch):
    coef = [0.0 for i in range(len(train[0]))]
    for epoch in range(n_epoch):
        for row in train:
            yhat = predict(row, coef)
            error = row[-1] - yhat
            coef[0] = coef[0] + l_rate * error * yhat * (1.0 - yhat)
            for i in range(len(row)-1):
                coef[i + 1] = coef[i + 1] + l_rate * error * yhat * (1.0 - yhat) * row[i]
    return coef

# Linear Regression Algorithm With Stochastic Gradient Descent
def logistic_regression(train, test, l_rate, n_epoch):
    predictions = list()
    coef = coefficients_sgd(train, l_rate, n_epoch)
    for row in test:
        yhat = predict(row, coef)
        yhat = round(yhat)
        predictions.append(yhat)
    return(predictions)

# Test the logistic regression algorithm on the diabetes dataset
```

```
seed(1)
# load and prepare data
filename = 'pima-indians-diabetes.csv'
dataset = load_csv(filename)
for i in range(len(dataset[0])):
	str_column_to_float(dataset, i)
# normalize
minmax = dataset_minmax(dataset)
normalize_dataset(dataset, minmax)
# evaluate algorithm
n_folds = 5
l_rate = 0.1
n_epoch = 100
scores = evaluate_algorithm(dataset, logistic_regression, n_folds, l_rate, n_epoch)
print('Scores: %s' % scores)
print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))
```

A k value of 5 was used for cross-validation, giving each fold 768/5 = 153.6 or just over 150 records to be evaluated upon each iteration. A learning rate of 0.1 and 100 training epochs were chosen with a little experimentation.

You can try your own configurations and see if you can beat my score.

Running this example prints the scores for each of the 5 cross-validation folds, then prints the mean classification accuracy.

We can see that the accuracy is about 77%, higher than the baseline value of 65% if we just predicted the majority class using the Zero Rule Algorithm.

```
Scores: [73.8562091503268, 78.43137254901961, 81.69934640522875, 75.81699346405229, 75.81699346405229]
Mean Accuracy: 77.124%
```

Extensions

This section lists a number of extensions to this tutorial that you may wish to consider exploring.

- **Tune The Example**. Tune the learning rate, number of epochs and even data preparation method to get an improved score on the dataset.
- **Batch Stochastic Gradient Descent**. Change the stochastic gradient descent algorithm to accumulate updates across each epoch and only update the coefficients in a batch at the end of the epoch.
- **Additional Classification Problems**. Apply the technique to other binary (2 class) classification problems on the UCI machine learning repository.

Did you explore any of these extensions?

Let me know about it in the comments below.

Review

In this tutorial, you discovered how to implement logistic regression using stochastic gradient descent from scratch with Python.

You learned.

- How to make predictions for a multivariate classification problem.
- How to optimize a set of coefficients using stochastic gradient descent.
- How to apply the technique to a real classification predictive modeling problem.