



머신러닝을 이용한 논도양 벗짚시용에 따른 토양탄소 축적량 추정에 관한 연구

The Study on Estimation of Soil Carbon Sequestration Analyzing Machine Learning at Paddy Fields Applied by Rice Straw

Abstract

서명철*, 조현숙, 김준환, 상완규, 이윤호, 신평, 이건휘

* 전라북도 완주군 이서면 국립식량과학원

최근 기후변화에 원인으로 화석연료 등 탄소배출이 가장 크게 대두되면서 온실가스 배출을 감축하거나 축적하여 온난화를 완화시키려는 노력들이 활발하게 이루어지고 있으며 농업분야에서도 온실가스 배출 감축과 함께 토양에 서의 온실가스 축적 능력에 대한 관심이 높아가고 있다. 벗짚은 가장 풍부한 유기자원으로 토양에 환원하여 비옥도를 증진시키는 가능 이외 장기적인 측면에서는 토양의 탄소함량을 증가시켜 온실가스를 축정 또는 저장하는 역할을 할 수 있어 이에 대한 장기적 축적시험과 이를 바탕으로 한 토양탄소 축적량 예측 기술 확립이 중요하다.

본 연구에서는 벗짚을 논 토양에 환원하였을 때 토양에서 분해되고 축적되는 양을 정량화하기 위하여 $50 \times 50 \text{ cm}^2$ 의 블록을 만들고 벗짚을 0, 0.5, 1, 1.5, 2 톤/㏊의 수준으로 4년간 사용하면서 벼를 4주씩 재배하였다. 월 1회 토양을 채취하여 토양내 전탄소 함량을 TOC 분석기로 분석하였으며 얻어진 결과를 가지고 머신러닝을 이용한 예측값을 분석하기 위하여 데이터 셋을 준비하였다. 머신러닝을 구동하기 위하여 텐서플로우의 선형회귀와 다변량 선형회귀 알고리듬을 이용하였다.

먼저 단변량 선형회귀의 경우 학습 횟수를 증가하여도 일반 통계에서 얻어지는 선형회귀 계수와 거의 동일한 결과를 얻을 수 있었다. 다변량 선형회귀를 위해 사용된 변수는 시간과 투입량 변수를 사용하였으며 결과값은 전탄소 함량을 데이터 셋으로 총 1201개의 데이터를 구성하였다. 추정식으로는 $W_1 \times X_1 + W_2 \times X_2 + b$ 로 하였으며 이때 W_1 과 W_2 의 비용함수가 최소화가 될 수 있도록 학습을 하였으며 배치의 크기는 100, 전체 데이터의 학습 (epoch)은 400회로 학습을 실시하였으며 비용함수를 최소화 하는 속도라 할 수 있는 learning rate는 0.001로 하였다. 총 1201개의 데이터 셋을 난수처리하여 200개의 검증시험 자료 셋으로 구성하였으며 나머지 1001개의 데이터 셋은 학습자료로 이용하였다. 다변량 선형회귀의 텐서플로우 분석 알고리듬은 아래 그림 1과 같다.

학습 초기 비용함수의 값은 0.34 이었으며 학습이 진행될수록 아래 그림 2와 같이 비용함수의 값이 감소하여 최종적으로는 0.02까지 감소하여 나름의 정확도는 높아지는 것을 알 수 있었다. 최종적인 W1과 W2의 계수값은 각각 0.2995와 -0.00017로 분석되었다.

시간과 투입량을 변수로 하여 다변량 선형회귀에 대한 텐서플로우를 이용하여 머신러닝을 구동하여 토양의 전탄소함량을 추정한 결과 $0.2995 * \text{시간} + -0.00017 * \text{투입량} + 0.604$ 의 결과를 얻었다. 그러나 이에 대한 학습자
로와 견즈자료를 통해 초점치의 전환도를 더 학습하는 것이 필요하다

결과 및讨論

Result and Discussion

Fig. 1

The Algorythm for multivariate regression with tensorflow to simulate machine learning

Fig. 2

The graph of simulation flow with tensor programmed by multivariate regression training

```

In [1]: with tf.Session() as sess:
    sess.run(init_op)

    log_dir = "./board/sample3"
    os.system("taskkill /f /im cmd.exe")

    if tf.gfile.Exists(log_dir):
        tf.gfile.DeleteRecursively(log_dir)

    writer = tf.summary.FileWriter(log_dir, sess.graph)

    coord = tf.train.Coordinator()
    threads = tf.train.start_queue_runners(coord=coord)

    # For matplotlib
    plt.figure(1)
    plt.title("Multivariate Linear Regression")
    plt.xlabel("Number of iterations")
    plt.ylabel("Cost_C")
    plt.ylim(0)

    try:
        step = 0
        while not coord.should_stop():
            training_set = sess.run(features)

            X_data = training_set[:, 0:-1]
            y_data = training_set[:, -1]

            # Mean Normalization
            X_data, mu, sigma = featureNormalization(X_data)

            # Add Bias term to data X
            X_data = addBiasTerm(X_data)

            sess.run(train, feed_dict={X: X_data, y: y_data})
            cur_cost = sess.run(cost, feed_dict={X: X_data, y: y_data})

            summary = sess.run(tb_merged, feed_dict={X: X_data, y: y_data})
            writer.add_summary(summary, step + 1)

            # For matplotlib
            iter_history[step] = step
            cost_history[step] = cur_cost
            print("iteration = ", step + 1, "cost function = ", cur_cost, sess.run(W), sess.run(b))

            step += 1

    except tf.errors.OutOfRangeError:
        print("Done -- epoch limit reached.")
        print("learning_rate=", learning_rate)
        print("batch size=", batch_size)
        print("num_of_epochs=", num_of_epochs)
        print("iterated=", step)

        print("\nW =", sess.run(W))

        feed_x = tf.constant([[1, (2104 - mu[0]) / sigma[0], (3 - mu[1]) / sigma[1]]], dtype=tf.float32)
        answer = sess.run(tf.matmul(feed_x, W))

        #print(sess.run(feed_x), " -> ", answer)

        # for matplotlib
        plt.plot(iter_history[0:step], cost_history[0:step])
        plt.show()

    finally:
        coord.request_stop()

```

Fig. 3

Simulation of training of multivariate regression on changes of soil total carbon

Fig. 6

Changes of cost function according to training of parameter to time and input
(left) no calibration (right) with calibration

```
iteration = 1 cost_function =  0.3366913 [[ 0.36750054]
[-0.5367747 ]] [0.672018]
iteration = 2 cost_function =  0.33374587 [[ 0.36724988]
[-0.53570116]] [0.6717673]
iteration = 3 cost_function =  0.3300046 [[ 0.3669814 ]
[-0.53462976]] [0.67149884]
iteration = 4 cost_function =  0.33127 [[ 0.36674824]
[-0.5335605 ]] [0.6712657]
iteration = 5 cost_function =  0.32649946 [[ 0.36647874]
[-0.5324934 ]] [0.67099625]
iteration = 6 cost_function =  0.32474697 [[ 0.3662169]
[-0.5314284]] [0.6707344]
iteration = 7 cost_function =  0.32390755 [[ 0.36593407]
[-0.5303655 ]] [0.6704516]
iteration = 8 cost_function =  0.32100716 [[ 0.36567748]
[-0.5293048 ]] [0.670195]
iteration = 9 cost_function =  0.31966132 [[ 0.3653745 ]
[-0.52824616]] [0.669892]
iteration = 10 cost_function =  0.3198021 [[ 0.36509976]
```

```
iteration = 4003 cost_function =  0.017965019 [[ 2.9953876e-01]
[-1.7791767e-04]] [0.6040546]
iteration = 4004 cost_function =  0.023332126 [[ 2.9952157e-01]
[-1.7756183e-04]] [0.60403734]
Done -- epoch limit reached
```

