Predicting Droughts using Weather Data with Regression

Brenner Lim
University of California, San Diego
b1lim@ucsd.edu

Steve Gisqu
University of California, San Diego
sgisqu@ucsd.edu

ECE 196, Spring 2021, April 28, 2021

ABSTRACT

This paper will go over the problem-solving and solution to the accurate prediction of droughts solely using weather data through machine learning technique of regression. The algorithm that was used was Stochastic Gradient Descent Regression and was implemented with scikit-learn's library. Although the solution ended up being unsuccessful, it highlights some important things to consider when working with weather data and data imbalancing.

1. Introduction

One of the main but unappreciated uses of technology is the capability to detect natural disasters before they occur. These tools include seismometers that detect seismic movements generated by earthquakes, and satellite imagery in order to monitor hurricanes. The importance of detecting natural disasters lies in the fact that their dangerous conditions and effects can be mitigated with preparation and prior emergency notice. However in many cases, these tools can only detect natural disasters right before they occur.

With these in mind, we sought out to create a machine learning algorithm that would be able to predict natural disasters before they occur. Out of the natural disasters that occur in California, the two most prominent are droughts and earthquakes. Due to the mass amounts of data collected from the weather, we sought out to create a machine learning model that can predict if the area is in a drought.

2. Problem-Solving Approach

2.1 Dataset

The first step before the project can take fruition is to find a sizable dataset to train, validate, and test our model comprehensively. The dataset we selected is **Predict Droughts using Weather & Soil Data** on Kaggle.com. This dataset encompasses daily weather data readings starting January 1st, 2000 to the end of 2020 in the United States. The goal of the dataset is to predict droughts and classify them into 4 categories based on severity. Days that are not categorized as a drought are instead represented as null values, or *NaN*. Their classifications are stored in a column as "score". Despite the large amounts of samples, this dataset was imbalanced with the majority being classified as None.

Category	Description	Possible Impacts
D0	Abnormally Dry	Going into drought: ▪ short-term dryness slowing planting, growth of crops or pastures Coming out of drought: ▪ some lingering water deficits ▪ pastures or crops not fully recovered
D1	Moderate Drought	▪ Some damage to crops, pastures ▪ Streams, reservoirs, or wells low, some water shortages developing or imminent ▪ Voluntary water-use restrictions requested
D2	Severe Drought	▪ Crop or pasture losses likely ▪ Water shortages common ▪ Water restrictions imposed
D3	Extreme Drought	▪ Major crop/pasture losses ▪ Widespread water shortages or restrictions
D4	Exceptional Drought	▪ Exceptional and widespread crop/pasture losses ▪ Shortages of water in reservoirs, streams, and wells creating water emergencies

[1]

The data was already conveniently split in training, validation, and testing csv files for machine learning. In order to manipulate this data, we used Pandas in order to import it into Jupyter Notebook.

2.2 Design Process

After selecting the dataset, we know that we have to utilize a supervised learning algorithm since the data has been labeled with clearly defined inputs and outputs. Under supervised learning, our problem could be viewed as either a classification or a regression problem. Since the drought's severity is measured as a continuous

score value from 0 to 4 and we wanted to stay as close to the original data, we found it more reasonable to treat it as a regression problem.

For the specific implementation, we looked into different regression algorithms included in scikit-learn, a popular machine learning library for Python. These include linear approaches to regression such as Ordinary least squares, Lasso, ElasticNet, and Stochastic Gradient Descent Regression (SGD Regression) and nonlinear models such as Support Vector Regression (SVR). Due to our lack of familiarity with machine learning and knowing that the number of training samples are in the millions, we decided to use SGD Regression due to its efficiency with large training samples [2].

2.2.1 Stochastic Gradient Descent Regression

The first assumption when we were working with Stochastic Gradient Descent Regression, or SGD Regression, is that the features we were analyzing are linearly correlated to the output, and in this case, the output is a continuous scalar number. With this method, the model can be represented as a linear function with an input vector x multiplied or dot produced with weight matrix ω with a bias b:

$$f(x) = w^T x + b \qquad [2]$$

By fine tuning the weights or parameters based on a loss function that measures the difference between the expected and the predicted output, the model should approach the most optimal weights that'll be able to accurately predict the output. For simplicity, we went with the squared-loss function,

$$\mathrm{SE} = (\mathrm{Y}_i - \hat{Y}_i)^2 \qquad [3]$$

where Y_i is the expected output and $\hat{Y}_i$ is the predicted output.

2.2.2 Stochastic Gradient Descent

For SGD Regression to optimize the weights based on the loss function, it implements Stochastic Gradient Descent (SGD). SGD is the optimization method used to iteratively approach the optimal function by finding the gradient of the regularized training error, 'E', given below. L refers to the loss function, R is known as the regularization term, or the penalty, and **α** us a non-negative hyperparameter.

$$E(w,b) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, f(x_i)) + \alpha R(w) \qquad [2]$$

The penalty we opted to use is the L2 norm. Also known as the Euclidean norm, it calculates the weight's magnitude squared. The penalty's function is to make sure that the weights stay near zero and penalizes large weights. As a result, features that may not be as important will approach zero rather than having the other weights approaching infinity [3].

$$R(w) := \tfrac{1}{2}\textstyle\sum_{i=1}^{n} w_i^2 \qquad [2]$$

After computing the gradient, it is multiplied with a set learning rate, η and subtracts from the current weights, ω, in order to get our new weights for the next iteration.

$$w \leftarrow w - \eta(\alpha\frac{\partial R(w)}{\partial w} + \frac{\partial L(w^T x_i + b, y_i)}{\partial w}) \qquad [2]$$

2.2.3 PCA Dimensionality Reduction

Due to the multiple features of the dataset, we decided to use Principal component analysis (PCA) in order to reduce the dimensionality of the problem and make processing and training more efficient. PCA uses Singular Value Decomposition in order to project multiple features into a lower dimensional space [5].

2.2.4 RobustScaler

One of the main concerns for using SGD Regression is its sensitivity to feature scaling. This means that if each feature isn't normalized to have mean 0 and unit variance, the algorithm will be biased by features with large numbers [6].

The reason we used RobustScaler rather than the other types of scalers (StandardScaler, MinMaxScaler, etc.) is due to its robustness with outliers. By basing its standardization on the percentiles, it becomes less influenced by any outliers that may occur in the data [6].

3. Solution

Our solution was to use SGD Regression in order to output a single continuous scalar, which represents the severity of drought from the weather data with 0 being no drought whatsoever, and numbers 1 to 5

corresponding to the 5 drought severity categories D0 to D4. It is important to note that this output “score” value is a continuous number (1.203, 3.705) and not discrete integers in the original dataset.

3.1 Preprocessing Data

After importing our training, validation, and testing CSV files into Jupyter Notebook via Pandas as dataframes, we had to format and split each one into their respective inputs and outputs.

In the dataset, the targeted value column is labelled as “score”, so we were able to copy them to new data frames train_y, test_y, and val_y. However, the current range of numbers in “score” aren’t usable since days without droughts are represented as null values, or NaN. In order to format it to have days without droughts represented as 0 and categories D0 to D4 as 1-5, we can add 1 to the output. Since null values aren’t affected by this, we could replace them with zeros afterwards.

```
# Formatting scores into own dataframes
train_y = train['score'] + 1
train_y.fillna(0,inplace=True)

test_y = test['score'] + 1
test_y.fillna(0,inplace=True)

val_y = val['score'] + 1
val_y.fillna(0,inplace=True)
```

[7]

For the inputs, we had to drop columns “date” and “score” since they were not relevant to the input features we wanted to analyze. These input data frames are named train_x, test_x, and val_x.

After initializing our input and output data structures, we then used RobustScaler from sklearn.preprocessing in order to normalize our input features to have mean 0 and unit variance. This is done by fitting the scaler to our train_x, and transforming each input. The fitting is only done once with train_x in order to scale all inputs to the same transformation. This transformation returns a numpy array of our input features instead of the pandas Dataframe.

```
# Normalize the features with train_x and save transform into 'scaler'
scaler = RobustScaler().fit(train_x)

# Apply transformation to each input matrix
train_x_norm = scaler.transform(train_x)
test_x_norm = scaler.transform(test_x)
val_x_norm = scaler.transform(val_x)
```

[7]

After feature scaling, we can reduce the dimensionality of our inputs with PCA. We chose PCA to return our features into 2 dimensions, or n_components = 2, to allow us to plot our data later on while making the training process less demanding.

3.2 Performance and Outcome

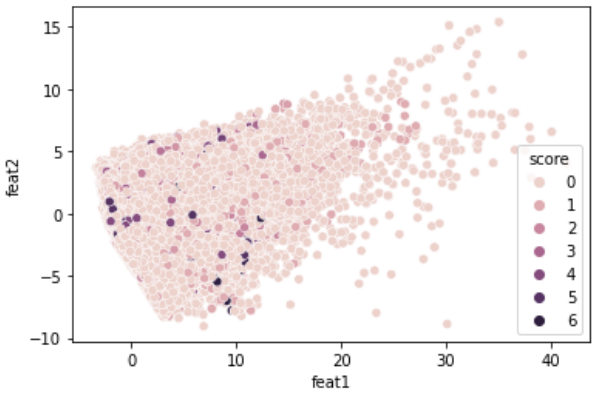
The performance of the SGD Regression model was unsatisfactory with an R^2 score of -0.03799485920837, which means that we don’t have the right approach to our problem. Although our model was able to handle the large training and test data sizes efficiently, the accuracy highlights a flaw in the assumption that the dataset would be linear.

This performance could also be attributed to the large data imbalance, where the majority of the samples are classified as not in a state of drought.

4. Conclusion

Although the project’s performance is lackluster, it is noted that it can be improved with the following:

1) We approached it with the assumption that the problem would be solved with a linear approach to regression. However, based on the graph of the input’s below, it can be inferred that the data isn’t linear.



2) Since the classes are imbalanced, the dataset can be fixed by minimizing the number of “no drought” samples or adding more “drought” samples.

3) Since the dataset is structured so that each sample is taken everyday, a machine learning algorithm like Long Short-Term Memory (LSTM) that uses a neural network can be used in order to get a rolling average over a set amount of time would perform better instead of our approach to predicting droughts based on a day’s worth of information [8].

4) Although most of the hyperparameters we utilized were not stated or changed from the ones SGDRegressor defaults to, we hypothesized that the large degree of error wouldn't be fixed by fine tuning these parameters.

4.1 Setbacks and Challenges

One of the first challenges for this project was our inexperience with Machine Learning in general. This meant that understanding the documentation of scikit-learn was very cumbersome and difficult. However, this helped us learn more about the hyperparameters and nuances of machine learning.

The second challenge of the project was deciding which model to implement. This also is correlated to our inexperience with machine learning. In the end, we decided to use SGD Regression due to its simplicity and robustness with large datasets.

A setback we faced was that any processing of the data in its entirety would freeze the computer and would make us restart due to the large sizes of data. Instead of importing all of the data and our work onto datahub.ucsd.edu, we worked with a smaller sample size of around 400,000 training samples for testing our implementation.

References:

[1] Minixhofer, Christoph. "Predict Droughts Using Weather & Soil Data." Kaggle, 3 Mar. 2021, www.kaggle.com/cdminix/us-drought-meteorological-data.

[2] "1.5. Stochastic Gradient Descent¶." 1.5. Stochastic Gradient Descent - Scikit-Learn 0.19.1 Documentation, sklearn.org/modules/sgd.html.

[3] Bernstein, Matthew. Squared Loss. 2017, pages.cs.wisc.edu/~matthewb/pages/notes/pdf/lossfunctions/SquaredLoss.pdf.

[4] Kassambara, and Chandrakumaran. "Penalized Regression Essentials: Ridge, Lasso & Elastic Net." STHDA, 11 Mar. 2018, www.sthda.com/english/articles/37-model-selection-essentials-in-r/153-penalized-regression-essentials-ridge-lasso-elastic-net/.

[5] "Sklearn.decomposition.PCA¶." Scikit, scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html.

[6] "Compare the Effect of Different Scalers on Data with Outliers." Scikit, scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html?highlight=scaler+differences

[7] SGisqu. "SGisqu/ECE-196-SPR21." GitHub, github.com/SGisqu/ECE-196-SPR21.

[8] "Long Short-Term Memory." Wikipedia, Wikimedia Foundation, 25 Mar. 2021, en.wikipedia.org/wiki/Long_short-term_memory.