HIGH SCHOOL EDITION
Journal of Student Research
Volume 12 Issue 1 (2023)

How Can Coaches and Players Tell the Outcome of a Pitch Based On Observed Data?

Aarushi Kulkarni

[1]Polygence, USA
#NO Advisor

ABSTRACT

Machine learning and the development of artificial intelligence continues to grow at a rapid pace with the age of technology and computer science jobs growing in demand. With this development the use of predictive technology becomes more realistic to make systems more efficient. For example, in the future of the medical field, a software program may diagnose patients given an input of symptoms and previous health issues more effectively than even humans can.

Sports are an area with vast possibilities of outcomes, meaning a lot of predictions to be made in all sorts of sports. Humans don't have the analytical ability to examine every aspect of each player to predict the next situation in a sports game. For example in baseball and softball, there are thousands of possible scenarios that can occur from a singular pitch.

How can coaches and players tell the outcome of a pitch based on observed data?

A simple machine learning predictive model using a technique called gradient descent takes features and examples as input in order to predict one of a couple possible outcomes for a specific pitch with its various features. Results show that upon running such programs numerous times with different iterations the model actually grows stronger in accuracy.

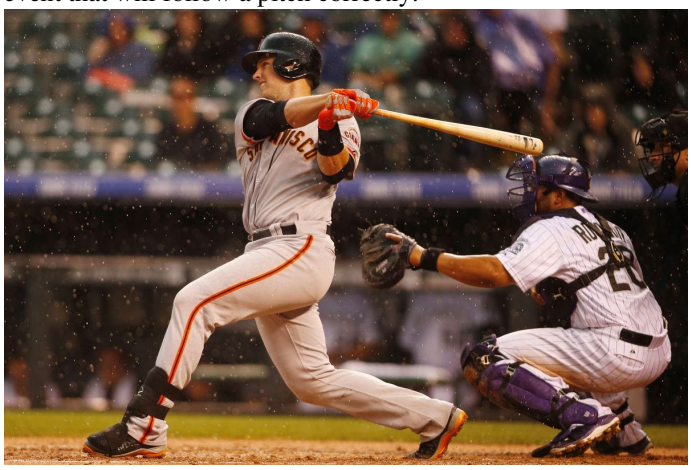
Introduction

In a baseball or softball game, the people on defense do their best at each batter to adjust positioning based on what they think the outcome of the specific pitch will be. The machine learning model predicts the event which occurs based on a pitch type, release position, and release speed of the pitch in a baseball scenario. An event would equate to a strikeout, a strike, a ball, a hit, or an out. For "hit", for hit specifies what type of hit, as in single, double, etc.)

Background

If the pitch called to throw is a changeup (a slow pitch) the defense will move to the right if it is a right-handed batter, and to the left otherwise, because there is a bigger chance that the batter will hit the slower pitch to that area. However, the defense makes mistakes which costs them in the overall game.

Coaches and team managers' goal is to maximize the amount of people getting out on the opposing team, and can do so if they know the outcome of a certain pitch for a player by telling their pitcher to throw the particular pitch. Again, this is extremely difficult for a human to predict accurately. Personally, the defense and coaches almost never predict the event that will follow a pitch correctly.



Method

The project is split into two parts for numerical data as input: a Linear Regression machine learning model implemented by scratch on one hand, and on the other a built-in Python machine learning library (scikit-learn) used to internally handle the linear regression for the prediction. Both implementations split the original dataset into 80 percent training and 20 percent testing data. They use training data to train on and the testing data to test the trained model.

Dataset

The dataset includes statistical numerical data about the "at bats" of one specific player, where one example includes a list of features and characteristics regarding the pitch (release speed with respect to the horizontal/vertical, acceleration, release position, pitch zone, etc.), and the outcome of the pitch with those specific attributes.

There are eleven classes for the Multi-Class Classification, meaning eleven possible outcomes of a pitch that can be predicted which are used as the ground truth values.

	description	release_speed	release_pos_x	zone	vx0	vy0	ax	ay	release_spin_rate	effective_speed	release_pos_y
0	swinging_strike	95.7	-2.3581	14.0	10.5318	-138.4377	-12.0431	34.6021	2270.0	95.025	54.0129
1	called_strike	96.3	-2.5577	9.0	9.6649	-139.7614	-11.7176	26.7988	2252.0	96.812	54.1114
2	called_strike	95.2	-2.3861	2.0	9.2416	-138.0840	-17.6628	28.7817	2291.0	95.468	54.0285
3	ball	93.4	4.2502	14.0	-10.6258	-135.1424	23.4366	30.9796	2175.0	91.825	54.7309
4	called_strike	92.8	4.1144	14.0	-13.1954	-134.2390	19.6908	28.6883	2171.0	91.790	54.5228
...	...	...	...	...	...	...	...	...	...	...	...
3431	ball	85.6	-1.9659	14.0	6.9600	-124.3710	-2.8210	23.6340	1552.0	84.459	54.8144
3432	hit_into_play_score	87.6	-1.9318	4.0	4.2870	-127.4520	-1.9720	24.6940	1947.0	86.412	54.8064
3433	ball	87.2	-2.0285	14.0	7.4910	-126.6650	-6.3930	21.9520	1761.0	86.368	54.7770
3434	foul	79.7	-1.7108	4.0	1.2540	-116.0620	5.1840	21.3280	2640.0	77.723	55.4756
3435	called_strike	93.2	-1.8476	8.0	5.9940	-135.4970	-9.3600	26.7820	2271.0	92.696	54.4299

The columns represent the different features of a pitch excluding the description column, which stores the possible output values, referred to as classes from here for the Multi-Class Classification. The rows represent examples of individual pitches, their attributes, and the outcome for this specific player.

Outcomes to a pitch refer to the action done by the player as a result of the specific features of the said pitch, and include strike out, hit and out, hit and on base, run batted in, etc.

The 11 possible classes to be predicted are shown below.

	0
0	swinging_strike
1	ball
2	called_strike
3	foul
4	hit_into_play
5	hit_into_play_score
6	foul_tip
7	hit_into_play_no_out
8	blocked_ball
9	swinging_strike_blocked
10	hit_by_pitch

Gradient Descent

The prediction model written from scratch uses “Multi-Class Classification” as the basis of the machine learning model. The body of the model uses a gradient descent algorithm and a hypothesis function for the predicted outputs and calculating loss (difference between the ground truth value and predicted value).

To get a predicted outcome, the hypothesis function takes a matrix of examples by features (examples on rows and features on columns) which represents the input values of the features, and another matrix of features by classes to represent the weight of each feature for each class for input.

Include a theta = table (which you can hand make with example values in matrix form)

The body of the function uses a logistic regression equation called the sigmoid function to predict an output:

$$h = \frac{1}{1 + e^{-\theta X}}$$

θ refers to the weight and X is the input feature. Since the inputs are matrices they are multiplied through dot products and the predicted class becomes a matrix of examples by classes. If h is more than 0.5 for each input feature the prediction is marked as correct, and this way the algorithm builds a relationship with the input and output. To compare predicted outputs to the ground truth values another matrix is created outside of the function with examples on the rows and classes on the columns, and is filled on the basis that one example can only have one ground truth value, or output. If that specific pitch has the outcome the intersection of those corresponding indices gets marked with a one.

Example of ground truth value matrix for training dataset:

	0	1	2	3	4	5	6	7	8	9	10
0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...
2566	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2567	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2568	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2569	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
2570	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

[2571 rows x 11 columns]

Example of ground truth value matrix for testing dataset:

	0	1	2	3	4	5	6	7	8	9	10
0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
..	...	...	...	...	...	...	...	...	...	...	...
638	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
639	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
640	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
641	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
642	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

[643 rows x 11 columns]

With this process the outcome matrix is built. To use the prediction function and the ground truth values for the actual linear regression the model uses gradient descent, implemented in another function.

This function takes the inputs of weights and ground truth matrices; learning rate denoted as alpha which is 0.0005; and the number of iterations for the gradient descent called epochs. The more epochs, the more times the gradient descent occurs establishing a relationship between input and output and therefore higher accuracy of prediction.

During the gradient descent for each epoch the weights of each feature (theta) get updated based on an equation. First the loss is calculated by simply subtracting the predicted output from the ground truth value of each corresponding indice in the respective matrices. The "gradient value" is given as the dot product of the input matrix transposed and the loss matrix just calculated, and this product divided by the length of the matrix of input features (the number of example inputs).

The weights update equation is quite simple: subtract the learning rate multiplied by the "gradient" from the current weight value for each index in the weights matrix.

$$\theta = \theta - \alpha \frac{(X.T \times (h - y))}{m}$$

Here, alpha is the learning rate as mentioned above, X is the input features matrix which is transposed, $h - y$ is the loss, and m is the learning rate. Doing so each value for the weights of the features of the pitch by the classes is updated and the function returns the matrix with these updated values when the same process occurs for each epoch. There are many results and quantities in the actual gradient descent and in the resulting updated weights that can be analyzed mathematically and graphically, the two shown to be calculated are the cost of the gradient descent as well as the accuracy of the actual model of linear regression that is implemented.

Cost can be defined as the error of the model (how far the predicted output was from the truth), and the goal of the model is to minimize the cost which can be done by hypertuning the parameters of the gradient descent function. The cost function takes the input features matrix, the ground truth values matrix, and the weights as input and calculates the model error using the following equation:

$$C = -\frac{1}{m}[\Sigma y^{i} \log h^{i} + (1-y)\log(1-h^{i})]$$

Generally, more epochs produces the lowest cost, and the goal of the model is to minimize the cost, meaning higher epochs improves accuracy. The accuracy of the model is calculated by the accuracy function which takes the weight, input feature, and ground truth matrices as well as the number of classes as parameters.

Firstly, the function makes predictions based on the model's understanding of the relationships between the input values and the actual values from the gradient descent by looping over the columns of the weight matrices, then compares these predictions to the ground truth values. As mentioned before, if the hypothesis value is .5 or higher the prediction is counted as correct if the ground truth value of the corresponding indices is also one.

```
for col in range(classes):
  for row in range(len(y)):
    if y.iloc[row,col] == 1 and output.iloc[col,row] >=.5:
      accuracy += 1
accuracy = accuracy/len(X)
```

The implementation of the gradient descent from scratch requires implementation of equations and iterating through data, and for high amounts of epochs (which are typically required to keep accuracy at a certain level) the runtime can become several hours for 100 epochs and above.

Using Built in Libraries

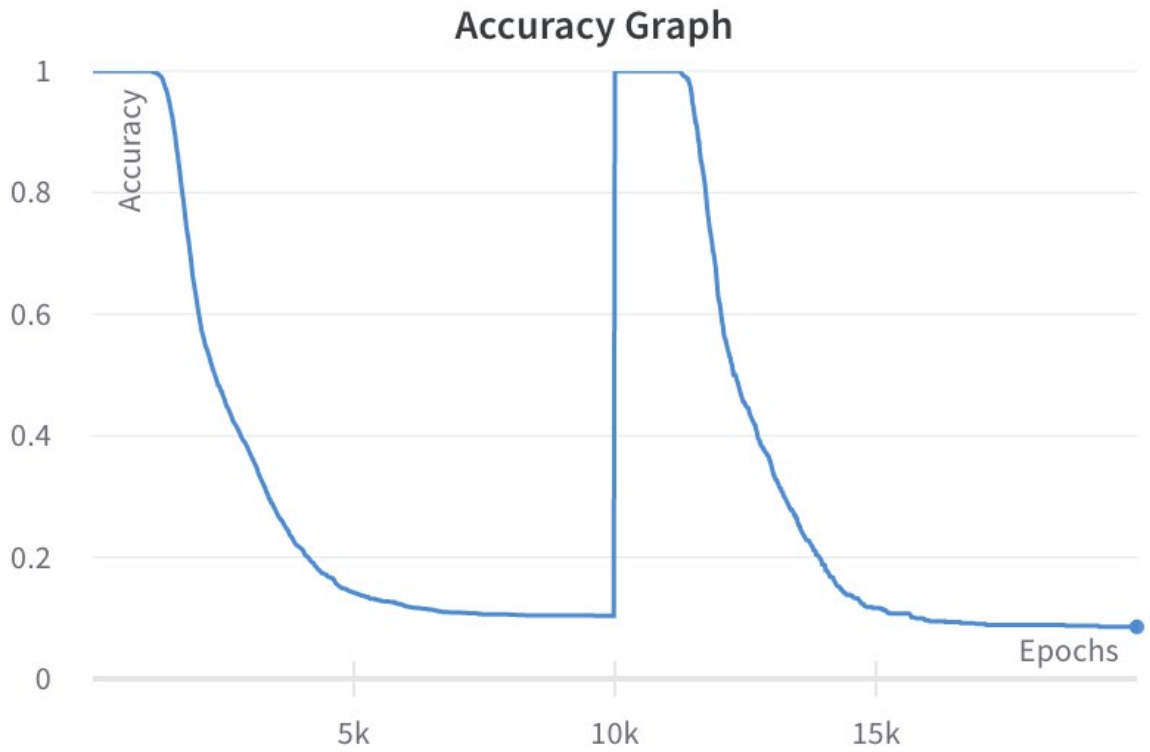
The other implementation of the same task utilizes the Linear Regression of a built in Python machine learning library called sci-kit learn and referred to as "sklearn" in the code. The library's LogisticRegression function first sets the maximum number of iterations, or epochs, the model should run for based on a parameter. For the ground truth values, instead of using Multi-Class Classification and denoting the output for a specific pitch with a one, the values are stored in a list of the actual strings representing the outcome. The actual job of the gradient descent function in terms of training the model is done by the fit function of the library's logistic regression, which takes the input values with the ground truth as inputs and trains the model to establish a relationship between the two. The predict function internally processes the input using the same equation implemented in the hypothesis function from above.

```
logreg = LogisticRegression(max_iter = 1500)
scaler = preprocessing.StandardScaler().fit(train_data)

X_scaled = scaler.transform(train_data)
ys = []
for i in yTrainMatrix:
  ys.append(i)

logreg.fit(X_scaled,ys)
h = logreg.predict(test_data)
```

Results

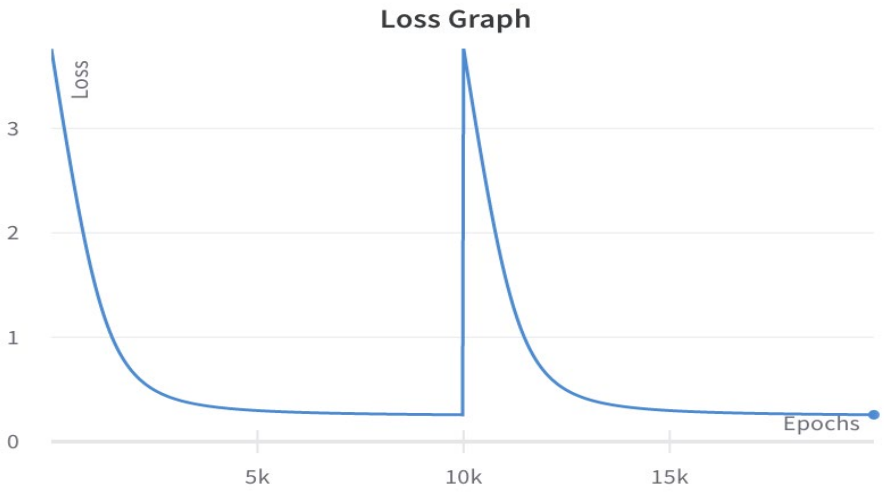
The results analyzed from the first implementation are the accuracy of the model and the cost in order to find the number of epochs which maximize this accuracy while simultaneously minimizing the cost. The accuracy is particularly low for this dataset because it is large with a lot of input values which vary in range of magnitude and actual quantity. Hypertuning the parameters passed to the gradient descent function can improve accuracy over several runs. When the accuracy is graphed for each epoch starting at 10000 epochs the accuracy starts at one (because the weights matrix is initialized to all ones) and comes down as the model trains on the data. The accuracy comes down abruptly because for each epoch the model makes a prediction then updates the weights to make another one.

Example of graphed accuracy function:



The first half of the graph represents the accuracy of the training data and the jump up in the middle is the start of the testing data. The cost looks similar—graphically speaking—to the accuracy for both training and testing data. The loss function's graph slopes down as the epochs increase because the model learns to minimize the loss by adjusting its weights for its different features based on each example it sees. At some point, the loss function starts to plateau, meaning that the model is done training. If at some point the loss function's graph starts to increase after plateauing, this indicates that the model is memorizing the data instead of learning the relationship between the input and output. Using this logic, the ideal number of epochs is the minimum of the graph.

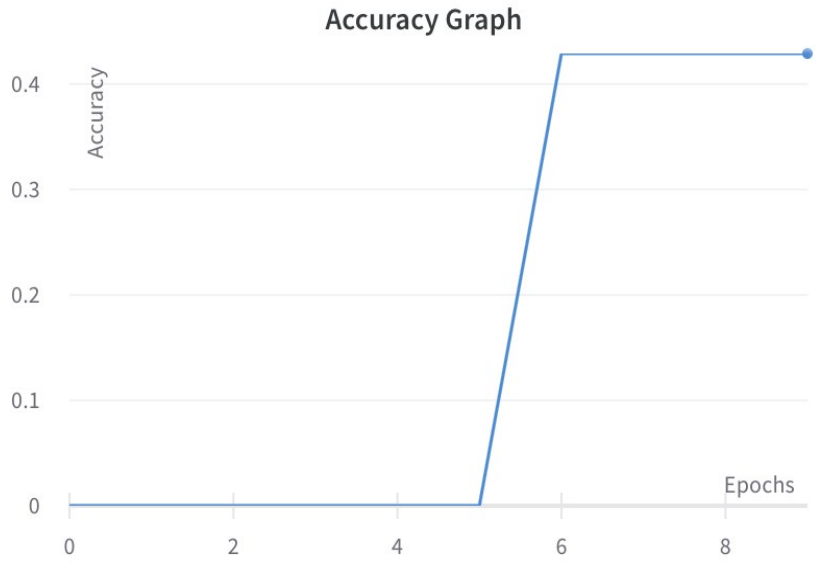
Example of graphed loss function:



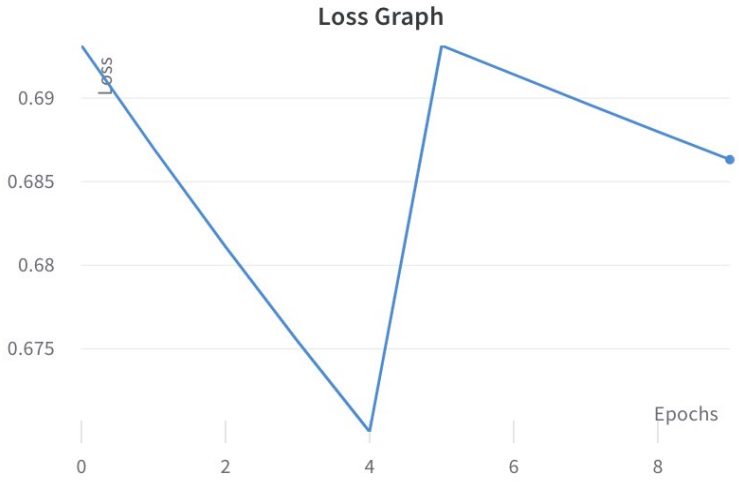
The loss is minimized at approximately 10,000 epochs by looking at the local minimum of the graph, so to maximize accuracy the gradient descent should be run for multiple experiments with the value of epochs set around 10,000. Since this value is the maximum number of epochs, it indicates that for more epochs the loss could possibly go lower.

For a much lower number of epochs (five in the case), the graphs of accuracy and loss take the same shape but the values differ because the model has less time to make accurate predictions with so little iterations to learn the relationship between input features and output.

Accuracy graph for 5 epochs:



Loss graph 5 epochs:



For the implementation of the model which uses the built in Python machine learning library, the accuracy and loss values are very similar. The sci-kit learn library also uses optimization, which will slowly optimize some of the parameters automatically. Although the parameters are set by the library function and changed using its optimization, the more times it runs the higher accuracy it will have, which is why for the first few runs the accuracy looks the same as gradient descent algorithm accuracy; the function has not had enough time to hypertune the parameters.

Conclusion

Essentially, in order to use predictive technology to help coaches and players in a baseball game scenario predict the outcome of a certain pitch machine learning is used. A from scratch implementation using the machine learning technique gradient descent along with Multi-Class Classification for a large dataset has around 10% accuracy for initial parameters and no hypertuning of features or number of iterations. Similarly, using a built in machine learning Python library has an accuracy with very little variance, because of the lack of runs. The more time a model has to train on the same data in order to establish a relationship for the ideal number of epochs (number which causes a local minimum in the loss graph) the higher the accuracy will be.

Acknowledgments

I would like to thank my advisor for the valuable insight provided to me on this topic.

References

1. Aarushikulkarni. (2022). *Aarushikulkarni/baseball-prediction-model*. GitHub. Retrieved January 25, 2023, from https://github.com/aarushikulkarni/Baseball-Prediction-Model
2. Dorsey, J. (2018, June 11). *Judge hit data*. Kaggle. Retrieved January 25, 2023, from https://www.kaggle.com/datasets/jidbro1/judge-hit-data
3. Kulkarni, A. (2022). *Graphical Results*. W&B. Retrieved January 25, 2023, from https://wandb.ai/aarushi-k/nmd-1?workspace=user-aarushi-k
4. Madan 86011 gold badge1111 silver badges1818 bronze badges, R., Jungblut 20.7k66 gold badges6868 silver badges9191 bronze badges, T., Muatik 3, M., Gomes 42511 gold badge88 silver badges2020 bronze badges, M., Coallier 67611 gold badge88 silver badges2222 bronze badges, N., & Gabrieli 98133 gold badges1515 silver badges3131 bronze badges, F. (1960, July 1). *Gradient descent using python and numpy*. Stack Overflow. Retrieved January 25, 2023, from https://stackoverflow.com/questions/17784587/gradient-descent-using-python-and-numpy
5. Sucky, R. N. (2020, November 3). *Multiclass classification algorithm from scratch with a project in Python: Step by step guide*. Medium. Retrieved January 25, 2023, from https://towardsdatascience.com/multiclass-classification-algorithm-from-scratch-with-a-project-in-python-step-by-step-guide-485a83c79992
6. © 2007 - 2023, scikit-learn developers (BSD License). (n.d.). *1. supervised learning*. scikit. Retrieved January 25, 2023, from https://scikit-learn.org/stable/supervised_learning.html#supervised-learning