ajdillhoff / CSE6363 Public

Code Issues Pull requests Actions Projects Security Insights

main CSE6363 / assignments / assignment1 /

ajdillhoff Updated instructions and added traffic data demo. ... last month History

..	
LinearRegression.py	last month
README.md	last month
TrafficDataDemo.ipynb	last month

README.md

CSE 6363: Assignment 1

This assignment covers Linear Regression and Gradient Descent. Linear Regression is a method of predicting real values given some input. Gradient descent is the algorithm we will use to optimize our linear model.

These models have been implemented over and over again and are available in many popular machine learning frameworks. It is important to implement the models yourself so that you gain a deeper understanding of them.

1. Implementation of the `LinearRegression` class

Before evaluating any data, we need some code to actually `fit`, `predict`, and `score` samples. This will be implemented in `LinearRegression.py` provided in this repository. The skeleton of the class is already there. In part 1, you will need to implement the `fit`, `predict`, and `score` functions.

After implementing these 3 functions, you will be able to use this model simply with any regression task.

1.2 The `fit` method

The `fit` method should accept 6 parameters:

1. the input data
2. the target values
3. `batch_size, int` - The size of each batch during training
4. `regularization, int` - The factor of L2 regularization to add, default to 0
5. `max_epochs, int` - The maximum number of times the model should train through the entire training set
6. `patience, int` - The number of epochs to wait for the validation set to decrease

Other parameters can be added as long as they are optional.

This method should use gradient descent to optimize the model parameters using mean squared error as the loss function. So that the model will converge to a solution, early stopping must be used. To do this, set aside 10% of the training data as a validation set. After each step of gradient descent, evaluate the loss on the validation set. If the loss on the validation set increases for 3 consecutive steps, stop training. If it decreases, save the current model parameters. After training is complete, used the saved parameters to set the model parameters.

1.3 The `predict` method

The `predict` method should accept 1 parameter:

1. the input data

This method should run a forward pass of the model and return the predicted values. Given n samples with d features each and m output values, let $X \in \mathbb{R}^{n \times d}$ be the input data, $W \in \mathbb{R}^{d \times m}$ be the model parameters, and $\mathbf{b} \in \mathbb{R}^{n \times m}$ be the bias terms. The predicted values are given by:

$$\mathbf{y} = XW + \mathbf{b} \in \mathbb{R}^{n \times m}$$

1.4 The `score` method

The `score` method should accept 2 parameters:

1. the input data
2. the target values

This method will predict the values for the input data and then compute the mean squared error between the predicted values and the target values. The mean squared error is given by:

$$\mathrm{MSE} = \frac{1}{nm} \sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2$$

where n is the number of samples, m is the output size, y_i is the target value for the $i$ \hat{y}_i$ is the predicted value for the $i$th sample.

2. Regression with a single output

The Iris flower dataset (https://en.wikipedia.org/wiki/Iris_flower_data_set) was organized by Ronald Fisher in 1936. It is a commonly used dataset for introductory machine learning concepts. You will use this dataset for fitting and evaluating your regression model.

The training and testing should be contained in a single evaluation script so that the results can be easily reproduced.

2.1 Preparing the Data

Much of machine learning is in understanding the data you are working with. For our regression task, we want to predict continuous outputs given some input feature(s).

Each sample in the Iris dataset has 4 features:

- sepal length
- sepal width
- petal length
- petal width

We may want to know which feature is most predictive of another. Does knowledge of the sepal length provide good estimates of the sepal width? What about the petal width? What if we use the sepal length and width to predict the petal length or width?

In this section of the assignment, you will create 4 different regression models to answer some of these questions. This will be trivial to do once the code for the model is finished.

To begin, load the data using `scikit-learn`. In order to verify the models that you will create in the following two sections, you will need to take some portion of the dataset and reserve it for testing. Randomly select 10% of the dataset, ensuring an even split of each class. This will be your **test** set. Note that this is different than the random 10% that is taken from the training set when in the `fit` method. The rest of the data will serve as your **training** set.

2.2 Training

Select 4 different combinations of input and output features to use to train 4 different models. For example, one model could predict the petal width given petal length and sepal width. Another could predict sepal length using only petal features. It does not matter which combination you choose as long as you have 4 unique combinations.

Your models should be trained using batch gradient descent with a batch size (optional parameter) of 32 using mean squared error as your loss function.

For each model, train for $n = 100$ steps (optional parameter) OR until the loss on the validation set increases for a number of consecutive epochs determined by `patience` (default to 3).

As each model trains, record the loss averaged over the batch size for each **step**. A single step is the processing of a single batch. One way to save this data is to either return an array from the fit method or save it as an internal class member that can be retrieved after training is complete.

After each model trains, plot the loss against the step number and save it. These plots should also be added to your report.

To observe the effects of regularization, pick one of your trained models and inspect the weights. Train an identical model again, except this time you will add L2 regularization to the loss. Record the difference in parameters between the regularized and non-regularized model.

Record these values into your report so they can be verified.

2.3 Testing

For each model you created, test its performance on unseen data by evaluating the mean squared error against the test dataset that you set aside previously.

In your report, briefly describe which input feature is most predictive of its corresponding output feature based on your experiments.

3. Regression with Multiple Outputs

Some tasks will require a model that can produce multiple outputs. One such task is predicting traffic flow for 36 spatial locations 15 minutes into the future. The output of your linear regression model will a 36 dimensional vector representing the relative traffic congestion of each location.

The training and testing should be contained in a single evaluation script so that the results can be easily reproduced.

3.1 Preparing the Data

To get started, download the dataset from here and extract the `.mat` file into the local `data` directory. A notebook has been provided which demonstrates loading the data into a `numpy` array.

3.2 Training

Use your linear regression class from part 1 to train a model using the provided data. Feel free to choose whatever hyperparameters you think would work best for this. It is recommended that you try a range of possible combinations to see what yields the best performance.

In your report, describe your strategy for training.

3.3 Testing

This dataset already has a test set prepared. Use that test set to evaluate your trained model's performance.

Record the result and include it in your report.

3.4 BONUS: Improving Accuracy

In the paper "Spatial Auto-regressive Dependence Interpretable Learning Based on Spatial Topological Constraints" by Zhao et al., they report a best error of 0.0454 RMSE (Root Mean Squared Error) using a linear regression model. If your implementation can get close to or lower than this error, an extra 20 bonus points is yours.

Submission

Create a zip file that includes all of your code as well as your report. The TA should be able to easily run the code to reproduce all plots and results. Include any additional instructions, if necessary.

Give feedback