C1_W2_Lab05_Sklearn_GD_Soln

June 3, 2024

1 Optional Lab: Linear Regression using Scikit-Learn

There is an open-source, commercially usable machine learning toolkit called scikit-learn. This toolkit contains implementations of many of the algorithms that you will work with in this course.

1.1 Goals

In this lab you will: - Utilize scikit-learn to implement linear regression using Gradient Descent

1.2 Tools

You will utilize functions from scikit-learn as well as matplotlib and NumPy.

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import SGDRegressor
from sklearn.preprocessing import StandardScaler
from lab_utils_multi import  load_house_data
from lab_utils_common import dlc
np.set_printoptions(precision=2)
plt.style.use('./deeplearning.mplstyle')
```

2 Gradient Descent

Scikit-learn has a gradient descent regression model sklearn.linear_model.SGDRegressor. Like your previous implementation of gradient descent, this model performs best with normalized inputs. sklearn.preprocessing.StandardScaler will perform z-score normalization as in a previous lab. Here it is referred to as 'standard score'.

2.0.1 Load the data set

```
[ ]: X_train, y_train = load_house_data()
     X_features = ['size(sqft)','bedrooms','floors','age']
```

2.0.2 Scale/normalize the training data

```
[ ]: scaler = StandardScaler()
     X_norm = scaler.fit_transform(X_train)
     print(f"Peak to Peak range by column in Raw        X:{np.ptp(X_train,axis=0)}")
     print(f"Peak to Peak range by column in Normalized X:{np.ptp(X_norm,axis=0)}")
```

2.0.3 Create and fit the regression model

```
[ ]: sgdr = SGDRegressor(max_iter=1000)
     sgdr.fit(X_norm, y_train)
     print(sgdr)
     print(f"number of iterations completed: {sgdr.n_iter_}, number of weight␣
      ↪updates: {sgdr.t_}")
```

2.0.4 View parameters

Note, the parameters are associated with the *normalized* input data. The fit parameters are very close to those found in the previous lab with this data.

```
[ ]: b_norm = sgdr.intercept_
     w_norm = sgdr.coef_
     print(f"model parameters:                   w: {w_norm}, b:{b_norm}")
     print( "model parameters from previous lab: w: [110.56 -21.27 -32.71 -37.97], b:
      ↪ 363.16")
```

2.0.5 Make predictions

Predict the targets of the training data. Use both the `predict` routine and compute using w and b.

```
[ ]: # make a prediction using sgdr.predict()
     y_pred_sgd = sgdr.predict(X_norm)
     # make a prediction using w,b.
     y_pred = np.dot(X_norm, w_norm) + b_norm
     print(f"prediction using np.dot() and sgdr.predict match: {(y_pred ==␣
      ↪y_pred_sgd).all()}")
```

```
print(f"Prediction on training set:\n{y_pred[:4]}" )
print(f"Target values \n{y_train[:4]}")
```

2.0.6 Plot Results

Let's plot the predictions versus the target values.

[ ]:
```
# plot predictions and targets vs original features
fig,ax=plt.subplots(1,4,figsize=(12,3),sharey=True)
for i in range(len(ax)):
    ax[i].scatter(X_train[:,i],y_train, label = 'target')
    ax[i].set_xlabel(X_features[i])
    ax[i].scatter(X_train[:,i],y_pred,color=dlc["dlorange"], label = 'predict')
ax[0].set_ylabel("Price"); ax[0].legend();
fig.suptitle("target versus prediction using z-score normalized model")
plt.show()
```

2.1 Congratulations!

In this lab you: - utilized an open-source machine learning toolkit, scikit-learn - implemented linear regression using gradient descent and feature normalization from that toolkit

[ ]:
```

```