Data Science 311

Lab 5 (10 points)
Due at 10pm on May 17, 2022

Read all of the instructions. Late work will not be accepted.

Overview

In this lab, you will work on a regression problem, using linear regression (and some of the tricks that can be used to make it more effective) to make some predictions. Then, you'll do some analysis to validate and interpret the model.

Collaboration

For this lab, you are encouraged to spend the lab period working together with a partner. Together means synchronously and collaboratively: no divide an conquer. After the lab period ends, you will work independently and submit your own solution, though you may continue to collaborate the same partner if you wish. Your submission must acknowledge which person you worked with, if any, and for what parts of the lab (this should be included as a statement at the top of your notebook).

Data

For this lab, we will work with the mpg dataset that comes built into Seaborn. Your job is to build a model that effectively predicts the miles per gallon column based on the values in the other columns. You may want to spend a couple minutes getting familiar with the dataset and its columns.

Tasks

There are two parts to this lab. In the first, you will create data splits and try out various tricks to get linear regression to predict miles per gallon (the mpg column) with high accuracy. In the second part, you will examine your trained model to validate (i.e., convince yourself that it's doing its job well) and interpret it (i.e., use the model to learn which features are most significant predictors of mpg).

Both parts should be documented in an iypnb titled `lab5.ipynb`.

Part I: Developing the Model

Start by splitting the dataset into training, validation, and testing sets as we discussed in lecture. Since the dataset is sorted by model year, you should probably randomize the splits. Remember that your test set is sacred: until you're sure you're done modifying your model, do not touch your test set! You may find the `sklearn.model_selection.train_test_split` function helpful; I strongly recommend passing a number into the `random_state` argument so that if you run the code again, you get the same splits – otherwise you risk having data in

a test set that you previously trained or validated a model on. I split my data into train, val, and test sets of 250, 75, and 67, respectively.

Your next task is to train a successful linear regression model to predict the mpg column. You'll likely want to make use of the `sklearn.linear_model.LinearRegression` model for this.

Evaluating Your Model Before we even train a model, we need to know how to tell if a model is good. For this will we use two metrics:

- R^2. This metric is conveniently computed by `LinearRegression`'s `score` method.
- $RMSE$. Though it is not too hard to compute this yourself, you can also use `sklearn.metrics.mean_squared_error` with the `squared` kwarg set to `False`.

When evaluating your model, check its performance on both the training and validation sets. Large differences in training and validation accuracy can suggest overfitting, so keep an eye out for that.

Trying Out Tricks Though we rare using a standard linear regression model, there are still a lot of decisions we can make that may affect the performance of your model. I recommend training the simplest possible model first, then trying out different ideas for preprocessing that might help your model perform better. These are listed in no particular order, and each one may or may not help – it is up to you to play around and find what works.

- **Feature choice**: What is the effect of including or excluding certain features (columns) from your training data?
- **Categorical features**: Relatedly, the categorical columns are not immediately applicable to a regression problem because they are not numerical. But you can convert them to numbers using one-hot encodings. Refer to the class notes for a reminder how one-hot encodings work. While one-hot encodings can be done manually, `sklearn.preprocessing` has this functionality built in: `OneHotEncoder`. For fun, you could try the approach I *do not* recommend, which is simply enumerating categorical variables as 0, 1, 2, etc. There is an `OrdinalEncoder` that can do this for you. For this data you can see if it works better or worse.
- **Data Scaling**: If the magnitudes of your input features differ by a lot, the model may do a better job of fitting when the features are all scaled to z-scores. Even if this does not affect model performance, it can help with interpretability (see Part II). You can use `sklearn.preprocessing.StandardScaler()`, or there's also a `RobustScaler` that is less sensitive to outliers.
- **Feature Expansions** We saw an example of fitting a polynomial. This involved a "feature expansion": from x we added new non-linear features x^2 and x^3. This approach allows you to use a linear model to fit non-linear functions, and the technique also works when the input dimension $D > 1$. `sklearn` has a `sklearn.preprocessing.PolynomailFeatures` function that does this. If $D = 2$ and you want second order polynomial features, it yields the following values:

$$[1, x_1, x_2, x_1x_2, x_1^2, x_2^2].$$

Be careful when using this – it explodes the number of features and increases your model complexity quickly, which increases the danger of overfitting.

You are not limited to the above – feel free to explore the other preprocessing features built into scikit-learn, or come up with your own ideas. The scores you achieve will depend on your data splits, but based on my experiments, you'll likely be able to achieve a coefficient of variation score of well over 0.8 on both training and validation – ideally you can go even higher than that.

Part II: Validation and Interpretation

One of the nice things about a linear regression models (in contrast to many more powerful models) is that they are relatively explainable. This means we can probe the model and understand some things about how it is working, which is useful both to build confidence that it is a well-behaved model, and also to help us understand things about the underlying data.

Try out the following "sanity checks" to help validate your model, and comment on whether each shows the "good outcome" – namely, that our model is behaving as expected.

- Calculate the *residuals* – that is, the difference between the ground-truth values and your predictions ($y - h(x)$) – on the validation set, and then plot their distribution. If the model is working well and its assumptions hold, we should see a bell-curve-like distribution of errors with a mean near zero.

- We cannot scatterplot all our features vs the predictions because our features are too high-dimensional. But if we scatterplot our predictions vs the ground-truth values, we should see a roughly linear relationship.

- We can check for *homoscedasticity* – the property that residuals do not vary depending on the value of y – by plotting the residuals versus the ground-truth value. If we get a scatterplot with no visible patterns or correlation, that is a good sign.

Finally, linear regression gives us a directly interpretable signal about what features the model found useful: the coefficients themselves. You can access these via `linear_regression_model.coef_` and see the weight applied to each input feature to compute the output value. If this number is large in magnitude for a given feature, that feature was important in computing the result; if it was close to zero, then that feature didn't matter much.

There are a couple caveats to keep in mind:

1. The scale of your features also affects the coefficient. If two equally important features vary from 0 to 1,000 (feature 1) and 0 to 10 (feature 2), the coefficient on feature 1 will be 1/100th of the feature 2 coefficient. For this reason, coefficients are best interpreted when you have normalized your input features to z-scores before fitting the model so they have around the same range. If you did not do that above, go ahead and add a preprocessing step that scales the features so we can interpret the coefficients.

2. Keep in mind that other preprocessing, especially feature expansions, will also affect the coefficients; if you turned your original features into a new set of features, you will get coefficients for the new features. You will need to find some way to interpret the new coefficients in terms of the original columns, based on the transformations you did.

Show (e.g., with a bar plot) the coefficients on each input feature and comment on which features were most important to the model; does this make intuitive sense?

Evaluating on the Test Set When you have refined your preprocessing and model, performed the above validation steps, and you are happy with its performance, go ahead and run it on the test set and comment on your test accuracy. If it is similar to your validation accuracy, great! If it is not, that's okay – you won't lose credit. However, if there is evidence that you cheated and ran on the test set more than once, you will lose credit.

Submitting Your Work

Please zip your lone ipynb file (using the naming conventions stated above, where spelling, spacing and capitalization matter) and upload the zip via Canvas.

Grading

70% of the assignment grade is based on correctness, with 30% on clarity. Points will be lost on correctness if there are technical issues, techniques applied or interpreted incorrectly, questionable assumption made, or important information ignored. Points will be lost on clarity if decisions are not justified, or if explanations are difficult to understand.

Acknowledgments

This lab assignment was created by Scott Wehrwein in Fall 2021.