ISTA 331 SPECIAL TOPICS ASSIGNMENT: PENALIZED LINEAR MODELS

1. Introduction

In HW 4, we fit linear and nonlinear models by minimizing the mean square error:

$$MSE = \frac{1}{N}\sum_i (y_i - \hat{y}_i)^2$$

This is a measure of how well the model fits the data. By fixing a function that depends on some parameters, $\hat{y} = f(\mathbf{x}; \beta)$, we can find the parameter vector β for which the RSS is smallest. We saw in HW 7 that this may be accomplished by gradient descent even when the model doesn't allow us to solve the optimization problem directly.

Note: you might conceptualize this as minimizing RMSE instead. The two give equivalent answers, but RSS is easier computationally because you don't need to take the square root. Below, though, as we modify the cost function, we'll specifically want RSS, not RMSE.

1.1. Improving models by shrinking parameters.

It turns out we can often improve our models by making the entries of the parameter vector β smaller (closer to 0). This works in two ways:

- *Bias-variance tradeoff.* We discussed in a lecture video the idea of bias-variance tradeoff. Ordinary least squares is an unbiased estimate. Preferring smaller coefficients introduces some bias but also reduces variance, and we can come out ahead in the tradeoff.
- *Interpretability.* In some cases we may have a very large number of predictors, but only some of them are useful in making predictions. By removing some predictors, we may slightly reduce the variance of our model and also make it easier to understand.

1.2. Bias-variance tradeoffs.

Penalized regression models are all attempts to exploit the *bias-variance tradeoff* and improve the accuracy of future predictions. Error in future predictions is called *generalization error* and can, broadly, be broken down into three parts:

- *irreducible error*: error due to the inherent random variability in the target variable. In general, there's nothing we can do about this; the best possible model may still have some irreducible error.
- *bias*: systematic error due to a model failing to capture a feature or relationship in the data. This is generally a result of incorrect assumptions, using a model type that isn't well suited to the task, or using a model that doesn't have enough flexibility to describe the data – in short, it is *underfitting* the training data. Bias can often be reduced by increasing the number of parameters and thereby allowing the model to fit the training data more closely.
- *variance*: random error due to a model's sensitivity to the training data. A highly flexible model (one with a lot of parameters) may produce very different fits for different samples of training data. If this happens, this means the model is fitting the random variation in the training data rather than fitting the true relationship – in short, it is *overfitting* the training data. Variance may be reduced by using a simpler model.

More complex models often reduce bias but increase variance. This is known as the *bias-variance tradeoff*.

It is a provable fact that if there is truly a linear relationship between a set of predictors $\mathbf{x}$ and a target variable y; that is,

$$y = \beta_0 + \beta_1 x_1 + \ldots + \beta_m x_m + \text{noise}$$

then ordinary least squares (OLS) gives unbiased estimates of the parameters β, and moreover, OLS has the lowest variance among all possible unbiased models.[1]

However, the surprising fact is that the best possible *unbiased* model is not necessarily the best possible model *overall*. Because of the bias-variance tradeoff, it may be the case that introducing a modest amount of bias can improve generalization error. This is especially true when the training data is limited, which is when the variance of OLS will be highest.

1.3. **Ridge regression.** Ridge regression is the most "classical" of the common penalized linear models. In this model, we replace the usual cost function for ordinary least squares with

$$\mathcal{L}_{ridge}(\beta) = MSE + \alpha \sum_m \beta_m^2$$

where α is a tuning parameter. (In some literature this α is denoted λ. I'm using α here because the `sklearn` implementations call it `alpha`.)

1.4. **Lasso.** The lasso method is similar to ridge regression, but uses the cost function

$$\mathcal{L}_{lasso}(\beta) = MSE + \alpha \sum_m |\beta_m|$$

This is a similar basic idea: penalizing the total size of the coefficients. Using the sum of absolute values instead of the sum of squares of the coefficients leads to two major differences:

- Bad news: it's harder to solve the optimization problem. In ridge regression, if the X variables are transformed correctly, there is an algebraic solution for the best coefficient vector. This is not true for the lasso; there is no exact solution. Thus it *must* be solved using gradient descent or another iterative optimization method.
- Good news: the lasso can do something that ridge regression can't. In ridge regression, no matter how big α is, the model will never set any coefficients to 0. On the other hand, the lasso method can result in some coefficients β_i being set to 0. Why is this good? It means that the lasso can perform the task of *variable selection*: it can suggest which predictors should be dropped from the model entirely. When we have a large number of predictors and it's not obvious which ones are useful, the lasso can help us decide which to use and which to drop.

1.5. **Elastic net regularization.** Elastic net regularization is an attempt to combine the ridge and lasso methods. The cost function for elastic net is

$$\mathcal{L}_{net}(\beta) = tL_{lasso}(\beta) + (1-t)L_{ridge}(\beta)$$

where $0 \leq t \leq 1$ is a tuning parameter that controls the mixture of the two cost functions. When $t = 0$, this is just a ridge; when $t = 1$, it is just a lasso.

[1] This fact is called the Gauss-Markov theorem.

Note that we still have the tuning parameter α "hidden" in this cost function, so there are two parameters to tune.

1.6. **Models in `sklearn`.** Each of the above models is present in `sklearn.linear_model`, under the classes `LinearRegression` (OLS), `Ridge`, `Lasso`, and `ElasticNet`. Documentation for these can be found at

- `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html`
- `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html`
- `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html`
- `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html`

The important parts are that each expose `fit` and `predict` models that we can use for model fitting and for making predictions, and the three regularized models take a parameter `alpha` (passed at model initialization). `ElasticNet` also takes a value for the parameter t under the name `l1_ratio`.

2. Instructions

2.1. **Code and submission.** Create a Python module called `penalized.py` and upload it to the assignment folder on D2L. Do the following imports:

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error
```

Then implement the functions below. Your `main` function must run when I type `python penalized.py` at the terminal.

2.2. **Documentation.** Your script must contain a header docstring containing your name, ISTA 331, the date, and a brief description of the module. Each function must contain a docstring. Each function docstring should include a description of the function's purpose, the name, type, and purpose of each parameter, and the type and meaning of the function's return value.

2.3. **Grading.** Your submission will be human-graded. I'll run the python script with `python penalized.py` and look at the output produced, and then read the code. Make sure that the script runs! I will deduct points if I have to chase down syntax errors to get output from your code, or add a call to `main`, etc. Code should be clear and concise, but you will only lose style points if your code is a real mess. Include inline comments to explain tricky lines and summarize sections of code.

2.4. **Collaboration.** Collaboration is **not** permitted on this assignment. However, you may use your own notes, documentation for `sklearn`, `numpy`, and `pandas`, and any resources available on the D2L site (including videos, worksheets, Jupyter notebooks, etc.).

3. Function specifications

- `get_frames`: This function takes no arguments, loads the Boston housing data set from `Boston_Housing.csv` and splits it into training and testing `X` and `y` data frames/series. Set the random seed using `np.random.seed(95)` and then use `np.random.choice` to select 100 rows from the data frame (without replacement!) to be the training set. The remainder of the data frame will be the testing set.

  The target variable is the median housing price, `MEDV`, so drop this from your training and testing frames to produce the training and testing `X`; and take `train['MEDV']` and `test['MEDV']` to be your training and testing `y`.

  Return the training `X`, training `y`, testing `X`, and testing `y`.
- `best_ridge`: This function takes a training `X`, a training `y`, and an array or list of `alpha` values to test. Use 5-fold cross-validation, using `cross_val_score` with the scoring `neg_mean_squared_error` (see HW6 if you need a refresher on this), to find which value of `alpha` performs best for a ridge regression. You can initialize a ridge regression model using `Ridge(alpha = xxxx)`. Return the best value of `alpha` out of the ones considered.
- `best_lasso`: Same as the previous function, but create a lasso model using `Lasso(alpha = xxxx)`.
- `best_net`: Same as the previous functions, but create an elastic net model using `ElasticNet(alpha = xxxx)`. The tuning parameter which I called t above is called `l1_ratio` here, but you can leave it at the default value of `0.5`.
- `main`: Load the data frames and split them into training and testing data.

  Use `best_ridge`, `best_lasso`, and `best_net` to find good values of `alpha`. It's up to you to decide what values of `alpha` are reasonable to try, and this may require a bit of experimentation. Then initialize one instance of each model type, using the `alpha` values you found.

  Once the models have been trained, evaluate them all by calling `predict` with your testing `X`. Calculate the RMSE of the predictions relative to the testing `y` values. Print your results in the following format:

```
Summary:

Ordinary Least Squares
RMSE: xxxx

Ridge Regression
Best alpha value: xxxx
RMSE: xxxx

Lasso
Best alpha value: xxxx
RMSE: xxxx

Elastic Net
Best alpha value: xxxx
RMSE: xxxx

The best test accuracy was achieved by: [model name]
```

  where `xxxx` is replaced by the appropriate numerical value and `[model name]` is the model which gave the smallest RMSE on the testing set. (This is human graded so don't worry about getting the formatting perfect.) A small part of your score will come from finding

good parameter values for each model and determining which model appears to perform best in this scenario.