Near-Term Forecasting of Solar Radiation Using Machine Learning Techniques

Overview

In this paper, the theory behind five common machine learning models is presented, as well as their application to the problem of forecasting near-term solar irradiance. The models examined are: linear regression, ridge regression, lasso regression, decision trees, and feed-forward neural networks. The challenge of accurately forecasting solar radiation will become increasingly relevant to society as solar photovoltaics are continually added to the electrical grid, thus motivating the application of these models to this task. While none of the models presented here can match the accuracy of state-of-the-art weather models, solar forecasting provides an important test case on which to demonstrate their relative capabilities.

Table of Contents

Background

Solar photovoltaics (PV) have been becoming a more significant part of the electric grid in the United States over the past 20 years and its role is only expected to increase. The Solar Energy Industries Association (SEIA) estimates there were close to 90,000 MWdc of installations in the US in 2020.

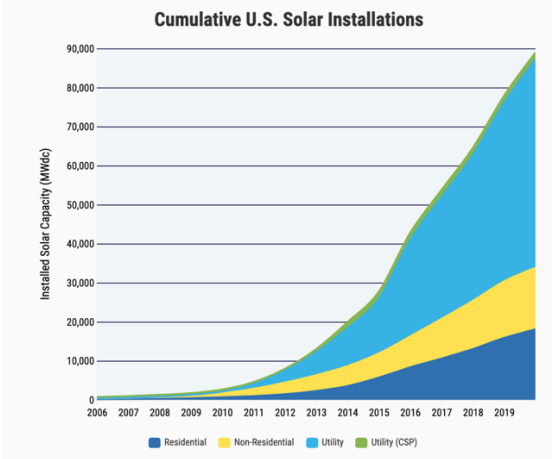


Fig. 1. Cumulative U.S. Solar Installations by sector, 2006-2020 [1]

With more and more solar PV contributing to electricity production in the United States, better forecasting models will be needed to maintain the stability of the electric grid and optimize the utilization of grid resources such as energy storage devices as well as other types of generating facilities.

This paper utilizes 30-minute interval weather data from various locations around Pittsburgh, Pennsylvania, shown in Figure 3, from the National Renewable Energy Laboratory's (NREL) National Solar Resource Database (NSRDB) [2] [3] in order to forecast the Global Horizontal Irradiance (GHI) in Pittsburgh four hours ahead of the most recent measurement. GHI is the combination of Direct Normal Irradiance (DNI) and Diffuse Horizontal Irradiance (DHI) and is the most important single value determining the production of a PV array [4]. Figure 2 provides a depiction of varying types of irradiance.

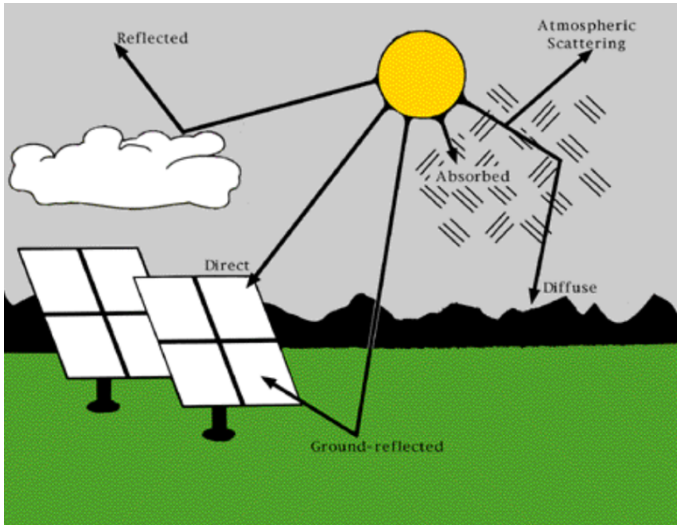


Fig. 2. Direct, diffuse and reflected solar irradiance [5]

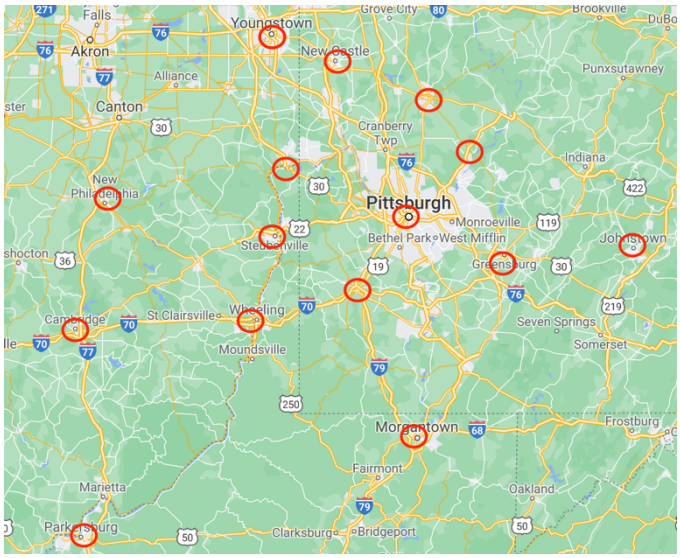


Fig. 3. Locations around Pittsburgh, PA from which weather data was downloaded. Locations west of Pittsburgh were oversampled due to weather patterns generally traveling west to east. The following city abbreviations are used throughout the paper: YGT=Youngstown, NCS=New Castle, BUT=Butler, NPH=New Philadelphia, ELV=East Liverpool, KIT=Kittanning, STU=Steubenville, PIT=Pittsburgh, CBG=Cambridge, WHE=Wheeling, WAS=Washington, GRE=Greensburg, JON=Johnstown, MGT=Morgantown, PKS=Parkersburg

The primary purpose of this paper is to present and analyze a few machine learning algorithms and demonstrate their utility when applied to the collected data. The algorithms that will be presented are: Linear Regression, Ridge Regression, Lasso Regression, Decision Tree, and Feed-Forward Neural Network, also known as a Multi-Layer Perceptron.

Data Acquisition and Preparation

As stated previously, data was acquired from NREL's NSRDB for various locations around Pittsburgh, PA. Data was collected and reported in 30-minute increments for the years 1998-2019. The features of the dataset include the following:

Table I
List of Features Collected

Hour	Minute	Day	Month	Year
DHI	DNI	GHI	Clearsky DHI	Clearsky DNI
Clearsky GHI	Dew Point	Surface Albedo	Wind Speed	Wind Direction
Relative Humidity	Temperature	Pressure	Global Horizontal UV Irradiance (280-400nm)	Global Horizontal UV Irradiance (295-385nm)

These features were collected for each of the locations shown in Fig. 3. To prepare the data for processing, the following steps were taken:

1. For each city, append each year's data to the end of the previous year's data.

City A Data, 1998
City A Data, 1999
…
City A Data, 2019

2. Remove the *Year* attribute from each dataframe

3. Concatenate the data from all cities along axis 1, creating a regional weather snapshot

City A Data	**City B Data**	…

4. Since the *Hour, Minute, Day,* and *Month* attributes will all be the same across each row of the dataset, remove all but one appearance of this feature.

5. Convert *Hour* and *Minute* to *Time_X* and *Time_Y* where the time is mapped to the unit circle, shown in Figure 4, where 0:00 is represented by the coordinates (0, 1), 12:00 by (0, -1) and time progresses in a clockwise fashion. The purpose of this is to explicitly tell the model that 23:30 and 0:00 are very close to one another, just as 4:00 and 4:30 are, rather than have a discontinuity in this variable at the end of each day.

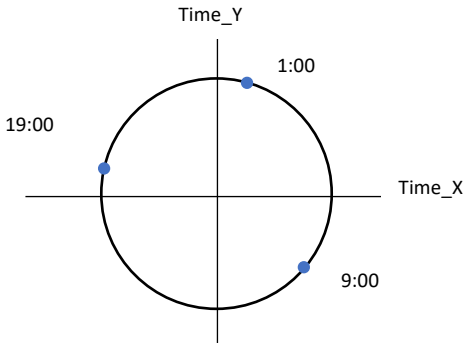


Fig. 4. Depiction of conversion from *Time* to *Time_X* and *Time_Y*

6. Similarly, convert *Day* and *Month* to *Day_X* and *Day_Y* where the day is mapped to the unit circle where January 1st is at the top of the unit circle and the day proceeds in a clockwise fashion. Again, this is to remove discontinuities and to emphasize that March 31st and April 1st are adjacent dates, just as December 31st and January 1st are.

7. Convert *Wind Speed* and *Wind Direction* to *Wind_X* and *Wind_Y* where the latter are cartesian coordinates rather than polar coordinates. Again, this is to remove the discontinuity between wind with a direction of 359º and 0º.

Creation of Features and Targets

The objective of each machine learning algorithm in this project is the same: to predict the GHI in Pittsburgh four hours ahead of time. However, three different datasets with an increasing number of features were used in order to determine if additional historical information can help improve prediction accuracy.

The features of the first dataset are the regional weather measurements from all cities previously shown in Figure 3. As seen in Table II, this collection of features, or input, is matched up with a target, or output, being the GHI in Pittsburgh four hours (eight timesteps) after the regional weather snapshot was observed.

The second dataset repeats the same features as the first, but also includes the regional weather snapshot from the immediately preceding timestep (30-minutes prior). This now larger set of features, shown in Table III, is again matched up with the GHI in Pittsburgh four hours later.

The third dataset contains the same features as the second dataset, but includes two additional historical regional weather snapshots (30-minutes, 1-hour, and 1.5-hours prior), as seen in Table IV. These additionally features allow the machine learning algorithm to effectively have a 1.5 hour memory of weather data.

Table II
Dataset 1: Current measurements and target 4 hours ahead

Features	Target
Data from all cities at timestep 0	Pittsburgh GHI at timestep 8
Data from all cities at timestep 1	Pittsburgh GHI at timestep 9
Data from all cities at timestep 2	Pittsburgh GHI at timestep 10
…	…

Table III
Dataset 2: Current and previous measurements and target 4 hours ahead

Features		Target
Data from all cities at timestep 0	Data from all cities at timestep 1	Pittsburgh GHI at timestep 9
Data from all cities at timestep 1	Data from all cities at timestep 2	Pittsburgh GHI at timestep 10
Data from all cities at timestep 2	Data from all cities at timestep 3	Pittsburgh GHI at timestep 11
…	…	…

Table IV
Dataset 3: Current and previous three measurements and target 4 hours ahead

Features				Target
Data from all cities at timestep 0	Data from all cities at timestep 1	Data from all cities at timestep 2	Data from all cities at timestep 3	Pittsburgh GHI at timestep 11
Data from all cities at timestep 1	Data from all cities at timestep 2	Data from all cities at timestep 3	Data from all cities at timestep 4	Pittsburgh GHI at timestep 12
Data from all cities at timestep 2	Data from all cities at timestep 3	Data from all cities at timestep 4	Data from all cities at timestep 5	Pittsburgh GHI at timestep 13
…	…	…	…	…

Note how the target timesteps in each dataset are shifted to always be four hours after the most recent timestep in the corresponding set of features. To differentiate timesteps, feature labels include a -0, -1, -2, or -3 to indicate how many timesteps previous to the current time the measurement was recorded. For example, the GHI in Pittsburgh recorded in the previous timestep is labeled "PIT GHI-1".

Splitting into Training, Validation, and Testing Sets

When developing a machine learning model, it is critical to set aside training data on which to evaluate the model to ensure that comparisons between algorithms can be made. These two sets are called the training and testing sets. Additionally, it is also often advantageous to set aside a portion of the training dataset to make intermediate observations of how the trained model generalizes to yet-unseen data. This set-aside portion of the training set is called the validation set and is especially useful when it comes to tuning the hyperparameters of a model.

With this in mind, two splits of each dataset (Tables II, III, and IV) were made. The first was to split the combined training and validation set from the testing set. This was done using a testing size ratio of 0.2 (approximately 4.5 years of data) without shuffling (in order to be able to visualize sequential data when testing and validation were performed). Next, another split was made between the training and validation sets, again with validation size of 0.2 (making the validation

set contain approximately 3.5 years of data) also without shuffling. This splitting process was performed for each of the three datasets.

Scaling

Almost all machine learning algorithms require that the data be normalized or scaled in order to perform well. The reason for this is to ensure each feature has similar range and variance, so one feature does not appear to be excessively significant due to unit selection. For example, a model which forecasts the price of a plane ticket based upon distance of the flight should not change whether that distance is measured in miles, feet, or centimeters.

A standard scaler was utilized in this project, which centers each feature on zero and scales them to have approximately unit variance. This is performed thorough the following process.

Let there be *m* features in the dataset, each with *n* instances

$$X_{unscaled} = [\boldsymbol{x}_{feature\ 1,unscaled} \quad \boldsymbol{x}_{feature\ 2,unscaled} \quad \cdots \quad \boldsymbol{x}_{feature\ m,unscaled}]$$

The mean of feature *j* is

$$\mu_j = \frac{1}{n}\sum_{i=1}^{n} x^{(i)}_{feature\ j,unscaled} \tag{1}$$

Its standard deviation is

$$\sigma_j = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x^{(i)}_{feature\ j,unscaled} - \mu_j)^2} \tag{2}$$

The scaled dataset becomes

$$X_{scaled} = \left[\frac{(\boldsymbol{x}_{feature\ 1,unscaled} - \mu_1)}{\sigma_1} \quad \frac{(\boldsymbol{x}_{feature\ 2,unscaled} - \mu_2)}{\sigma_2} \quad \cdots \quad \frac{(\boldsymbol{x}_{feature\ m,unscaled} - \mu_m)}{\sigma_m}\right] \tag{3}$$

Where each operation is performed component-wise. Unless otherwise stated, it can be assumed throughout this paper that all vectors $\boldsymbol{x}$ are already scaled using the standard scaler.

Importantly, so as not to allow any information leakage between datasets, one standard scaler was fit only on the training data, and then the training and validation sets were transformed using this scaler. A second scaler was fit on the combination of the training and validation set and this set as well as the testing set were transformed using this scaler.

Model Analysis

This paper will present and explain five machine learning models. While all five are fairly basic in the grand scheme of machine and deep learning, understanding how these work can be foundational to being able to determine when and why to use more complex models. Additionally, these simpler models often deliver sufficiently satisfactory performance on many machine learning tasks.

The first three of five models to be presented are all closely related. They are: Linear Regression, Ridge Regression, and Lasso Regression.

Linear Regression

A linear regression model is a model where the output $\hat{y}$ is predicted by the following equation given an input $\boldsymbol{x}$

$$\hat{y} = \theta_0 + x_1\theta_1 + \cdots + x_m\theta_m \tag{4}$$

Or,

$$\hat{y} = \tilde{\boldsymbol{x}}^{\mathrm{T}}\boldsymbol{\theta} \tag{5}$$

where,

$$\tilde{\boldsymbol{x}} = \begin{bmatrix} 1 \\ \boldsymbol{x} \end{bmatrix} = [1 \;\; x_1 \ldots x_m]^{\mathrm{T}}$$

And $\boldsymbol{\theta}$ is selected to "best-fit" the data. The most common measurement of fit is the sum of the squared errors between predicted output and actual output for all data instances. This is represented by the objective function

$$J(\boldsymbol{\theta}) = (\boldsymbol{y} - \hat{\boldsymbol{y}})^{\mathrm{T}}(\boldsymbol{y} - \hat{\boldsymbol{y}}) \tag{6}$$

Or,

$$J(\boldsymbol{\theta}) = (\boldsymbol{y} - \tilde{\mathrm{X}}\boldsymbol{\theta})^{\mathrm{T}}(\boldsymbol{y} - \tilde{\mathrm{X}}\boldsymbol{\theta}) \tag{7}$$

Where $\boldsymbol{y}$ is the vector of observed outputs, $y^{(i)}$ for given actual inputs, $\boldsymbol{x}^{(i)}$, and

$$\tilde{\mathrm{X}} = [\mathbf{1_n} \quad \mathrm{X}] = \begin{bmatrix} 1 & \boldsymbol{x}^{(1)^{\mathrm{T}}} \\ \vdots & \vdots \\ 1 & \boldsymbol{x}^{(n)^{\mathrm{T}}} \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(1)} & \cdots & x_m^{(1)} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_1^{(n)} & \cdots & x_m^{(n)} \end{bmatrix}$$

Where $x_1^{(1)}$ is the 1st instance of the 1st attribute, $x_1^{(n)}$ is the n^{th} instance of the 1st attribute, and so on.

Linear regression is a method by which the objective function, $J(\boldsymbol{\theta})$ is minimized over $\boldsymbol{\theta}$. If a minimum exists, the optimal $\boldsymbol{\theta}$ is denoted $\boldsymbol{\theta}^*$ and is used in the final model Eq. (5).

A necessary condition of a point $\boldsymbol{\theta}^*$ to be a minima of Eq. (7) is that the gradient of J evaluated at $\boldsymbol{\theta}^*$ is equal to $\mathbf{0}$.

$$\frac{\partial J(\boldsymbol{\theta}^*)}{\partial \boldsymbol{\theta}} = \mathbf{0} \tag{8}$$

$$\frac{\partial J(\boldsymbol{\theta}^*)}{\partial \boldsymbol{\theta}} = \frac{\partial}{\partial \boldsymbol{\theta}}\left(\left(\boldsymbol{y} - \widetilde{\mathrm{X}}\boldsymbol{\theta}\right)^{\mathrm{T}}\left(\boldsymbol{y} - \widetilde{\mathrm{X}}\boldsymbol{\theta}\right)\right)|_{\boldsymbol{\theta}^*} = \mathbf{0}$$

$$\frac{\partial}{\partial \boldsymbol{\theta}}\left(\boldsymbol{y}^{\mathrm{T}}\boldsymbol{y} - \boldsymbol{y}^{\mathrm{T}}\widetilde{\mathrm{X}}\boldsymbol{\theta} - \boldsymbol{\theta}^{\mathrm{T}}\widetilde{\mathrm{X}}^{\mathrm{T}}\boldsymbol{y} + \boldsymbol{\theta}^{\mathrm{T}}\widetilde{\mathrm{X}}^{\mathrm{T}}\widetilde{\mathrm{X}}\boldsymbol{\theta}\right)|_{\boldsymbol{\theta}^*} = \mathbf{0} \tag{9}$$

Note that since,

$$\widetilde{\mathrm{X}} \in \mathbb{R}^{nx(m+1)}, \boldsymbol{\theta} \in \mathbb{R}^{(m+1)} \; and \; \boldsymbol{y} \in \mathbb{R}^{n}$$

where m is the number of features of the data and n is the number of instances.

Then,

$$\boldsymbol{y}^{\mathrm{T}}\widetilde{\mathrm{X}}\boldsymbol{\theta} \in \mathbb{R}^{1}$$

which implies,

$$\boldsymbol{y}^{\mathrm{T}}\widetilde{\mathrm{X}}\boldsymbol{\theta} = (\boldsymbol{\theta}^{\mathrm{T}}\widetilde{\mathrm{X}}^{\mathrm{T}}\boldsymbol{y})^{\mathrm{T}} = \boldsymbol{\theta}^{\mathrm{T}}\widetilde{\mathrm{X}}^{\mathrm{T}}\boldsymbol{y}$$

Therefore, we can rewrite Eq. (9) as

$$\frac{\partial}{\partial \boldsymbol{\theta}}\left(\boldsymbol{y}^{\mathrm{T}}\boldsymbol{y} - 2\boldsymbol{y}^{\mathrm{T}}\widetilde{\mathrm{X}}\boldsymbol{\theta} + \boldsymbol{\theta}^{\mathrm{T}}\widetilde{\mathrm{X}}^{\mathrm{T}}\widetilde{\mathrm{X}}\boldsymbol{\theta}\right)|_{\boldsymbol{\theta}^*} = \mathbf{0} \tag{10}$$

Now performing the differentiation yields

$$-2(\boldsymbol{y}^{\mathrm{T}}\widetilde{\mathrm{X}})^{\mathrm{T}} + \widetilde{\mathrm{X}}^{\mathrm{T}}\widetilde{\mathrm{X}}\boldsymbol{\theta}^* + (\widetilde{\mathrm{X}}^{\mathrm{T}}\widetilde{\mathrm{X}})^{\mathrm{T}}\boldsymbol{\theta}^* = \mathbf{0}$$

$$-2\widetilde{\mathrm{X}}^{\mathrm{T}}\boldsymbol{y} + 2\widetilde{\mathrm{X}}^{\mathrm{T}}\widetilde{\mathrm{X}}\boldsymbol{\theta}^* = \mathbf{0}$$

$$2\widetilde{\mathrm{X}}^{\mathrm{T}}\widetilde{\mathrm{X}}\boldsymbol{\theta}^* = 2\widetilde{\mathrm{X}}^{\mathrm{T}}\boldsymbol{y}$$

$$\boldsymbol{\theta}^* = (\widetilde{\mathrm{X}}^{\mathrm{T}}\widetilde{\mathrm{X}})^{-1}\widetilde{\mathrm{X}}^{\mathrm{T}}\boldsymbol{y} \tag{11}$$

Sufficient conditions mandate that for $\boldsymbol{\theta}^*$ to be a minima of J($\boldsymbol{\theta}$), its Hessian matrix must be positive-definite when evaluated at $\boldsymbol{\theta}^*$.

$$\boldsymbol{s}^{\mathrm{T}}\frac{\partial^2 J(\boldsymbol{\theta}^*)}{\partial \boldsymbol{\theta}^2}\boldsymbol{s} > \mathbf{0} \; \forall \boldsymbol{s} \neq \mathbf{0} \tag{12}$$

Extending Eq. (9), we have

$$\frac{\partial^2}{\partial \boldsymbol{\theta}^2}(\boldsymbol{y}^{\mathrm{T}}\boldsymbol{y} - \boldsymbol{y}^{\mathrm{T}}\tilde{\mathrm{X}}\boldsymbol{\theta} - \boldsymbol{\theta}^{\mathrm{T}}\tilde{\mathrm{X}}^{\mathrm{T}}\boldsymbol{y} + \boldsymbol{\theta}^{\mathrm{T}}\tilde{\mathrm{X}}^{\mathrm{T}}\tilde{\mathrm{X}}\boldsymbol{\theta})|_{\boldsymbol{\theta}^*} = 2\tilde{\mathrm{X}}^{\mathrm{T}}\tilde{\mathrm{X}} \quad (13)$$

Note that $\tilde{\mathrm{X}}^{\mathrm{T}}\tilde{\mathrm{X}}$ is positive semi-definite since for *any* non-zero vector $\boldsymbol{s}$

$$\boldsymbol{s}^{\mathrm{T}}\tilde{\mathrm{X}}^{\mathrm{T}}\tilde{\mathrm{X}}\boldsymbol{s} = (\tilde{\mathrm{X}}\boldsymbol{s})^{\mathrm{T}}(\tilde{X}\boldsymbol{s}) = \|\tilde{\mathrm{X}}\boldsymbol{s}\|_2^2 \geq 0 \quad (14)$$

And

$$\|\tilde{\mathrm{X}}\boldsymbol{s}\|_2^2 = 0$$

If and only if

$$\tilde{\mathrm{X}}\boldsymbol{s} = \boldsymbol{0}$$

Which can only be true if $\tilde{\mathrm{X}}$ is *not* full rank (since we restrict $\boldsymbol{s}$ to be a non-zero vector). Therefore, if $\tilde{\mathrm{X}}$ is full rank, we observe that $\tilde{\mathrm{X}}^{\mathrm{T}}\tilde{\mathrm{X}}$ is positive-definite. Which implies that $\boldsymbol{\theta}^*$ from Eq. (11) is a minima of J($\boldsymbol{\theta}$).

While this provides us all we need to develop a best-fit linear model on the data, it is interesting, and helpful, to develop a more intuitive understanding of what Eq. (11) represents (presented again below).

$$\boldsymbol{\theta}^* = (\tilde{\mathrm{X}}^{\mathrm{T}}\tilde{\mathrm{X}})^{-1}\tilde{\mathrm{X}}^{\mathrm{T}}\boldsymbol{y}$$

First, we can observe that

$$\tilde{\mathrm{X}}^{\mathrm{T}}\boldsymbol{y} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \boldsymbol{x}^{(1)} & \boldsymbol{x}^{(2)} & \cdots & \boldsymbol{x}^{(n)} \end{bmatrix}\boldsymbol{y} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(n)} \\ \vdots & \vdots & \cdots & \vdots \\ x_m^{(1)} & x_m^{(2)} & \cdots & x_m^{(n)} \end{bmatrix}\begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

Making a change of variables where $\tilde{\mathrm{X}}^{\mathrm{T}}\boldsymbol{y} = \boldsymbol{z} \in \mathbb{R}^{(m+1)}$, we can say that z_i can be thought of as the unnormalized projection of all instances of feature *i* onto the corresponding instances of output *y*. Since all features have the same scale, $\boldsymbol{z}$ represents an approximate measure of correlation strength of each feature to the output. If the largest positive and negative instances of $y^{(i)}$ tend to correspond with the largest positive and negative instances of feature *j*, respectively, then z_j will have a relatively larger positive value than other elements of $\boldsymbol{z}$. If most of the instances of feature *k* with values close to zero (meaning those unscaled feature values near the mean of feature *k*) correspond with the highest magnitude instances of the output, $\boldsymbol{y}$, and the highest magnitude instances of feature *k* correspond to the instances of the output near zero, then z_k will be closer to zero than z_j and would be considered to be lesser correlated to the output.

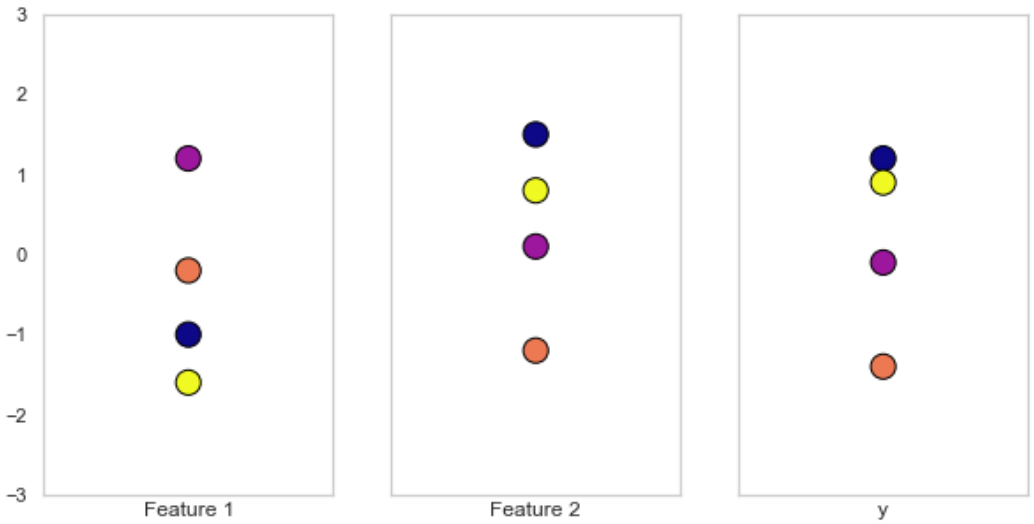


Fig. 5. With all features having approximately the same scale, it is clear that *feature 2* is more highly correlated to the output, *y*, than *feature 1*. This would result in $z_2 > z_1$. Colors indicate like data instances.

With $\boldsymbol{z}$ having the same dimension as $\boldsymbol{\theta}$, and being a measure of how correlated each feature is to the output, one would think this would be a viable option for $\boldsymbol{\theta}^*$. However, simply looking at the value of $z_o = \sum y^{(i)}$ shows us that some scaling needs to happen to $\boldsymbol{z}$ before it can be used as the coefficients in a linear model.

This is the role of the $(\tilde{\mathrm{X}}^{\mathrm{T}}\tilde{\mathrm{X}})^{-1}$ term in Eq. (11). Understanding this term will aid in understanding how to arrive at $\boldsymbol{\theta}^*$, and will also be useful when analyzing ridge regression in the next section. Using the suggested change of variables, $\tilde{\mathrm{X}}^{\mathrm{T}}\boldsymbol{y} = \boldsymbol{z}$, Eq. (11) can be rewritten as

$$\boldsymbol{\theta}^* = (\tilde{\mathrm{X}}^{\mathrm{T}}\tilde{\mathrm{X}})^{-1}\boldsymbol{z} \qquad (15)$$

To analyze the $\tilde{\mathrm{X}}^{\mathrm{T}}\tilde{\mathrm{X}}$ term, we will step back to our assumption that all data is initially scaled by the standard scaler, which yields

$$x_j^{(i)} = \frac{\hat{x}_j^{(i)} - \mu_j}{\sigma_j} \qquad (16)$$

Where $x_j^{(i)}$ is the scaled value of the i^{th} instance of the j^{th} feature, $\hat{x}_j^{(i)}$ is the unscaled value of the i^{th} instance of the j^{th} feature, μ_j is the mean of the instances of the j^{th} feature, and σ_j is the standard deviation of the j^{th} feature. Then

$$\tilde{\mathrm{X}} = [\mathbf{1}_{\boldsymbol{n}} \quad X] = \begin{bmatrix} 1 & \frac{\hat{x}_1^{(1)} - \mu_1}{\sigma_1} & \frac{\hat{x}_2^{(1)} - \mu_2}{\sigma_2} & \cdots & \frac{\hat{x}_m^{(1)} - \mu_m}{\sigma_m} \\ 1 & \frac{\hat{x}_1^{(2)} - \mu_1}{\sigma_1} & \frac{\hat{x}_2^{(2)} - \mu_2}{\sigma_2} & \cdots & \frac{\hat{x}_m^{(2)} - \mu_m}{\sigma_m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \frac{\hat{x}_1^{(n)} - \mu_1}{\sigma_1} & \frac{\hat{x}_2^{(n)} - \mu_2}{\sigma_2} & \cdots & \frac{\hat{x}_m^{(n)} - \mu_m}{\sigma_m} \end{bmatrix}$$

And $\widetilde{\mathrm{X}}^{\mathrm{T}}\widetilde{\mathrm{X}}$ can be expanded as

$$\widetilde{\mathrm{X}}^{\mathrm{T}}\widetilde{\mathrm{X}} = [\mathbf{1}_n \quad \mathrm{X}]^T[\mathbf{1}_n \quad \mathrm{X}] = \begin{bmatrix} \mathbf{1}_n{}^{\mathrm{T}} \\ \mathrm{X}^T \end{bmatrix}[\mathbf{1}_n \quad \mathrm{X}] = \begin{bmatrix} n & \mathbf{1}_n{}^{\mathrm{T}}\mathrm{X} \\ (\mathbf{1}_n{}^{\mathrm{T}}\mathrm{X})^{\mathrm{T}} & \mathrm{X}^{\mathrm{T}}\mathrm{X} \end{bmatrix}$$

$$= \begin{bmatrix} n & \frac{\sum(\hat{x}_1^{(i)} - \mu_1)}{\sigma_1} & \frac{\sum(\hat{x}_2^{(i)} - \mu_2)}{\sigma_2} & \cdots & \frac{\sum(\hat{x}_m^{(i)} - \mu_m)}{\sigma_m} \\ \frac{\sum(\hat{x}_1^{(i)} - \mu_1)}{\sigma_1} & \frac{\sum(\hat{x}_1^{(i)} - \mu_1)^2}{{\sigma_1}^2} & \frac{\sum(\hat{x}_1^{(i)} - \mu_1)(\hat{x}_2^{(i)} - \mu_2)}{\sigma_1\sigma_2} & \cdots & \frac{\sum(\hat{x}_1^{(i)} - \mu_1)(\hat{x}_m^{(i)} - \mu_m)}{\sigma_1\sigma_m} \\ \frac{\sum(\hat{x}_2^{(i)} - \mu_2)}{\sigma_2} & \frac{\sum(\hat{x}_1^{(i)} - \mu_1)(\hat{x}_2^{(i)} - \mu_2)}{\sigma_1\sigma_2} & \frac{\sum(\hat{x}_2^{(i)} - \mu_2)^2}{{\sigma_2}^2} & \cdots & \frac{\sum(\hat{x}_2^{(i)} - \mu_2)(\hat{x}_m^{(i)} - \mu_m)}{\sigma_2\sigma_m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\sum(\hat{x}_m^{(i)} - \mu_m)}{\sigma_m} & \frac{\sum(\hat{x}_1^{(i)} - \mu_1)(\hat{x}_m^{(i)} - \mu_m)}{\sigma_1\sigma_m} & \frac{\sum(\hat{x}_2^{(i)} - \mu_2)(\hat{x}_m^{(i)} - \mu_m)}{\sigma_2\sigma_m} & \cdots & \frac{\sum(\hat{x}_m^{(i)} - \mu_m)^2}{{\sigma_m}^2} \end{bmatrix}$$

$$= \begin{bmatrix} n & \frac{\sum \hat{x}_1^{(i)} - \sum \mu_1}{\sigma_1} & \frac{\sum \hat{x}_2^{(i)} - \sum \mu_2}{\sigma_2} & \cdots & \frac{\sum \hat{x}_m^{(i)} - \sum \mu_m}{\sigma_m} \\ \frac{\sum \hat{x}_1^{(i)} - \sum \mu_1}{\sigma_1} & \frac{n\mathbb{E}[(\hat{x}_1 - \mu_1)^2]}{{\sigma_1}^2} & \frac{n\mathbb{E}[(\hat{x}_1 - \mu_1)(\hat{x}_2 - \mu_2)]}{\sigma_1\sigma_2} & \cdots & \frac{n\mathbb{E}[(\hat{x}_1 - \mu_1)(\hat{x}_2 - \mu_m)]}{\sigma_1\sigma_m} \\ \frac{\sum \hat{x}_2^{(i)} - \sum \mu_2}{\sigma_2} & \frac{n\mathbb{E}[(\hat{x}_1 - \mu_1)(\hat{x}_2 - \mu_2)]}{\sigma_1\sigma_2} & \frac{n\mathbb{E}[(\hat{x}_2 - \mu_2)^2]}{{\sigma_2}^2} & \cdots & \frac{n\mathbb{E}[(\hat{x}_m - \mu_m)(\hat{x}_2 - \mu_2)]}{\sigma_m\sigma_2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\sum \hat{x}_m^{(i)} - \sum \mu_m}{\sigma_m} & \frac{n\mathbb{E}[(\hat{x}_1 - \mu_1)(\hat{x}_m - \mu_m)]}{\sigma_1\sigma_m} & \frac{n\mathbb{E}[(\hat{x}_m - \mu_m)(\hat{x}_2 - \mu_2)]}{\sigma_m\sigma_2} & \cdots & \frac{n\mathbb{E}[(\hat{x}_m - \mu_m)^2]}{{\sigma_m}^2} \end{bmatrix}$$

$$= \begin{bmatrix} n & 0 & 0 & \cdots & 0 \\ 0 & \frac{n\mathbb{E}[(\hat{x}_1 - \mu_1)^2]}{{\sigma_1}^2} & \frac{n\mathbb{E}[(\hat{x}_1 - \mu_1)(\hat{x}_2 - \mu_2)]}{\sigma_1\sigma_2} & \cdots & \frac{n\mathbb{E}[(\hat{x}_1 - \mu_1)(\hat{x}_2 - \mu_m)]}{\sigma_1\sigma_m} \\ 0 & \frac{n\mathbb{E}[(\hat{x}_1 - \mu_1)(\hat{x}_2 - \mu_2)]}{\sigma_1\sigma_2} & \frac{n\mathbb{E}[(\hat{x}_2 - \mu_2)^2]}{{\sigma_2}^2} & \cdots & \frac{n\mathbb{E}[(\hat{x}_m - \mu_m)(\hat{x}_2 - \mu_2)]}{\sigma_m\sigma_2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \frac{n\mathbb{E}[(\hat{x}_1 - \mu_1)(\hat{x}_m - \mu_m)]}{\sigma_1\sigma_m} & \frac{n\mathbb{E}[(\hat{x}_m - \mu_m)(\hat{x}_2 - \mu_2)]}{\sigma_m\sigma_2} & \cdots & \frac{n\mathbb{E}[(\hat{x}_m - \mu_m)^2]}{{\sigma_m}^2} \end{bmatrix}$$

$$= n\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & \frac{\mathbb{E}[(\hat{x}_1 - \mu_1)^2]}{{\sigma_1}^2} & \frac{\mathbb{E}[(\hat{x}_1 - \mu_1)(\hat{x}_2 - \mu_2)]}{\sigma_1\sigma_2} & \cdots & \frac{\mathbb{E}[(\hat{x}_1 - \mu_1)(\hat{x}_2 - \mu_m)]}{\sigma_1\sigma_m} \\ 0 & \frac{\mathbb{E}[(\hat{x}_1 - \mu_1)(\hat{x}_2 - \mu_2)]}{\sigma_1\sigma_2} & \frac{\mathbb{E}[(\hat{x}_2 - \mu_2)^2]}{{\sigma_2}^2} & \cdots & \frac{\mathbb{E}[(\hat{x}_m - \mu_m)(\hat{x}_2 - \mu_2)]}{\sigma_m\sigma_2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \frac{\mathbb{E}[(\hat{x}_1 - \mu_1)(\hat{x}_m - \mu_m)]}{\sigma_1\sigma_m} & \frac{\mathbb{E}[(\hat{x}_m - \mu_m)(\hat{x}_2 - \mu_2)]}{\sigma_m\sigma_2} & \cdots & \frac{\mathbb{E}[(\hat{x}_m - \mu_m)^2]}{{\sigma_m}^2} \end{bmatrix}$$

$$= n\begin{bmatrix} 1 & \mathbf{0}^{\mathrm{T}} \\ \mathbf{0} & \Pi \end{bmatrix} \qquad (17)$$

where Π is the Pearson correlation matrix for a population. Since we already determined that $\widetilde{\mathrm{X}}^{\mathrm{T}}\widetilde{\mathrm{X}}$ is positive-definite if $\widetilde{\mathrm{X}}$ is full rank (which we will assume for now), we know that $\widetilde{\mathrm{X}}^{\mathrm{T}}\widetilde{\mathrm{X}}$ has $m+1$ eigenvectors, all with positive eigenvalues. We can now easily observe from

(17) that one of the eigenvectors of $\widetilde{\mathrm{X}}^{\mathrm{T}}\widetilde{\mathrm{X}}$ is $\boldsymbol{u}_1 = \begin{bmatrix}1\\ \mathbf{0}\end{bmatrix}$ with eigenvalue *n,* where $\boldsymbol{u}_1$ is the first unit vector. We can represent the remaining eigenvalues and corresponding eigenvectors as λ_j and $\begin{bmatrix}0\\ \boldsymbol{v_j}\end{bmatrix}$ where $\boldsymbol{v}_\mathrm{j}$ is an eigenvector of $n\Pi = \mathrm{X}^{\mathrm{T}}\mathrm{X}$.

We can show that the eigenvectors of $\mathrm{X}^{\mathrm{T}}\mathrm{X}$ (and thus, fundamentally related to those of $\widetilde{\mathrm{X}}^{\mathrm{T}}\widetilde{\mathrm{X}}$) are such that the eigenvector corresponding with the largest eigenvalue is the direction of maximum variance of the data, the eigenvector corresponding to the next largest eigenvalue is the direction of second-most variance of the data, and so on.

First, we will show that the direction of maximum variance of the data is an eigenvector of $\mathrm{X}^{\mathrm{T}}\mathrm{X}$.

Let $\boldsymbol{w}$ be a vector in *m*-space such that $\|\boldsymbol{w}\|_2 = 1$, since we are only concerned with the direction of maximum variance, the magnitude of the vector does not matter.

The projection of a data instance onto $\boldsymbol{w}$ is

$$\left(\boldsymbol{x}^{(i)^{\mathrm{T}}}\boldsymbol{w}\right)\boldsymbol{w}$$

With the constraint that $\|\boldsymbol{w}\|_2 = 1$, we see that the magnitude of this projection is $\boldsymbol{x}^{(i)^{\mathrm{T}}}\boldsymbol{w}$.

The variance of a population of a scaler, *z,* is defined as

$$\sigma_z{}^2 = \frac{1}{n}\sum_{i=1}^{n}(z_i - \mu_z)^2 \qquad (18)$$

Where μ is the mean of z. Therefore, the variance of the magnitude of the projections of all standardized data instances onto $\boldsymbol{w}$ is

$$\sigma_{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{w}}{}^2 = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x}^{(i)^{\mathrm{T}}}\boldsymbol{w} - \mu_{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{w}})^{\mathbf{2}} \qquad (19)$$

where

$$\mu_{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{w}} = \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x}^{(i)^{\mathrm{T}}}\boldsymbol{w}$$

$$= \frac{1}{n}((x_1^{(1)}\boldsymbol{w}_1 + \cdots + x_m^{(1)}\boldsymbol{w}_m) + (x_1^{(2)}\boldsymbol{w}_1 + \cdots + x_m^{(2)}\boldsymbol{w}_m) + \cdots + (x_1^{(n)}\boldsymbol{w}_1 + \cdots + x_m^{(n)}\boldsymbol{w}_m))$$

$$= \frac{1}{n}(\boldsymbol{w}_1(x_1^{(1)} + \cdots + x_1^{(n)}) + \boldsymbol{w}_2(x_2^{(1)} + \cdots + x_2^{(n)}) + \cdots + \boldsymbol{w}_m(x_m^{(1)} + \cdots + x_m^{(n)}))$$

and since we assume X is standardized according to the standard scaler, then we can conclude that

$$\mu_{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{w}} = \mathbf{0} \tag{20}$$

and therefore Eq.
(19) becomes

$$\sigma_{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{w}}{}^2 = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x}^{(i)^{\mathrm{T}}}\boldsymbol{w})^2$$

$$= \frac{1}{n}((\boldsymbol{x}^{(1)^{\mathrm{T}}}\boldsymbol{w})^2 + (\boldsymbol{x}^{(2)^{\mathrm{T}}}\boldsymbol{w})^2 + \cdots + (\boldsymbol{x}^{(n)^{\mathrm{T}}}\boldsymbol{w})^2)$$

$$= \frac{1}{n}[\boldsymbol{x}^{(1)^{\mathrm{T}}}\boldsymbol{w} \quad \cdots \quad \boldsymbol{x}^{(n)^{\mathrm{T}}}\boldsymbol{w}]\begin{bmatrix}\boldsymbol{x}^{(1)^{\mathrm{T}}}\boldsymbol{w}\\ \vdots \\ \boldsymbol{x}^{(n)^{\mathrm{T}}}\boldsymbol{w}\end{bmatrix}$$

$$= \frac{1}{n}(\mathrm{X}\boldsymbol{w})^{\mathrm{T}}(\mathrm{X}\boldsymbol{w}) = \frac{1}{n}\boldsymbol{w}^{\mathrm{T}}\mathrm{X}^{\mathrm{T}}\mathrm{X}\boldsymbol{w} \tag{21}$$

To maximize this variance, we can try setting the derivative of the variance with respect to $\boldsymbol{w}$ equal to zero to find a critical point. However, upon closer inspection, we see that $\sigma_{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{w}}{}^2 \geq 0$ and a minimum critical point will occur at $\boldsymbol{w} = \mathbf{0}$. Additionally, the problem is unbounded above as it is quadratic in $\boldsymbol{w}$ and we already know that a critical point is a local minimum. However, we recall that we want to maximize the variance with the constraint that $\boldsymbol{w}$ have unit length since we only are concerned with the direction of maximum variance. With that condition in mind, we can write the optimization problem as

$$minimize\ \frac{1}{n}\boldsymbol{w}^{\mathrm{T}}\mathrm{X}^{\mathrm{T}}\mathrm{X}\boldsymbol{w} \tag{22}$$

$$\boldsymbol{s.t.}\ \|w\|_{\boldsymbol{2}} = 1$$

Since this is a constrained optimization problem, we can utilize the Lagrangian formulation

$$\mathcal{L}(\boldsymbol{w}, \lambda) = f(\boldsymbol{w}) - \lambda h(\boldsymbol{w}) \tag{23}$$

where

$$f(\boldsymbol{w}) = \frac{1}{n}\boldsymbol{w}^{\mathrm{T}}\mathrm{X}^{\mathrm{T}}\mathrm{X}\boldsymbol{w}, \quad h(\boldsymbol{w}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{w} - 1$$

Finding a stationary point, which is a necessary condition for $\boldsymbol{w}^*$ to be a solution to (22), results in the following equations being satisfied

$$\frac{\partial \mathcal{L}(\boldsymbol{w}^*, \lambda)}{\partial \lambda} = 0 \tag{24}$$

$$\frac{\partial \mathcal{L}(\boldsymbol{w}^*, \lambda)}{\partial \boldsymbol{w}} = \mathbf{0} \tag{25}$$

Observing Eq. (23), we see Eq. (24) amounts to satisfying the constraint

$$\frac{\partial \mathcal{L}(\boldsymbol{w}^*, \lambda)}{\partial \lambda} = -h(\boldsymbol{w}^*) = 1 - \boldsymbol{w}^{*\mathrm{T}}\boldsymbol{w}^* = 0 \Rightarrow \boldsymbol{w}^{*\mathrm{T}}\boldsymbol{w}^* = 1$$

and Eq. (25) yields

$$\frac{\partial \mathcal{L}(\boldsymbol{w}^*, \lambda)}{\partial \boldsymbol{w}} = \frac{\partial f(\boldsymbol{w}^*)}{\partial \boldsymbol{w}} - \lambda \frac{\partial h(\boldsymbol{w}^*)}{\partial \boldsymbol{w}} = 2\mathrm{X}^{\mathrm{T}}\mathrm{X}\boldsymbol{w}^* - 2\lambda \boldsymbol{w}^* = \mathbf{0}$$

$$2\mathrm{X}^{\mathrm{T}}\mathrm{X}\boldsymbol{w}^* = 2\lambda \boldsymbol{w}^*$$

$$\mathrm{X}^{\mathrm{T}}\mathrm{X}\boldsymbol{w}^* = \lambda \boldsymbol{w}^* \tag{26}$$

which demonstrates that $\boldsymbol{w}^*$ satisfying (22) is actually an eigenvector, $\boldsymbol{v}$, of $\mathrm{X}^{\mathrm{T}}\mathrm{X}$ with eigenvalue λ. However, we don't know which eigenvector $\boldsymbol{v}_j$ is equal to $\boldsymbol{w}^*$. We do know that $\mathrm{X}^{\mathrm{T}}\mathrm{X}$ has m eigenvectors and eigenvalues. Let us assume $\lambda_1, \dots, \lambda_m$ are the eigenvalues of $\mathrm{X}^{\mathrm{T}}\mathrm{X}$ (not necessarily, but very likely distinct when dealing with a dataset drawn from a continuous distribution), and label them such that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_m$.

Remember that Eq. (21) tells us that the variance of the magnitude of the projection of the data along $\boldsymbol{w}$ is

$$\sigma_{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{w}}{}^2 = \frac{1}{n}\boldsymbol{w}^{\mathrm{T}}\mathrm{X}^{\mathrm{T}}\mathrm{X}\boldsymbol{w}$$

Evaluating at $\boldsymbol{w}^*$ and utilizing the result from Eq. (26) we get

$$\sigma_{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{w}^*}{}^2 = \frac{1}{n}\boldsymbol{w}^{*\mathrm{T}}(\lambda \boldsymbol{w}^*) = \frac{\lambda}{n}\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{w}^* = \frac{\lambda}{n} = \frac{\lambda_j}{n} \tag{27}$$

Since $\|\boldsymbol{w}^*\|_2 = 1$ and we know that $\boldsymbol{w}^*$ is equal to some $\boldsymbol{v}_j$, the eigenvector of $\mathrm{X}^{\mathrm{T}}\mathrm{X}$ corresponding to eigenvalue λ_j.

Therefore, we can see that the variance of the magnitude of the projection of the data is maximized (Eq. (27)) when projected onto the eigenvector corresponding with λ_1, and minimized when projected onto the eigenvector corresponding with λ_m.

Furthermore, note that since $\mathrm{X}^{\mathrm{T}}\mathrm{X}$ is a symmetric, full-rank matrix, we can see that for eigenvectors $\boldsymbol{v}_i$ and $\boldsymbol{v}_k$ corresponding to eigenvalues λ_i *and* λ_k, respectively,

$$\boldsymbol{v}_i{}^{\mathrm{T}}(\mathrm{X}^{\mathrm{T}}\mathrm{X})\boldsymbol{v}_k = \boldsymbol{v}_i{}^{\mathrm{T}}(\mathrm{X}^{\mathrm{T}}\mathrm{X})^{\mathrm{T}}\boldsymbol{v}_k = ((\mathrm{X}^{\mathrm{T}}\mathrm{X})\boldsymbol{v}_i)^{\mathrm{T}}\boldsymbol{v}_k = \lambda_i \boldsymbol{v}_i{}^{\mathrm{T}}\boldsymbol{v}_k \quad (28)$$

and

$$\boldsymbol{v}_i{}^{\mathrm{T}}(\mathrm{X}^{\mathrm{T}}\mathrm{X})\boldsymbol{v}_k = \boldsymbol{v}_i{}^{\mathrm{T}}\lambda_k \boldsymbol{v}_k = \lambda_k \boldsymbol{v}_i{}^{\mathrm{T}}\boldsymbol{v}_k \quad (29)$$

Since (28) and (29) are equal, we can write

$$\lambda_i \boldsymbol{v}_i{}^{\mathrm{T}}\boldsymbol{v}_k - \lambda_k \boldsymbol{v}_i{}^{\mathrm{T}}\boldsymbol{v}_k = (\lambda_i - \lambda_k)\boldsymbol{v}_i{}^{\mathrm{T}}\boldsymbol{v}_k = 0 \quad (30)$$

Which implies that either $\lambda_i = \lambda_k$, or $\boldsymbol{v}_i$ and $\boldsymbol{v}_k$ are orthogonal. Note that even if $\lambda_i = \lambda_k$, we can still choose $\boldsymbol{v}_i$ and $\boldsymbol{v}_k$ to be orthogonal since the eigenvectors of $\mathrm{X}^{\mathrm{T}}\mathrm{X}$ span $\mathbb{R}^{mxm}$.

Finally, to summarize, we can say that the eigenvector $\boldsymbol{v}_1$ of $\mathrm{X}^{\mathrm{T}}\mathrm{X}$ is the direction of maximum variance of the data, X, while $\boldsymbol{v}_2$ is the direction maximum variance along all directions orthogonal to $\boldsymbol{v}_1$, and so on such that $\boldsymbol{v}_{\mathrm{j}}$ is the direction of maximum variance of the data along all directions orthogonal to all $\boldsymbol{v}_i$, where $i < j$. This can be seen for a two-dimensional example in Fig. 6.

Finally, now to return to Eq. (15) to understand the effect of $(\widetilde{\mathrm{X}}^{\mathrm{T}}\widetilde{\mathrm{X}})^{-1}$ on $\boldsymbol{z}$.

We have determined that the eigenvectors of $\widetilde{\mathrm{X}}^{\mathrm{T}}\widetilde{\mathrm{X}}$ are $\begin{bmatrix}1\\ \mathbf{0}\end{bmatrix}, \begin{bmatrix}0\\ \boldsymbol{v}_1\end{bmatrix}, \ldots, \begin{bmatrix}0\\ \boldsymbol{v}_m\end{bmatrix}$, with corresponding eigenvalues $n, \lambda_1, \ldots, \lambda_m$, where $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_m$.

With $\widetilde{\mathrm{X}}^{\mathrm{T}}\widetilde{\mathrm{X}}$ being a real-symmetric full-rank matrix, we can say that $\widetilde{\mathrm{X}}^{\mathrm{T}}\widetilde{\mathrm{X}}$ is diagonalizable, which means $\exists\ \mathrm{D}, \mathrm{P} \in \mathbb{R}^{(m+1)\,x\,(m+1)}\ such\ that$

$$\widetilde{\mathrm{X}}^{\mathrm{T}}\widetilde{\mathrm{X}} = \mathrm{PDP}^{-1} \quad (31)$$

where

$$D = \begin{bmatrix} n & 0 & \cdots & 0 \\ 0 & \lambda_1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \lambda_m \end{bmatrix}$$

$$P = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ \mathbf{0} & \boldsymbol{v}_1 & \cdots & \boldsymbol{v}_m \end{bmatrix}$$

which implies that $(\widetilde{\mathrm{X}}^{\mathrm{T}}\widetilde{\mathrm{X}})^{-1}$ can be written as

$$(\widetilde{\mathrm{X}}^{\mathrm{T}}\widetilde{\mathrm{X}})^{-1} = (\mathrm{PDP}^{-1})^{-1} = \mathrm{PD}^{-1}\mathrm{P}^{-1} \quad (32)$$

where

$$\mathrm{D}^{-1} = \begin{bmatrix} \frac{1}{n} & 0 & \cdots & 0 \\ 0 & \frac{1}{\lambda_1} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \frac{1}{\lambda_m} \end{bmatrix}$$

So we can rewrite Eq. (15) as

$$\boldsymbol{\theta}^* = \mathrm{A}\boldsymbol{z} \tag{33}$$

where

$$\mathrm{A} = \mathrm{P}\mathrm{D}^{-1}\mathrm{P}^{-1} \tag{34}$$

and

$$\boldsymbol{z} = \widetilde{\mathrm{X}}^{\mathrm{T}}\boldsymbol{y} \tag{35}$$

This implies that $\boldsymbol{\theta}^*$ is found by transforming $\boldsymbol{z}$ via a matrix which has eigenvectors equal to those of $\widetilde{\mathrm{X}}^{\mathrm{T}}\widetilde{\mathrm{X}}$, but has corresponding eigenvalues $\frac{1}{n}, \frac{1}{\lambda_1}, \ldots, \frac{1}{\lambda_m}$. To understand how this transformation works, we observe that since the eigenvectors of A span $\mathbb{R}^{m+1}$, $\boldsymbol{z}$ can be written as a linear combination of these eigenvectors

$$\boldsymbol{z} = \rho_0 \begin{bmatrix} 1 \\ \mathbf{0} \end{bmatrix} + \rho_1 \begin{bmatrix} 0 \\ \boldsymbol{v}_1 \end{bmatrix} + \cdots + \rho_m \begin{bmatrix} 0 \\ \boldsymbol{v}_m \end{bmatrix} \tag{36}$$

Note that ρ_0 is simply equal to the sum of all outputs $y^{(i)}$, and ρ_i is the magnitude of the projection of $\boldsymbol{z}$ onto $\begin{bmatrix} 0 \\ \boldsymbol{v}_i \end{bmatrix}$, since all eigenvectors of A are orthogonal.

Eq. (33) can now be written as

$$\boldsymbol{\theta}^* = \mathrm{A}\boldsymbol{z}$$

$$= \boldsymbol{A}\left(\rho_0 \begin{bmatrix} 1 \\ \mathbf{0} \end{bmatrix} + \rho_1 \begin{bmatrix} 0 \\ \boldsymbol{v}_1 \end{bmatrix} + \cdots + \rho_m \begin{bmatrix} 0 \\ \boldsymbol{v}_m \end{bmatrix}\right)$$

$$= \rho_0 A \begin{bmatrix} 1 \\ \mathbf{0} \end{bmatrix} + \rho_1 A \begin{bmatrix} 0 \\ \boldsymbol{v}_1 \end{bmatrix} + \cdots + \rho_m A \begin{bmatrix} 0 \\ \boldsymbol{v}_m \end{bmatrix}$$

$$= \rho_0 \frac{1}{n} \begin{bmatrix} 1 \\ \mathbf{0} \end{bmatrix} + \rho_1 \frac{1}{\lambda_1} \begin{bmatrix} 0 \\ \boldsymbol{v}_1 \end{bmatrix} + \cdots + \rho_m \frac{1}{\lambda_m} \begin{bmatrix} 0 \\ \boldsymbol{v}_m \end{bmatrix}$$

$$= \bar{y} \begin{bmatrix} 1 \\ \mathbf{0} \end{bmatrix} + \frac{\rho_1}{\lambda_1} \begin{bmatrix} 0 \\ \boldsymbol{v}_1 \end{bmatrix} + \cdots + \frac{\rho_m}{\lambda_m} \begin{bmatrix} 0 \\ \boldsymbol{v}_m \end{bmatrix}$$

$$= \bar{y}\begin{bmatrix}1\\\mathbf{0}\end{bmatrix} + \gamma_1\begin{bmatrix}0\\\boldsymbol{v}_1\end{bmatrix} + \cdots + \gamma_m\begin{bmatrix}0\\\boldsymbol{v}_m\end{bmatrix} \quad (37)$$

where $\bar{y}$ is the average of all the output values, and is thus the value of θ_0.

A simple example showing the roles of the eigenvectors and eigenvalues of the features, as well as the representation of ρ_i *and* γ_i is shown in Figure 6. The example dataset has two features and a single target with no bias term. The relationship between each feature and the target variable is shown in Figure 7.

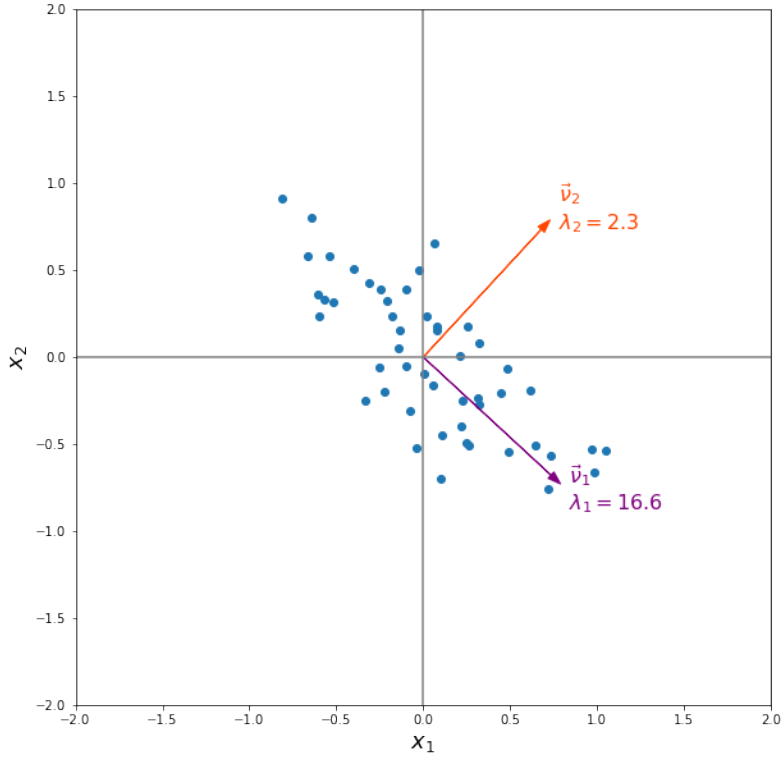


Fig. 6. Distribution of feature values, x_1 and x_2. The eigenvectors and eigenvalues of X^TX are shown. The relationship between eigenvectors and directions of maximum variance of the data can be seen as well.

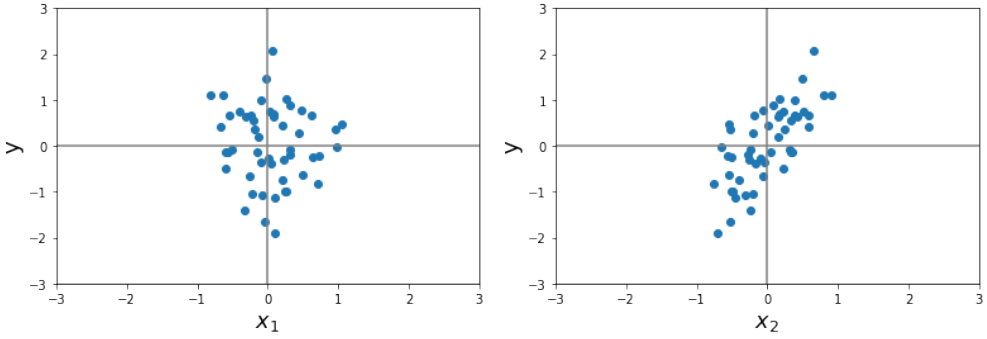


Fig. 7. Feature instances plotted individually versus output, y. Since x_2 appears to have a fairly strong positive correlation with y, while x_1 appears to have a weaker, slightly

negative correlation with y, we can see that z_2 will be a relatively large, positive value, while z_1 will be slightly negative.

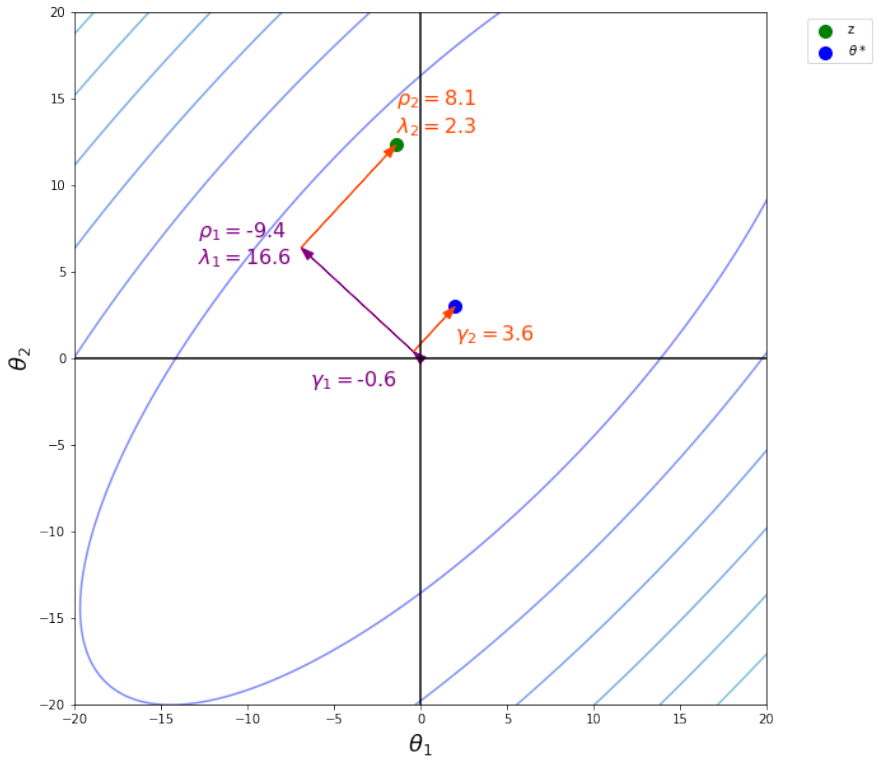


Fig. 8. The contours of the objective function (7) are shown along with the locations of $\boldsymbol{z}$ and $\boldsymbol{\theta}^*$ in $\boldsymbol{\theta}$-space. Both points are shown as a linear combination of the eigenvalues of X^TX. It can be seen that $\boldsymbol{\theta}^*$ is achieved by reducing the $\boldsymbol{v}_1$ component of $\boldsymbol{z}$ by a factor of λ_1, and reducing the $\boldsymbol{v}_2$ component of $\boldsymbol{z}$ by a factor of λ_2.

The application of linear regression to the weather dataset will be shown in the next section along with ridge regression.

Ridge Regression

Ridge regression, or Tikhonov regularization, is a regression technique similar to linear regression, but with a regularization factor included. Regularization is a technique used to better generalize a model to yet-unseen data. This is especially important in situations where the amount of data collected is limited. In the case of ridge regression, the regularization factor is applied to the squared 2-norm of $\boldsymbol{\theta}$. It will be seen that this is equivalent to restricting the magnitude of $\boldsymbol{\theta}$.

The objective function for ridge regression is generally stated in a form similar to

$$J(\boldsymbol{\theta}) = (\boldsymbol{y} - \tilde{\mathrm{X}}\boldsymbol{\theta})^{\mathrm{T}}(\boldsymbol{y} - \tilde{\mathrm{X}}\boldsymbol{\theta}) \; + \; \alpha\|\boldsymbol{\theta}\|_2^2 \tag{38}$$

where $\alpha \geq 0$.

This can be seen to be equivalent to the linear regression objective function, Eq. (7), when $\alpha = 0$.

By rewriting Eq. (38) as a constrained optimization problem, and utilizing Lagrangian optimization techniques [6], the solution, $\boldsymbol{\theta}^*$, which minimizes Eq. (38) can be seen to be equivalent to the solution of

$$\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta})$$

$$f(\boldsymbol{\theta}) \; = (\boldsymbol{y} - \tilde{\mathrm{X}}\boldsymbol{\theta})^{\mathrm{T}}(\boldsymbol{y} - \tilde{\mathrm{X}}\boldsymbol{\theta})$$

$$\text{subject to: } \|\boldsymbol{\theta}\|_2^2 \leq c \tag{39}$$

The Lagrangian formulation of this problem is

$$\mathcal{L}(\boldsymbol{\theta}, \mu) = \; (\boldsymbol{y} - \tilde{\mathrm{X}}\boldsymbol{\theta})^{\mathrm{T}}(\boldsymbol{y} - \tilde{\mathrm{X}}\boldsymbol{\theta}) + \mu(\|\boldsymbol{\theta}\|_2^2 - c) \tag{40}$$

The solution to this is found using the Karush-Kuhn-Tucker (KKT) conditions

1. Stationarity:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}|_{\boldsymbol{\theta}^*} = \frac{\partial}{\partial \boldsymbol{\theta}}((\boldsymbol{y} - \tilde{\mathrm{X}}\boldsymbol{\theta})^{\mathrm{T}}(\boldsymbol{y} - \tilde{\mathrm{X}}\boldsymbol{\theta}) + \mu(\|\boldsymbol{\theta}\|_2^2 - c))|_{\boldsymbol{\theta}^*} = \mathbf{0} \tag{41}$$

2. Primal Feasibility:

$$\|\boldsymbol{\theta}^*\|_2^2 - c \leq 0$$

3. Dual Feasibility:

$$\mu \geq 0$$

4. Complementary Slackness:

$$\mu(\|\boldsymbol{\theta}^*\|_2^2 - c) = 0$$

If μ rather than c is selected by the user, then by letting c always equal $\|\boldsymbol{\theta}^*\|_2^2$, we can see that with a change of variables, $\mu = \alpha$ the solution to (39) is identical to the solution to (38). Thus, we can view ridge regression as a constrained optimization problem. This constraint on the magnitude of $\boldsymbol{\theta}^*$, is included to not only potentially help avoid overfitting $\boldsymbol{\theta}^*$ to the training data, and thereby helping the linear regression to generalize better to yet-unseen data, but it can also help to demonstrate which features are most important in predicting the output. Ridge regression accomplished this by reducing the coefficients of the lesser-significant features more and more as α is increased.

To find the ridge regression solution, $\boldsymbol{\theta}^*$, and see how it compares to $\boldsymbol{\theta}^{\mathrm{OLR}}$, where $\boldsymbol{\theta}^{\mathrm{OLR}}$ is the solution to the ordinary linear regression problem previously analyzed, we will perform the differentiation in Eq. (41) (proceeding with the replacement of μ with α).

First, to make this differentiation easier, Eq. (40) can be rewritten and expanded as

$$\mathcal{L}(\boldsymbol{\theta}, \alpha) = (\boldsymbol{y} - \tilde{\mathrm{X}}\boldsymbol{\theta})^{\mathrm{T}}(\boldsymbol{y} - \tilde{\mathrm{X}}\boldsymbol{\theta}) + \alpha(\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{\theta} - c)$$

$$\mathcal{L}(\boldsymbol{\theta}, \alpha) = \boldsymbol{y}^{\mathrm{T}}\boldsymbol{y} - 2\boldsymbol{\theta}^{\mathrm{T}}\tilde{\mathrm{X}}^{\mathrm{T}}\boldsymbol{y} + \boldsymbol{\theta}^{\mathrm{T}}\tilde{\mathrm{X}}^{\mathrm{T}}\tilde{\mathrm{X}}\boldsymbol{\theta} + \alpha\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{\theta} - \alpha c \qquad (42)$$

Now performing the differentiation in Eq. (41) yields

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}|_{\boldsymbol{\theta}^*} = -2\tilde{\mathrm{X}}^{\mathrm{T}}\boldsymbol{y} + \tilde{\mathrm{X}}^{\mathrm{T}}\tilde{\mathrm{X}}\boldsymbol{\theta}^* + (\tilde{\mathrm{X}}^{\mathrm{T}}\tilde{\mathrm{X}})^{\mathrm{T}}\boldsymbol{\theta}^* + 2\alpha\boldsymbol{\theta}^*$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}|_{\boldsymbol{\theta}^*} = -2\tilde{\mathrm{X}}^{\mathrm{T}}\boldsymbol{y} + 2\tilde{\mathrm{X}}^{\mathrm{T}}\tilde{\mathrm{X}}\boldsymbol{\theta}^* + 2\alpha\boldsymbol{\theta}^*$$

$$-2\tilde{\mathrm{X}}^{\mathrm{T}}\boldsymbol{y} + 2\tilde{\mathrm{X}}^{\mathrm{T}}\tilde{\mathrm{X}}\boldsymbol{\theta}^* + 2\alpha\boldsymbol{\theta}^* = \mathbf{0}$$

$$\left(\tilde{\mathrm{X}}^{\mathrm{T}}\tilde{\mathrm{X}} + \alpha\mathrm{I}_{m+1}\right)\boldsymbol{\theta}^* = \tilde{\mathrm{X}}^{\mathrm{T}}\boldsymbol{y}^{\mathrm{T}} \qquad (43)$$

where I_{m+1} is the (m+1) x (m+1) identity matrix.

Now solving for $\boldsymbol{\theta}^*$ yields

$$\boldsymbol{\theta}^* = (\tilde{\mathrm{X}}^{\mathrm{T}}\tilde{\mathrm{X}} + \alpha\mathrm{I}_{m+1})^{-1}\tilde{\mathrm{X}}^{\mathrm{T}}\boldsymbol{y} \qquad (44)$$

We can use the identity from Eq. (31) to rewrite this as

$$\boldsymbol{\theta}^* = (\mathrm{PDP}^{-1} + \alpha\mathrm{PP}^{-1})^{-1}\tilde{\mathrm{X}}^{\mathrm{T}}\boldsymbol{y}$$

$$\boldsymbol{\theta}^* = (\mathrm{PDP}^{-1} + \mathrm{P}(\alpha\mathrm{I}_{\mathrm{m+1}})\mathrm{P}^{-1})^{-1}\tilde{\mathrm{X}}^{\mathrm{T}}\boldsymbol{y}$$

$$\boldsymbol{\theta}^* = (\mathrm{P}(\mathrm{D} + \alpha\mathrm{I}_{\mathrm{m+1}})\mathrm{P}^{-1})^{-1}\tilde{\mathrm{X}}^{\mathrm{T}}\boldsymbol{y}$$

$$\boldsymbol{\theta}^* = \mathrm{P}(\mathrm{D} + \alpha \mathrm{I}_{\mathrm{m+1}})^{-1}\mathrm{P}^{-1}\widetilde{\mathrm{X}}^{\mathrm{T}}\boldsymbol{y}$$

$$\boldsymbol{\theta}^* = \mathrm{P}\mathrm{K}^{-1}\mathrm{P}^{-1}\widetilde{\mathrm{X}}^{\mathrm{T}}\boldsymbol{y} \tag{45}$$

where $\mathrm{K} = \mathrm{D} + \alpha \mathrm{I}_{\mathrm{m+1}}$

$$\mathrm{K} = \begin{bmatrix} n+\alpha & 0 & \cdots & 0 \\ 0 & \lambda_1 + \alpha & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \lambda_m + \alpha \end{bmatrix}$$

which implies

$$\mathrm{K}^{-1} = \begin{bmatrix} \frac{1}{n+\alpha} & 0 & \cdots & 0 \\ 0 & \frac{1}{\lambda_1 + \alpha} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \frac{1}{\lambda_m + \alpha} \end{bmatrix}$$

Eq. (45) can also be rewritten as

$$\boldsymbol{\theta}^* = \mathrm{S}\boldsymbol{z} \tag{46}$$

where

$$\mathrm{S} = \mathrm{P}\mathrm{K}^{-1}\mathrm{P}^{-1} \tag{47}$$

which implies that $\boldsymbol{\theta}^*$ is found by transforming $\boldsymbol{z}$ by a matrix with eigenvectors equal to those of $\widetilde{\mathrm{X}}^{\mathrm{T}}\widetilde{\mathrm{X}}$ as well as A, but has corresponding eigenvalues $\frac{1}{n+\alpha}, \frac{1}{\lambda_1+\alpha}, \ldots, \frac{1}{\lambda_m+\alpha}$.

Comparing linear regression to ridge regression, we can see that for both cases $\boldsymbol{\theta}^*$ is obtained by transforming $\boldsymbol{z}$ along the eigenvectors of $\widetilde{\mathrm{X}}^{\mathrm{T}}\widetilde{\mathrm{X}}$. However, during ridge regression, the eigenvalues are all smaller than those during linear regression (since $\alpha,\ \lambda_i > 0$). The amount each eigenvalue decreases is correlated to the magnitude of the original eigenvalue (λ_i). If λ_i is small compared to α, α will have a large impact and $\frac{1}{\lambda_i} \gg \frac{1}{\lambda_i+\alpha}$. However, if λ_i is large, then α will not have much impact and $\frac{1}{\lambda_i} \approx \frac{1}{\lambda_i+\alpha}$. We can say that to get to $\boldsymbol{\theta}^*$ from $\boldsymbol{\theta}^{\mathrm{OLR}}$, $\boldsymbol{\theta}^{\mathrm{OLR}}$ is reduced to a large extent along the directions of low variance of the feature data, while only slightly reduced along the directions of high variance.

As done in the linear regression section, we can express $\boldsymbol{\theta}^*$ as a linear combination of the eigenvectors of $\widetilde{\mathrm{X}}^{\mathrm{T}}\widetilde{\mathrm{X}}$. Using Eq. (36), we can rewrite Eq. (46) as

$$\boldsymbol{\theta}^* = S(\rho_0 \begin{bmatrix}1\\ \mathbf{0}\end{bmatrix} + \rho_1 \begin{bmatrix}0\\ \boldsymbol{v}_1\end{bmatrix} + \cdots + \rho_m \begin{bmatrix}0\\ \boldsymbol{v}_m\end{bmatrix})$$

$$= \rho_0 S \begin{bmatrix}1\\ \mathbf{0}\end{bmatrix} + \rho_1 S \begin{bmatrix}0\\ \boldsymbol{v}_1\end{bmatrix} + \cdots + \rho_m S \begin{bmatrix}0\\ \boldsymbol{v}_m\end{bmatrix}$$

$$= \rho_0 \frac{1}{n+\alpha} \begin{bmatrix}1\\ \mathbf{0}\end{bmatrix} + \rho_1 \frac{1}{\lambda_1+\alpha} \begin{bmatrix}0\\ \boldsymbol{v}_1\end{bmatrix} + \cdots + \rho_m \frac{1}{\lambda_m+\alpha} \begin{bmatrix}0\\ \boldsymbol{v}_m\end{bmatrix}$$

$$= \frac{\rho_0}{n+\alpha} \begin{bmatrix}1\\ \mathbf{0}\end{bmatrix} + \frac{\rho_1}{\lambda_1+\alpha} \begin{bmatrix}0\\ \boldsymbol{v}_1\end{bmatrix} + \cdots + \frac{\rho_m}{\lambda_m+\alpha} \begin{bmatrix}0\\ \boldsymbol{v}_m\end{bmatrix}$$

$$= \gamma_0^{(ridge)} \begin{bmatrix}1\\ \mathbf{0}\end{bmatrix} + \gamma_1^{(ridge)} \begin{bmatrix}0\\ \boldsymbol{v}_1\end{bmatrix} + \cdots + \gamma_m^{(ridge)} \begin{bmatrix}0\\ \boldsymbol{v}_m\end{bmatrix} \qquad (48)$$

Continuing the two-dimensional example from the linear regression section and Figure 8, we can see how an optimal solution is shifted to have lower magnitude in ridge regression in Figures 9 and 10.

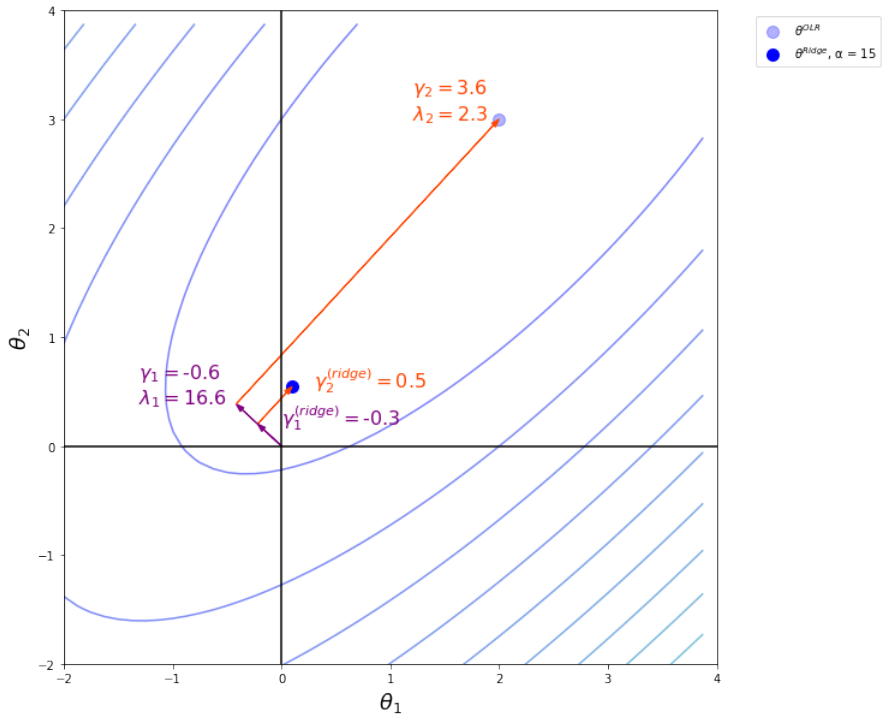


Fig. 9. The contours of the objective function (7) are shown along with the locations of $\boldsymbol{\theta}^{\mathrm{OLR}}$, the linear regression solution, and $\boldsymbol{\theta}^*$, the ridge regression solution when $\alpha = 15$. Both points are shown as a linear combination of the eigenvalues of $\mathrm{X}^{\mathrm{T}}\mathrm{X}$. It can be seen that the $\boldsymbol{v}_2$ component of $\boldsymbol{\theta}^*$ is greatly reduced compared to that of $\boldsymbol{\theta}^{\mathrm{OLR}}$. By observing Eq. (48), this can be seen to be due to $\boldsymbol{v}_2$ being associated with a much smaller eigenvalue than $\boldsymbol{v}_1$ with α being approximately 6.5 times larger than λ_2, while only about 0.90 times the size of λ_1.

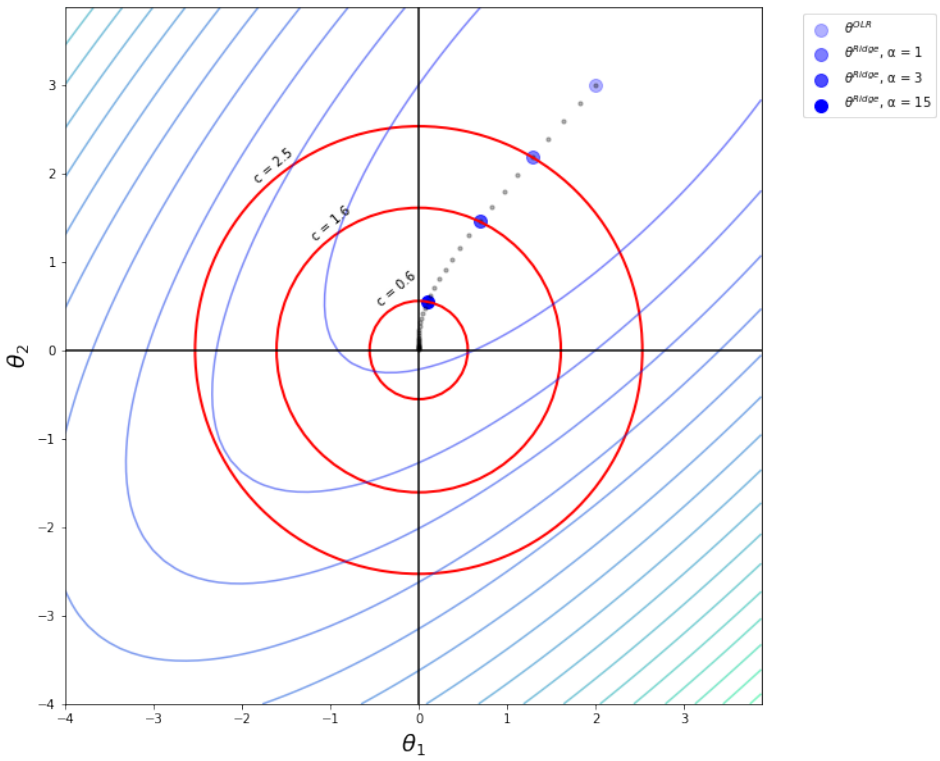


Fig. 10. Ridge regression as constrained optimization.
The contours of the objective function (7) are shown along with the trajectory of ridge regression solutions (small gray points) as α is increased from zero, where the solution is the same as that of linear regression, to a value great enough such that $\boldsymbol{\theta}^*$ converges to the origin. Selected constraints on the magnitude of $\boldsymbol{\theta}^*$ are shown in red, along with their corresponding values of α in the legend.

Note that in this example, the ridge regression solution is pulled more so along the θ_1 axis than θ_2. This indicates that the output, y, must be more sensitive to variation in x_2 than x_1. Observe, however, that the value of optimal θ_1 is pulled *towards* but not *to* zero. In fact, it can be seen that θ_1 actually becomes slightly negative for some high values of α before being pulled again towards zero.

We have seen how the optimal solution moves as α is increased, but we have not discussed how to select a good value of α. This is where the concept of a hyperparameter comes into play. We have seen model parameters (the components of $\boldsymbol{\theta}$ are all model parameters**)** and how they are solved for. Now we have our first hyperparameter. This is a variable (or variables) that must be selected by the user that impact what will be the ultimate model parameters. Selecting the best hyperparameters to use depends on the data and the goal of the model. Care must be taken to create a good metric and method by which to measure hyperparameter selection.

Recall that we split our data into training, validation, and testing sets. Our workflow will be as follows to select optimal hyperparameters:

1. Select a set of candidate hyperparameter values to test. A smaller set will result in faster training, while a larger set will have a lower chance of missing an optimal hyperparameter selection if the range and density of each hyperparameter set are chosen appropriately

2. Fit the model using the first hyperparameter options and only the training dataset.

3. Evaluate the performance of the model on the validation set using a pre-selected objective function (which ought to be very similar to the objective function internal to each model) and record both the value of the objective and the set of hyperparameter values used to fit this model.

4. Repeat this process for all hyperparameters, or until some benchmark is met, such as validation performance becoming consistently worse as a hyperparameter is adjusted in a particular direction.

5. Select the set of hyperparameters which resulted in the best model performance on the validation set.

6. Fit a final model, using the hyperparameters selected in step 5, on the combination of the training and validation sets.

7. Finally, evaluate the performance of this final model on the testing set. It is important to not go back and adjust hyperparameters further based on this performance. A primary objective of machine learning is to generalize well to yet-unseen data. "Peeking" at the testing set and then going back and tweaking the model based on this would corrupt this workflow and not tell us much about how the model is likely to generalize.

For ridge regression and our weather dataset, the optimal choice of α can be made by fitting the training data to ridge regression models with a range of α values, calculating the RMSE error for this model on the validation data, and comparing the models. Finally, the optimal choice of α can be made, and then a new model can be fit on the combination of the training and validation data. Ultimately the RMSE can be calculated on the testing data.

The plots of validation set RMSE vs alpha parameters are below:

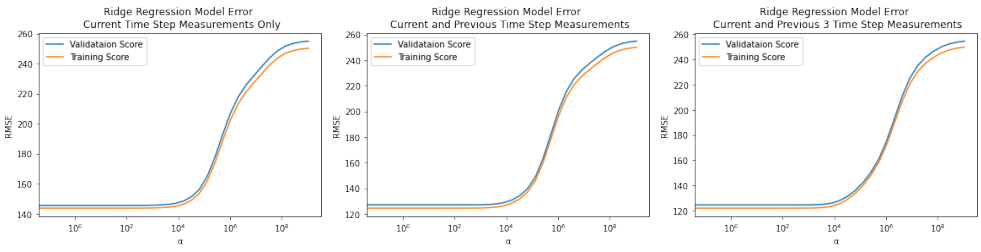


Fig. 11. Ridge regression model performance as α is adjusted.
Validation and training score (RMSE) vs α for ridge regression models based on all three sets of features. In all cases, the best model appears to be one with $\alpha = 0$, which is simply linear regression.

As can be seen, for all of these datasets, the model performed best as a simple linear model without regularization. This is likely due to the large number of instances in the dataset already proving sufficient information for generalization of the linear model to yet-unseen data. While in each of these models there are too many coefficients, θ_i, to visualize how they change in $\boldsymbol{\theta}$-space as α changes, we can plot the values for a few select coefficients, θ_i, vs. α. The plot in Figure 12 was constructed using the dataset consisting of only current measurements.

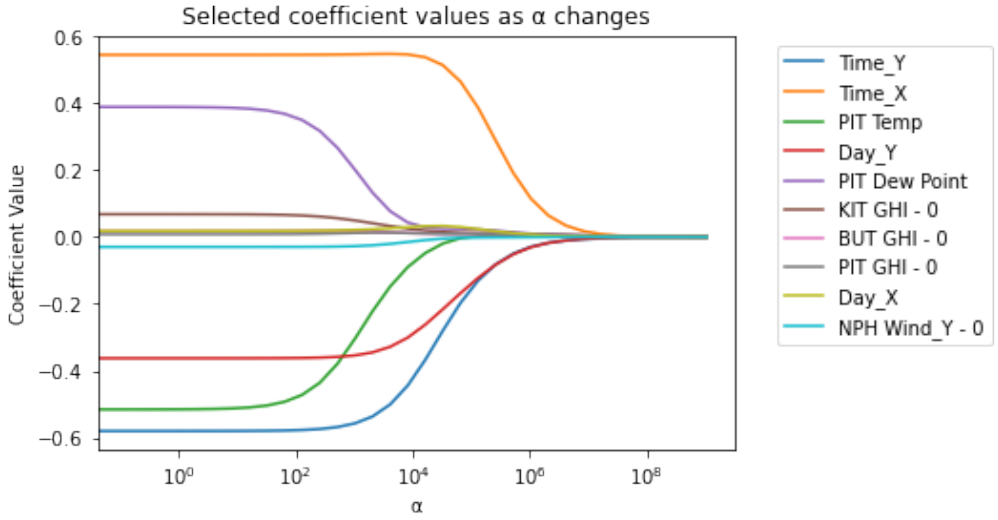


Fig. 12. Visualization of how the magnitudes of selected coefficients are reduced as α is increased.

As can be seen, high values of α drive the magnitude of all parameters towards zero, but some have their magnitude decreased sooner, or at a greater rate (see PIT Temp compared to Time_Y in Figure 12, for example). Note that none of the parameter values ever equal zero, they simply approach zero asymptotically. We can also see the reduction of the 2-norm of $\boldsymbol{\theta}^*$ as α increases in Figure 13.

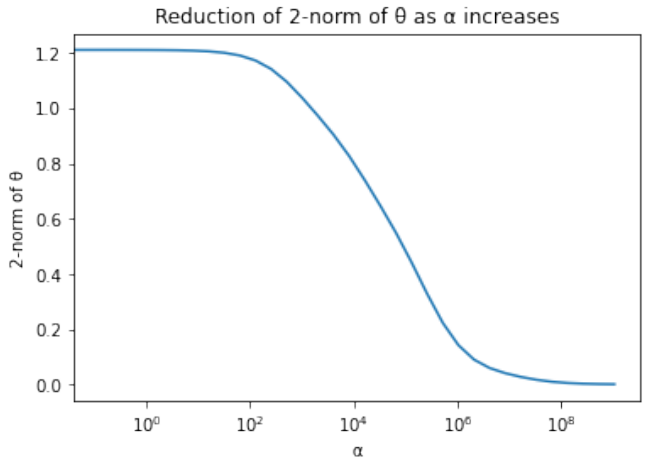


Fig. 13. Visualization of how the overall magnitude of the optimal parameter vector ($\boldsymbol{\theta}^*$) decreases as α increases.

Since a simple linear regression model performed best on validation data for all three models, all three models were fit to the combined training-validation set with alpha set to 0, and then evaluated on the testing set.

Table V
Evaluation of Linear Models on Testing Sets

Linear model fit with:	Current measurements only	Current and previous measurements	Current and previous three measurements
RMSE	141.44	121.47	119.71

As can be seen in Table V, by including past weather measurements, a linear model can more accurately forecast the GHI in Pittsburgh 4 hours ahead of time. An example of the predicted GHI on the testing set is shown on the next page in Figure 14. This segment of testing data was selected due to the presence of a few days of highly volatile GHI (indicating intermittent cloud cover) and a few days of very smooth GHI curves (indicating clear, sunny days), so that the impacts of both can be seen.

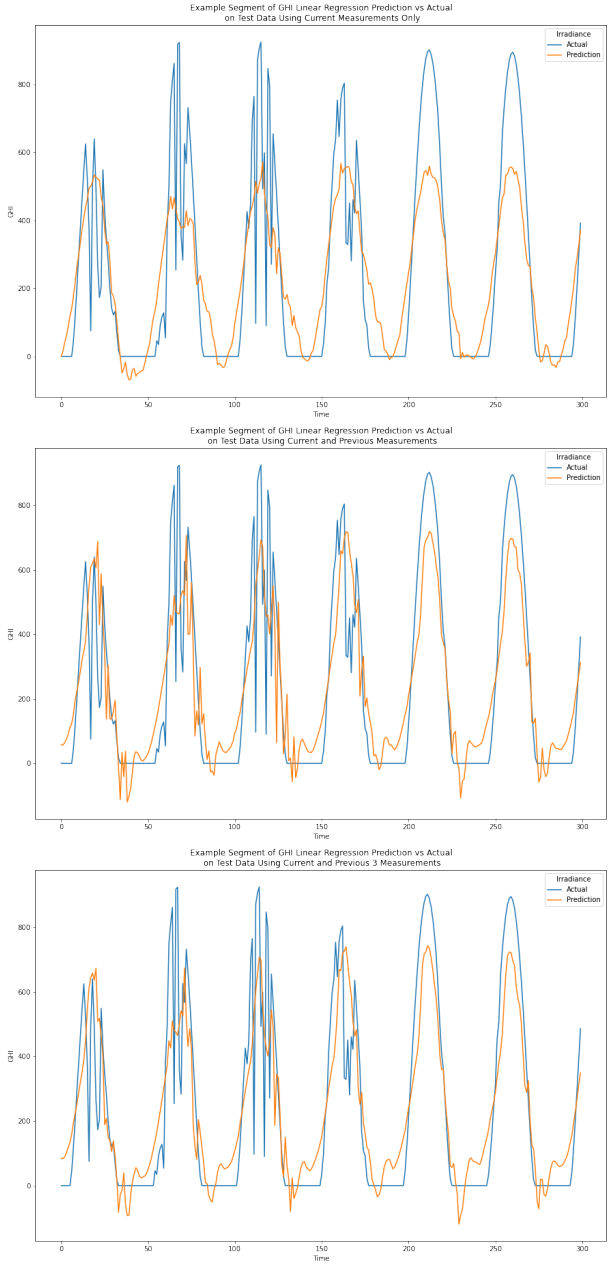


Fig. 14. Comparisons of linear regression model predictions (orange) and actual observed values (blue) of GHI in Pittsburgh. The top plot shows the model when only current weather measurements are used to make predictions. The middle plot shows the model when the current and previous timestep (30 minutes prior) measurements are used. The bottom plot shows the model when the current and previous three timestep (30 minutes, 1 hour, and 1.5 hours prior) measurements are used. The last model has the best predictive capability of the three.

Note that all models often predict negative GHI values (something that is physically impossible) and struggle to predict a consistent value of zero at night, something that would be intuitive to a human, but the model does not readily learn. General trends in the GHI are successfully predicted, but rapid changes, such as those seen in the first 4 days shown, are not necessarily predicted well.

To examine more quantitatively the values of the coefficients in the models, we can look at the five largest-magnitude coefficients in the various models, both when linear regression is applied, and when ridge regression with a high α value is applied.

Table VI
High-Magnitude Coefficient Values in Various Linear and Ridge Regression Models

Current-only measurements			
Linear Model Highest-magnitude coefficients		Ridge-regression with high α ($\alpha = 2^{15}$) Highest-magnitude coefficients	
Time_Y	-0.583	Time_X	0.525
Time_X	0.547	Time_Y	-0.309
PIT Temperature – 0	-0.523	Day_Y	-0.230
PIT Dew Point – 0	0.400	PIT Clearsky DNI – 0	-0.075
Day_Y	-0.363	CBG Clearsky DHI - 0	-0.053

Current and previous measurements			
Linear Model Highest-magnitude coefficients		Ridge-regression with high α ($\alpha = 2^{15}$) Highest-magnitude coefficients	
Time_Y	-0.602	Time_X	0.376
PIT Clearsky DNI - 0	-0.367	Time_Y	-0.310
Day_Y	-0.355	Day_Y	-0.222
Time_X	0.350	PIT Clearsky DNI – 0	-0.100
YGT GHI - 1	-0.199	PKS Global Horzontal UV Irradiance - 1	-0.063

Current and previous three measurements			
Linear Model Highest-magnitude coefficients		Ridge-regression with high α ($\alpha = 2^{15}$) Highest-magnitude coefficients	
Time_Y	-0.663	Time_Y	-0.331
PIT Clearsky DNI - 0	-0.406	Time_X	0.312
Day_Y	-0.375	Day_Y	-0.233
Time_X	0.297	PIT Clearsky DNI – 0	-0.121
PIT Clearsky GHI - 0	-0.186	Day_X	0.061

For all of these models, Time_Y, Time_X, and Day_Y are among the most impactful features. Recall that large negative Time_Y values correspond with mid-day, while large positive values correspond with night-time. Large negative Time_X values correspond with the evening, while large positive values correspond with the morning. Summertime is represented by large negative Day_Y values, and winter by large positive Day_Y values. Intuitively, all these features have a consistent, predictable impact on GHI.

A primary difference to note between the linear models and the ridge-regression models is the relative significance of the feature with the highest-magnitude coefficient. For example, when using the current and previous three measurements, the ratio of the absolute values of the first and fifth largest coefficients in the linear model is 3.56, while for the ridge regression model it is 5.43. This indicates how ridge regression tries to maintain a large magnitude coefficient for only the

most significant features. It should also be noted that for none of these models were any coefficients identically zero.

Since for this weather data, linear regression was the best model to use (compared to any ridge regression model with non-zero α), one could wonder why we went through the trouble of analyzing ridge regression at all. For many datasets, ridge regression does indeed help the model generalize better to unseen data. In particular, when there are a more limited number of data instances (especially in relation to the number of features), a linear regression model is likely to give too much significance to relative outliers in the data.

We can see how ridge regression with non-zero α can perform better on yet-unseen data for smaller datasets by sampling a limited number of instances from our training set, and then repeating the same hyperparameter tuning process as above where fitted models are evaluated on the validation set.

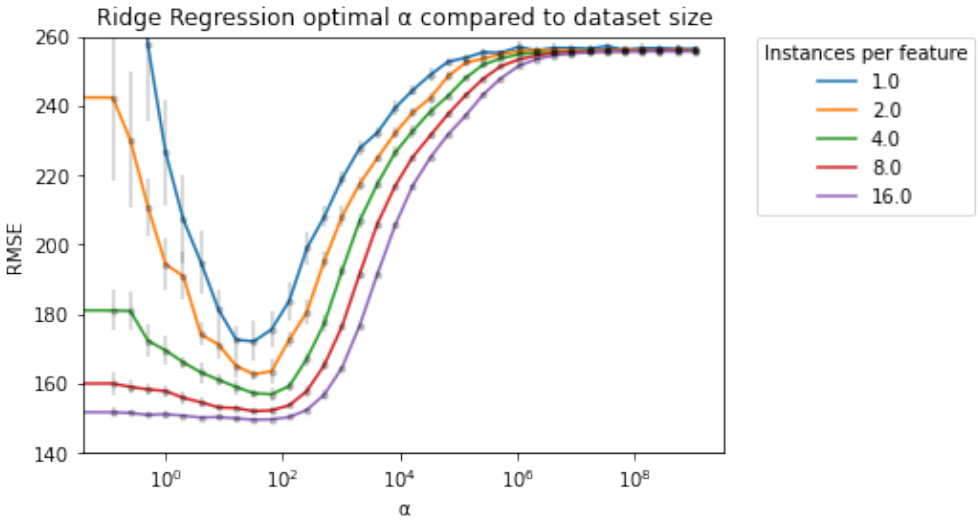


Fig. 15. Comparison of the benefits of ridge regression for datasets of varying size
Each dataset size is represented in terms of number of instances per number of features. Gray error bars represent the standard deviation of validation RMSE scores for random sub-sampling of the training set data used to fit the model.
For this data, training sets with a low number of instances have a fairly well-defined optimal value of α for which the model generalized best. But as the training set becomes larger, the optimal α becomes less well defined as low- α models perform better and better until linear regression becomes the best-generalizing model. For all values of α, however, a model trained with a larger dataset generalizes better than a smaller dataset. This demonstrates the usefulness of ridge regression, but the value of additional data.

Lasso Regression

Lasso Regression, or Least Absolute Shrinkage and Selection Operator, is a regularization technique similar to ridge regression, but which uses the 1-norm rather than the 2-norm constraint on $\boldsymbol{\theta}$.

The objective function for ridge regression is generally stated in a form similar to

$$J(\boldsymbol{\theta}) = (\boldsymbol{y} - \tilde{\mathrm{X}}\boldsymbol{\theta})^{\mathrm{T}}(\boldsymbol{y} - \tilde{\mathrm{X}}\boldsymbol{\theta}) \; + \; \alpha\|\boldsymbol{\theta}\|_1 \tag{49}$$

where $\alpha \geq 0$.

This can again be seen to be equivalent to the linear regression objective function, Eq. (7), when $\alpha = 0$.

By rewriting Eq. (49) as a constrained optimization problem, and utilizing Lagrangian optimization techniques, the solution, $\boldsymbol{\theta}^*$, which minimizes Eq. (49) can be seen to be equivalent to the solution of [6]

$$\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta})$$

$$f(\boldsymbol{\theta}) \; = (\boldsymbol{y} - \tilde{\mathrm{X}}\boldsymbol{\theta})^{\mathrm{T}}(\boldsymbol{y} - \tilde{\mathrm{X}}\boldsymbol{\theta})$$

$$\text{subject to: } \|\boldsymbol{\theta}\|_1 \leq c \tag{50}$$

The Lagrangian formulation of this problem is

$$\mathcal{L}(\boldsymbol{\theta}, \mu) = \; (\boldsymbol{y} - \tilde{\mathrm{X}}\boldsymbol{\theta})^{\mathrm{T}}(\boldsymbol{y} - \tilde{\mathrm{X}}\boldsymbol{\theta}) + \mu(\|\boldsymbol{\theta}\|_1 - c) \tag{51}$$

The solution to this is found using KKT conditions

1. Stationarity:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}|_{\boldsymbol{\theta}^*} = \frac{\partial}{\partial \boldsymbol{\theta}}((\boldsymbol{y} - \tilde{\mathrm{X}}\boldsymbol{\theta})^{\mathrm{T}}(\boldsymbol{y} - \tilde{\mathrm{X}}\boldsymbol{\theta}) + \mu(\|\boldsymbol{\theta}\|_1 - c))|_{\boldsymbol{\theta}^*} = \mathbf{0} \tag{52}$$

2. Primal Feasibility:

$$\|\boldsymbol{\theta}^*\|_1 - c \leq 0$$

3. Dual Feasibility:

$$\mu \geq 0$$

4. Complementary Slackness:

$$\mu(\|\boldsymbol{\theta}^*\|_1 - c) = 0$$

If μ rather than c is selected by the user, then by letting c always equal $\|\boldsymbol{\theta}^*\|_1$, we can see that with a change of variables, $\mu \equiv \alpha$, the solution to (50) is identical to the solution to (49). Thus, we can view lasso regression as a constrained optimization problem. This constraint on the 1-norm, or the sum of the absolute values of the components of $\boldsymbol{\theta}$, is included to potentially help avoid overfitting $\boldsymbol{\theta}$ to the training data, just like with ridge regression, and thereby helping the linear regression to generalize better to yet-unseen data. Lasso regression also can perform the task of feature selection by which a limited number of features are selected in the data from which to build a model. This can help save on time and memory requirements of the model. Lasso regression accomplishes this by reducing the coefficients of the lesser-significant features to identically zero.

To help see how lasso regression reduces the elements of $\boldsymbol{\theta}^*$, we can use a change of variable and add a second constraint to (50)

$$\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) \tag{53}$$

$$f(\boldsymbol{\theta}) = (\boldsymbol{y} - \widetilde{\mathrm{X}}\boldsymbol{\theta})^{\mathrm{T}}(\boldsymbol{y} - \widetilde{\mathrm{X}}\boldsymbol{\theta})$$

$$\text{subject to: } \|\boldsymbol{\beta}\|_1 \leq c$$

$$\boldsymbol{\beta} = \boldsymbol{\theta}$$

Now, the Lagrangian becomes (replacing μ with α)

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\beta}) = (\boldsymbol{y} - \widetilde{\mathrm{X}}\boldsymbol{\theta})^{\mathrm{T}}(\boldsymbol{y} - \widetilde{\mathrm{X}}\boldsymbol{\theta}) + \alpha(\|\boldsymbol{\beta}\|_1 - c) + \boldsymbol{u}^{\mathrm{T}}(\boldsymbol{\theta} - \boldsymbol{z}) \tag{54}$$

This can be rewritten as

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\beta}) = \mathcal{L}_1(\boldsymbol{\theta}) + \mathcal{L}_2(\boldsymbol{\beta}) \tag{55}$$

where

$$\mathcal{L}_1(\boldsymbol{\theta}) = (\boldsymbol{y} - \widetilde{\mathrm{X}}\boldsymbol{\theta})^{\mathrm{T}}(\boldsymbol{y} - \widetilde{\mathrm{X}}\boldsymbol{\theta}) - \alpha c + \boldsymbol{u}^{\mathrm{T}}\boldsymbol{\theta} \tag{56}$$

$$\mathcal{L}_2(\boldsymbol{\beta}) = \alpha\|\boldsymbol{\beta}\|_1 - \boldsymbol{u}^{\mathrm{T}}\boldsymbol{\beta} \tag{57}$$

Let the primal variables $\boldsymbol{\theta}$ and $\boldsymbol{\beta}$ be combined

$$\boldsymbol{\phi} = \begin{bmatrix} \boldsymbol{\theta} \\ \boldsymbol{\beta} \end{bmatrix}$$

The stationarity condition then requires

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\phi}}|_{\boldsymbol{\phi}^*} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} \\ \frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} \end{bmatrix}|_{\boldsymbol{\theta}^*,\boldsymbol{\beta}^*} = \begin{bmatrix} \frac{\partial \mathcal{L}_1(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \\ \frac{\partial \mathcal{L}_2(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} \end{bmatrix}|_{\boldsymbol{\theta}^*,\boldsymbol{\beta}^*} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix} \tag{58}$$

To solve the upper portion of Eq. (58), we can expand Eq. (56)

$$\mathcal{L}_1(\boldsymbol{\theta}) = \boldsymbol{y}^{\mathrm{T}}\boldsymbol{y} - 2\boldsymbol{y}^{\mathrm{T}}\widetilde{\mathrm{X}}\boldsymbol{\theta} + \boldsymbol{\theta}^{\mathrm{T}}\widetilde{\mathrm{X}}^{\mathrm{T}}\widetilde{\mathrm{X}}\boldsymbol{\theta} - \alpha c + \boldsymbol{u}^{\mathrm{T}}\boldsymbol{\theta} \tag{59}$$

And now differentiating Eq. (59) in accordance with Eq. (58)

$$\frac{\partial \mathcal{L}_1}{\partial \boldsymbol{\theta}}|_{\boldsymbol{\theta}^*} = -2\widetilde{\mathrm{X}}^{\mathrm{T}}\boldsymbol{y} + 2\widetilde{\mathrm{X}}^{\mathrm{T}}\widetilde{\mathrm{X}}\boldsymbol{\theta}^* + \boldsymbol{u} = \mathbf{0} \tag{60}$$

Solving this for $\boldsymbol{u}$ yields

$$\boldsymbol{u} = 2\widetilde{\mathrm{X}}^{\mathrm{T}}\boldsymbol{y} - 2\widetilde{\mathrm{X}}^{\mathrm{T}}\widetilde{\mathrm{X}}\boldsymbol{\theta}^* \tag{61}$$

or when solving for $\boldsymbol{\theta}^*$, produces

$$\boldsymbol{\theta}^* = (\widetilde{\mathrm{X}}^{\mathrm{T}}\widetilde{\mathrm{X}})^{-1}\widetilde{\mathrm{X}}^{\mathrm{T}}\boldsymbol{y} - \frac{1}{2}(\widetilde{\mathrm{X}}^{\mathrm{T}}\widetilde{\mathrm{X}})^{-1}\boldsymbol{u} \tag{62}$$

which, by using the results from Eqs. (11), (32), and (34) can be rewritten as

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}^{\mathrm{OLR}} - \frac{1}{2}(\widetilde{\mathrm{X}}^{\mathrm{T}}\widetilde{\mathrm{X}})^{-1}\boldsymbol{u} = \boldsymbol{\theta}^{\mathrm{OLR}} - \frac{1}{2}\mathrm{A}\boldsymbol{u} \tag{63}$$

where $\boldsymbol{\theta}^{\mathrm{OLR}}$ is the solution to the ordinary linear regression problem. Thus, we can see that the solution to the lasso regression problem is equal to the linear regression solution minus $\frac{1}{2}\mathrm{A}\boldsymbol{u}$.

Now looking at the lower portion of Eq. (58), we can observe that

$$\frac{\partial \mathcal{L}_2}{\partial \boldsymbol{\beta}}|_{\boldsymbol{\beta}^*} = \begin{bmatrix} \frac{\partial \mathcal{L}_2(\boldsymbol{\beta})}{\partial \beta_0} \\ \vdots \\ \frac{\partial \mathcal{L}_2(\boldsymbol{\beta})}{\partial \beta_m} \end{bmatrix}|_{\boldsymbol{\beta}^*} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \tag{64}$$

Additionally, Eq. (57) can be rewritten as

$$\mathcal{L}_2(\boldsymbol{\beta}) = \alpha\|\boldsymbol{\beta}\|_1 - \boldsymbol{u}^{\mathrm{T}}\boldsymbol{\beta}$$

$$\mathcal{L}_2(\boldsymbol{\beta}) = \alpha\sum_{i=0}^{m}|\beta_i| - \sum_{i=0}^{m}u_i\beta_i \tag{65}$$

$$\mathcal{L}_2(\boldsymbol{\beta}) = \sum_{i=0}^{m}\alpha|\beta_i| - u_i\beta_i \tag{66}$$

$$\mathcal{L}_2(\boldsymbol{\beta}) = \sum_{i=0}^{m}(\alpha - u_i sgn(\beta_i))|\beta_i| \tag{67}$$

The gradient of $\mathcal{L}_2(\boldsymbol{\beta})$ with respect to $\boldsymbol{\beta}$ is equal to the zero vector (as required at a solution point by Eq.
(64)) when $\mathcal{L}_2(\boldsymbol{\beta})$ is at either a minimum, maximum, or saddle point. By searching for a minimum to $\mathcal{L}_2(\boldsymbol{\beta})$ over $\boldsymbol{\beta}$, we can quickly observe from Eq.
(67)

$$\min_{\boldsymbol{\beta}}\mathcal{L}_2(\boldsymbol{\beta}) = \begin{cases} 0, & if\ \forall i, |u_i| \le \alpha \\ -\infty, & otherwise \end{cases} \tag{68}$$

because if for any *i,* $|u_i| > \alpha$, then $\mathcal{L}_2(\boldsymbol{\beta})$ could be minimized further over $\boldsymbol{\beta}$ by simply continuously increasing (or decreasing) β_i.

Thus, for the problem to be feasible, the following constraint on the dual variable, $\boldsymbol{u}$, must be observed

$$\|\boldsymbol{u}\|_\infty \le \alpha \tag{69}$$

Returning to Eq.
(64) and using the expression for $\mathcal{L}_2$ from Eq.
(66), we observe that for a particular element of the gradient, we have

$$\frac{\partial\mathcal{L}_2(\boldsymbol{\beta})}{\partial\beta_j} = \frac{\partial}{\partial\beta_j}\left(\sum_{i=0}^{m}\alpha|\beta_i| - u_i\beta_i\right) = \frac{\partial}{\partial\beta_j}\left(\alpha|\beta_j| - u_j\beta_j\right) \tag{70}$$

The partial derivative of the right-hand side of Eq. (70) is not continuous at 0, but by letting ζ be a subgradient of the function at 0, we can say

$$\frac{\partial \mathcal{L}_2(\widehat{\boldsymbol{\beta}})}{\partial \beta_j} = \begin{cases} \alpha - u_j, & if\ \beta_j > 0 \\ -\alpha - u_j, & if\ \beta_j < 0 \\ \alpha\zeta - u_j, \quad where\ \zeta \in [-1, 1], & if\ \beta_j = 0 \end{cases} \tag{71}$$

Thus for Eq.
(64) to be satisfied, we can say that $if\ \beta_j > 0\ then\ u_j = \alpha$; $if\ \beta_j < 0\ then\ u_j = -\alpha$; and if $|u_j| \neq \alpha$ then $\beta_j = 0$.
Combining the above equation, observing the equality relationship between $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$, and noting that according to Eq. (63), the solution to the lasso regression problem occurs at

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}^{\mathrm{OLR}} - \frac{1}{2}\mathrm{A}\boldsymbol{u}$$

We see that $\boldsymbol{\theta}^*$ cannot be solved for analytically, but is defined in terms of itself (through the equations governing $\boldsymbol{u}$). However, we can envision iterating to a solution.

Let's consider $\lim_{i\to\infty} \widehat{\boldsymbol{\theta}}^{(i)} = \boldsymbol{\theta}^*$ and assume an initial guess

$$\widehat{\boldsymbol{\theta}}^{(0)} = \boldsymbol{\theta}^{\mathrm{OLR}}$$

Let's also assume all elements of $\boldsymbol{\theta}^{\mathrm{OLR}}$ are all positive (although similar arguments hold regardless of the sign of these elements, as long as they are non-zero). This would imply that all elements of $\boldsymbol{\beta}$ are positive and therefore,

$$\boldsymbol{u} = \begin{bmatrix} \alpha \\ \vdots \\ \alpha \end{bmatrix}$$

From observations while analyzing ridge regression, we recall that A is positive-definite, which implies that

$$\boldsymbol{s}^{\mathrm{T}} A \boldsymbol{s} > 0, \qquad \forall \boldsymbol{s} \neq \boldsymbol{0}$$

This relationship implies further that

$$sgn(s_i) = \mathrm{sgn}((\mathrm{A}\boldsymbol{s})_i)\ \forall i$$

Therefore, all elements of $\frac{1}{2}\mathrm{A}\boldsymbol{u}$ will be positive and we can observe that our next guess,

$$\widehat{\boldsymbol{\theta}}^{(1)} = \boldsymbol{\theta}^{\mathrm{OLR}} - \frac{1}{2}\mathrm{A}\boldsymbol{u}_{\widehat{\boldsymbol{\theta}}^{(0)}}$$

where $\boldsymbol{u}_{\widehat{\boldsymbol{\theta}}^{(0)}}$ indicates $\boldsymbol{u}$ evaluated at $\widehat{\boldsymbol{\theta}}^{(0)}$, will have pulled all elements of $\widehat{\boldsymbol{\theta}}^{(0)} = \boldsymbol{\theta}^{\mathrm{OLR}}$ closer to zero. Now observe that if an element of $\widehat{\boldsymbol{\theta}}^{(1)}$ gets pulled so far that it switches sign from $\widehat{\boldsymbol{\theta}}^{(0)}$, then upon the next iteration the corresponding element of $\boldsymbol{u}$ will now be equal to $-\alpha$, which will cause that element of $\widehat{\boldsymbol{\theta}}^{(1)}$ to be pushed again towards zero, this time from the opposite direction. This would happen continuously until that element settles upon zero. Furthermore, we can see that

larger values of α will cause larger shifts from $\boldsymbol{\theta}^{\mathrm{OLR}}$. Thus, we can say that lasso regression drives an increasing number of elements of $\boldsymbol{\theta}^*$ to zero, given increasing values of α.

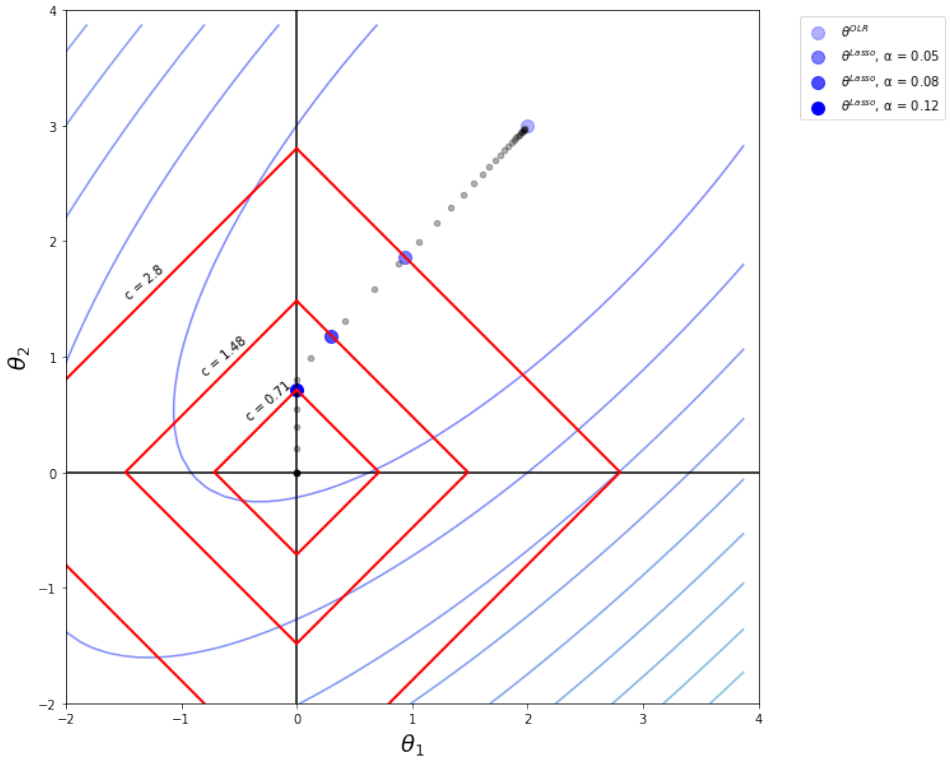


Fig. 16. Lasso regression as constrained optimization.
Recall from Fig. 10 where the path of optimal $\boldsymbol{\theta}^*$ was shown as α increased for ridge regression. Figure 16 contains the contours of the same objective function, Eq. (7) (identical to function $f(\boldsymbol{\theta})$ in Eq. (53)), for the same two-dimensional example dataset. Gray points indicate the trajectory of lasso regression solutions as α is increased from zero to a value great enough such that $\boldsymbol{\theta}^*$ converges to the origin. Selected constraints, c, on the 1-norm of $\boldsymbol{\theta}^*$ are shown in red, along with their corresponding values of α in the legend.

Note that in this example, the lasso regression solution is pulled linearly until it hits the θ_2 axis. From that point on, θ_1 remains at zero as the solution approaches and eventually reaches the origin.

The workflow outlined in the ridge regression section was again followed to determine the optimal value of α for a lasso regression model on all three weather datasets.

The plots of validation set RMSE vs alpha parameters are shown in Figure 17.

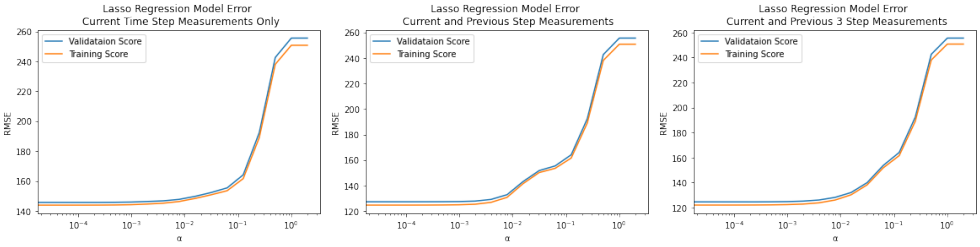


Fig. 17. Lasso regression model performance as α is adjusted.
Validation and training score errors (RMSE) vs α for lasso regression models based on all three datasets. In all cases, the best model appears to be one with $\alpha = 0$, which is again just linear regression.

As can be seen, for all of these datasets, the model again performed best as a simple linear model without regularization. This, as it was for ridge regression, is likely due to the large number of examples in the dataset. To see how α affected the values of various coefficients, we can plot the coefficient value vs α for a few selected coefficients. The plots in Figures 18 and 19 were constructed using the dataset consisting of only current measurements.

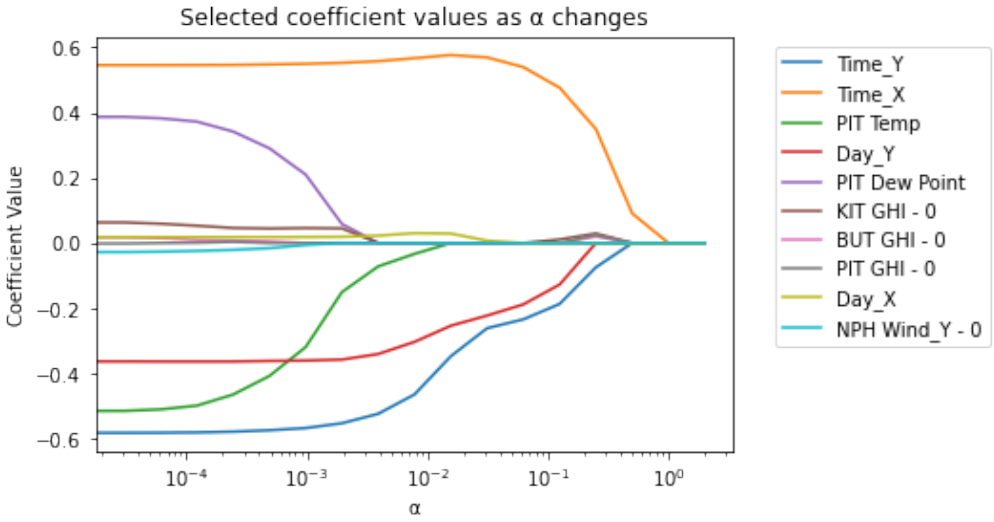


Fig. 18. Unlike for ridge regression, the coefficients in a lasso regression model do not approach zero asymptotically as α is increased, but become identically zero in a piece-wise linear fashion. This generates a sparse solution which can be seen by plotting the number of non-zero elements in the optimal $\boldsymbol{\theta}$ vs α seen in Figure 19.

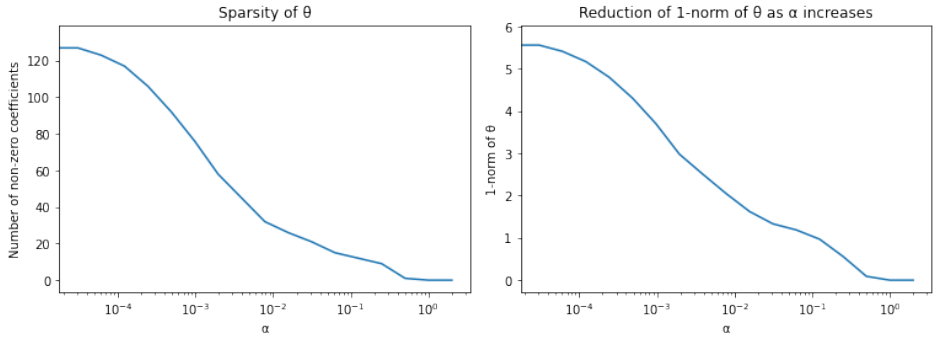


Fig. 19. The optimal solution to the lasso regression problem becomes more and more sparse as its 1-norm is reduced. This is a primary difference between ridge and lasso regression. In ridge regression, no components of the solution become and remain zero.

By reducing coefficients of various components to zero, lasso regression is often used to perform feature selection. This is a process by which less-impactful features are dropped from the dataset either due to time, computing power, and or memory constraints. Additionally, this process can inform the researcher as to the *m* most impactful features if desired.

For each dataset, we can observe the five highest-magnitude coefficients for various settings of α, including pure linear regression ($\alpha = 0$) in Table VII

Table VII
High-Magnitude Coefficient Values in Various Linear and Lasso Regression Models

Current-only measurements					
Linear Model Highest-magnitude coefficients		Lasso regression with $\alpha = 2^{-8}$ Highest-magnitude coefficients[1]		Lasso regression with $\alpha = 2^{-2}$ Highest-magnitude coefficients[2]	
Time_Y	-0.583	Time_X	0.559	Time_X	0.351
Time_X	0.547	Time_Y	-0.526	Time_Y	-0.080
PIT Temperature – 0	-0.523	Day_Y	-0.339	YGT DNI - 0	0.030
PIT Dew Point – 0	0.400	PIT Clearsky DNI – 0	-0.251	KIT GHI - 0	0.028
Day_Y	-0.363	CBG Clearsky DHI -0	-0.088	BUT GHI – 0	0.025

Current and previous measurements					
Linear Model Highest-magnitude coefficients		Lasso regression with $\alpha = 2^{-8}$ Highest-magnitude coefficients[3]		Lasso regression with $\alpha = 2^{-2}$ Highest-magnitude coefficients[2]	
Time_Y	-0.602	Time_Y	-0.539	Time_X	0.351
PIT Clearsky DNI - 0	-0.367	Time_X	0.403	Time_Y	-0.080
Day_Y	-0.355	Day_Y	-0.319	YGT DNI - 0	0.030
Time_X	0.350	PIT Clearsky DNI – 0	-0.313	KIT GHI - 0	0.028
YGT GHI - 1	-0.199	YGT GHI - 1	-0.119	BUT GHI – 0	0.025

Current and previous three measurements					
Linear Model Highest-magnitude coefficients		Lasso regression with $\alpha = 2^{-8}$ Highest-magnitude coefficients[4]		Lasso regression with $\alpha = 2^{-2}$ Highest-magnitude coefficients[2]	
Time_Y	-0.663	Time_Y	-0.620	Time_X	0.351
PIT Clearsky DNI - 0	-0.406	PIT Clearsky DNI – 0	-0.398	Time_Y	-0.080
Day_Y	-0.375	Day_Y	-0.349	YGT DNI - 0	0.030
Time_X	0.297	Time_X	0.323	KIT GHI - 0	0.028
PIT Clearsky GHI - 0	-0.186	NPH GHI - 3	-0.080	BUT GHI – 0	0.025

1. Has 44 non-zero coefficients
2. Has 9 non-zero coefficients
3. Has 128 non-zero coefficients
4. Has 125 non-zero coefficients

Interestingly, for all three datasets, a value of $\alpha = 2^{-2}$ created an identical model, with only 9 non-zero coefficients.

As with ridge regression, one can wonder why we went through the trouble developing the model if linear regression seems to have better performance. This is again due to the large amount data used. If we were to only have access to a subset of this data, it is likely non-zero α lasso regression models could generalize better than linear regression.

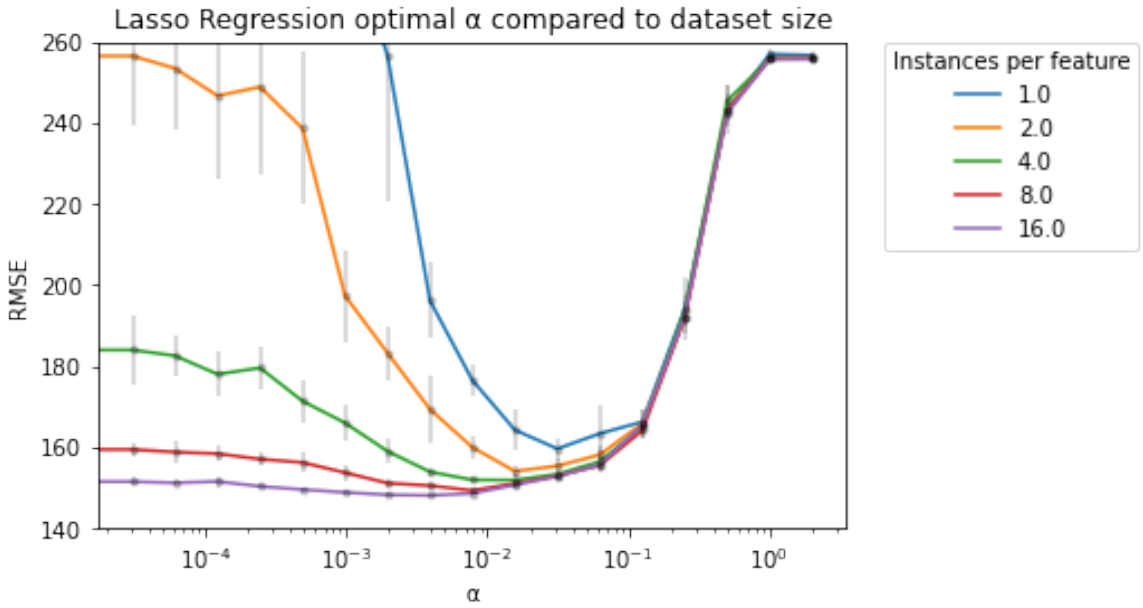


Fig. 20. Comparison of the benefits of lasso regression for datasets of varying size
Each dataset size is represented in terms of number of instances per number of features. Gray error bars represent the standard deviation of validation RMSE scores for random sub-sampling of the training set data used to fit the model.
For this data, training sets with a low number of instances have a fairly well-defined optimal value of α for which the model generalized best. But as the training set becomes larger, the optimal α becomes less well defined as low-α models perform better and better until linear regression becomes the best-generalizing model. For all values of α, however, a model trained with a larger dataset generalizes better than or equal to a smaller dataset. This demonstrates the usefulness of lasso regression, but the value of additional data.

Additionally, note the comparison of this figure to its companion for ridge regression, Figure 15. It can be seen that for smaller datasets (1-4 instances per feature) lasso regression outperforms ridge regression.

Decision Tree

A decision tree is an algorithm which, through a series of sequential binary decisions can be used to perform regression. These individual decision points are called *nodes*. An initial node is selected to be the root of the tree, from which two branches emerge. The decision on which branch to take is based on whether, for a single data instance, a particular feature, k_i, has a value less than or equal to a selected threshold value, t_{k_i}. If this statement is true, the branch to the left is taken. If false, the branch on the right is taken. Both branches can lead to additional binary decision nodes where similar evaluations are performed. This process repeats until a *leaf* is reached. A leaf is the end of a sequence of branches, where an ultimate prediction is made. All paths terminate eventually with a leaf. Once the decision tree is formed, it is quite simple and quick to evaluate new instances as no computations are involved.

An example of a decision tree is shown in Figure 21.

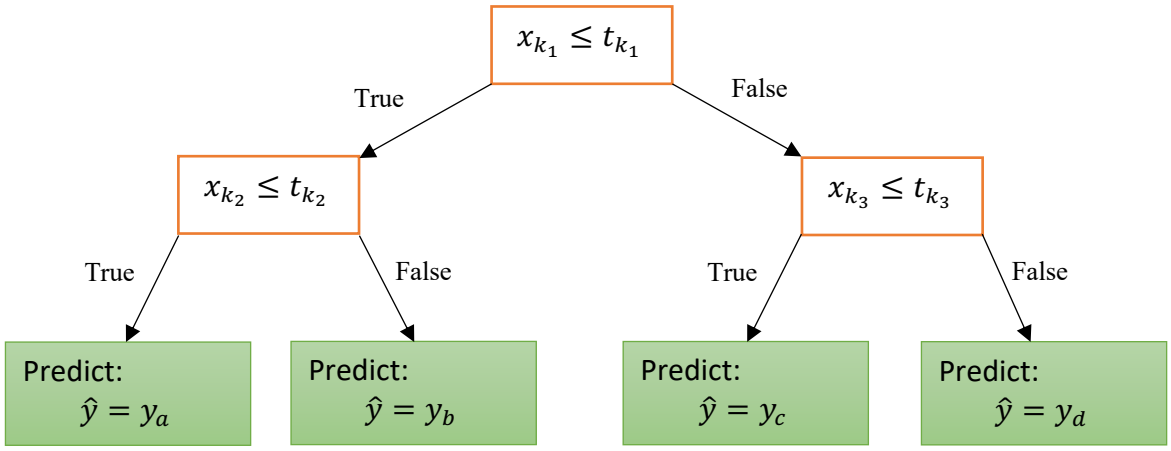


Fig. 21. Decision Tree Example
At the root of the decision tree is the initial node (nodes indicated with orange-outlined boxes), where all instances must be evaluated. In this case, the value of feature k_1 of a data instance, which is x_{k_1}, is compared to the scaler t_{k_1}. If x_{k_1} is less than or equal to t_{k_1}, then the branch on the left is taken. Otherwise, the branch on the right is taken. This sequence traveling along branches to nodes is continued until a leaf (indicated by green boxes) is reached. All data instances that arrive at a particular leaf are assigned to the same prediction value, y_a in the case of the leftmost leaf above. The same feature can be used for evaluation at multiple nodes (k_1 *and* k_2 and so on are not necessarily distinct), but the same feature will not be compared to the same threshold value twice on any individual pathway, since this comparison would not yield any new information about the data instances to reach it.

As can be seen in Figure 21, all decisions are made on the basis of a split along an axis of the data (the comparison is never made on a linear combination of features). This means that there is no

need to scale features to all have similar means and variances. Additionally, this fact gives the decision tree one of its major attributes: interpretability. Decision tree models are considered to be "white box" models, meaning observation of the model structure and its learned parameters can provide insight into why certain predictions are made by the model. Consider a dataset containing patient medical information such as age, height, weight, blood pressure, cholesterol levels, etc. If the regression target is patient annual health care expenses, then a decision tree can be a fairly easy model to understand. It would read something like: "*if* the patient is over *x* years old *and* weighs less than *y* pounds, *and* has cholesterol below *z*, *then* their predicted health care expenses will be *d* dollars. Of course, as the decision tree gains more and more layers of these comparisons, the potential pathways increase exponentially, and the interpretability of the model decreases accordingly. This brings us to an important hyperparameter to consider when creating decision trees: maximum depth. The maximum depth is the greatest number of possible decisions the model would be allowed to perform to make a prediction for a data instance. The tree in Figure 21 has a depth of two. When fitting a decision tree to a dataset, without any restrictions on the maximum depth hyperparameter, the tree will keep increasing its depth until all training data is perfectly classified, meaning the training set could be evaluated by the tree with zero error (that is, unless two or more instances in the data have identical feature values, but different target values). However, this strategy is likely to severely overfit the model since the decision tree would essentially memorize the training set, causing it to generalize poorly to new data. Therefore, hyperparameters such as those restricting the maximum depth of the model are utilized. Other hyperparameters are available to help avoid overfitting as well, such as requiring a minimum number of training instances for a new leaf to be formed.

Decision trees are very easy to evaluate once built, but one of their drawbacks is their difficulty to construct. Decision trees are generally considered to be greedy algorithms, meaning that they are not globally optimized, but optimized at each step for the most immediate improvement in objective function. When performing regression, the objective function of each node is generally the Classification and Regression Tree (CART) function. This function searches for the pair of k_i and t_{k_i}, where t_{k_i} is the threshold value at which feature k_i is split, that minimizes the CART cost function [6]

$$J(k_i, t_{k_i}) = \frac{N_{left}}{N} SSE_{left} + \frac{N_{right}}{N} SSE_{right} \quad (72)$$

with N being the total number of training data instances to arrive at the given node, N_{left} and N_{right} being the number sorted to the nodes connected to the left branch and right branch, respectively, and

$$SSE_{node} = \sum_{i \in node} (\hat{y}_{node} - y^{(i)})^2 \quad (73)$$

where the sum of squared errors (SSE) is performed over all data instances to be sorted to the left or right node, and

$$\hat{y}_{node} = \frac{1}{N_{node}} \sum_{i \in node} y^{(i)} \quad (74)$$

If a branch terminates in a leaf rather than a node, then the predicted output for any data instance reaching that leaf will be $\hat{y}_{node}$, where $\hat{y}_{node}$ is the average output of all training instances sorted into that leaf.

Consider a node *i,* which could be a leaf or a node leading to further branches. Let $D^{(i)}$ represent the set of all training data that reach node *i*. Minimizing the CART function at this node is equivalent to selecting the feature k_i that can best predict *y* among all $D^{(i)}$ when $D^{(i)}$ can only be split into two groups. All data instances falling into a particular group will be assigned the same prediction value, $\hat{y}_{node}$. The threshold t_{k_i} is the value of feature k_i at which the two groups are partitioned. It should be noted that the same feature can be used as the partitioning feature multiple times in a decision tree. With no analytic solution to find the optimal k_i and t_{k_i} at each node, this algorithm relies on a significant amount of iterating through possible feature and threshold values at each node. For this reason, decision trees can be quite time consuming to train. Often in implementation, only a random subset of all features is searched over at each node.

Returning to the two-dimensional example of the previous sections, we can observe how decision trees of varying depths would subdivide the space and make predictions. The underlying function which was sampled is

$$y = 2x_1 + 3x_2$$

The true function outputs can be visualized for the input space forming a square around the origin and extending from -2 to 2 in each coordinate in Figure 22.

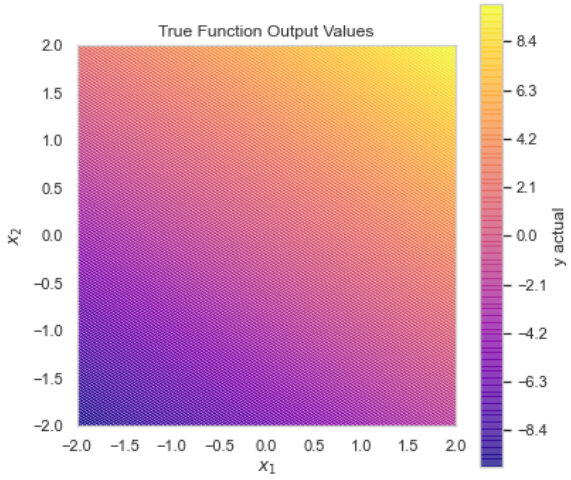


Fig. 22. Plot displaying the output value (*y*) vs. a two-dimensional input (x_1 and x_2).

When a decision tree is fit to this data but has its maximum depth limited, the decision boundaries and predictions of the output can be seen in Figure 23.

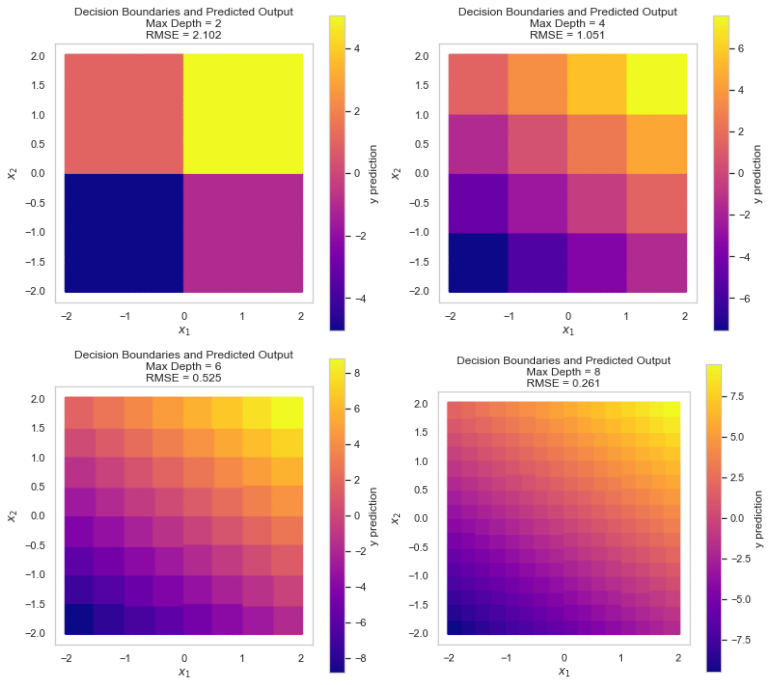


Fig. 23. Shallow decision trees (top row) are generally poor approximators for an underlying function (shown in Figure 22). Decision boundaries can be seen to always be orthogonal to the axes. Note the RMSE values listed in the title of each subplot.

The decision boundaries are always orthogonal to the component axes, which can yield a poor approximation of the underlying function, especially for shallow decision trees. To alleviate this, the input data could be preprocessed in such a way as to better align the function gradient with the axes. For our two-dimensional example, this could be a simple rotation around the origin. Applying a decision tree to this rotated data can smooth decision boundaries and not require as deep of a tree to achieve a high-accuracy model, as shown by the plots in Figure 24. While preprocessing the input data can increase model accuracy, especially for shallow decision trees, much of the interpretability of the model is lost. The decision trees developed in the top right subplot of Figure 23 could have an example be interpreted as: "when $1 < x_1 \leq 2$ and $1 < x_2 \leq 2$, the output is predicted to be 7." Given the units of measurement and context, this could likely be well-understood by someone with domain knowledge. Compare this to the top right plot in Figure 24, which with the change in principle axes would read as: "when $2 < w_1 \leq 3$ and w_2 is any value, the output is predicted to be 8, where $w_1 = \frac{2}{\sqrt{13}}x_1 + \frac{3}{\sqrt{13}}x_2$ and $w_2 = \frac{-3}{\sqrt{13}}x_1 + \frac{2}{\sqrt{13}}x_2$. Clearly, this change in components has decreased the interpretability of the model by the researcher.

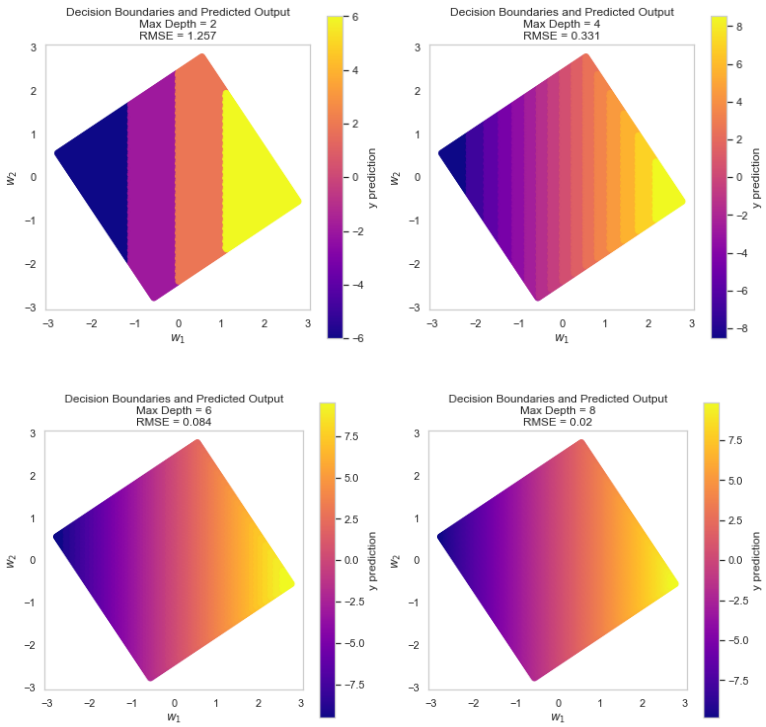


Fig. 24. When input data is rotated prior to constructing the decision tree, the tree can be much more efficiently fit to the data. In this case the data was rotated clockwise by $tan^{-1}(\frac{3}{2}) \approx 35.8°$. Observe the much lower RMSE values for these models compared to those of similar depth in Figure 22. While this process improved the accuracy of low-depth decision trees, the interpretability of the model is largely lost, which is generally a major advantage of decision trees.

As previously stated, in the absence of any restrictions, a decision tree will perfectly fit the training data. Therefore, care must be taken to avoid creating a model which is overfit and does not generalize well. To accomplish this, a similar workflow to that laid out in the ridge regression section ought to be followed. A range of hyperparameters will be selected, and the optimal hyperparameters will be chosen as those which yield the best performance on the validation set. The hyperparameter we will use to control for generalization is the maximum depth allowed. Utilizing this parameter is equivalent to stopping the model early, when it could continue to better fit the training data. Halting the training of a model early is a technique known as early-stopping, and can be utilized with many machine learning models. Figure 25 shows how a decision tree fit to the weather training data performs on both the training and validation sets as the maximum depth parameter is increased. It can be seen that the training error can be decreased by continually increasing depth. However, at some point this model would cease to generalize well to unseen data. All it will have done is memorize the training data. This demonstrates the importance of utilizing a validation set to select the hyperparameters that lead to the best generalization of the model.

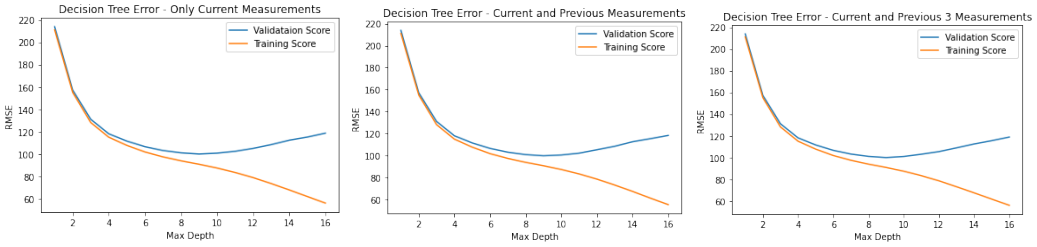


Fig. 25. Decision tree model performance as the maximum depth hyperparameter is adjusted
Validation and training score (RMSE) vs maximum depth for decision tree models based on all three datasets. It can be seen how training performance continues to improve as the depth of the model is permitted to increase (orange line). However, the generalization of the model ceases to improve beyond a depth of about nine (blue line) for all three datasets. This demonstrates the importance of utilizing a validation set when tuning hyperparameters; otherwise, the researcher would be misled into believing the best model is one with no restriction on the depth of the decision tree.

In Figure 25, all three charts appear identical and have an optimal depth of nine, corresponding to a validation error score of about 110. This would indicate that adding the extra previous measurements does not improve the performance of a decision tree in forecasting the GHI in Pittsburgh four hours ahead. Decision trees with maximum depth of nine were fit to the combined training and validation set for the three datasets. All three models were then evaluated on their respective testing sets.

Table VIII records the RMSE values for each model performance on the testing set, and Figures 26, 27, and 28 display a visualization of the first few layers of each tree. Since no standardization of the input data is required, all threshold values can be seen in their original units, which is helpful for model interpretation.

Table VIII
Evaluation of Decision Trees on Testing Sets

Linear model fit with:	Current measurements only	Current and previous measurements	Current and previous three measurements
RMSE	98.07	98.13	98.27

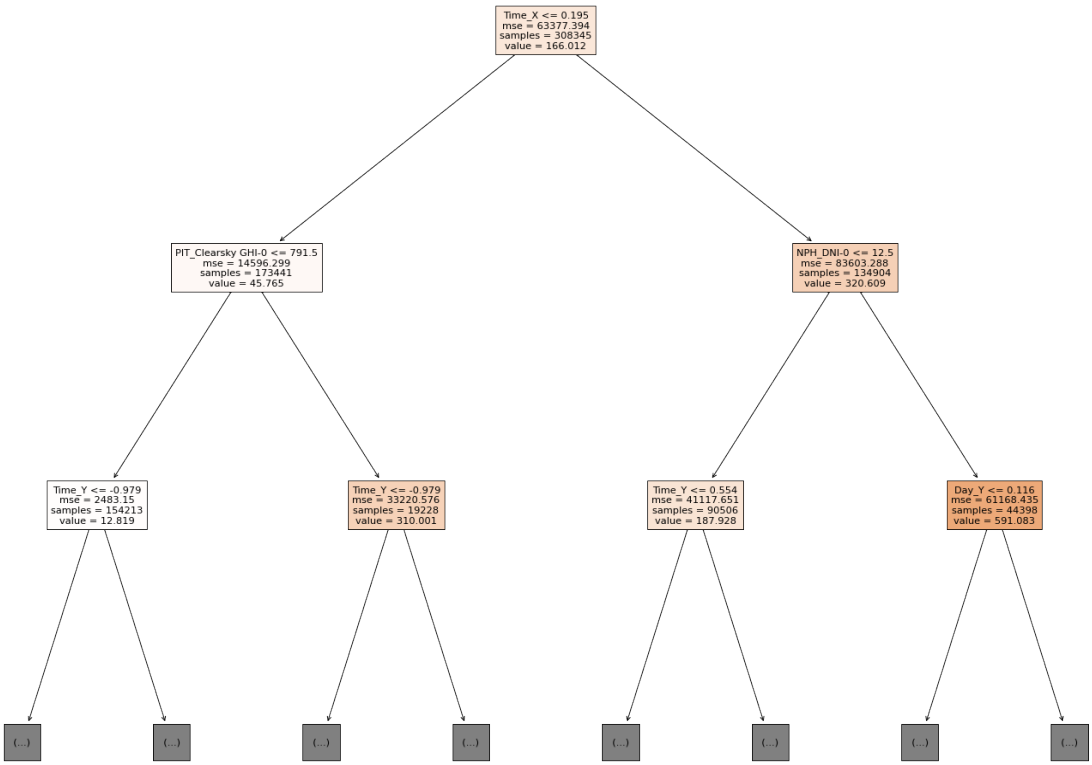


Fig. 26. Visualization of the first few layers of the decision tree with maximum depth of nine built on the combined training and validation set of the current-measurements-only dataset.

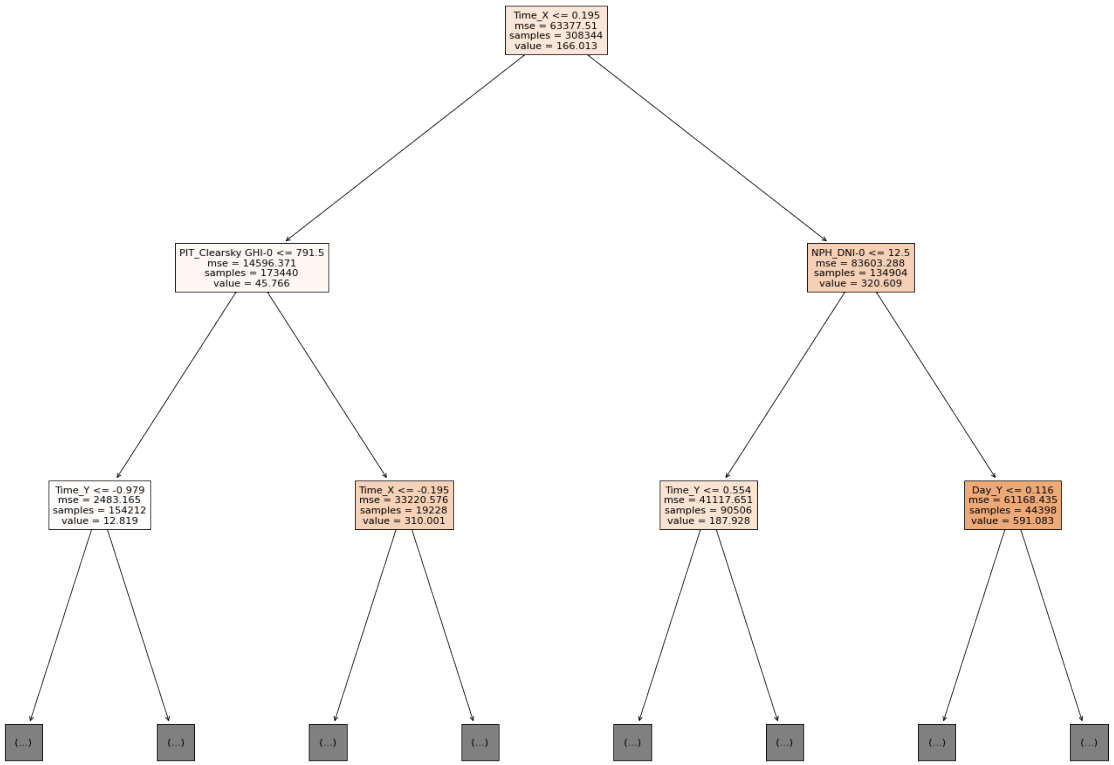


Fig. 27. Visualization of the first few layers of the decision tree with maximum depth of nine built on the combined training and validation set of the current-and-previous-measurements dataset.

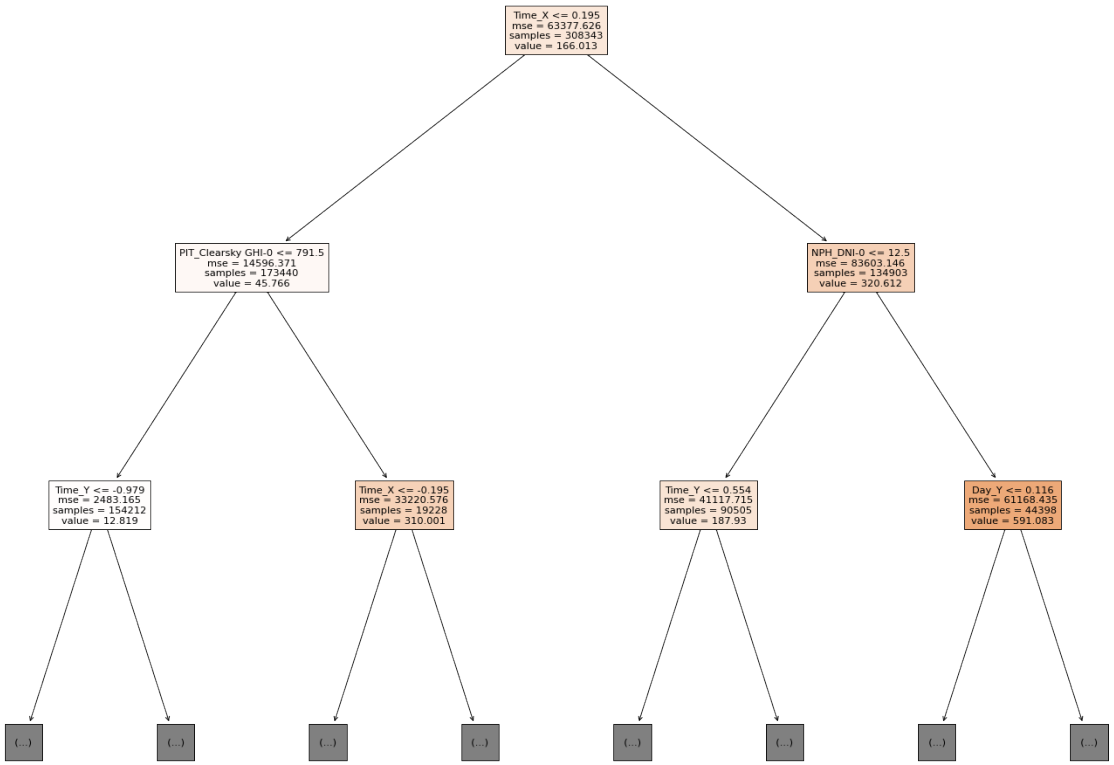


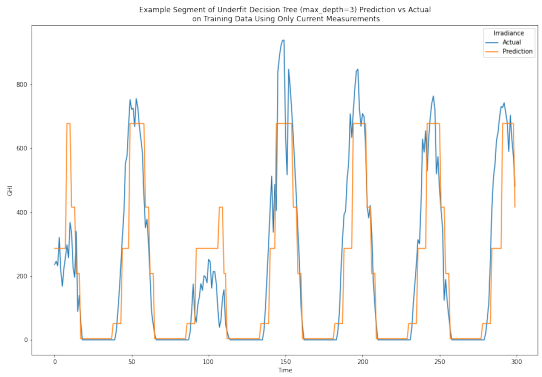
Fig. 28. Visualization of the first few layers of the decision tree with maximum depth of nine built on the combined training and validation set of the current-and-previous-three-measurements dataset.

As can be seen, each of these three models appear almost identical in both how they scored when evaluated on the testing set, and in how their first few layers are structured. Upon closer inspection, there is only one different feature that was used for a split in the first few layers, and that was Time_Y for the current-measurements-only dataset, while Time_X was used in its place for the other two datasets. It should be noted that most implementations of the decision tree algorithm utilize some level of randomness in selecting optimal splitting features, so if two features are nearly identical in performance when used as a splitting feature, the algorithm may choose randomly between the two.

To observe the differences between underfit, optimal, and overfit models on both the training data and validation data, Figure 29 contains comparisons for the current-measurements-only dataset on sample portions of the training and validation sets. The model fits the training set better and better as model depth is increased, but at some point (as shown in Figure 25), the model performance on the validation set starts to worsen.

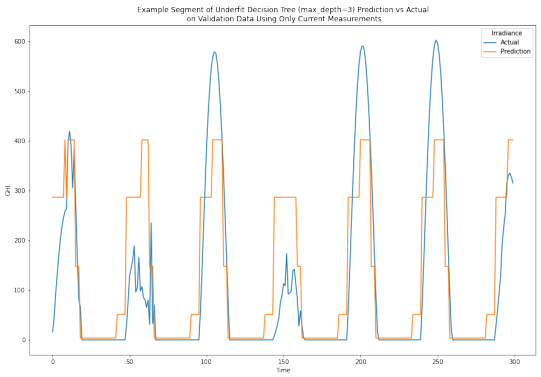
Underfit Model (maximum depth = 3)

Training set



RMSE = 128.14

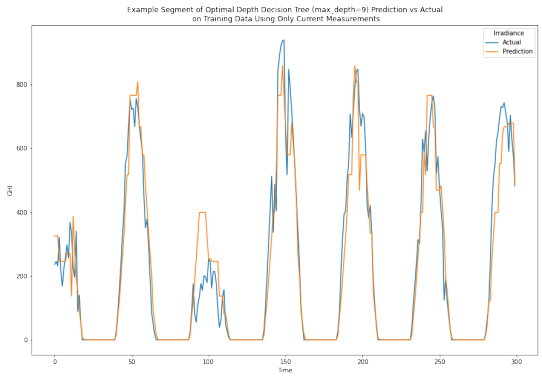
Validation set



RMSE = 131.05

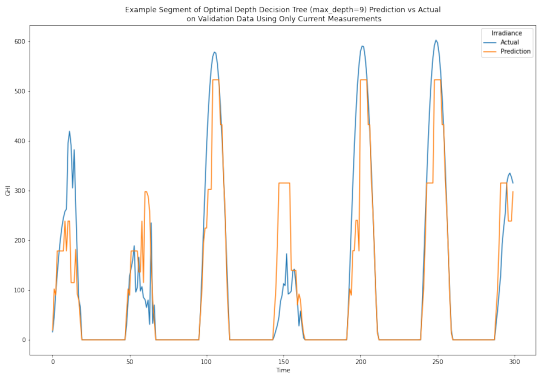
Optimal Model (maximum depth = 9)

Training set



RMSE = 90.77

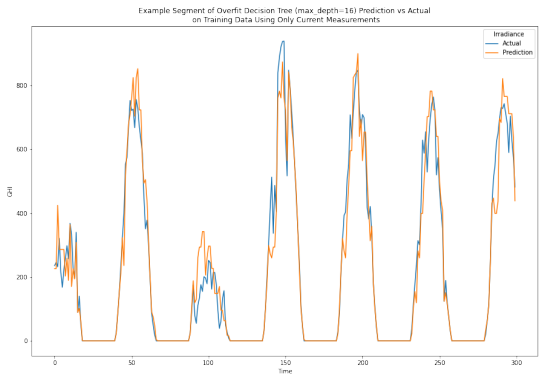
Validation set



RMSE = 99.95

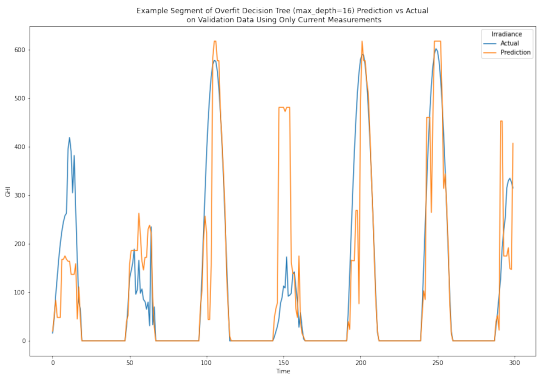
Overfit Model (maximum depth = 16)

Training set



RMSE = 55.98

Validation set



RMSE = 118.62

Fig. 29. Visualization of the impact of underfitting, optimal fitting, and overfitting to the training set. Training set performance continues to improve as model depth is increased, while there appears to be an optimal depth for the validation set (data not used to fit the model) after which prediction errors increase.

Figure 29 demonstrates why it is important to monitor the model error on the validation set while adjusting hyperparameters, before selecting optimal hyperparameters and finally fitting the model to the combined training and validation set. Failure to monitor this error will likely lead to large discrepancies between model performance during training versus in deployment.

Feed-Forward Neural Network

A feed-forward neural network, or a multi-layer perceptron, is a network of nodes, or "neurons," connected by weights, or "synapses," and activated by an activation function. Generally, neural networks have three types of layers: input, output, and hidden. While there are only one input and output layer, there can be any number of hidden layers. In the following section, we will only consider a network with a single hidden layer.

The input layer is represented by a series of nodes, one for each feature of a data instance. These nodes are connected via a matrix of synapses to the hidden layer. The hidden nodes are then connected through further synapses to the output node. Figure 30 has a visualization of the structure of this network. This type of neural network is called "feed-forward" because information only travels in one direction; there is no manner of feedback via looped or reversed synapse connections in the network.

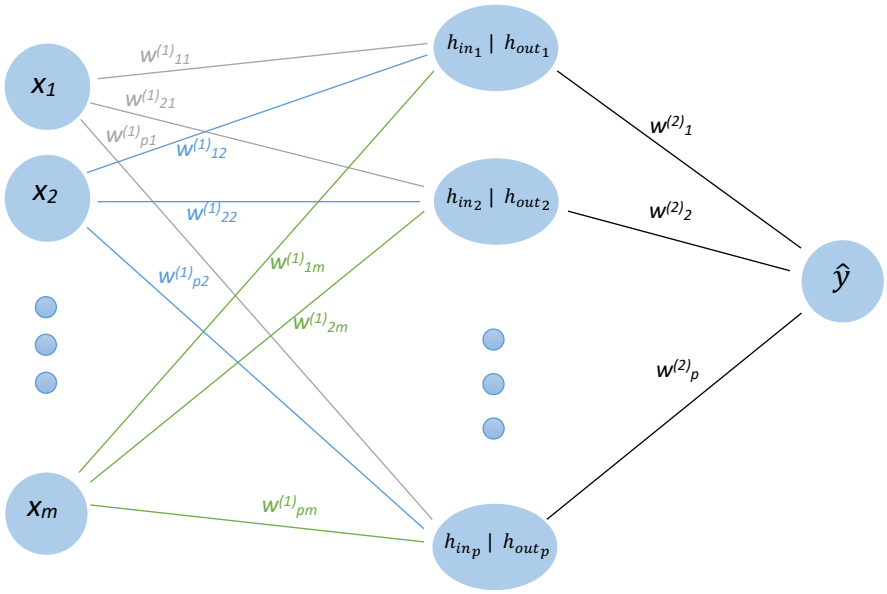


Fig. 30. Visualization of a feed-forward neural network
Neural networks can be visualized as a series of layers of nodes (or "neurons"), connected by weights (or "synapses"). The input layer can be seen on the left where each node represents the scaled value of a feature for a particular instance. This input layer is "fully-connected" to the single hidden layer via weights $w^{(1)}{}_{ij}$, meaning each input node has a connection to each hidden node in the next layer. Each hidden layer node has an input, which is determined by the sum of products of the input nodes and weights, and an output, which is the result of the activation function applied to the input of the node. Finally, the output layer, the single neuron on the right, is the result of a summation of each hidden layer node multiplied by a weight, $w^{(2)}{}_i$.

Mathematically, the equations that create the neural network are as follows

$$\hat{y} = w_1^{(2)} h_{out_1} + w_2^{(2)} h_{out_2} + \cdots + w_p^{(2)} h_{out_p} + b^{(2)} = \boldsymbol{w}^{(2)^{\mathrm{T}}} \boldsymbol{h}_{out} + b^{(2)} \quad (75)$$

$$\boldsymbol{h}_{out} = \sigma(\boldsymbol{h}_{in}) \quad (76)$$

$$\boldsymbol{h}_{in} = W^{(1)} \boldsymbol{x} + \boldsymbol{b}^{(1)} \quad (77)$$

where σ is referred to as the activation function. A common function to use for this is the rectified linear unit (ReLU)

$$\sigma(z) = ReLU(z) = max(0, z) \quad (78)$$

The activation function is applied component-wise to its input, so $\boldsymbol{h}_{out}$ has the same dimension as $\boldsymbol{h}_{in}$. For example,

$$h_{out_1} = \sigma(h_{in_1}) = \sigma(w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + \cdots + w_{1m}^{(1)} x_m + b_1^{(1)}) \quad (79)$$

The activation function is the secret behind the neural network. Without it, the entire network reduces to a simple linear transformation.

The primary goal of training a neural network is to determine the optimal parameters $W^{(1)}$, $\boldsymbol{b}^{(1)}$, $\boldsymbol{w}^{(2)}$, and $b^{(2)}$. Collectively, we will call these parameters $\boldsymbol{\theta}$. The determination of "good" parameters is based on the evaluation of a cost function. For a regression problem, this cost function is generally the mean squared error of the predicted vs actual outputs. In order to find the most optimal set of parameters, the cost function must be minimized over $\boldsymbol{\theta}$. Since the parameters of $\boldsymbol{\theta}$ exist in each layer of the network, the chain rule of calculus is employed to back-propagate the gradient of the cost function to the parameters in the earlier layers. Once this gradient is calculated, each parameter is updated by the gradient multiplied by a learning rate. There are several ways in which this process can take place, however. One option is to send a single training example through the network (with randomly initialized parameters), calculate the cost function and its gradient, apply the update to all parameters, then repeat for the second training example, and so on. This is known as on-line gradient descent. The problem with on-line gradient descent is that sending only one example through the network at a time may cause the parameters to move rather wildly and not in a smooth or efficient manner. Another option is to send the entire training data though the network, saving all calculated variable values, then perform a single batch gradient descent step at the end. This will yield a much smoother parameter update, but is a rather slow and memory-intensive method. A common middle ground is found by using a mini-batch method. This is where mini-batches of approximately 10-100 examples are sent though the network before updating the parameters. This is generally the preferred method as a trade-off between efficiency, speed, and memory requirements.

For the neural network defined in Equations (75)-(79), the gradients of all parameters in $\boldsymbol{\theta}$ can be calculated for a single example as shown below [7].

Define the cost function to be

$$J(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}) = (\hat{y}^{(i)} - y^{(i)})^2 \quad (80)$$

To simplify this, we can let

$$\delta^{(i)} = \hat{y}^{(i)} - y^{(i)} \quad (81)$$

then

$$J(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}) = (\delta^{(i)})^2 \quad (82)$$

First, note that the gradient of J with respect to $\hat{y}^{(i)}$ is

$$\frac{\partial J(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})}{\partial \hat{y}^{(i)}} = \frac{\partial}{\partial \hat{y}^{(i)}}(\hat{y}^{(i)} - y^{(i)})^2$$

$$= 2(\hat{y}^{(i)} - y^{(i)}) = 2\delta^{(i)} \quad (83)$$

Then the gradient of J with respect to $b^{(2)}$ is

$$\frac{\partial J(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})}{\partial b^{(2)}} = \frac{\partial J(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})}{\partial \hat{y}^{(i)}} \frac{\partial \hat{y}^{(i)}}{\partial b^{(2)}}$$

$$= 2\delta^{(i)} \quad (84)$$

Next, the gradient of J with respect to $\boldsymbol{w}^{(2)}$ is

$$\frac{\partial J(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})}{\partial \boldsymbol{w}^{(2)}} = \frac{\partial J(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})}{\partial \hat{y}^{(i)}} \frac{\partial \hat{y}^{(i)}}{\partial \boldsymbol{w}^{(2)}}$$

$$= 2\delta^{(i)} \boldsymbol{h}_{out}{}^{(i)} \quad (85)$$

Now, stepping back a layer, we will need to consider the gradient of the activation function. In the case of the ReLU, we observe that the derivative is not continuous at 0. However, for the sake of our implementation, we can define the derivative at this point to be 0.

$$\frac{\partial \sigma(z)}{\partial z} = \begin{cases} 1 \; for \; z > 0 \\ 0 \; for \; z \leq 0 \end{cases} = sgn(\sigma(z)) \quad (86)$$

Now we can calculate the gradient of J with respect to $\boldsymbol{b}^{(1)}$

$$\frac{\partial J\left(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}\right)}{\partial \boldsymbol{b}^{(1)}} = \frac{\partial J\left(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}\right)}{\partial \hat{y}^{(i)}} \frac{\partial \hat{y}^{(i)}}{\partial \boldsymbol{h}_{out}{}^{(i)}} \frac{\partial \boldsymbol{h}_{out}{}^{(i)}}{\partial \boldsymbol{h}_{in}{}^{(i)}} \frac{\partial \boldsymbol{h}_{in}{}^{(i)}}{\partial \boldsymbol{b}^{(1)}}$$

$$= 2\delta^{(i)} \boldsymbol{w}^{(2)} \odot sgn(\boldsymbol{h}_{out}{}^{(i)}) \tag{87}$$

($\odot$ is used here to designate component-wise multiplication).

Finally, we can calculate the gradient of J with respect to $\mathrm{W}^{(1)}$

$$\frac{\partial J\left(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}\right)}{\partial \mathrm{W}^{(1)}} = \frac{\partial J\left(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}\right)}{\partial \hat{y}^{(i)}} \frac{\partial \hat{y}^{(i)}}{\partial \boldsymbol{h}_{out}{}^{(i)}} \frac{\partial \boldsymbol{h}_{out}{}^{(i)}}{\partial \boldsymbol{h}_{in}{}^{(i)}} \frac{\partial \boldsymbol{h}_{in}{}^{(i)}}{\partial \mathrm{W}^{(1)}}$$

$$= (2\delta^{(i)} \boldsymbol{w}^{(2)} \odot sgn(\boldsymbol{h}_{out}{}^{(i)})) \boldsymbol{x}^{(i)\mathrm{T}} \tag{88}$$

Once all of these gradients are calculated, all parameters, $\boldsymbol{\theta}$, can be updated before the process is repeated once again. For example

$$\boldsymbol{\theta}^{(new)} = \boldsymbol{\theta}^{(old)} - \alpha \frac{\partial \mathrm{J}(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}^{(old)})}{\partial \boldsymbol{\theta}^{(old)}} \tag{89}$$

where α is the learning rate hyperparameter. The learning rate can be a set value, or can be adjusted throughout the training process.

Now, we will look at how these gradients differ from those calculated when sending a batch of M examples though the network prior to updating any parameters. This is known as mini-batch gradient descent.

$$J\left(\left(\boldsymbol{x}^{(j)}, y^{(j)}\right), \ldots, \left(\boldsymbol{x}^{(j+M)}, y^{(j+M)}\right), \boldsymbol{\theta}\right) = \frac{1}{M} \sum_{i=j}^{j+M} (\hat{y}^{(i)} - y^{(i)})^2$$

$$= \frac{1}{M} \sum_{i=j}^{j+M} (\delta^{(i)})^2 = \frac{1}{M} \sum_{i=j}^{j+M} J(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}) \tag{90}$$

The relevant gradients of the cost function become

$$\frac{\partial J(X^{(batch)}, \boldsymbol{y}^{(batch)}, \boldsymbol{\theta})}{\partial b^{(2)}} = \frac{1}{M}\sum_{i=j}^{j+M} \frac{\partial}{\partial b^{(2)}} J(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$$

$$= \frac{1}{M}\sum_{i=j}^{j+M} \frac{\partial J\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)}{\partial \hat{y}^{(i)}} \frac{\partial \hat{y}^{(i)}}{\partial b^{(2)}} = \frac{1}{M}\sum_{i=j}^{j+M} 2\delta^{(i)} \quad (91)$$

$$\frac{\partial J\left(X^{(batch)}, \boldsymbol{y}^{(batch)}, \boldsymbol{\theta}\right)}{\partial \boldsymbol{w}^{(2)}} = \frac{1}{M}\sum_{i=j}^{j+M} \frac{\partial}{\partial \boldsymbol{w}^{(2)}} J\left(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}\right)$$

$$= \frac{1}{M}\sum_{i=j}^{j+M} \frac{\partial J\left(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}\right)}{\partial \hat{y}^{(i)}} \frac{\partial \hat{y}^{(i)}}{\partial \boldsymbol{w}^{(2)}} = \frac{1}{M}\sum_{i=j}^{j+M} 2\delta^{(i)} \boldsymbol{h}_{out}{}^{(i)} \quad (92)$$

$$\frac{\partial J\left(X^{(batch)}, \boldsymbol{y}^{(batch)}, \boldsymbol{\theta}\right)}{\partial \boldsymbol{b}^{(1)}} = \frac{1}{M}\sum_{i=j}^{j+M} \frac{\partial}{\partial \boldsymbol{b}^{(1)}} J\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)$$

$$= \frac{1}{M}\sum_{i=j}^{j+M} \frac{\partial J\left(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}\right)}{\partial \hat{y}^{(i)}} \frac{\partial \hat{y}^{(i)}}{\partial \boldsymbol{h}_{out}{}^{(i)}} \frac{\partial \boldsymbol{h}_{out}{}^{(i)}}{\partial \boldsymbol{h}_{in}{}^{(i)}} \frac{\partial \boldsymbol{h}_{in}{}^{(i)}}{\partial \boldsymbol{b}^{(1)}}$$

$$= \frac{1}{M}\sum_{i=j}^{j+M} 2\delta^{(i)} \boldsymbol{w}^{(2)} \odot sgn(\boldsymbol{h}_{out}{}^{(i)}) \quad (93)$$

$$\frac{\partial J\left(X^{(batch)}, \boldsymbol{y}^{(batch)}, \boldsymbol{\theta}\right)}{\partial W^{(1)}} = \frac{1}{M}\sum_{i=j}^{j+M} \frac{\partial}{\partial \boldsymbol{b}^{(1)}} J\left(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}\right)$$

$$= \frac{1}{M}\sum_{i=j}^{j+M} \frac{\partial J\left(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}\right)}{\partial \hat{y}^{(i)}} \frac{\partial \hat{y}^{(i)}}{\partial \boldsymbol{h}_{out}{}^{(i)}} \frac{\partial \boldsymbol{h}_{out}{}^{(i)}}{\partial \boldsymbol{h}_{in}{}^{(i)}} \frac{\partial \boldsymbol{h}_{in}{}^{(i)}}{\partial \mathrm{W}^{(1)}}$$

$$= \frac{1}{M}\sum_{i=j}^{j+M} (2\delta^{(i)}\boldsymbol{w}^{(2)} \odot sgn(\boldsymbol{h}_{out}{}^{(i)}))\boldsymbol{x}^{(i)\mathrm{T}} \qquad (94)$$

So as can be seen, batch gradient descent is simply a way to smooth out the fluctuations of on-line gradient descent by taking an average. While this is a smoother method of updating $\boldsymbol{\theta}$ rather than updating after every single example, there is still an opportunity for $\boldsymbol{\theta}$ to move inefficiently towards its optima. The update step is still highly dependent on the examples most recently seen. Additionally, it is very sensitive to a good selection of α. Too large, and the optima or valleys in the objective function are continually overshot. Too small, and the optimization proceeds exceedingly slowly, and can potentially become stuck in flat regions of the objective function.

We may like the gradient of the cost function to be less sensitive to α, and to have some memory of previous updates, similar to the momentum a ball rolling down a hillside would experience. This is where various solver techniques come into play. A popular and very successful solver is known as Adam (Adaptive Moment Estimation), which is outlined below [10]

Select α as the step-size (commonly set to $\alpha = 0.001$)

Select the exponential decay rates for the moment estimates $\beta_1, \beta_1 \in [0, 1)$
(commonly set to β_1= 0.9, $\beta_2 = 0.999$)

Select ϵ (commonly set to $\epsilon = 10^{-8}$), used to avoid division by zero

Let $J_t(\boldsymbol{\theta})$ be the cost function of a mini-batch (mini-batch could have size 1 to n) at timestep *t*.

Let $\nabla_{\boldsymbol{\theta}} J_t(\boldsymbol{\theta}_{t-1})$ be the gradient of the cost function with respect to $\boldsymbol{\theta}$ at timestep *t*.

Select initial parameters $\boldsymbol{\theta_0}$

Initialize the 1st moment vector: $\mathbf{m}_0 \leftarrow \mathbf{0}$

Initialize the 2nd moment vector: $\mathbf{v}_0 \leftarrow \mathbf{0}$

Initialize timestep: $t = 0$

While $\boldsymbol{\theta}_t$ not converged:

- $t \leftarrow t + 1$
- $\boldsymbol{g}_t \leftarrow \nabla_{\boldsymbol{\theta}} J_t(\boldsymbol{\theta}_{t-1})$
- $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$ (update biased first moment estimate)
- $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t \odot \boldsymbol{g}_t$ (update biased second moment estimate)
- $\widehat{\boldsymbol{m}}_t \leftarrow \frac{\boldsymbol{m}_t}{(1-\beta_1{}^t)}$ (compute bias-corrected first moment estimate)
- $\widehat{\boldsymbol{v}}_t \leftarrow \frac{\boldsymbol{v}_t}{(1-\beta_2{}^t)}$ (compute bias-corrected second moment estimate)
- $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \alpha \frac{\widehat{\boldsymbol{m}}_t}{\sqrt{\widehat{\boldsymbol{v}}_t} + \epsilon}$ (update parameters, the division and square root are performed component-wise)

End while

Return $\boldsymbol{\theta}_t$

In the Adam algorithm, the $\boldsymbol{v}_t$ term stores an exponentially decaying average of all previous squares of the gradient, while the $\boldsymbol{m}_t$ term represents an exponentially decaying average of all previous gradients (not squared). The bias-correction terms are included since both $\boldsymbol{v}_t$ and $\boldsymbol{m}_t$ are initialized to $\mathbf{0}$. A benefit of Adam is that the step sizes of all updates of $\boldsymbol{\theta}$ approximate the step-size parameter, α, since $|\widehat{\boldsymbol{m}}_t| \sim \sqrt{\widehat{\boldsymbol{v}}_t}$. Once $\sqrt{\widehat{\boldsymbol{v}}_t}$ approaches ϵ (meaning the weighted average of the gradient is very small), the size of the update of $\boldsymbol{\theta}$ will become smaller and smaller as $\boldsymbol{\theta}$ settles in a local minimum.

There are a number of hyperparameters that must be chosen when training a neural network. Some of these hyperparameters include, but are not limited to: number of hidden neurons, learning rate, batch size, a regularization parameter value, and any solver-specific parameters. Selecting the best hyperparameters is an optimization problem in itself. There are several ways one could go about doing this. One way is to simply manually adjust hyperparameters and observe any changes in model performance. This works well if the individual has experience with similar problems and has a solid understanding of the impact of each hyperparameter. However, this method can also be tedious and potentially inefficient.

Another method is to perform a grid search. This is where hyperparameters are segmented into lists of potential values and then all possible combinations of hyperparameter values are iterated over, and then the best performing set is selected.

Bergstra and Bengio note in their 2012 paper [6] that this method has some limitations in that it may oversample from non-impactful hyperparameters, and completely miss the optimal combinations (see Figure 31). For this reason, a random search is suggested. This can allow more sampling along the high-impact hyperparameters without an exponential increase in training time.

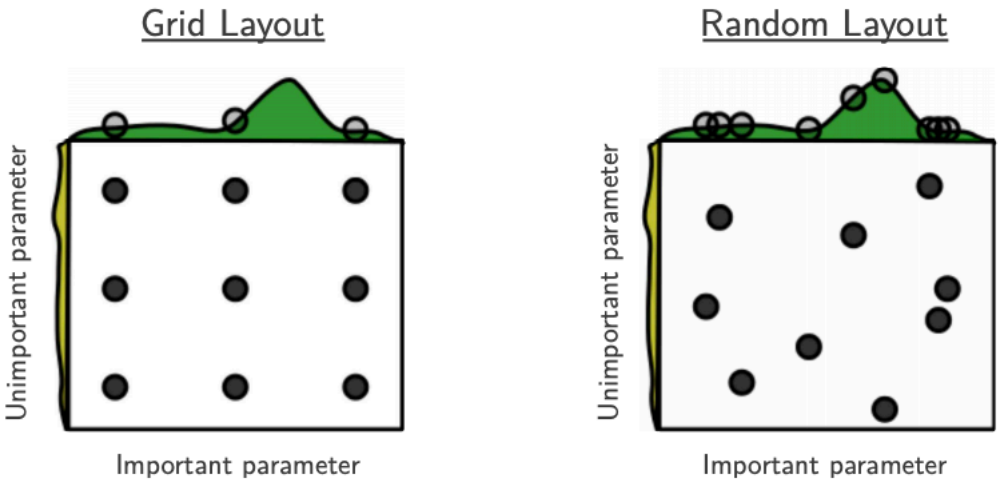


Fig. 31. Figure from Bergstra and Bengio [6]. By performing a random sampling of hyperparameters, the likelihood of under-sampling the most important parameters decreases without exponentially increasing computational time by simply using a finer grid search. Note, however, that the range of sampling is important. If optimal parameter values occur at an extreme of a sampling window, the window ought to be shifted or widened to ensure no further improvement is possible.

For our weather dataset, we will observe the impact of four different hyperparameters: number of hidden neurons, batch size, learning rate, and a regularization parameter, alpha, which, similarly to ridge regression, encourages the sum of squared parameter values to be decrease. Note that this alpha is different than that shown in the Adam optimization algorithm. The alpha value used in Adam is here referred to as the initial learning rate.

Critical to finding optimal values for hyperparameters is selecting an appropriate range from which to sample. Additionally, many hyperparameters are sampled from a log distribution, as their impact is typically only observed as their order of magnitude is changed.

A random grid search was performed for the training set of the dataset containing current-measurements-only. While the joint distribution of performance for all four hyperparameters cannot be visualized as was done in Figure 31, Figure 32 contains plots of the model performance on the validation set with respect to only one of the marginal variables at a time.

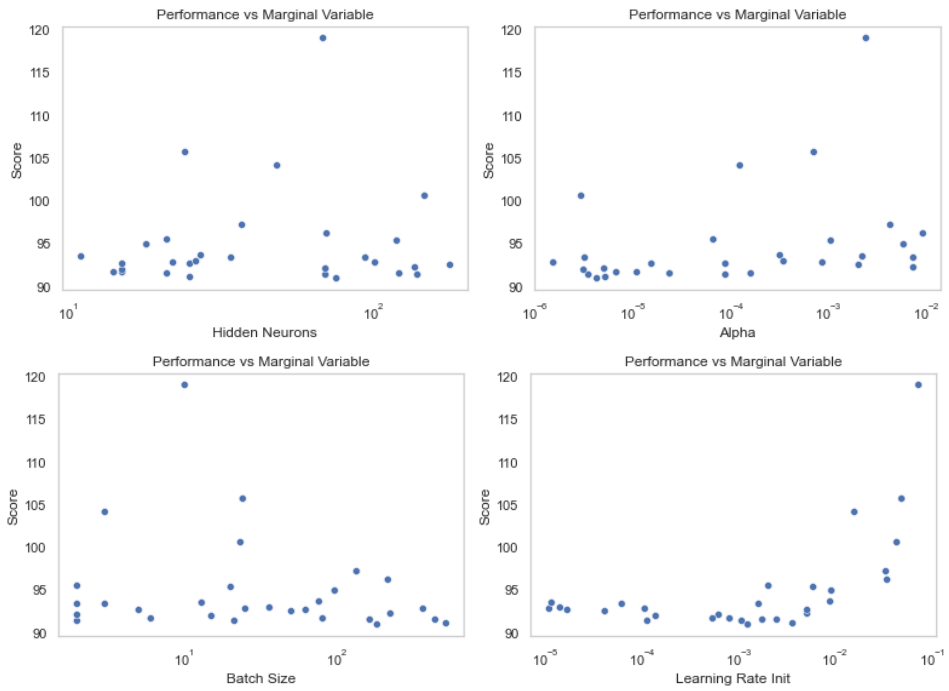


Fig. 32. Model performance on validation set (current-measurements-only dataset). Note that each subplot is simply a view of the marginal distribution of performance with respect to a single hyperparameter – other hyperparameter values are not fixed as one is adjusted.

It is difficult to determine the optimal combination of hyperparameters from simply observing the performance versus a single marginal hyperparameter. However, these plots can inform us as to particularly poor hyperparameter selections, and which ranges of values seem to be inconsequential.

From Figure 32 we can conclude that an initial learning rate greater than 10^{-2} is generally a poor selection, and that there is not much variance in model performance when this value is between 10^{-5} and 10^{-3}. From this, we can surmise that the default value of this hyperparameter of 10^{-3} ought to be acceptable. Furthermore, we observe that while some models with low batch size perform well, there is much less variance in model performance when this value is greater than ~30. This fact as well as the long training time for models with low batch size settings encourages us to select a batch size in the range of 30 to 1,000. No strong trends can be observed from this amount of sampling for the optimal (or consistently poor) settings of alpha or the number of hidden neurons. To choose the best model settings for which to train a final model consisting of the training and validation set combination, we can simply select the best performing combination of hyperparameters from these trials. The top 5 models can be seen in Table IX

Table IX
Top 5 Performing Hyperparameter Combinations
For Model Using Only Current Timestep Measurements

Rank	Number of Hidden Neurons	Alpha (regularization parameter)	Batch Size	Initial Learning Rate	Validation Set RMSE
1	75	0.000004	180	0.001216	90.99
2	25	0.000005	506	0.003584	91.17
3	138	0.000003	21	0.001058	91.37
4	69	0.000087	2	0.000111	91.47
5	21	0.000023	161	0.002424	91.52

It should be noted that the final model generated by a set of hyperparameters and given dataset is not deterministic. There are several sources of randomness in the training of a multi-layer perceptron. One source is the random initialization of all model parameters. Another is the random selection of data instances for each batch update. This variation can be observed in Figure 33, where five models were trained for each number of hidden neurons shown, while keeping all other hyperparameters fixed.

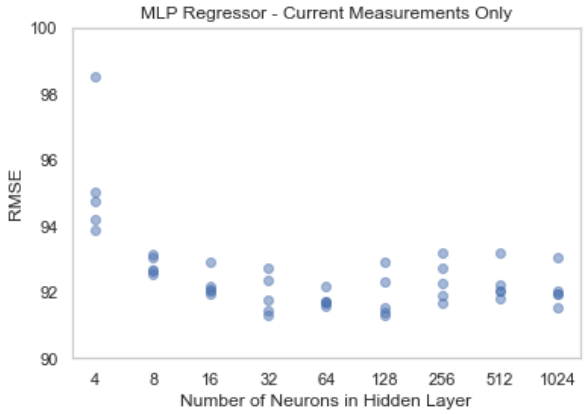


Fig. 33. The final parameter values of a multi-layer perceptron can be different each time the model is trained, even when all hyperparameters are unchanged. In this example, all hyperparameters were fixed, except for the number of hidden neurons in the hidden layer. For each of these values, the same model was trained five times. The performance of each model on the validation set is plotted. Observe how some hyperparameter values can result in a larger variance of ultimate model performance than others. An optimal hyperparameter selection is one with both low average error and low variance.

Returning to Table IX and Figure 32, we observe that the best performing set of hyperparameters (from Table IX) appears to also reside in a low-variance region of hyperparameter-space (Figure 32). Due to this, we will select this set of hyperparameters to train a final model on the combined training and validation set for our current-measurements-only dataset.

Moving now to the dataset containing current and previous timestep measurements, as well as that containing the current and previous three timestep measurements, we can observe that each feature distribution in these datasets is identical to one in the current-timestep-only dataset. Therefore, it ought to be sufficient to maintain the previously selected values for learning rate and batch size.

However, with more features in these two datasets, we may want to see the effect of adjusting the number of hidden neurons and the regularization parameter, alpha. Figures 34 and 35 display the marginal distributions of model performance as only two hyperparameters are adjusted. Tables X and XI list the top-five performing sets of hyperparameters for each dataset.

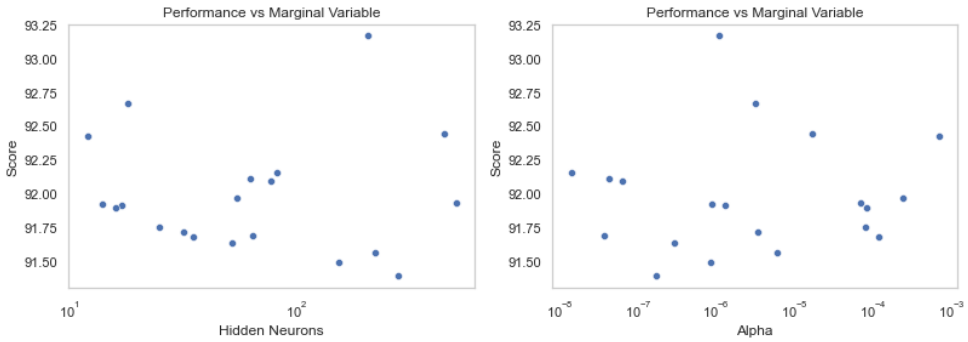


Fig. 34. Model performance on validation set (current-and-previous-measurements dataset). Note that each subplot is simply a view of the marginal distribution of performance with respect to a single hyperparameter – the other hyperparameter value are not fixed as one is adjusted. Note the y-axis range on each of the subplots above compared to those in Figure 32. It appears that there is very little performance variance in models trained using any selection of hyperparameters in the given ranges.

Table X
Top 5 Performing Hyperparameter Combinations
For Model Using Current and Previous Timestep Measurements

Rank	Number of Hidden Neurons	Alpha (regularization parameter)	Batch Size	Initial Learning Rate	Validation Set RMSE
1	281	1.7207e-07	180	0.001	91.39
2	153	8.4901e-07	180	0.001	91.49
3	224	5.9601e-06	180	0.001	91.57
4	52	2.9358e-07	180	0.001	91.64
5	35	1.1840e-04	180	0.001	91.68

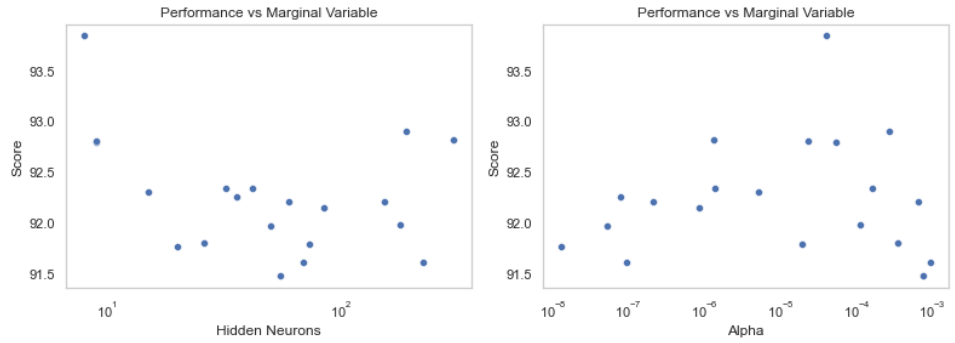


Fig. 35. Model performance on validation set (current-and-previous-three-measurements dataset). Note that each subplot is simply a view of the marginal distribution of performance with respect to a single hyperparameter – the other hyperparameter value are not fixed as one is adjusted. Note the y-axis range on each of the subplots above compared to those in Figure 32. It appears that there is very little performance variance in models trained using any selection of hyperparameters in the given ranges.

Table XI
Top 5 Performing Hyperparameter Combinations
For Model Using Current and Previous Three Timestep Measurements

Rank	Number of Hidden Neurons	Alpha (regularization parameter)	Batch Size	Initial Learning Rate	Validation Set RMSE
1	55	7.4436e-04	180	0.001	91.48
2	224	9.3320e-04	180	0.001	91.61
3	69	9.1638e-08	180	0.001	91.61
4	20	1.3044e-08	180	0.001	91.77
5	73	1.8843e-05	180	0.001	91.80

For each of the three datasets, a final model was trained on their combined training and validation sets using the top-performing hyperparameters seen in Tables IX, X, and XI. The evaluations of these trained models on the testing sets are shown in Table XII.

Table XII
Final Hyperparameter Selection and Model Evaluation on Testing Sets

Dataset	Number of Hidden Neurons	Alpha (regularization parameter)	Batch Size	Initial Learning Rate	Testing Set RMSE
Current Measurements Only	75	4.0e-06	180	0.001	89.76
Current and Previous Measurements	281	1.7e-07	180	0.001	90.84
Current and Previous Three Measurements	55	7.4e-04	180	0.001	88.78

While for each dataset, the model performance improved when trained using the combination of training and validation sets (demonstrating the value of additional data), it is not clear that utilizing more historical data improves the predictive capabilities of a single-layer neural network. Figure 36 shows the performance of each model on a selection of the testing set.

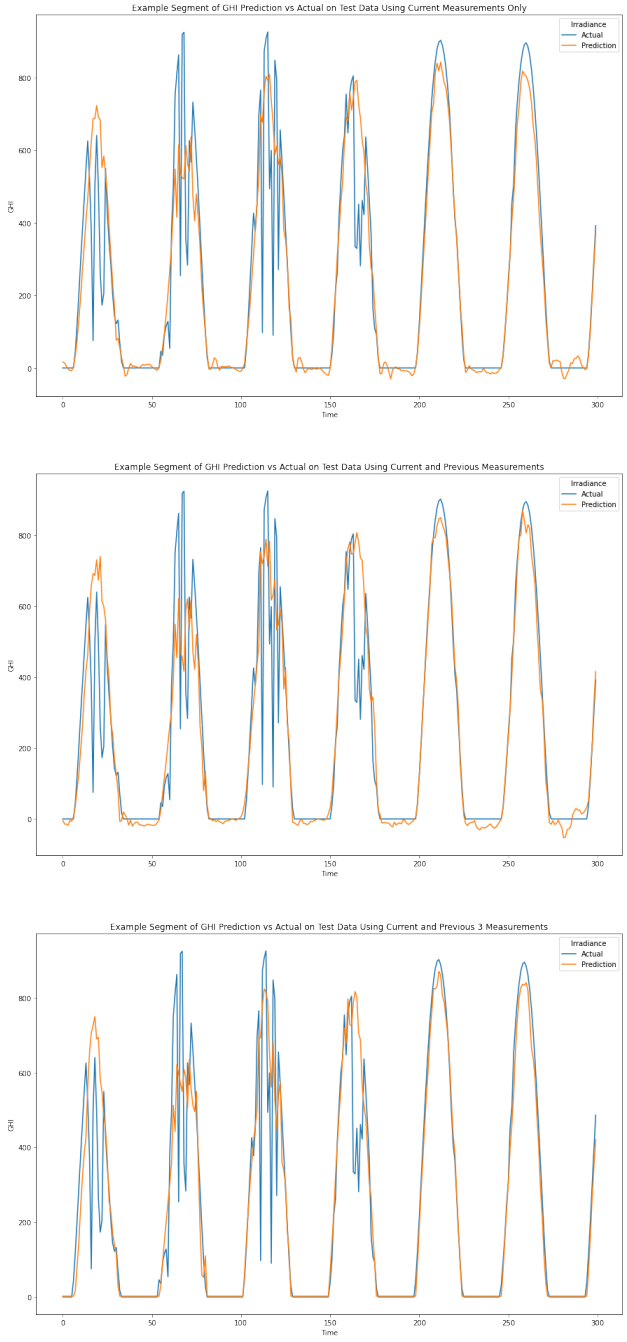


Fig. 36. Comparisons of single-layer neural network predictions (orange) and actual observed values (blue) of GHI in Pittsburgh. The top plot shows the model when only current weather measurements are used to make predictions. The middle plot shows the model when the current and previous timestep (30 minutes prior) measurements are used. The bottom plot shows the model when the current and previous three timestep (30 minutes, 1 hour, and 1.5 hours prior) measurements are used. Each model has a very similar predictive accuracy. Interestingly, however, the model using the current and previous three timesteps (bottom plot) is the only one that seems to have learned to predict a constant GHI value of zero during the night.

Conclusion

Throughout this paper we have explored the theory and application of five classes of regression models: linear regression, ridge regression, lasso regression, decision trees, and neural networks. We observed that the size of the weather dataset caused the optimal ridge and lasso regression models to be equivalent to a linear regression model with no regularization hyperparameter. The performance of a linear model, as well as decision trees and neural networks on the testing set are displayed in Table XIII.

Table XIII
Final Model Evaluation on Testing Sets

Dataset	Linear Model RMSE	Decision Tree RMSE	Neural Network RMSE
Current Measurements Only	141.44	98.07	89.76
Current and Previous Measurements	121.47	98.13	90.84
Current and Previous Three Measurements	119.71	98.27	88.78

Table XIII demonstrates that neural networks provided the most accurate irradiance forecasts out of all models explored. However, contrary to intuition, adding additional historical weather measurements to the input did not appear to significantly improve the predictive accuracy of the neural network. Further work could explore the use of additional locations from which weather data could be obtained. Candidate locations could either be cities further away from Pittsburgh than those currently utilized, or additional locations which increase the spatial density of the sampled data. Greatly increasing the density of locations sampled could motivate utilizing more advanced models such as convolutional neural networks. Furthermore, rather than simply adding further historical measurements to the input (thus greatly increasing the number of model parameters), specific models designed for sequence forecasting could be utilized, such as recurrent neural networks (RNNs) or long short-term memory networks (LSTMs).

Bibliography

[1] Renewables, SEIA/Wood Mackenzie Power &, "U.S. Solar Market Insight 2020 Q4," SEIA/Wood Mackenzie Power & Renewables, 2020.

[2] National Renewable Energy Laboratory, "NSRDB Data Viewer," [Online]. Available: https://maps.nrel.gov/nsrdb-viewer/. [Accessed 20 July 2020].

[3] M. Sengupta, Y. Xie, A. Lopez, A. Habte, G. Maclaurin and J. Shelby, "The National Solar Radiation Data Base (NSRDB)," *Renewable and Sustainable Energy Reviews,* vol. 89, no. June, pp. 51-60, 2018.

[4] D. P. N. Nguyen and J. Lauwaert, "Calculating the Energy Yield of Si-Based Solar Cells for Belgium and Vietnam Regions at Arbitrary Tilt and Orientation under Actual Weather Conditions," *Energies,* vol. 13, no. 12, p. 3180, 2020.

[5] Homer Energy, "Homer Pro 3.14," [Online]. Available: https://www.homerenergy.com/products/pro/docs/latest/global_horizontal_irradiance_ghi.html. [Accessed 19 February 2021].

[6] D. G. Luenberger and Y. Ye, Linear and Nonlinear Programming, Switzerland: Springer International Publishing, 2016.

[7] C. Mecklenbrauker, P. Gerstoft and E. Zochmann, "Using the LASSO's Dual for Regularization in Sparse Signal Reconstruction from Array Data," *IEEE Transactions on Signal Processing,* 2015.

[8] A. Geron, Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow, Sebastopol, CA: O'Reilly Media, Inc., 2019.

[9] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, Cambridge, MA: MIT Press, 2016.

[10] D. P. Kingma and J. L. Ba, "Adam: A Method for Stochastic Optimization," in *ICLR*, 2015.

[11] J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization," *Journal of Machine Learning Research,* vol. 13, pp. 281-305, 2012.