TUTORIAL 1

Linear regression

Recap from the lectures

The setup We have data of the form \( \{(\mathbf{x}_i, y_i)\}_{i=1}^n \) where the \( \mathbf{x}_i \)'s are predictors and the \( y_i \)'s are outcomes. We want to estimate the relationship between the outcome variable and the predictors. This is called regression analysis. In linear regression, we assume the relationship is linear

\[
y_i = \beta_0 + \sum_{j=1}^{d} \beta_j x_{i,j} + \epsilon_i
\]

where \( \epsilon_i \) is noise. We use the least-squares approach of finding coefficients that minimize

\[
\sum_{i=1}^{n} \left( y_i - \left\{ \beta_0 + \sum_{j=1}^{d} \beta_j x_{i,j} \right\} \right)^2.
\]

We re-write it in matrix form. Let

\[
\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad \mathbf{A} = \begin{pmatrix} 1 & \mathbf{x}_1^T \\ 1 & \mathbf{x}_2^T \\ \vdots & \vdots \\ 1 & \mathbf{x}_n^T \end{pmatrix} \quad \text{and} \quad \mathbf{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_d \end{pmatrix}.
\]

Then the problem is

\[
\min_{\mathbf{\beta}} \| \mathbf{y} - \mathbf{A} \mathbf{\beta} \|^2.
\]
**The theory** In words, we are looking for a linear combination of the columns of $A$ that is closest to $y$ in 2-norm.

**Theorem (Gram-Schmidt):** Let $a_1, \ldots, a_m$ be linearly independent. Then there exist an orthonormal basis $q_1, \ldots, q_m$ of $\text{span}(a_1, \ldots, a_m)$.

In matrix form, let

$$A = \begin{pmatrix} | & \cdots & | \\ a_1 & \cdots & a_m \end{pmatrix} \quad \text{and} \quad Q = \begin{pmatrix} | & \cdots & | \\ q_1 & \cdots & q_m \end{pmatrix}.$$  

The output of the Gram-Schmidt algorithm above can then be written in the following compact form, known as a **QR decomposition** (https://en.wikipedia.org/wiki/QR_decomposition),

$$A = QR$$

where column $i$ of the $m \times m$ matrix $R$ contains the coefficients of the linear combination of $q_j$'s that produces $a_i$. By the proof of Gram-Schmidt, $a_i \in \text{span}(q_1, \ldots, q_i)$. So column $i$ of $R$ has only zeros below the diagonal. Hence $R$ has a special structure: it is upper triangular.

**Definition (Orthogonal projection):** Let $U \subseteq V$ be a linear subspace with orthonormal basis $q_1, \ldots, q_m$. The orthogonal projection of $v \in V$ on $U$ is defined as

$$P_U v = \sum_{j=1}^{m} \langle v, q_j \rangle q_j.$$  

\<

**Theorem (Orthogonal Projection is Closest in Subspace):** Let $U \subseteq V$ be a linear subspace with orthonormal basis $q_1, \ldots, q_m$ and let $v \in V$. For any $u \in U$

$$\|v - P_U v\| \leq \|v - u\|.$$  

Furthermore, if $u \in U$ and the inequality above is an equality, then $u = P_U v$.  

From this result we get:

**Theorem (Least Squares via QR):** Let \( A \in \mathbb{R}^{n \times m} \) be an \( n \times m \) matrix with linearly independent columns, let \( b \in \mathbb{R}^n \) be a vector, and let \( A = QR \) be a QR decomposition of \( A \). The solution to the least-squares problem

\[
\min_{x \in \mathbb{R}^m} \|Ax - b\|
\]

satisfies

\[
Rx^* = Q^Tb.
\]

**The algorithm**

```python
In [2]:
def mmids_gramschmidt(A):
    n, m = size(A)
    Q = zeros(Float64, n, m)
    R = zeros(Float64, m, m)
    for j = 1:m
        v = A[:,j]
        for i = 1:j-1
            R[i,j] = dot(Q[:,i],A[:,j])
            v -= R[i,j]*Q[:,i]
        end
        R[j,j] = norm(v)
        Q[:,j] = v/R[j,j]
    end
    return Q, R
end

Out[2]: mmids_gramschmidt (generic function with 1 method)
```

```python
In [3]:
def mmids_backsubs(U,b):
    m = length(b)
    x = zeros(Float64, m)
    for i=m:-1:1
        x[i] = (b[i] - dot(U[i,i+1:m],x[i+1:m]))/U[i,i] # assumes non-zero diagonal
    end
    return x
end

Out[3]: mmids_backsubs (generic function with 1 method)
```

```python
In [4]:
def ls_by_qr(A, b):
    Q,R = mmids_gramschmidt(A)
    return mmids_backsubs(R,Q'*b)
end

Out[4]: ls_by_qr (generic function with 1 method)
```
1 Predicting sales

The following dataset is from [ISLR] textbook. Quoting [ISLR, Section 2.1]:

Suppose that we are statistical consultants hired by a client to provide advice on how to improve sales of a particular product. The Advertising data set consists of the sales of that product in 200 different markets, along with advertising budgets for the product in each of those markets for three different media: TV, radio, and newspaper. [...] It is not possible for our client to directly increase sales of the product. On the other hand, they can control the advertising expenditure in each of the three media. Therefore, if we determine that there is an association between advertising and sales, then we can instruct our client to adjust advertising budgets, thereby indirectly increasing sales. In other words, our goal is to develop an accurate model that can be used to predict sales on the basis of the three media budgets.

In [5]: df = CSV.read("advertising.csv")
first(df,5)

Out[5]: 5 rows × 4 columns

<table>
<thead>
<tr>
<th></th>
<th>TV</th>
<th>radio</th>
<th>newspaper</th>
<th>sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>230.1</td>
<td>37.8</td>
<td>69.2</td>
<td>22.1</td>
</tr>
<tr>
<td>2</td>
<td>44.5</td>
<td>39.3</td>
<td>45.1</td>
<td>10.4</td>
</tr>
<tr>
<td>3</td>
<td>17.2</td>
<td>45.9</td>
<td>69.3</td>
<td>9.3</td>
</tr>
<tr>
<td>4</td>
<td>151.5</td>
<td>41.3</td>
<td>58.5</td>
<td>18.5</td>
</tr>
<tr>
<td>5</td>
<td>180.8</td>
<td>10.8</td>
<td>58.4</td>
<td>12.9</td>
</tr>
</tbody>
</table>

We will focus for now on the TV budget.
We form the matrix $A$ and use our least-squares code to solve for $\beta$.

```python
In [7]:
n = length(sales)
A = hcat(ones(n), TV)
coeff = ls_by_qr(A, sales)

Out[7]:
2-element Array{Float64,1}:
7.032593549127698
0.047536640433019715
In [8]:
TVgrid = LinRange(minimum(TV), maximum(TV), 100)
scatter(TV, sales, legend=false, alpha=0.5)
plot!(TVgrid, coeff[1].+coeff[2]*TVgrid, lw=2)  # recall that array indices start at 1

Out[8]:

A degree-two polynomial might be a better fit.

In [9]:
A = reduce(hcat, [ones(n), TV, TV.^2])
coeff = ls_by_qr(A, sales)

Out[9]:
3-element Array{Float64,1}:
6.114120128333173
0.06726592695624341
-6.846933732938574e-5
In [10]:
scatter(TV, sales, legend=False, alpha=0.5)

Out[10]:

The fit looks slightly better than the linear one. This is not entirely surprising though given that the linear model is
a subset of the quadratic one. But, in adding more parameters, one must worry about overfitting
(http://en.wikipedia.org/wiki/Overfitting#cite_note-1). To quote Wikipedia:

In statistics, overfitting is "the production of an analysis that corresponds too closely or exactly
to a particular set of data, and may therefore fail to fit additional data or predict future
observations reliably".[1 (http://en.wikipedia.org/wiki/Overfitting#cite_note-1)] An overfitted
model is a statistical model that contains more parameters than can be justified by the data.[2
(http://en.wikipedia.org/wiki/Overfitting#cite_note-CDS-2)] The essence of overfitting is to have
unknowingly extracted some of the residual variation (i.e. the noise) as if that variation
represented underlying model structure.[3 (http://en.wikipedia.org/wiki/Overfitting#cite_note-
BA2002-3)]

To illustrate, let’s see what happens with a degree-20 polynomial fit.
```python
In [11]:
    deg = 20
    A = reduce(hcat, [TV.^i for i=0:1:deg])
    coeff = ls_by_qr(A, sales)

Out[11]:
21-element Array{Float64,1}:
    3.5485182942335296
    1.9983587342791638
   -0.12293286246561463
    0.0035915780264108243
    5.608819661563708e-5
    5.014719202946552e-7
   -2.5949492351362565e-9
    7.241091423601946e-12
   -7.455713695819655e-15
   -9.418178511077042e-18
    2.4591311142030467e-20
   -6.316259228977537e-24
   -2.7466282663160562e-27
   -2.4036415725501296e-30
   -2.9082523810373025e-33
   -4.2946537585142373e-36
   -7.27662549565319e-39
   -1.3642164068864983e-41
   -2.7643672469104974e-44
   -5.957199891770676e-47
   -1.349448254017876e-49

Out[12]:

In [12]:
    saleshat = sum([coeff[i+1]*TVgrid.^i for i=0:1:deg])
    scatter(TV, sales, legend=false, alpha=0.5)
    plot!(TVgrid, saleshat, lw=2)

Out[12]:
```
We could use cross-validation ([cross-validation](https://www.textbook.ds100.org/ch/15/bias_cv.html)) to choose a suitable degree.

We return to the linear case, but with the full set of predictors.

```python
In [13]: radio, newspaper = df[:, :radio], df[:, :newspaper];
```

```python
In [14]: p1 = scatter(radio, sales, legend='false', alpha=0.5, title='radio')
p2 = scatter(newspaper, sales, legend='false', alpha=0.5, title='newspaper')
plot(p1, p2, layout=(2, 1), size=(750, 375))
```

Out[14]:

![Plots of radio and newspaper against sales](image)

```python
In [15]: A = reduce(hcat, [ones(n), TV, radio, newspaper])
coeff = ls_by_qr(A, sales)
```

Out[15]:

4-element Array{Float64,1}:

2.9388893694594307
0.04576464545539755
0.18853001691820379
-0.0010374930424762435

Newspaper advertising (the last coefficient) seems to have a much weaker effect on sales per dollar spent. We will come back later in the course to assessing the statistical significance ([statistical significance](https://en.wikipedia.org/wiki/Statistical_significance)) of such a conclusion.
2 Analyzing flight delays [optional]

Suppose we are interested in gaining a better understanding of flight delays. Let's get some data from the DOT's Bureau of Transportation Statistics (https://www.transtats.bts.gov/DL_SelectFields.asp). The following dataset contains information about every flight out of Madison, WI between December 1, 2018 and November 30, 2019 (that were not cancelled or diverted). This information includes the date (date), day of week (day_of_week; 1 is Monday, 2 is Tuesday and so on), carrier (carrier), flight number (fl_num), destination (dest), scheduled departure time (sch_dep; e.g. 20.5 is 8:30pm), scheduled arrival time (sch_arr), and arrival delay (arr_delay; e.g. -33.0 means the flight arrived 33 minutes early).

In [16]:
df = CSV.read("msn-flight-data-19.csv")
first(df, 5)

Out[16]: 5 rows × 8 columns

<table>
<thead>
<tr>
<th>date</th>
<th>day_of_week</th>
<th>carrier</th>
<th>fl_num</th>
<th>dest</th>
<th>sch_dep</th>
<th>sch_arr</th>
<th>arr_delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-12-10</td>
<td>1</td>
<td>DL</td>
<td>949</td>
<td>DTW</td>
<td>13.6667</td>
<td>16.1</td>
<td>-33.0</td>
</tr>
<tr>
<td>2018-12-10</td>
<td>1</td>
<td>DL</td>
<td>1223</td>
<td>ATL</td>
<td>17.2667</td>
<td>20.4833</td>
<td>-16.0</td>
</tr>
<tr>
<td>2018-12-10</td>
<td>1</td>
<td>DL</td>
<td>1386</td>
<td>MSP</td>
<td>6.41667</td>
<td>7.78333</td>
<td>-21.0</td>
</tr>
<tr>
<td>2018-12-10</td>
<td>1</td>
<td>DL</td>
<td>1495</td>
<td>DTW</td>
<td>6.58333</td>
<td>9.1</td>
<td>-27.0</td>
</tr>
<tr>
<td>2018-12-10</td>
<td>1</td>
<td>DL</td>
<td>1547</td>
<td>MSP</td>
<td>17.3333</td>
<td>18.6333</td>
<td>72.0</td>
</tr>
</tbody>
</table>

In [17]:
describe(df)

Out[17]: 8 rows × 8 columns

<table>
<thead>
<tr>
<th>variable</th>
<th>mean</th>
<th>min</th>
<th>median</th>
<th>max</th>
<th>nunique</th>
<th>nmissing</th>
<th>eltype</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>2018-12-01</td>
<td>2019-11-30</td>
<td>365</td>
<td>Date</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>day_of_week</td>
<td>3.90154</td>
<td>1</td>
<td>4.0</td>
<td>7</td>
<td>Int64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>carrier</td>
<td>9E</td>
<td>YX</td>
<td>11</td>
<td>String</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fl_num</td>
<td>3515.57</td>
<td>159</td>
<td>3651.0</td>
<td>6117</td>
<td>Int64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dest</td>
<td>ATL</td>
<td>SLC</td>
<td>26</td>
<td>String</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sch_dep</td>
<td>12.7728</td>
<td>5.08333</td>
<td>13.3167</td>
<td>21.9</td>
<td>Float64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sch_arr</td>
<td>14.9175</td>
<td>6.23333</td>
<td>15.7833</td>
<td>23.9833</td>
<td>Float64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>arr_delay</td>
<td>6.59901</td>
<td>-61.0</td>
<td>-8.0</td>
<td>1540.0</td>
<td>Float64</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Let's say we suspect that the average delay gets worse throughout the day. By using linear regression, we can try to quantify the relationship between scheduled departure times and arrival delays. Let's extract the relevant columns first.
In [18]:
sch_dep = df[:, :sch_dep];
arr_delay = df[:, :arr_delay];
n = nrow(df)

Out[18]: 13579

The average delay in minutes was:

In [19]:
mean(arr_delay)

Out[19]: 6.5990131821194495

Here is an histogram of arrival delays (truncated at 4 hours).

In [20]:
histogram(arr_delay, legend=false, nbins=300, xlims=(-60, 240))

Out[20]:

To get a sense of the relationship between scheduled departure times and arrival delays, we use a scatter plot. (This is a rather large dataset. To avoid making the notebook too sluggish, we only plot a random samples of 1000 flights.)
Finally, we form the matrix $A$ and use our linear regression algorithm.

```python
In [22]:
A = reduce(hcat, [ones(n), sch_dep])
beta = ls_by_qr(A, arr_delay)
```

```
Out[22]:
2-element Array{Float64,1}:
-3.906848102051145
0.8225205619598612
```

So this suggest that delays increase by roughly 0.8 minutes per hour. Over 17-hour day, that adds up to about 14 minutes by the end of the day.
In [23]:
    
x = LinRange(minimum(sch_dep), maximum(sch_dep), 100)
y = beta[1] .+ beta[2]*x;
scatter(sch_dep[subsample], arr_delay[subsample],
    legend=false, alpha=0.2, ylims=(-60,240))
plot!(x, y, lw=2)

Out[23]:

We may suspect that other factors have a greater impact on delays, for instance, the destination. Let's look at O'Hare. The average delay when going from Madison to O'Hare is:

In [24]:
    
dest = df[:, dest]
mean(arr_delay[findall(x -> x == "ORD", dest)])

Out[24]: 12.519916868721857

To account for the effect of the time of day (https://en.wikipedia.org/wiki/Controlling_for_a_variable) (perhaps flights to ORD tend to be later), we perform a multiple linear regression with both sch_dep and whether or not the destination is ORD. How do we encode the latter in a numerical value? We use dummy coding (https://en.wikipedia.org/wiki/Categorical_variable#Dummy_coding). That is, we introduce a new variable which is 1 if a flight is headed to ORD and 0 otherwise.

In [25]:
    
to_ord = Float64.(dest .== "ORD");

We then add to_ord to our $A$ matrix and perform multiple linear regression.
In [26]:
A = reduce(hcat, [ones(n), sch_dep, to_ord])
beta = ls_by_qr(A, arr_delay)

Out[26]:
3-element Array{Float64,1}:
-5.331027103829616
  0.8114827228893247
  7.361740408240546

So going to ORD adds roughly 7 minutes to the baseline of all other flights.

Exercise: Suppose that we add two extra columns: one which encodes "going to MSP", that is, it is 0 if a flight is going to MSP and 1 otherwise; one which encodes every other flight, that is, it is 1 if the flight is going to neither ORD or MSP, and 0 otherwise.

In [27]:
to_msp = Float64.(dest .== "MSP")
to_others = Float64.((dest .!= "ORD") .& (dest .!= "MSP"))
A = reduce(hcat, [ones(n), sch_dep, to_ord, to_msp, to_others])
beta = ls_by_qr(A, arr_delay)

Out[27]:
5-element Array{Float64,1}:
4.62010164573336e15
  2.0560850358459306
-4.62010164573337e15
-4.620101645733381e15
-4.620101645733378e15

What happened? Which assumption is not satisfied in this case? Check your answer numerically. Find the right solution by modifying the matrix $A$. △
3 Bootstrapping for linear regression [optional]

Our coefficients are estimated from a sample. There is intrinsic variability in our sampling procedure. We would like to understand how our estimated coefficients compare to the true coefficients. This is set up beautifully in [Data8 (https://www.inferentialthinking.com/chapters/13/2/Bootstrap.html), Section 13.2]:

A data scientist is using the data in a random sample to estimate an unknown parameter. She uses the sample to calculate the value of a statistic that she will use as her estimate. Once she has calculated the observed value of her statistic, she could just present it as her estimate and go on her merry way. But she’s a data scientist. She knows that her random sample is just one of numerous possible random samples, and thus her estimate is just one of numerous plausible estimates. By how much could those estimates vary? To answer this, it appears as though she needs to draw another sample from the population, and compute a new estimate based on the new sample. But she doesn’t have the resources to go back to the population and draw another sample. It looks as though the data scientist is stuck. Fortunately, a brilliant idea called the bootstrap can help her out. Since it is not feasible to generate new samples from the population, the bootstrap generates new random samples by a method called resampling: the new samples are drawn at random from the original sample.

Without going into full details (see [DS100 (https://www.textbook.ds100.org/ch/18/hyp_regression.html), Section 18.3] for more), it works as follows. Let \( \{(x_i, y_i)\}_{i=1}^n \) be our data. We assume that our sample is representative of the population and we simulate our sampling procedure by resampling from the sample. That is, we take a random sample with replacement \( \mathcal{X}_{\text{boot},1} = \{(x_i, y_i) : i \in I\} \) where \( I \) is a multi-set (https://en.wikipedia.org/wiki/Multiset) of elements from \([n]\) of size \( n \). We recompute our estimated coefficients on \( \mathcal{X}_{\text{boot},1} \). Then we repeat independently for a desired number of replicates \( \mathcal{X}_{\text{boot},1}, \ldots, \mathcal{X}_{\text{boot},r} \). Plotting a histogram of the resulting coefficients gives some idea of the variability of our estimates.

We implement the bootstrap for linear regression in Julia next.
In [28]:
```python
function mmids_linregboot(A,b;replicates=10000)
    n,m = size(A)
    coeff_boot = zeros(Float64,m,replicates)
    for i =1:replicates
        resample = rand(1:n,n)
        Aboot = A[resample,:)  
        bboot = b[resample]
        coeff_boot[:,i] = ls_by_qr(Aboot,bboot)
    end
    return coeff_boot
end
```

Out[28]: mmids_linregboot (generic function with 1 method)

First, let's use a simple example from the lecture with a known ground truth.

In [29]:
```python
n, b0, b1 = 1000, -1, 1
x = LinRange(0,10,n)
y = b0 + b1*x + randn(n)
A = hcat(ones(n),x);
```

The estimated coefficients are the following.

In [30]:
```python
ls_by_qr(A,y)
```

Out[30]: 2-element Array{Float64,1}:
-0.9600306265441597
0.9960724718988153

Now we apply the bootstrap and plot histograms of the two coefficients.

In [31]:
```python
coeff_boot = mmids_linregboot(A,y);
```
We see in the histograms that the true coefficient values $-1$ and $1$ fall within the likely range.
We return to the Advertising dataset and apply the bootstrap.

```
In [34]:
n = length(sales)
A = reduce(hcat, [ones(n), TV, radio, newspaper])
ls_by_gr(A,sales)
```

```
Out[34]: 4-element Array{Float64,1}:
 2.938893694594307
 0.04576464545539755
 0.18853001691820379
-0.0010374930424762435
```

```
In [35]: coeff_boot = mmids_linregboot(A,sales);
```

Plotting a histogram of the coefficients corresponding to newspaper advertising shows that 0 is a plausible value, while it is not for TV advertising.

```
In [36]: histogram(coeff_boot[2,:], legend=false)
```

```
Out[36]:
```

![Histogram of coefficients for newspaper advertising](image)

```
In [36]: hist...
```

```
Out[36]:
```
In [37]: `histogram(coeff_boot[4,:], legend=false)`
Out[37]: