10-315 Notes - Optimization and Linear Regression

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[ ]:
```
# Bump up the default font size for matplotlib
plt.rcParams.update({'font.size': 10})
```

1 Optimization and Linear Regression

There is an incredible amount of ML that we can learn from just looking at $y = mx + b$. Well, of course, we also have to look at data as well as this linear model. But, seriously, mastering the details behind fitting this linear model to data will give us the backbone for so so much in this course and machine learning, including all the fanciest neural nets.

1.1 Table of Contents

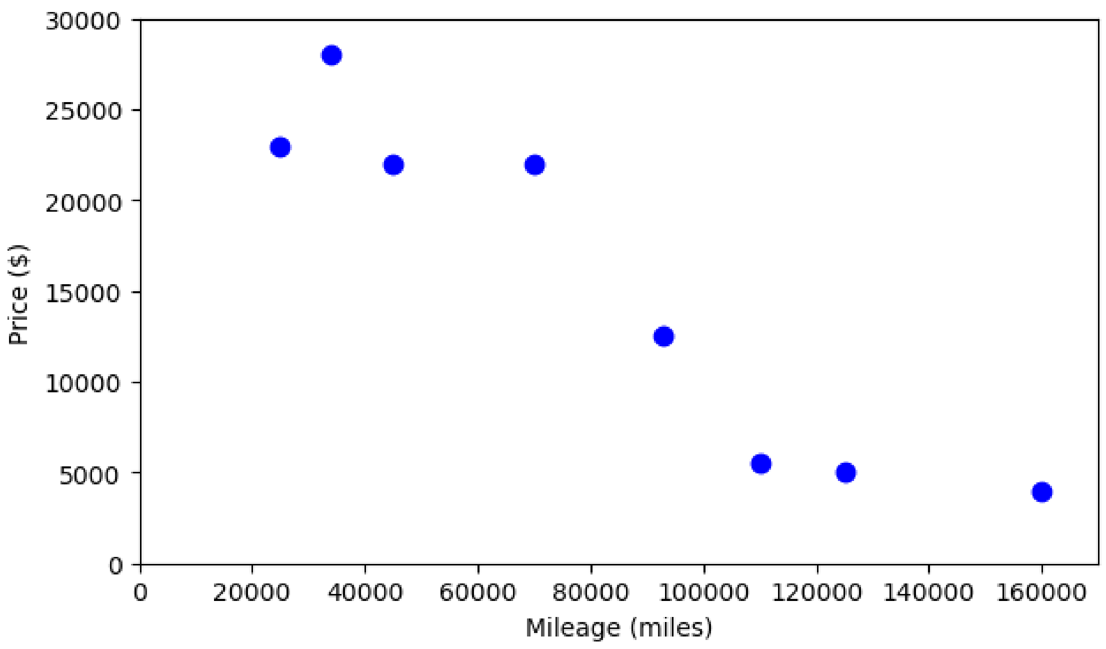
2 Selling my car

Suppose I'm trying to sell my sweet Pontiac Vibe to get some extra spending cash. I looked through some sites that sell cars online and grabbed a few data points, specifically the mileage and price for eight cars. I want to create an ML model (a **hypothesis function** h) that takes a car's mileage and predicts the selling price, $\hat{y} = h(x)$.

[ ]:
```
# Mileage values
x_train = np.array([25000, 34000, 45000, 70000, 93000, 110000, 125000, 160000])
y_train = np.array([23000, 28000, 22000, 22000, 12500, 5500, 5000, 4000])
```

[ ]:
```
def plot_points(x, y):
    plt.plot(x, y, 'b.', markersize=15, label="Measured")

    plt.xlim(0, 170000)
    plt.ylim(0, 30000)
    plt.xlabel('Mileage (miles)')
    plt.ylabel('Price ($)')
#     plt.locator_params(nbins=4)
```

[ ]:
```
plt.figure(figsize=(7,4))
plot_points(x_train, y_train)
```



2.1 Linear model

It looks like it might be reasonable to model this data with a linear1 function. Here is a linear hypthesis function with input variable x and parameters w and b, $h(x; w, b) = wx + b$. For now these are all scalars. The parameter w represents the slope of the line. We often use w in machine learning; this is the same as the m that you may have seen used for slope in algebra.

Notation alert: sometimes we'll separate arguments of a function with a semi-colon to highlight the input variables from parameters.

1Or affine. We'll talk more about linear and affine later.

[ ]:
```
def predict(x, w, b):
    return w*x + b
```

[ ]:
```
def plotPredictFunction(x_data, w, b,
                        x_min=0, x_max=200000):
```

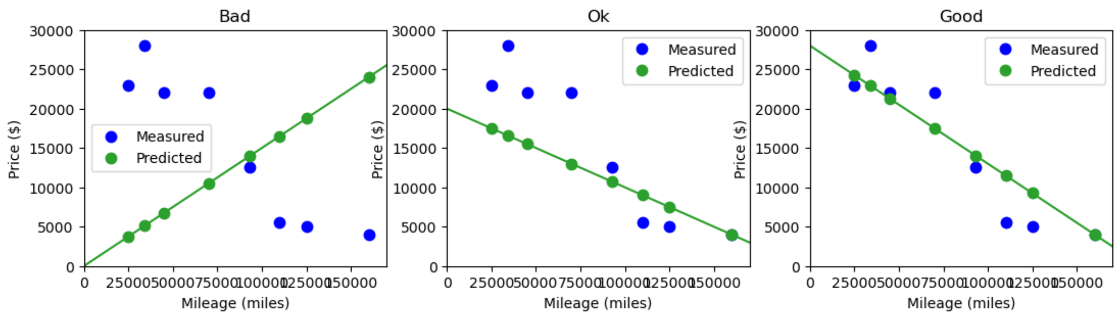
```
    # The way we implemented the predict function above, it happens to work when␣
↪we pass
    # it the numpy array with the x values for many data points, correctly␣
↪returning a
    # numpy array with the corresponding predicted y values for those data points

    y_pred = predict(x_data, w, b)

    # Create an evenly space array of x points that we can use to plot a line␣
↪for our
    # model
    x_grid = np.linspace(x_min, x_max, 100)
    y_grid_pred = predict(x_grid, w, b)

    plt.plot(x_grid, y_grid_pred, '-', color="tab:green")
    plt.plot(x_data, y_pred, '.', color="tab:green", markersize=15,␣
↪label="Predicted")
```

### Plotting three different models (three different sets of parameters)

We get to change values for **parameters** m and b to find the best line!

[ ]:
```
w_bad = 0.15
b_bad = 0

w_ok = -0.1
b_ok = 20000

w_good = -0.15
b_good = 28000

plt.figure(figsize=(13, 3))

plt.subplot(1, 3, 1)
plot_points(x_train, y_train)
plotPredictFunction(x_train, w_bad, b_bad)
plt.legend()
plt.title("Bad")

plt.subplot(1, 3, 2)
plot_points(x_train, y_train)
plotPredictFunction(x_train, w_ok, b_ok)
plt.legend()
plt.title("Ok")

plt.subplot(1, 3, 3)
plot_points(x_train, y_train)
plotPredictFunction(x_train, w_good, b_good)
```

```
plt.legend()
plt.title("Good");
```



The “ok” values for parameters w and b are certainly better than the “bad” ones, and “pretty good” visually seems to be better than “ok”. But, we can’t rely on eyeballing ML models to see if they are any good.

To allow a machine to find the best parameters settings for us, we need a quantitative to compare with model is best. A **performance measure**!!

2.2 Performance measure for regression

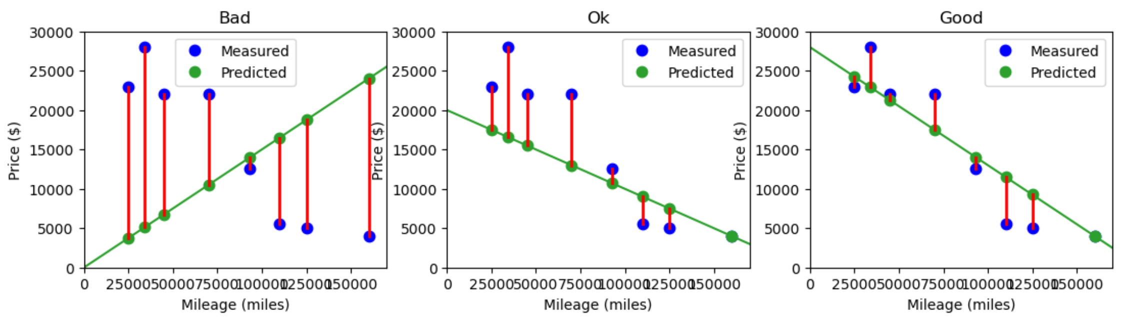
In regression, **error** is the difference between the predicted output and the measured output in our data. We do this for each data point:

$$\hat{y}^{(i)} = mx^{(i)} + b \tag{1}$$

$$error^{(i)} = y^{(i)} - \hat{y}^{(i)} \tag{2}$$

[ ]:
```
# Updated to include error lines
def plotPredictFunction(x_data, y_data, w, b,
                        x_min=0, x_max=200000):

    # The way we implemented the predict function above, it happens to work when␣
↪we pass
    # it the numpy array with the x values for many data points, correctly␣
↪returning a
    # numpy array with the corresponding predicted y values for those data points

    y_pred = predict(x_data, w, b)

    # Create an evenly space array of x points that we can use to plot a line␣
↪for our
    # model
    x_grid = np.linspace(x_min, x_max, 100)
    y_grid_pred = predict(x_grid, w, b)
    plt.plot(x_grid, y_grid_pred, '-', color="tab:green")
    plt.plot(x_data, y_pred, '.', color="tab:green", markersize=15,␣
↪label="Predicted")

    # Error lines
    plt.plot((x_data, x_data), (y_pred, y_data), 'r-', linewidth=2)
```

[ ]:
```
# Same as above but includine error lines

w_bad = 0.15
b_bad = 0

w_ok = -0.1
b_ok = 20000

w_good = -0.15
b_good = 28000

plt.figure(figsize=(13, 3))

plt.subplot(1, 3, 1)
plot_points(x_train, y_train)
plotPredictFunction(x_train, y_train, w_bad, b_bad)
plt.legend()
plt.title("Bad")

plt.subplot(1, 3, 2)
plot_points(x_train, y_train)
plotPredictFunction(x_train, y_train, w_ok, b_ok)
plt.legend()
plt.title("Ok")

plt.subplot(1, 3, 3)
plot_points(x_train, y_train)
plotPredictFunction(x_train, y_train, w_good, b_good)
plt.legend()
plt.title("Good");
```



But now we have $N = 8$ different error numbers for each model. We really need to get down to one number that represents the performance for each model.

- Attempt 1 (fail): $\sum_{i=1}^{N} error^{(i)}$, **sum of errors**
- Because the error values can be positive or negative, summing large positive and large negative will cancel out, making the "bad" model, left, actually have a very low sum.
- Attempt 2 (ok): $\frac{1}{N}\sum_{i=1}^{N} \left|error^{(i)}\right|$, **mean absolute error**
- Attempt 3 (preferred): $\frac{1}{N}\sum_{i=1}^{N} \left(error^{(i)}\right)^2$, **mean squared error (MSE)**
- We'll learn more about why this is preferred over mean absolute error in a few weeks
- Note: Sometimes we use root mean squared error (RMSE), the square root of MSE, to make this number match the scale of the output range.

[ ]:

```

```

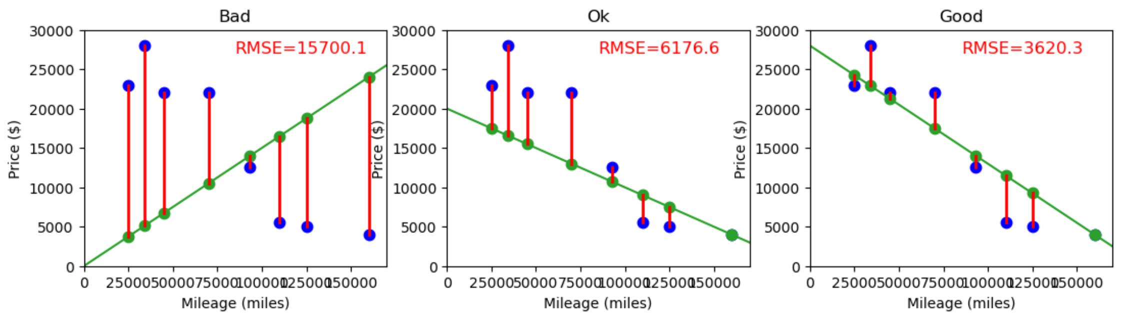
[ ]:

```
# Updated to include error lines and RMSE
def plotPredictFunction(x_data, y_data, w, b,
                        x_min=0, x_max=200000):

    # The way we implemented the predict function above, it happens to work when␣
↪we pass
    # it the numpy array with the x values for many data points, correctly␣
↪returning a
    # numpy array with the corresponding predicted y values for those data points

    y_pred = predict(x_data, w, b)

    # Create an evenly space array of x points that we can use to plot a line␣
↪for our
    # model
    x_grid = np.linspace(x_min, x_max, 100)
    y_grid_pred = predict(x_grid, w, b)

    plt.plot(x_grid, y_grid_pred, '-', color="tab:green")
    plt.plot(x_data, y_pred, '.', color="tab:green", markersize=15,␣
↪label="Predicted")

    # Error lines
    plt.plot((x_data, x_data), (y_pred, y_data), 'r-', linewidth=2)

    rmse = np.sqrt(np.mean((y_data - y_pred)**2))
    plt.text(85000, 27000, f"RMSE={rmse:.1f}", color='red', fontsize=12)
```

[ ]:

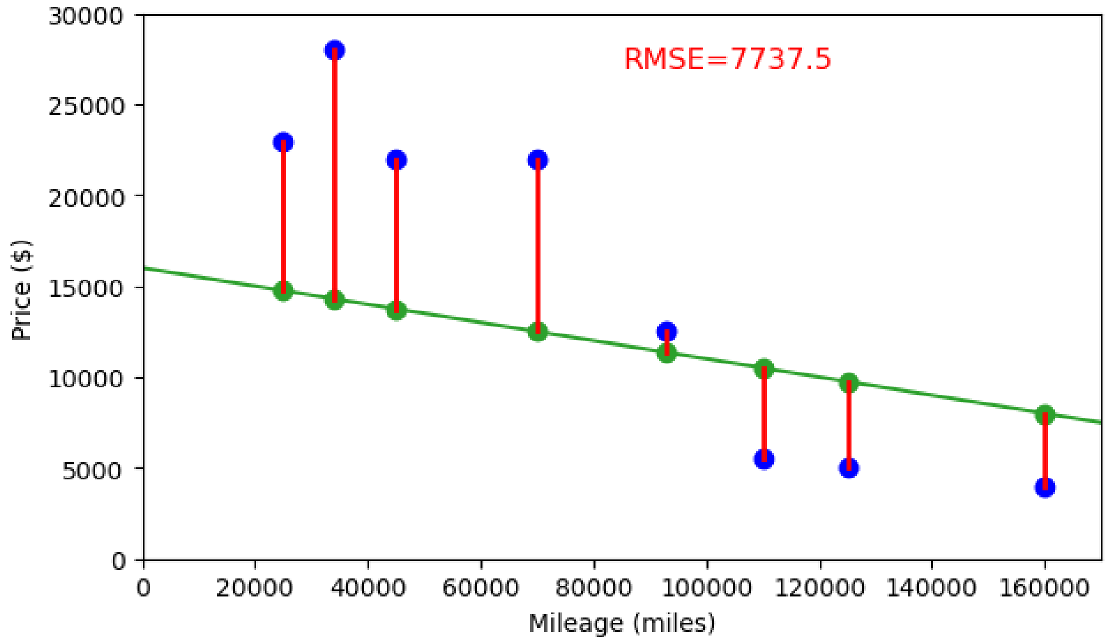
```
# Same as above but includine error lines

w_bad = 0.15
```

```
b_bad = 0

w_ok = -0.1
b_ok = 20000

w_good = -0.15
b_good = 28000

plt.figure(figsize=(13, 3))

plt.subplot(1, 3, 1)
plot_points(x_train, y_train)
plotPredictFunction(x_train, y_train, w_bad, b_bad)
plt.title("Bad")

plt.subplot(1, 3, 2)
plot_points(x_train, y_train)
plotPredictFunction(x_train, y_train, w_ok, b_ok)
plt.title("Ok")

plt.subplot(1, 3, 3)
plot_points(x_train, y_train)
plotPredictFunction(x_train, y_train, w_good, b_good)
plt.title("Good");
```



Now we can say that the "good" model is quantitatively better than the others, with a root mean squared error of 3,620.3.

```
[ ]: #@title Play with parameters { run: "auto"}
     w = -0.05 #@param {type:"slider", min:-0.5, max:0.5, step:0.01}
     b = 16000 #@param {type:"slider", min:-8000, max:40000, step:4000}

     plt.figure(figsize=(7,4))
     plot_points(x_train, y_train)
     plotPredictFunction(x_train, y_train, w, b)
```



2.3 Questions

Some questions you could certainly be asking:

- How do we find the parameters that give the *best* performance measure?

- What if my data doesn't fit in a line?

- What if my data has more than one input feature?

We'll get to these. But, first, let's talk through: 1. Math background 2. ML problem formulation and optimization notation 3. Optimization for linear regression

3 Math background

3.1 Math background topics

Linear (and affine) functions and geometry

Linear algebra

Multivariate calculus

Optimization notation

3.2 Linear (and affine) functions and geometry

3.2.1 Vocab / definitions:

linear combination: If $\mathbf{x}_1, ..., \mathbf{x}_K$ are vectors in $\mathbb{R}^M$, and $c_1, ..., c_K$ are scalars $\in \mathbb{R}$, then the resulting vector, $c_1\mathbf{x}_1 + ... + c_K\mathbf{x}_K \in \mathbb{R}^M$, is called a linear combination of the vectors $\mathbf{x}_1, ..., \mathbf{x}_M$ and $c_1, ..., c_K$ are called the coefficients of the linear combination.

linear function: a function $f : \mathbb{R}^M \to \mathbb{R}$ is a linear function, if for all vectors $\mathbf{x}, \mathbf{v} \in \mathbb{R}^M$ and scalars α and $\beta \in \mathbb{R}$, the property $f(\alpha\mathbf{x} + \beta\mathbf{v}) = \alpha f(\mathbf{x}) + \beta f(\mathbf{v})$ holds. If a function f is linear, this extends to linear combinations of any number of vectors, and not just linear combinations of two vectors.

affine function: (just saying linear sometimes includes affine) A linear function plus a constant is called an affine function.

3.2.2 Linear vs affine

What linear usually means depends on the domain!

Linear algebra

- Linear usually means strictly linear
- Linear: $\mathbf{y} = A\mathbf{x}$
- Affine: $\mathbf{y} = A\mathbf{x} + \mathbf{b}$

Geometry and algebra

- Linear usually means affine
- Line: $y = wx + b$
- Plane/hyperplane: $y = \mathbf{w}^T\mathbf{x} + b$

Vocab for linear (affine) geometry in various dimensions:

Note: $\mathbf{x}$ is the input variable vector

	$x \in \mathbb{R}$	$\mathbf{x} \in \mathbb{R}^2$	$\mathbf{x} \in \mathbb{R}^3$	$\mathbf{x} \in \mathbb{R}^M$
$y = \mathbf{w}^T\mathbf{x} + b$	line	plane	hyperplane	hyperplane
$\mathbf{w}^T\mathbf{x} + b = 0$	point	line	plane	hyperplane
$\mathbf{w}^T\mathbf{x} + b \geq 0$	half line	half plane	half space	half space

Machine learning Updated! In machine learning, the parameters are usually the variables and the data is considered constant. This makes us take a different look at our model equations.

$$f(x) = wx + b \text{ is affine when } x \text{ is the variable} \tag{3}$$
$$f(b, w) = wx + b \text{ is actually linear in the parameter variables } w,\ b \tag{4}$$

Similarly,

$$f(x) = w_2x^2 + w_1x + b \text{ is quadratic when } x \text{ is the variable} \tag{5}$$
$$f(b, w_1, w_2) = w_2x^2 + w_1x + b \text{ is linear in the parameter variables } w_1,\ w_2, \text{ and } b!! \tag{6}$$

We'll often use linear algebra to make these linear models more explicit and concise, where we put all of our parameters variables in a vector, $\boldsymbol{\theta}$.

For $\boldsymbol{\theta} = \begin{bmatrix} b \\ w \end{bmatrix}$ and $\mathbf{x} = \begin{bmatrix} 1 \\ x \end{bmatrix}$:

$$\begin{aligned} y &= wx + b && (7) \\ &= \begin{bmatrix} 1 & x \end{bmatrix} \begin{bmatrix} b \\ w \end{bmatrix} && (8) \\ &= \mathbf{x}^T\boldsymbol{\theta} && (9) \end{aligned}$$

For $\boldsymbol{\theta} = \begin{bmatrix} b \\ w_1 \\ w_2 \end{bmatrix}$ and $\mathbf{x} = \begin{bmatrix} 1 \\ x \\ x^2 \end{bmatrix}$

$$\begin{aligned} y &= w_2x^2 + w_1x + b && (10) \\ &= \begin{bmatrix} 1 & x & x^2 \end{bmatrix} \begin{bmatrix} b \\ w_1 \\ w_2 \end{bmatrix} && (11) \\ &= \mathbf{x}^T\boldsymbol{\theta} && (12) \end{aligned}$$

Again, these are linear because we are treating the data vectors, $\mathbf{x}$ as constant.

3.2.3 Linear algebra

Most of the linear algebra listed here should be prerequisite material for you. The exceptions might be vector and matrix norms and any notational changes.

Notation Matrix notation: $A \in \mathbb{R}^{N \times M}$:

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,M} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,M} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N,1} & a_{N,2} & \cdots & a_{N,M} \end{bmatrix}$$

Summation notation: Matrix multiplication with $A \in \mathbb{R}^{M \times K}, B \in \mathbb{R}^{K \times N}, C \in \mathbb{R}^{M \times N}$. If $C = AB$, then $C_{i,j} = \sum_{k=1}^{K} a_{i,k} b_{k,j}$. The number of columns in A must match the number of rows in B. C will have the same number of rows as A and the same number of columns as B.

Vector notation:

$\mathbf{u} \in \mathbb{R}^{M \times 1}$	Column vector of length M
$\mathbf{v} \in \mathbb{R}^{1 \times M}$	Row vector of length M
$\mathbf{z} \in \mathbb{R}^{M}$	Ambiguous. In this course, assume column vector unless stated otherwise.

$$\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_M \end{bmatrix}$$

Linear systems of equations **underdetermined**: If there are fewer equations than variables, the system is underdetermined and cannot have exactly 1 solution, it must have either infinitely many or no solutions.

overdetermined: A system with more equations than variables. An overdetermined system may have 1 solution, 0 solutions, or infinitely many solutions.

inconsistent: when the system of equations does not have a solution.

consistent: when the system of equations has at least one solution.

Vectors **dot product**: $\mathbf{a}^T\mathbf{b} = a_1b_1 + a_2b_2 + ... + a_Mb_M$ for two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^M$. It is the sum of the products of corresponding entries of the two vectors.

inner product (more general than dot product): is a way to multiply vectors, resulting in a scalar. Let $\mathbf{u}, \mathbf{v}, \mathbf{w}$ be vectors and let α be a scalar. Then, the inner product satisfies the following properties: $\backslash$ $\langle \mathbf{u} + \mathbf{v}, \mathbf{w} \rangle = \langle \mathbf{u}, \mathbf{w} \rangle + \langle \mathbf{v}, \mathbf{w} \rangle$ $\backslash$ $\langle \alpha\mathbf{v}, \mathbf{w} \rangle = \alpha \langle \mathbf{v}, \mathbf{w} \rangle$ $\backslash$ $\langle \mathbf{v}, \mathbf{w} \rangle = \langle \mathbf{w}, \mathbf{v} \rangle$ $\backslash$ $\langle \mathbf{v}, \mathbf{v} \rangle \geq 0$ and $\langle \mathbf{v}, \mathbf{v} \rangle = 0$ if and only if $\mathbf{v} = 0$

outer product: Outer product of vectors $\mathbf{u}$ and $\mathbf{v}$ is $Y = \mathbf{u} \otimes \mathbf{v} = \mathbf{u}\mathbf{v}^T$, and $Y_{i,j} = u_i v_j$.

magnitude: Magnitude (length) of vector $\mathbf{u}$ is $|\mathbf{u}| = \|\mathbf{u}\|_2 = \left(\sum_i u_i^2\right)^{\frac{1}{2}}$.

L2 norm: Also known as Euclidean norm $\|\mathbf{v}\|_2 = \left(\sum_i v_i^2\right)^{\frac{1}{2}} = \left(\mathbf{v}^T\mathbf{v}\right)^{\frac{1}{2}}$

L1 norm: The sum of absolute values of the entries of the vector $\|\mathbf{v}\|_1 = \sum_i |v_i|$

L0 "norm": Number of non-zero entries in a vector (not technically a norm) $\|\mathbf{v}\|_0 = \sum_i |v_i|^0$, where 0^0 is defined as being equal to zero.

p-norm: $\|\mathbf{v}\|_p = (\sum_i |v_i|^p)^{\frac{1}{p}}$ (Only a norm for $p \geq 1$).

span: Set of all linear combinations of a set of vectors. For example, given a set of vectors $\mathcal{S} = \{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\}, \mathbf{v_i} \in \mathbb{R}^M$, $span(\mathcal{S}) = \{\alpha_1\mathbf{v}_1 + \alpha_2\mathbf{v}_2 + \alpha_3\mathbf{v}_3 \mid \alpha_i \in \mathbb{R}\}$. Span is an example of a vector space.

vector space: Span of a set of vectors is an example of a vector space. Vector space is a more general term for the result of combining a set of vectors with addition and scalar multiplication. For example, if we changed span to include multiplication by complex scalars, that would be a different vector space.

linearly dependent: A vector, $\mathbf{u}$, is linearly dependent with a set of vectors, $\mathcal{S} = \{\mathbf{v}_k\}_{k=1}^K$, if it is possible to represent $\mathbf{u}$ as a linear combination of vectors in $\mathcal{S}$. For example, if $\mathbf{u} = 3\mathbf{v}_1 - 3.5\mathbf{v}_3$, then $\mathbf{u}$ is linearly dependent with $\mathcal{S}$.

A set of vectors in linearly dependent if any of the vectors in the set can be represented by a linear combination of the remaining vectors in the set.

linearly independent: A vector, $\mathbf{u}$ is linearly independent from a set of vectors, $\mathcal{S} = \{\mathbf{v}_k\}_{k=1}^K$, if it is not possible to represent $\mathbf{u}$ as a linear combination of vectors in $\mathcal{S}$.

A set of vectors is linearly independent if no single vector in the set can be represented by a linear combination of the remaining vectors in the set.

Matrices **identity matrix**: A matrix with all ones on the diagonal and zeros elsewhere. Represented as I, or more specifically I_N, where the N indicates that it is an $N \times N$ identity matrix.
$I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

matrix inverse: The inverse of matrix $A \in \mathbb{R}^{N \times N}$ is denoted A^{-1}. A^{-1} is also an $N \times N$ matrix. The inverse of a square, $N \times N$ matrix will exist if the matrix is full rank, i.e., the column rank and row rank is N. If the inverse of a square matrix exists then $A^{-1}A = AA^{-1} = I$, where I is the $N \times N$ identity matrix.

column rank of a matrix: the maximal number of linearly independent vectors among the column vectors in a given matrix. This is also the dimensionality of the vector space spanned by the column vectors of the matrix.

row rank of a matrix: the maximal number of linearly independent vectors among the row vectors in a given matrix. This is also the dimensionality of the vector space spanned by the row vectors of the matrix.

rank: the row rank and column rank of a matrix are always equal, so there are often just referred to as matrix "rank".

full rank: A matrix is full rank is the rank is equal to the minimum of its number of rows and number of columns. If a matrix is square and full rank then the inverse of that matrix exists.

singular matrix: A square matrix is singular if it is not full rank and thus it's inverse doesn't exist.

Frobenius norm of a matrix: Basically the L2 norm if we were to flatten the matrix into a vector. For matrix $A \in \mathbb{R}^{N \times M}$,

$$\|A\|_F = \left(\sum_{i=1}^{N} \sum_{j=1}^{M} {a_{i,j}}^2 \right)^{\frac{1}{2}}$$

$$\|A\|_F^2 = \sum_{i=1}^{N} \sum_{j=1}^{M} {a_{i,j}}^2$$

3.2.4 Multivariate calculus

While you may not have explicitly learned multivariate calculus, if very much just builds on top of normal (scalar) calculus. In multivariate calculus, we are just shifting to working with functions that have multiple inputs variables and potentially multiple outputs.

Partial derivative A **partial derivative** is when we take the derivative of a function f with respect to one of its many input variables. Notation-wise, you'll see it written as $\frac{\partial}{\partial z} f(x, z)$ or $\frac{\partial f}{\partial z}$. (It could also be written as $f_z(x, z)$, but we won't use that in this course.)

When we take the partial derivative with respect to one variable, we just hold all other variables constant.

For example:

$$\begin{aligned} f(x, z) &= 2x^3 z^5 && (13) \\ \frac{\partial f}{\partial x} &= 6x^2 z^5 && (14) \\ \frac{\partial f}{\partial z} &= 10x^3 z^4 && (15) \\ & && (16) \end{aligned}$$

You can think of linear algebra as having many individual variables. Take, for example, the L2 norm squared of $\mathbf{x} \in \mathbb{R}^3$:

$$\begin{aligned} f(\mathbf{x}) &= \|\mathbf{x}\|^2 = \mathbf{x}^T \mathbf{x} = \sum_i {x_i}^2 = {x_1}^2 + {x_2}^2 + {x_3}^2 && (17) \\ f(x_1, x_2, x_3) &= \|\mathbf{x}\|^2 = \mathbf{x}^T \mathbf{x} = \sum_i {x_i}^2 = {x_1}^2 + {x_2}^2 + {x_3}^2 && (18) \\ \frac{\partial f}{\partial x_1} &= 2x_1 && (19) \\ \frac{\partial f}{\partial x_2} &= 2x_2 && (20) \\ \frac{\partial f}{\partial x_3} &= 2x_3 && (21) \\ & && (22) \end{aligned}$$

Gradient Given a scalar function with vector input, $f : \mathbb{R}^M \to \mathbb{R}$, $f(\mathbf{x}) = f(x_1, ..., x_M)$, the **gradient** is a column vector where the i-th entry is the partial derivative of the function with respect to the i-th input entry in the input vector.

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_M} \end{bmatrix}$$

We call this the gradient of f with respect to $\mathbf{x}$. The $\mathbf{x}$ in the subscript is redundant if $\mathbf{x}$ is the only argument to f and would typically be dropped, $\nabla f(\mathbf{x})$.

Using the same example from above, the L2 norm squared of $\mathbf{x} \in \mathbb{R}^3$:

$$f(\mathbf{x}) = \|\mathbf{x}\|^2 = \begin{bmatrix} {x_1}^2 \\ {x_2}^2 \\ {x_3}^2 \end{bmatrix} \tag{23}$$

$$\frac{\partial f}{\partial x_1} = 2x_1 \tag{24}$$

$$\frac{\partial f}{\partial x_2} = 2x_2 \tag{25}$$

$$\frac{\partial f}{\partial x_3} = 2x_3 \tag{26}$$

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \frac{\partial f}{\partial x_3} \end{bmatrix} = \begin{bmatrix} 2x_1 \\ 2x_2 \\ 2x_3 \end{bmatrix} = 2\mathbf{x} \tag{27}$$

3.3 Optimization notation

For all possible values x in the set $\mathcal{X}$ and return the output of $f(x)$ that has the smallest value:

$$y^* = \min_{x \in \mathcal{X}}. f(x)$$

For all possible values x in the set $\mathcal{X}$ and return **the x corresponding to** the output of $f(x)$ that has the smallest value (i.e. return the argument, not the value of the function):

$$x^* = \underset{x \in \mathcal{X}}{\text{argmin}}. f(x)$$

For example:

$$y^* = \min_{x \in \mathbb{R}} 3(x-5)^2 - 200 \tag{28}$$

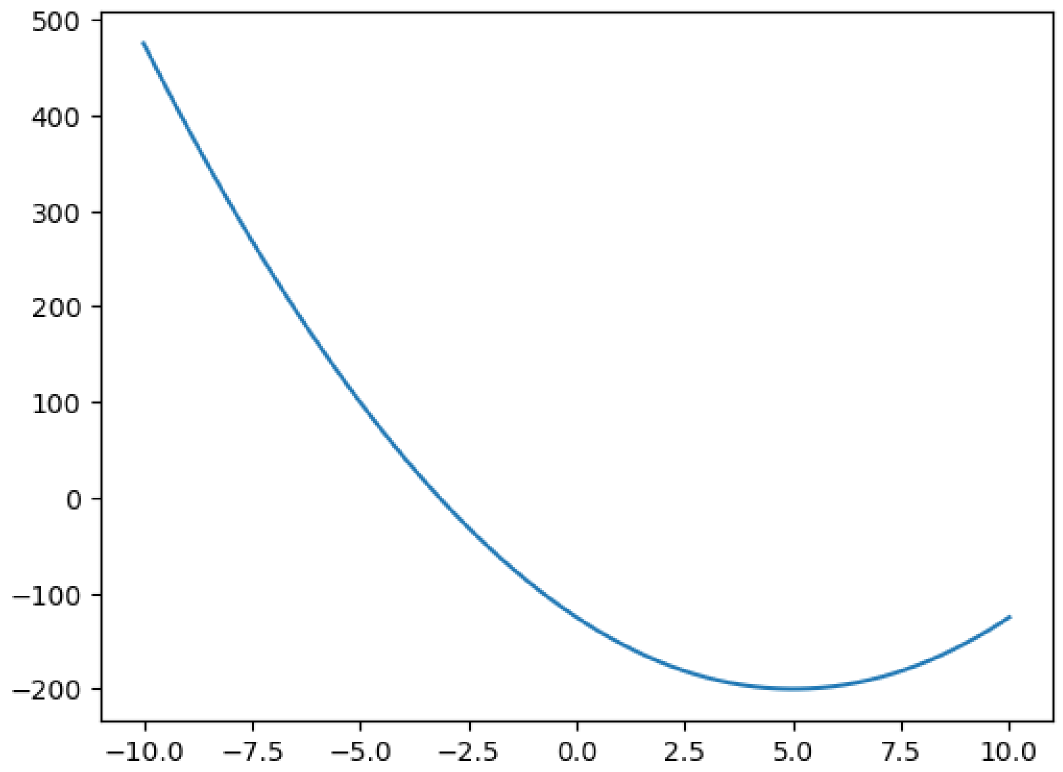
$$= -200 \tag{29}$$

$$x^* = \underset{x \in \mathbb{R}}{\text{argmin}}\ 3(x-5)^2 - 200 \tag{30}$$

$$= 5 \tag{31}$$

$$\tag{32}$$

[ ]:
```python
# Plot for above example
def f(x):
    return 3*(x-5)**2 - 200

x_grid = np.linspace(-10, 10, 100)
y_grid = f(x_grid)
plt.plot(x_grid, y_grid, '-');
```



4 ML problem formulation and notation

Ok, we're almost back to linear regression. We just have a few more machine learning formalities to define before we can stup up our linear regression optimization.

4.1 Loss

In machine learning, **loss** is a non-negative, scalar measure of how different a predicted output, $\hat{y}$, is from the corresponding true output (or the measured output in our dataset). The loss function, denoted $\ell(y, \hat{y})$, depends on both the type of machine learning problem (e.g. classification or regression) and the specific application. Note: the term **error** is also dependent on the type of problem and is very similar to loss as we'll see below.

4.1.1 Classification loss functions

For classification, **classification error** is when the predicted label doesn't match the measured label.

Zero-one loss is a common loss that simply returns zeros when the label match and one when they don't. $\ell(y, \hat{y}) = \mathbb{1}(y \neq \hat{y})$, where the indicator function, $\mathbb{1}(z)$, returns one when z is true and zero otherwise.

A more general classification loss function could put more weight on different types of label mismatches. For example, in medicine, there could be significant differences in the severity of making false positive versus false negative errors, where **false positive errors** are when the prediction is ill but the true output is healthy and **false netagive errors** are when the prediction is healthy but the true output is ill. This leads to an assymetric loss function, as in the following loss table:

$$\ell(y,\hat{y}) = \begin{array}{c|cc} & \hat{y}=0 & \hat{y}=1 \\ \hline y=0 & 0 & 2 \\ y=1 & 10 & 0 \end{array}$$

4.1.2 Regression loss functions

Squared error is commonly used for regression problems as the ouptput values are continuous and the size of the difference matters. The squared error loss function is $\ell(y,\hat{y}) = (y-\hat{y})^2$. Note: you may be correctly guessing that this is leading to our use of mean squared error.

4.2 Risk for ML models

4.2.1 Risk

The **risk** for ML hypothesis function h, $R(h)$, is the expected value of loss over distribution of input and output data represented by input and output random variable X and Y:

$$R(h) = \mathbb{E}_{X,Y}[\ell(Y,h(X))])$$

However, we basically never have sufficient models for the real world, so we do our best with empirical data samples, $\mathcal{D}$.

4.2.2 Empirical risk

However, we basically never have sufficient models for the real world, so we do our best with empirical data samples, $\mathcal{D} = \left\{\left(x^{(i)}, y^{(i)}\right)\right\}_{i=1}^{N}$. For a given dataset, $\hat{R}(h)$ is the empirical risk for the hypothesis function h:

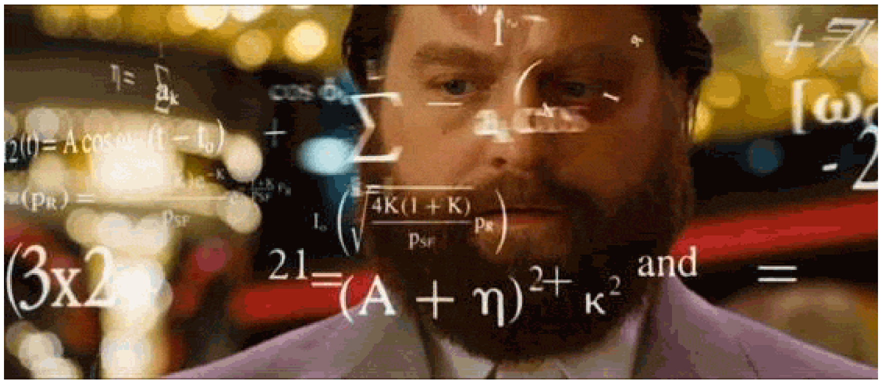
$$\hat{R}(h) = \frac{1}{N}\sum_{i=1}^{N} \ell\left(y^{(i)}, h\left(x^{(i)}\right)\right)$$

4.2.3 Empirical risk minimization

Optimization! Find the hypothesis function (ML model) that minimizes the empirical risk for a given dataset:

$$h^* = \underset{h\in\mathcal{H}}{\operatorname{argmin}}\ \hat{R}(h)$$

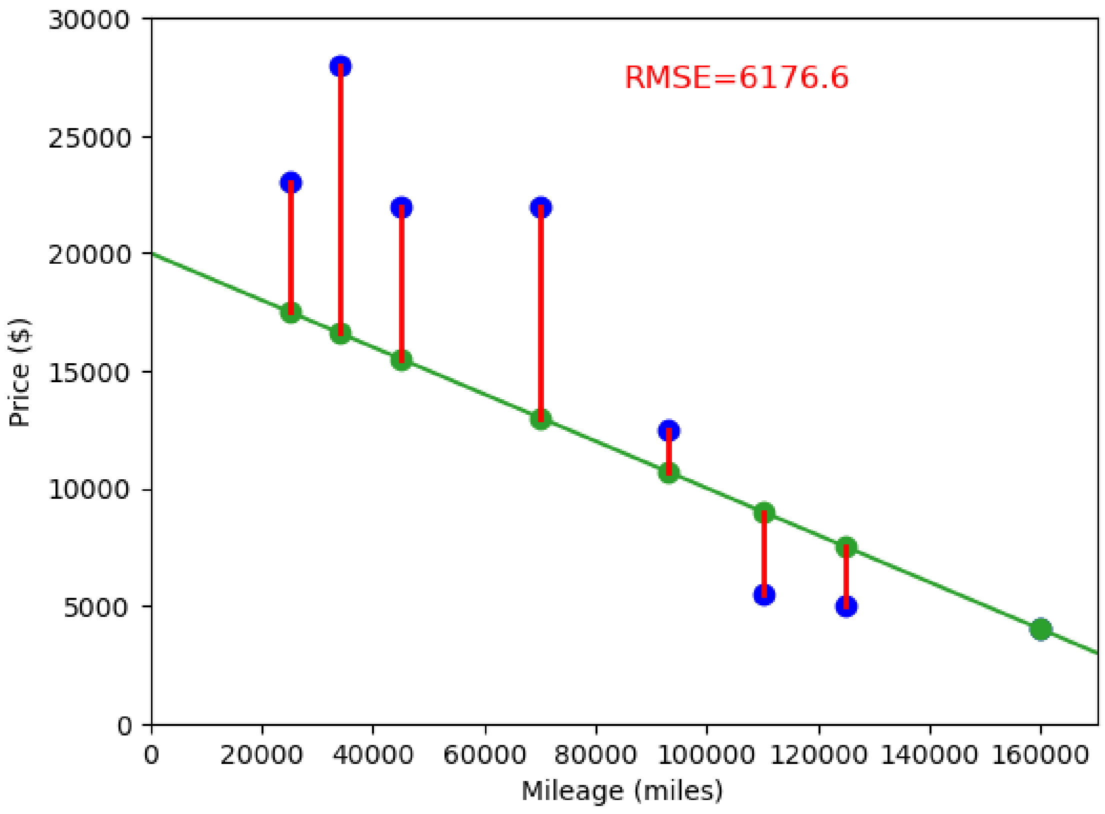
where $\mathcal{H}$ is the set of hypothesis functions that we are considering. Often $\operatorname{argmin}_{h\in\mathcal{H}}$ is replaced by $\operatorname{argmin}_\theta$ when we fix the structure of the hypothesis function (e.g., $h(x;w,b) = wx+b$) and then search over parameters values (e.g. w and b) to explore the space of linear hypothesis models, $\mathcal{H}$.



Holy cow, so much notation. This will be so much better when we apply this our simple linear regression problem!

5 Optimization for linear regression

```
[ ]: plot_points(x_train, y_train)
     plotPredictFunction(x_train, y_train, w_ok, b_ok)
```



[ ]:
```python
print("Training data:")
print("x:", x_train)
print("y:", y_train)
```

```
Training data:
x: [ 25000  34000  45000  70000  93000 110000 125000 160000]
y: [23000 28000 22000 22000 12500   5500   5000   4000]
```

5.1 Empircial risk minimization for linear regression model

5.1.1 Setup

Input: $x \in \mathbb{R}$

Output: $y \in \mathbb{R}$

Data:

$$\begin{aligned}\mathcal{D} = &\left\{\left(x^{(i)}, y^{(i)}\right)\right\}_{i=1}^{N=8} && (33)\\ =&(25000, 23000), (34000, 28000), (45000, 22000), (70000, 22000), && (34)\\ &(93000, 12500), (110000, 5500), (125000, 5000), (160000, 4000) && (35)\end{aligned}$$

Hypothesis function stucture: $h(x; w, b) = wx + b$

Parameters: w, b

Hypothesis space, $\mathcal{H}$: The set of all 1-D affine functions parameterized by w and b.

Loss function: Squared loss $\ell(y, \hat{y}) = (y - \hat{y})^2$

5.1.2 Optimization formulation

Empricial risk:

$$\begin{aligned}\hat{R}(h) &= \frac{1}{N}\sum_{i=1}^{N} \ell\left(y^{(i)}, h\left(x^{(i)}\right)\right) && (36)\\ &= \frac{1}{N}\sum_{i=1}^{N} \left(y^{(i)} - h\left(x^{(i)}\right)\right)^2 \quad \text{<— Mean squared error!} && (37)\\ &= \frac{1}{N}\sum_{i=1}^{N} \left(y^{(i)} - \left(wx^{(i)} + b\right)\right)^2 && (38)\\ & && (39)\end{aligned}$$

Empirical risk minimzation:

$$\begin{aligned}h^* &= \operatorname*{argmin}_{h \in \mathcal{H}} \hat{R}(h) && (40)\\ w^*, b^* &= \operatorname*{argmin}_{w \in \mathbb{R},\, b \in \mathbb{R}} \frac{1}{N}\sum_{i=1}^{N} \left(y^{(i)} - \left(wx^{(i)} + b\right)\right)^2 && (41)\\ & && (42)\end{aligned}$$

5.1.3 Optimization formulation (linear algebra version 1)

Parameters: $\boldsymbol{\theta} = \begin{bmatrix} b \\ w \end{bmatrix}$; Input data points: $\mathbf{x}^{(i)} = \begin{bmatrix} 1 \\ x^{(i)} \end{bmatrix}$; Ouput data points are still: $y^{(i)}$

Empricial risk:

$$\hat{R}(h) = \frac{1}{N} \sum_{i=1}^{N} \left(y^{(i)} - \mathbf{x}^{(i)T} \boldsymbol{\theta} \right)^2 \tag{43}$$

$$\tag{44}$$

Empirical risk minimzation:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta} \in \mathbb{R}^2}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} \left(y^{(i)} - \mathbf{x}^{(i)T} \boldsymbol{\theta} \right)^2 \tag{45}$$

$$\tag{46}$$

5.1.4 More to come this week in lecture!

In class we'll work on:

- Optimization formulation: linear algebra version 2 (utilizing the L2 norm)

- Methods to solve this optimization and find the optimal parameters

- Answering our questions about non-linear data and data with more than one input feature