Questions

- K-fold cross validation
  - K chunks of the data
  - Train how many times?
CHECKPOINT Questions

- **K-fold cross validation**
  - K chunks of the data
  - Train how many times?
  - K times!
  - Cross-validation error is the average of the validation errors over K folds
Clever idea: Use many small validation sets without losing too much training data.

Still need to break off our test set like before. After doing so, break the training set into chunks.

```
<table>
<thead>
<tr>
<th>Train</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chunk1</td>
<td></td>
</tr>
<tr>
<td>Chunk2</td>
<td></td>
</tr>
<tr>
<td>Chunk3</td>
<td>Chnk4</td>
</tr>
<tr>
<td>Test</td>
<td></td>
</tr>
</tbody>
</table>
```

For a given model complexity, train it times. Each time use all but one chunk and use that left out chunk to determine the validation error.

80-20 split. Take the 80% split and do a k-fold cross-validation on it.
Cross-Validation

The process generally goes

chunk_1, ..., chunk_k, test = split_data(dataset)

for each model complexity p:
  for i in [1, k]:
    model = train_model(model_p, chunks - i)
    val_err = error(model, chunk_i)
    avg_val_err = average val_err over chunks
  keep track of p with smallest avg_val_err
return model trained on train with best p + error(model, test)
K-fold cross validation

- K chunks of the data
- Train how many times?
- K times!
- Cross-validation error is the average of the validation errors over K folds
- E.g. K = 4 (split train data into 4)
  - Train on Chunk 1,2,3 and validate on 4 -> val1
  - Train on Chunk 1,3,4 and validate on 2 -> val2
  - Train on Chunk 1,2,4 and validate on 3 -> val3
  - Train on Chunk 2,3,4 and validate on 1 -> val4
  - Cross validation error - (val1 + val2 + val3 + val4)/4

- For M models - M*K trainings need to happen!
Validation vs Test Set - When to use what??
- Training done on train data
- Test error is an approximation of the true error
- So why do we need the validation data set?
- Validation data used to pick the right model

Generalization error
- Measures performance on unseen data
- What is the unseen data here?
- Why unseen data?
- Shopping example
Q2 - Using the regularizer
- Impact of regularizer on bias/model-complexity?
- What happens when you cube the weights and sum them up?
- Same as what happens when you sum the weights!!
Q2 - Using the regularizer

- Impact of regularizer on bias/model-complexity?
- What happens when you cube the weights and sum them up?
- Same as what happens when you sum the weights!!
- What parameters minimize the sum of weights?
- What parameters minimize the sum of squares of weights?
Notations

- **What? w-hat?**
  - Hat notation used for predicted parameters/coefficients

- **Weights vs coefficients vs parameters**

- **Input data vs features**

- **Regression vs Regularization**
  - Regularization is an objective function added on to regular regression that reduces over-fitting of the model!
  - Lambda needs to be picked - How?
How should we choose the best value of $\lambda$?

- Pick the $\lambda$ that has the smallest $RSS(\hat{\omega})$ on the training set
- Pick the $\lambda$ that has the smallest $RSS(\hat{\omega})$ on the test set
- Pick the $\lambda$ that has the smallest $RSS(\hat{\omega})$ on the validation set
- Pick the $\lambda$ that has the smallest $RSS(\hat{\omega}) + \lambda \| \hat{\omega} \|_2^2$ on the training set
- Pick the $\lambda$ that has the smallest $RSS(\hat{\omega}) + \lambda \| \hat{\omega} \|_2^2$ on the test set
- Pick the $\lambda$ that has the smallest $RSS(\hat{\omega}) + \lambda \| \hat{\omega} \|_2^2$ on the validation set
- Pick the $\lambda$ that results in the smallest coefficients
- Pick the $\lambda$ that results in the largest coefficients
- None of the above
RECAP

- Ridge Regression
- Choosing lambda
- What does high/low lambda mean?
For any particular setting of $\lambda$, use Ridge Regression objective

$$\hat{w}_{ridge} = \min_w RSS(w) + \lambda \|w_{1:D}\|_2^2$$

If $\lambda$ is too small, will overfit to training set. Too large, $\hat{w}_{ridge} = 0$.

How do we choose the right value of $\lambda$? We want the one that will do best on future data. This means we want to minimize error on the validation set.

Don’t need to minimize $RSS(w) + \lambda \|w_{1:D}\|_2^2$ on validation because you can’t overfit to the validation data (you never train on it).

Another argument is that it doesn’t make sense to compare those values for different settings of $\lambda$. They are in different “units” in some sense.
Choosing $\lambda$

The process for selecting $\lambda$ is exactly the same as we saw with using a validation set or using cross validation.

for $\lambda$ in $\lambda$s:

Train a model using using Gradient Descent

$$\hat{\mathbf{w}}_{\text{ridge}}(\lambda) = \min_{\mathbf{w}} \text{RSS}_{\text{train}}(\mathbf{w}) + \lambda \|\mathbf{w}_{1:D}\|_2^2$$

Compute validation error

$$\text{validation\_error} = \text{RSS}_{\text{val}}(\hat{\mathbf{w}}_{\text{ridge}}(\lambda))$$

Track $\lambda$ with smallest $\text{validation\_error}$

Return $\lambda^*$ & estimated future error $\text{RSS}_{\text{test}}(\hat{\mathbf{w}}_{\text{ridge}}(\lambda^*))$

There is no fear of overfitting to validation set since you never trained on it! You can just worry about error when you aren’t worried about overfitting to the data.
What exactly is Gradient Descent??
What exactly is Gradient Descent??
Moving in the direction of negative gradient
Gradient Descent

- What exactly is Gradient Descent??
- Moving in the direction of negative gradient
GRADIENT DESCENT
Consider the quadratic function in the previous figure. Which direction (left or right) does the gradient point to for a) a point on the left side of the global minimum and for b) a point on the right side of the global minimum

a) Left, Right
b) Right, Right
c) Left, Left
d) Right, Left
What exactly is Gradient Descent??
Moving in the direction of negative gradient
Gradient is a generalization of slope
Gradient Descent - 2 dimensions
If you have K features and a linear regression model fit to these K features - What will be the dimension of the gradient?
Gradient dimensions

- If you have K features and a linear regression model fit to these K features - What will be the dimension of the gradient?
- K+1
- Gradient is a vector (generalization of slope which is a scalar)
Gradient Descent - Non-convex functions
Feature Selection and All Subsets
Why do we care about selecting features? Why not use them all?

**Complexity**
Models with too many features are more complex. Might overfit!

**Interpretability**
Can help us identify which features carry more information.

**Efficiency**
Imagine if we had MANY features (e.g. DNA). $\mathbf{w}$ could have $10^{11}$ coefficients. Evaluating $\hat{y} = \mathbf{w}^T h(x)$ would be very slow!

If $\mathbf{w}$ is **sparse**, only need to look at the non-zero coefficients

$$\hat{y} = \sum_{\hat{w}_j \neq 0} \hat{w}_j h_j(x)$$
Might have many features to potentially use. Which are useful?

<table>
<thead>
<tr>
<th>Lot size</th>
<th>Dishwasher</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Family</td>
<td>Garbage disposal</td>
</tr>
<tr>
<td>Year built</td>
<td>Microwave</td>
</tr>
<tr>
<td>Last sold price</td>
<td>Range / Oven</td>
</tr>
<tr>
<td>Last sale price/sqft</td>
<td>Refrigerator</td>
</tr>
<tr>
<td>Finished sqft</td>
<td>Washer</td>
</tr>
<tr>
<td>Unfinished sqft</td>
<td>Dryer</td>
</tr>
<tr>
<td>Finished basement sqft</td>
<td>Laundry location</td>
</tr>
<tr>
<td># floors</td>
<td>Heating type</td>
</tr>
<tr>
<td>Flooring types</td>
<td>Jetted Tub</td>
</tr>
<tr>
<td>Parking type</td>
<td>Deck</td>
</tr>
<tr>
<td>Parking amount</td>
<td>Fenced Yard</td>
</tr>
<tr>
<td>Cooling</td>
<td>Lawn</td>
</tr>
<tr>
<td>Heating</td>
<td>Garden</td>
</tr>
<tr>
<td>Exterior materials</td>
<td>Sprinkler System</td>
</tr>
<tr>
<td>Roof type</td>
<td></td>
</tr>
<tr>
<td>Structure style</td>
<td></td>
</tr>
</tbody>
</table>

...
How happy are you? What part of the brain controls happiness?
Best Model
Size 0

Features
# bathrooms
# bedrooms
sq.ft. living
sq.ft lot
floors
year built
year renovated
waterfront

Features
# bathrooms
# bedrooms
sq.ft. living
sq.ft lot
floors
year built
year renovated
waterfront

Note only:
y_i = 0
Best Model
Size 1

Features
- # bathrooms
- # bedrooms
- sq.ft. living
- sq.ft lot
- floors
- year built
- year renovated
- waterfront

\[ R_{SS_{train}}(\hat{\omega}) \]

# of features
Best Model

Size 1

$\mathbf{RSS}_{train} (\hat{\mathbf{w}})$

# of features

Features

- # bathrooms
- # bedrooms
- sq.ft. living
- sq.ft lot
- floors
- year built
- year renovated
- waterfront
Best Model
Size 1

Features
# bathrooms
# bedrooms
sq.ft. living
sq.ft. lot
floors
year built
year renovated
waterfront

RSS_{train} (\hat{\mathbf{w}})

# of features
Best Model
Size 1

Features
# bathrooms
# bedrooms
sq.ft. living
sq.ft lot
floors
year built
year renovated
waterfront
Best Model
Size 1

Features
- # bathrooms
- # bedrooms
- sq.ft. living
- sq.ft lot
- floors
- year built
- year renovated
- waterfront

\[ \text{RSS}_{\text{train}}(\hat{w}) \]

# of features
Best Model
Size 1

Features
- # bathrooms
- # bedrooms
- sq.ft. living
- sq.ft lot
- floors
- year built
- year renovated
- waterfront

\[ RSS_{\text{train}}(\hat{\omega}) \]

# of features
Best Model
Size 1

Features
# bathrooms
# bedrooms
sq.ft. living
sq.ft lot
floors
year built
year renovated
waterfront

$RSS_{train}(\hat{\omega})$

# of features
Best Model
Size 1

Features
- # bathrooms
- # bedrooms
- sq.ft. living
- sq.ft lot
- floors
- year built
- year renovated
- waterfront

\[RSS_{\text{train}}(\hat{w})\]
Best Model
Size 1

Features
# bathrooms
# bedrooms
sq.ft. living
sq.ft lot
floors
year built
year renovated
waterfront

\[ RSS_{\text{train}}(\hat{w}) \]

# of features
Best Model
Size 2

Not necessarily nested!
Best Model – Size 1: sq.ft living
Best Model – Size 2:
# bathrooms & # bedrooms

Features
# bathrooms
# bedrooms
sq.ft. living
sq.ft lot
floors
year built
year renovated
waterfront

\[ \text{RSS}_{\text{train}}(\hat{\mathbf{w}}) \]

# of features
Best Model
Size 3

Features
- # bathrooms
- # bedrooms
- sq.ft. living
- sq.ft lot
- floors
- year built
- year renovated
- waterfront

$R_{\text{RSS}_{\text{train}}}(\hat{w})$

# of features
Best Model
Size 4

Features
# bathrooms
# bedrooms
sq.ft. living
sq.ft lot
floors
year built
year renovated
waterfront

$R_{\text{train}}(\hat{w})$

0 1 2 3 4

# of features
Best Model
Size 5

Features
# bathrooms
# bedrooms
sq.ft. living
sq.ft lot
floors
year built
year renovated
waterfront

$\mathcal{RSS}_{\text{train}}(\hat{\omega})$

# of features

0 1 2 3 4 5
Best Model
Size 6

Features
# bathrooms
# bedrooms
sq.ft. living
sq.ft lot
floors
year built
year renovated
waterfront
Best Model
Size 7

Features
- # bathrooms
- # bedrooms
- sq.ft. living
- sq.ft lot
- floors
- year built
- year renovated
- waterfront

\[ \text{RSS}_{\text{train}}(\hat{\theta}) \]

\[ \begin{align*}
0 & & 1 & & 2 & & 3 & & 4 & & 5 & & 6 & & 7 \\
\end{align*} \]

# of features
Best Model
Size 8

Features
# bathrooms
# bedrooms
sq.ft. living
sq.ft lot
floors
year built
year renovated
waterfront

![Graph showing the decrease in $\text{RSS}_{\text{train}}$ with increasing number of features.]
Choose Num Features?

Option 1
Assess on a validation set

Option 2
Cross validation

Option 3+
Other metrics for penalizing model complexity like BIC
Efficiency of All Subsets

How many models did we evaluate?

\[
\hat{y}_i = \epsilon_i \\
\hat{y}_i = w_0 h_0(x) + \epsilon_i \\
\hat{y}_i = w_1 h_1(x) + \epsilon_i \\
\ldots \\
\hat{y}_i = w_0 h_0(x) + w_1 h_1(x) + \epsilon_i \\
\ldots \\
\hat{y}_i = w_0 h_0(x) + w_1 h_1(x) + \ldots + w_D h_D(x) + \epsilon_i
\]

If evaluating all subsets of 8 features only took 5 seconds, then

- 16 features would take 21 minutes
- 32 features would take almost 3 years
- 100 features would take almost 7.5*10^{20} years
  - 50,000,000,000x longer than the age of the universe!
Knowing it’s impossible to find exact solution, approximate it!

**Forward stepwise**
Start from model with no features, iteratively add features as performance improves.

**Backward stepwise**
Start with a full model and iteratively remove features that are the least useful.

**Combining forward and backwards steps**
Do a forward greedy algorithm that eventually prunes features that are no longer as relevant

*And many many more!*
Example Greedy Algorithm

Start by selecting number of features $k$

\[
\begin{align*}
S_0 & \leftarrow \{\} \\
\text{for } i & \leftarrow 1..k: \\
\quad & \text{Find feature } f_i \text{ not in } S_{i-1}, \text{ that when combined with } S_{i-1}, \text{ minimizes the loss the most.} \\
\quad & S_i \leftarrow S_{i-1} \cup \{f_i\} \\
\text{Return } S_k
\end{align*}
\]

Called greedy because it makes choices that look best at the time.
GREEDY ALGORITHM

How many times would we need to train the model to follow the greedy forward procedure assuming there are N possible features and we stop the algorithm at K features?

a) O(K^2)
b) O(N^2)
c) O(NK)
d) O(NK^2)
e) O(N^2K)
Option 2
Regularization
Recap: Regularization

Before, we used the quality metric that minimized loss
\[ \hat{w} = \min_w L(w) \]

Change quality metric to balance loss with measure of overfitting
- \( L(w) \) is the measure of fit
- \( R(w) \) measures the magnitude of coefficients

\[ \hat{w} = \min_w L(w) + R(w) \]

How do we actually measure the magnitude of coefficients?
Recap: Magnitude

Come up with some number that summarizes the magnitude of the coefficients in $w$.

**Sum?**

$$R(w) = \sum_{j=0}^{D} w_j$$

**Sum of absolute values?**

$$R(w) = \sum_{j=0}^{D} |w_j| \triangleq \|w\|_1$$

$L_1$ norm (come back Wed!)

**Sum of squares?**

$$R(w) = \sum_{j=0}^{D} w_j^2 \triangleq \|w\|_2$$

$L_2$ norm (today)
We saw that Ridge Regression shrinks coefficients, but they don’t become 0. What if we remove weights that are sufficiently small?
Ridge for Feature Selection

Instead of searching over a **discrete** set of solutions, use regularization to reduce coefficient of unhelpful features.

Start with a full model, and then “shrink” ridge coefficients near 0. Non-zero coefficients would be considered selected as important.
Look at two related features #bathrooms and # showers. Our model ended up not choosing any features about bathrooms!
What if we had originally removed the # showers feature?

- The coefficient for # bathrooms would be larger since it wasn’t “split up” amongst two correlated features.
- Instead, it would be nice if there were a regularizer that favors sparse solutions in the first place to account for this…
LASSO Regression

Change quality metric to minimize

\[ \ell_1 \text{ norm: } \| w \|_1 = \sum_{j=0}^{D} |w_j| \]

is a tuning parameter that changes how much the model cares about the regularization term.

What if?

\[ \hat{w} = \min w \, \text{RSS}(w) \quad \Rightarrow \quad \hat{w}_{\text{LASSO}} \]

What if?

\[ \hat{w} = 0 \]

in between?

\[ 0 \leq \| \hat{w}_{\text{LASSO}} \|_1 \leq \| w \|_1 \]
Ridge Coefficient Paths

many small, but non-zero coeffs.
LASSO Coefficient Paths

Sparse = many 0 coefs.
Example from Google’s Machine Learning Crash Course
There is no poll to answer for this question. This is an open-ended question.

Why might the shape of the L1 penalty cause more sparsity than the L2 penalty?
When using the L1 Norm ($\| \cdot \|_1$) as a regularizer, it favors solutions that are **sparse**. Sparsity for regression means many of the learned coefficients are 0.

This has to do with the shape of the norm

When $\| \cdot \|_1$ is small, it is VERY small!
Another way to visualize why LASSO prefers sparse solutions
Sparsity
Geometry
Brain Break
Choosing $\lambda$

Exactly the same as Ridge Regression :) 

This will be true for almost every hyper-parameter we talk about

A hyper-parameter is a parameter you specify for the model that influences which parameters (e.g. coefficients) are learned by the ML algorithm

Hyperparameter tuning:

for each setting of HPs:
train model with current HP setting
validate predictor w/ validation set / cross-val
track HP w/ lowest val error
return best HP and estimate of test error.
A very common usage of LASSO is in feature selection. If you have a model with potentially many features you want to explore, you can use LASSO on a model with all the features and choose the appropriate to get the right complexity.

Then once you find the non-zero coefficients, you can identify which features are the most important to the task at hand*
De-biasing LASSO

LASSO adds bias to the Least Squares solution (this was intended to avoid the variance that leads to overfitting)

▪ Recall Bias-Variance Tradeoff

It’s possible to try to remove the bias from the LASSO solution using the following steps

1. Run LASSO to select which features should be used (those with non-zero coefficients)
2. Run regular Ordinary Least Squares on the dataset with only those features

Coefficients are no longer shrunk from their true values
1. Within a group of highly correlated features (e.g. # bathroom and # showers), LASSO tends to select amongst them arbitrarily.
   - Maybe it would be better to select them all together?
2. Often, empirically Ridge tends to have better predictive performance

**Elastic Net** aims to address these issues

\[
\hat{w}_{\text{ElasticNet}} = \min_w \text{RSS}(w) + \lambda_1 \|w\|_1 + \lambda_2 \|w\|_2^2
\]

Combines both to achieve best of both worlds!
Be careful when interpreting results of feature selection or feature importances in Machine Learning!

- Selection only considers features included
- Sensitive to correlations between features
- Results depend on the algorithm used!
Theme: Use regularization to do feature selection

Ideas:

- Describe “all subsets” approach to feature selection and why it’s impractical to implement.
- Formulate LASSO objective
- Describe how LASSO coefficients change as hyper-parameter is varied
- Interpret LASSO coefficient path plot
- Compare and contrast LASSO and ridge