Overfitting and Regularization

March 21, 2019
Data Science CSCI 1951A
Brown University
Instructor: Ellie Pavlick
HTAs: Wennie Zhang, Maulik Dang, Gurnaaz Kaur

Announcements

- MapReduce…the final announcement
- Final Project Schedule…sry
- Blog Post and other final project grading
- Mid-Semester feedback form

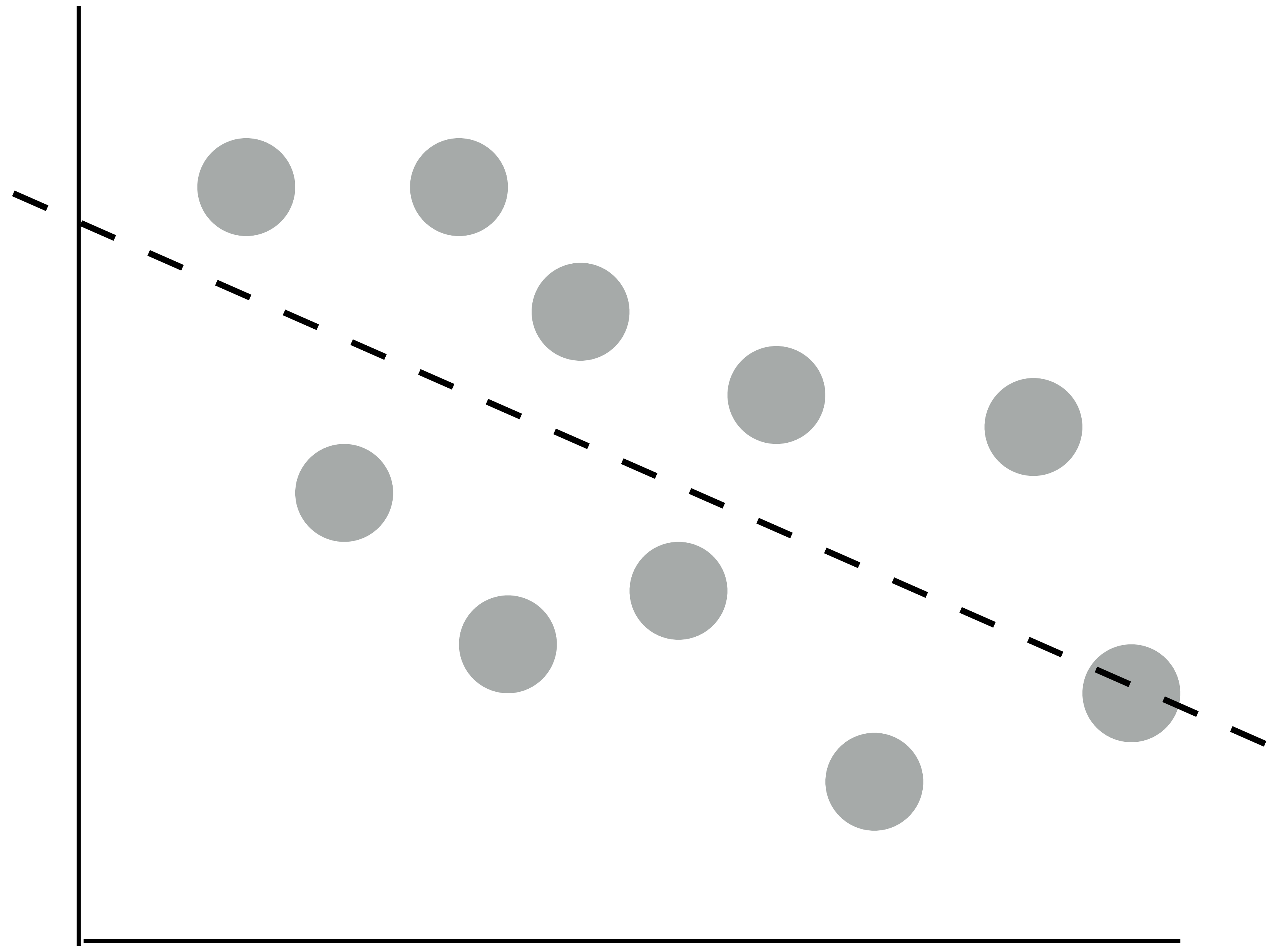
Today

- Overfitting and Regularization
- All “live demo” style
- Following slides contain fairly uninformative bullet points, those of you not present, please watch video/see ipython notebook for useful content :)

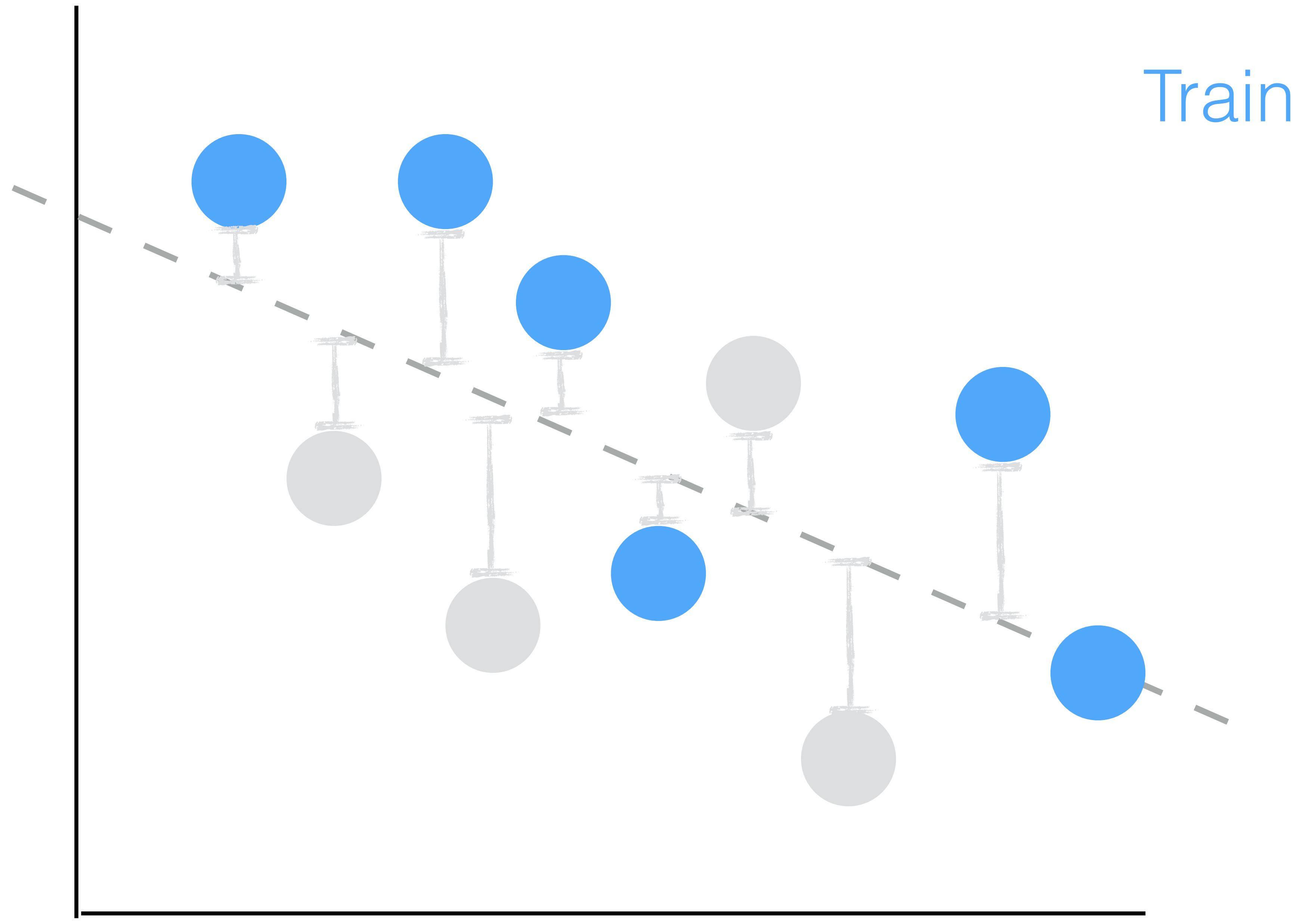
Train/Test Splits

- By definition, trained models are minimizing their objective for the data they see, but not for the data they don’t see
- What we really care about is how the model does on data we don’t see
- So we split our training data into disjoin sets—a train set and a test set—and assess performance on test given parameters set using train.

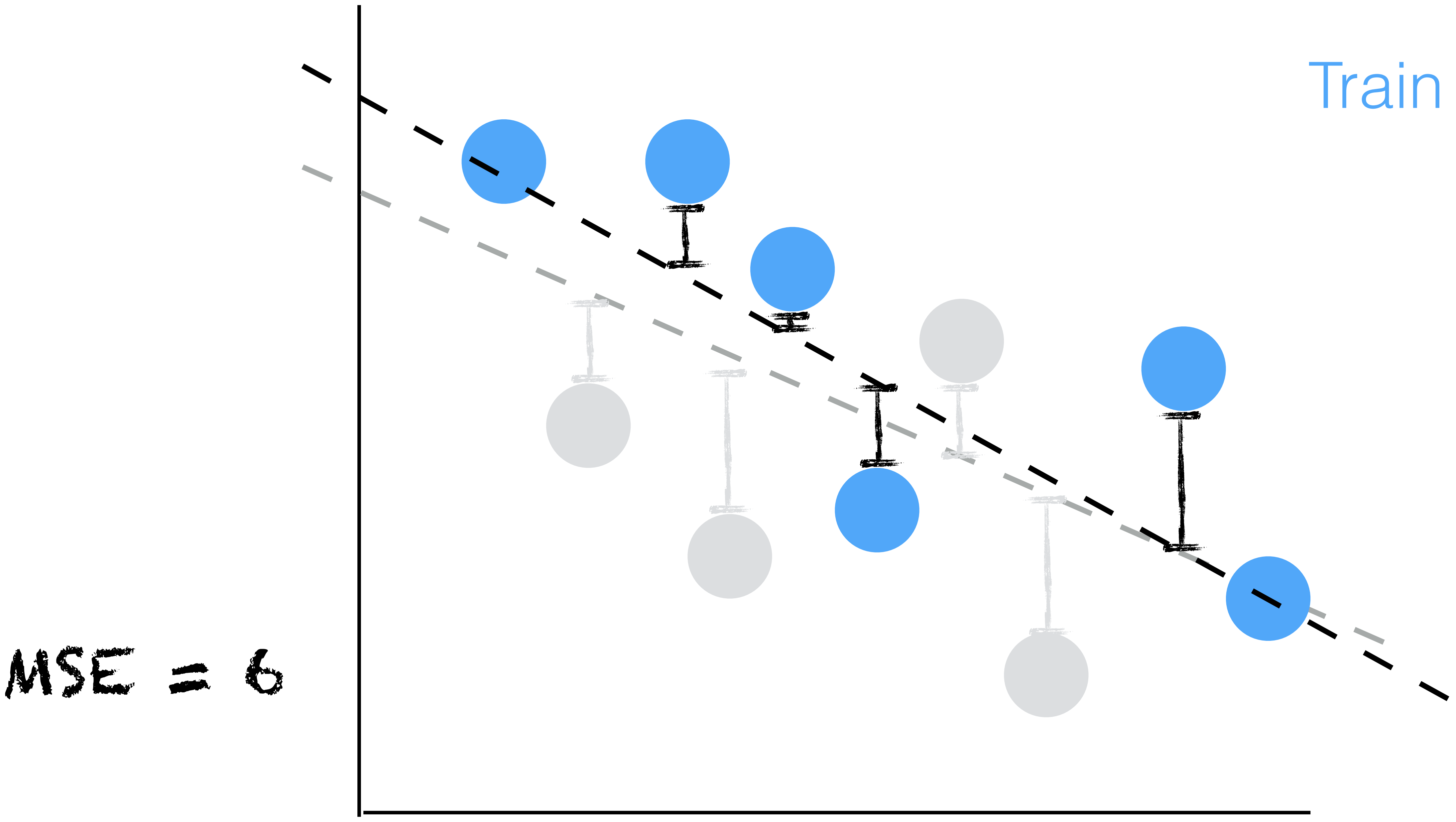
Train/Test Splits



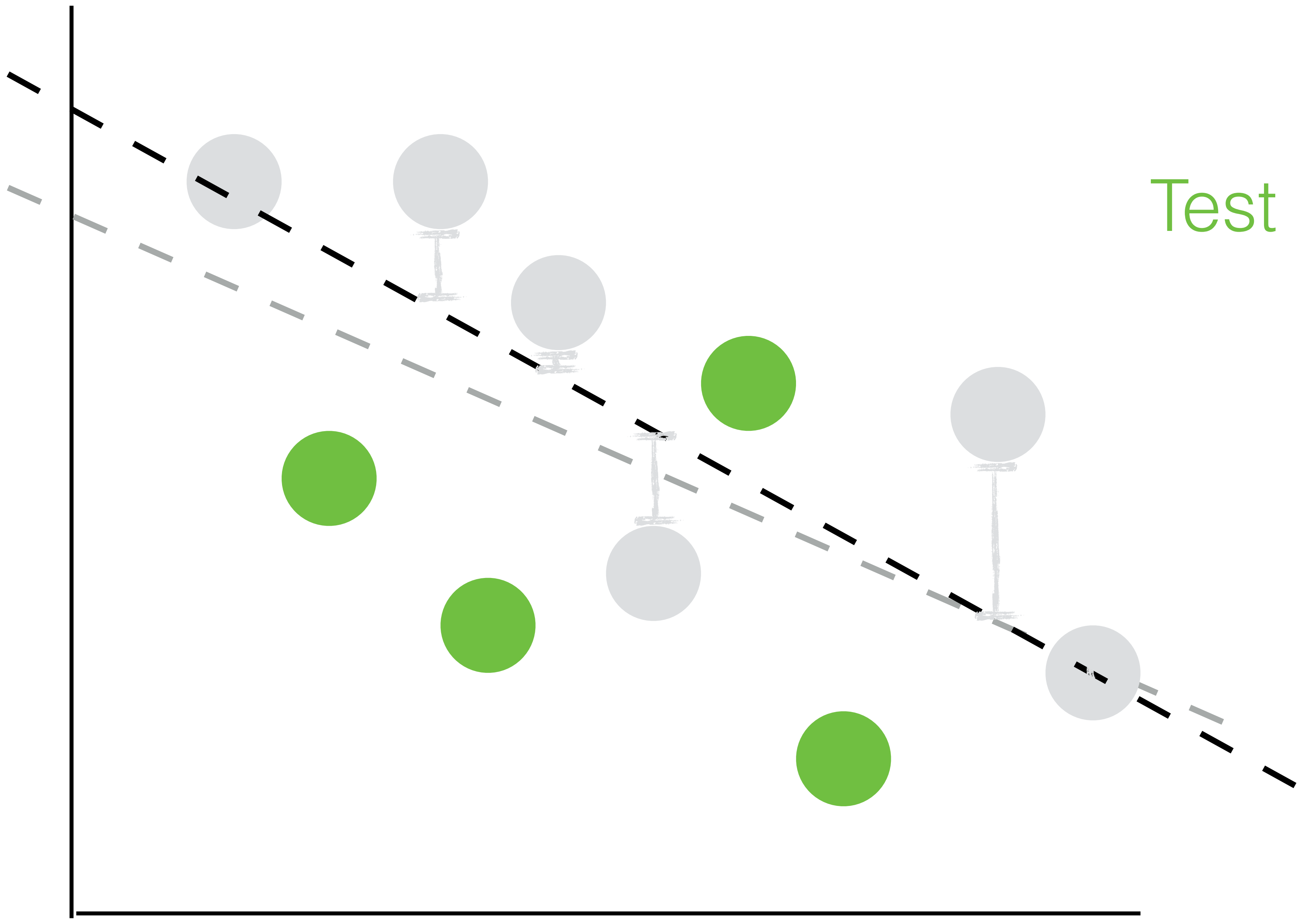
Train/Test Splits



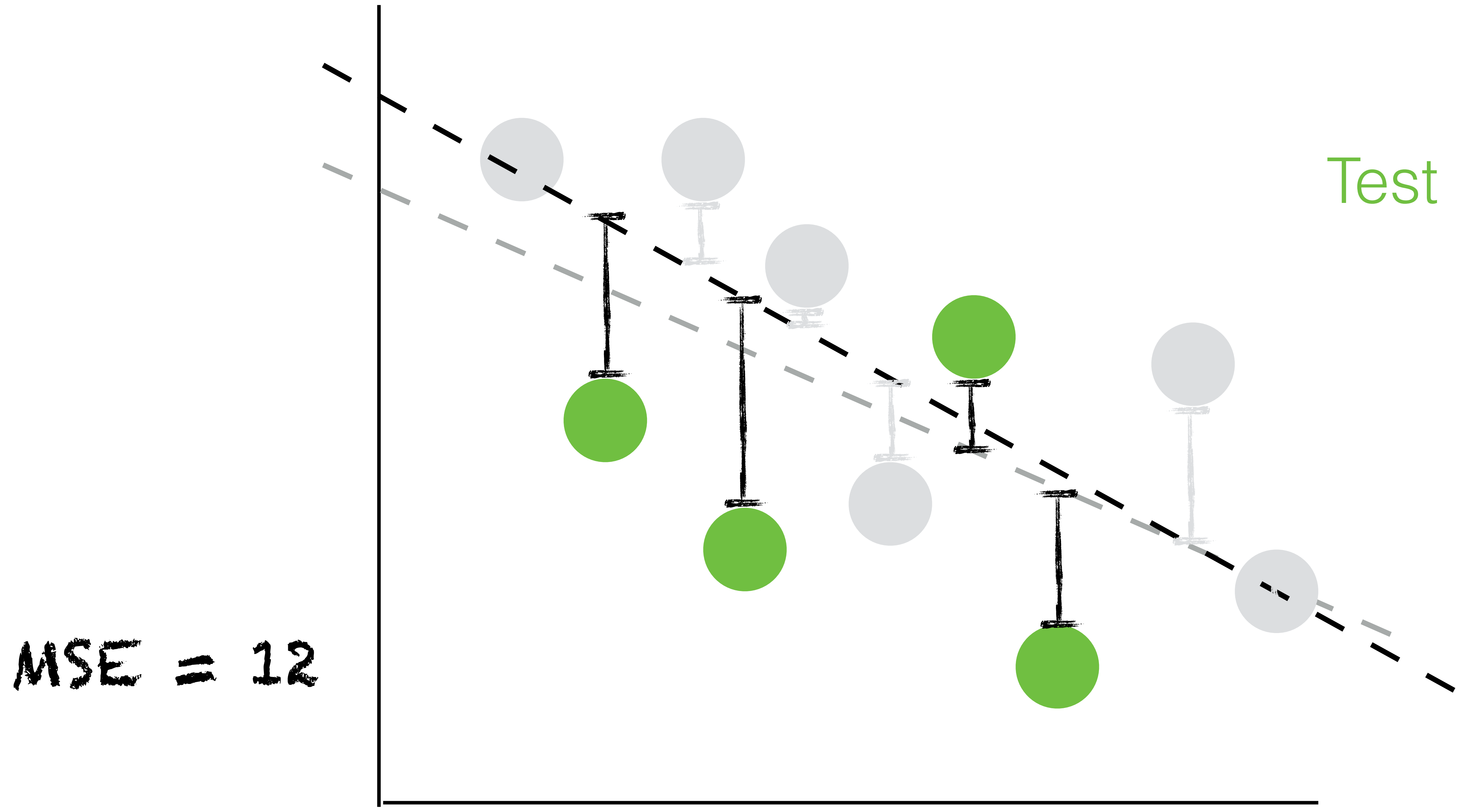
Train/Test Splits



Train/Test Splits



Train/Test Splits



Cross Validation

- Some train/test splits are harder than others
- To get a more stable estimate of test performance, we can use cross validation

```
accs = []
for i in range(num_folds):
    train, test = random.split(data)
    clf.fit(train)
    accs.append(clf.score(test))
```

Overfitting

- Models are likely to overfit when the model is more “complex” than is needed to explain the variation we care about

- “Complex” generally means the number of parameters (i.e. features) is high

- When the number of parameters is >= the number of observations, you can trivially memorize your training data, often without learning anything generalizable to test time

Regularization

- Incur a cost for including more features (more non-zero weights), or for assuming features are very important (more higher weights)

- Or “early stopping”—for iterative training procedures (i.e. gradient descent) stop before the model has fully converged (i.e. you assume the final steps are spent memorizing noise)

- *By definition* regularization will make your model worse during training…

- But hopefully better at test (which is what you really care about)

Regularization

$$min_\theta\Big(loss(x;\theta) + \lambda cost(\theta)\Big)$$

- Adds an extra “hyperparameter” which controls how much you penalize

Dev/Validation Sets

- Often you need to make meta-decisions (not just set the parameters), E.g.
  - Which model is better?
  - What regularization to use?
  - How many training iterations?
- Do do this, you have to split into train/dev/test, not just train/dev. If you use test to set these parameters, you are “peaking” at unseen data in order to fit the model, and thus test performance is no longer actually representative of how you would do in the real world

Norms

- L1 norm: $l_1 = \sum_i |x_i|$ encourages sparsity
- L2 norm: $l_2 = \sqrt{\sum_i x_i^2}$ more stable
- Lp norm: $l_p = \sqrt[p]{\sum_i x_i^p}$

Norms

- Linear Regression — No regularization
  $$min_w\big((y - w \cdot x)^2\big)$$

- Lasso Regression — Linear regression with L1 penalty on the loss
  $$min_w\big((y - w \cdot x)^2 + \lambda l_1(w)\big)$$

- Ridge Regression— Linear regression with L2 penalty on the loss
  $$min_w\big((y - w \cdot x)^2 + \lambda l_2(w)\big)$$

- Logistic Regression usually uses l1 or l2 regularization by default (e.g. in sklearn)

Feature Selection

- Explicitly remove features from model before training
- Lots of heuristic techniques (no magic solutions, requires trial and error)
- Some techniques:
  - Remove correlated features
  - Remove low-variance features
  - Iteratively add features with highest weight or information gain
  - Iteratively remove features with lowest weight or information gain
  - Dimensionality Reduction (e.g. PCA; will cover later in semester)