PROFESSIONAL & CONTINUING EDUCATION
UNIVERSITY of WASHINGTON

Introduction to Machine Learning

MLEARN 510A – Lesson 5

Recap of Lesson 4

- Data Preprocessing
- Dealing with Missing Data
- Detection of Outliers
- Exploratory Data Analysis
- Data Transformations
- Data Splitting

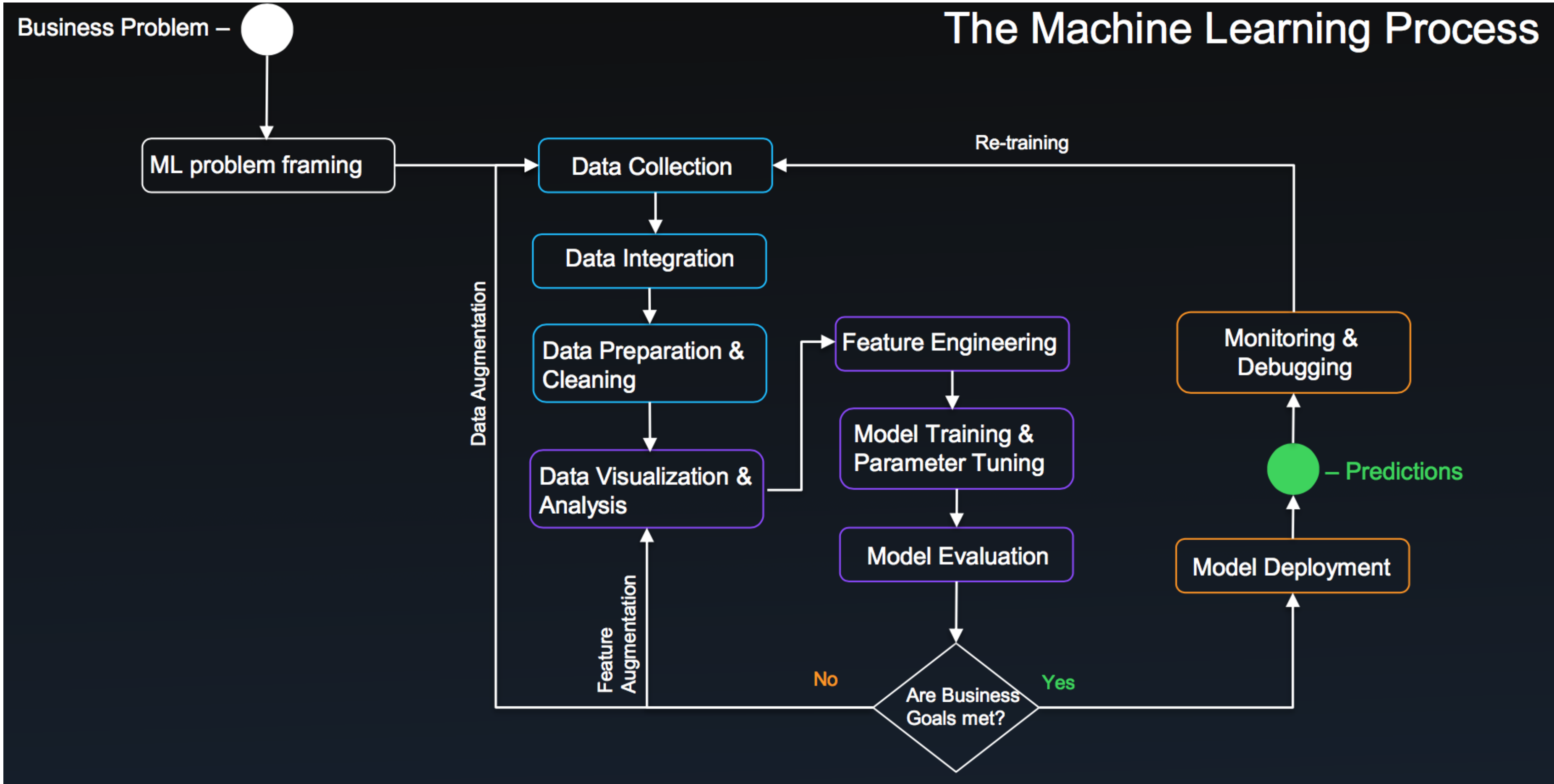
Course Outline

1. Introduction to Statistical Learning
2. Linear Regression
3. Classification
4. Model Building, Part 1
5. **Model Building, Part 2**
6. Resampling Methods
7. Linear Model Selection and Regularization
8. ~~Moving Beyond Linearity~~ Time Series Analysis
9. ~~Bayesian Analysis~~ Frequent Itemset Mining
10. Dimensionality Reduction

Outline of Lesson 5

- Feature Engineering
- Custom Feature Transformation
- Feature Selection
- Chaining Transformations Together
- Hyper-parameter Tuning
- Testing, Launching, Monitoring and Maintaining

The Machine Learning Process



Feature Engineering

- Transforming data to create features for ML models
- Most creative aspect of Data Science
- This process can be automated

"More data beats clever algorithms, but better data beats more data"
Peter Norvig

Categorical Features

- Nearly always need some preprocessing
- Difficult to impute missing data
- Can generate very sparse data due to high cardinality
- Two types:
  - - Ordinal Categorical Features
  - - Nominal Categorical Features

Feature Engineering – Categorical Features

- **Ordinal Encoding**
  - Label Encoding
  - Ordinal Encoding

- **Nominal Encoding**
  - One Hot Encoding
  - Dummy Encoding
  - Mean or Target Encoding
  - Frequency Encoding

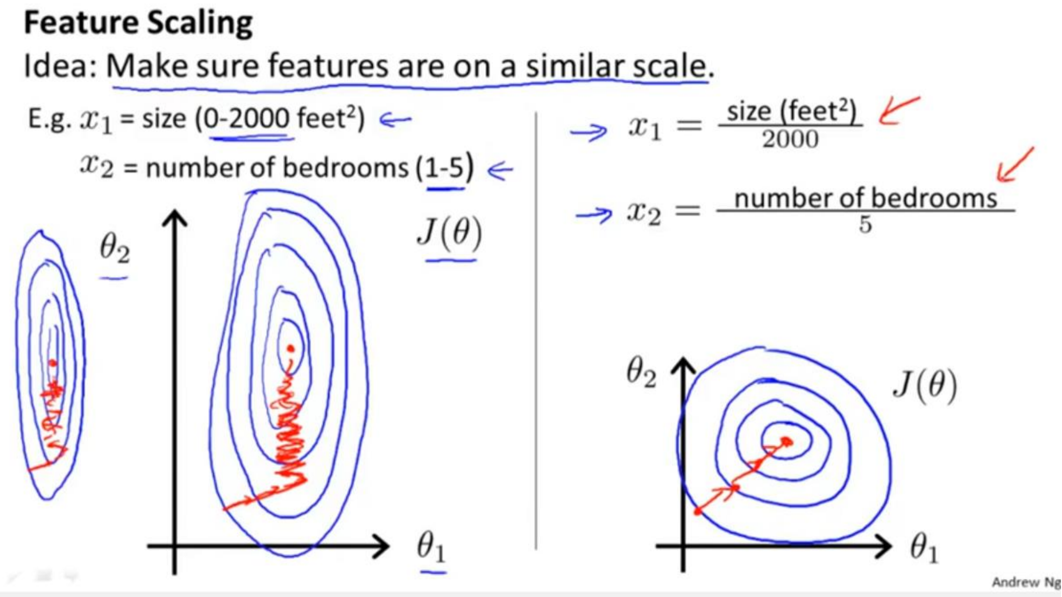
	pclass	survived	age	sibsp	parch	fare	sex_female	sex_male	embarked_C	embarked_Q	embarked_S
0	1	1	29.0000	0	0	211.3375	1	0	0	0	1
1	1	1	0.9167	1	2	151.5500	0	1	0	0	1
2	1	0	2.0000	1	2	151.5500	1	0	0	0	1
3	1	0	30.0000	1	2	151.5500	0	1	0	0	1
4	1	0	25.0000	1	2	151.5500	1	0	0	0	1

Feature Engineering – Numerical Features

- Floats, Counts, Numbers
- Can be readily fed into ML algorithms
- Easier to impute missing data

Feature Engineering – Numerical Features

- Quantization or Binning
- Log Transformation
- Power Transforms: Generalization of the Log Transform
- Feature Scaling or Normalization
  - Min-Max Scaling
  - Standardization
  - L2 Normalization



Feature Engineering – Text Features

- Bag-of-Words
- Bag-of-n-Grams
- Filtering for cleaner features
  - Stopwords
  - Stemming
  - Parsing and Tokenization
  - TF-IDF

Combining Features (Interaction)

- Add
- Subtract
- Multiply
- Ratios

- For example, given the number of rooms, number of bedrooms, we might create: Number of bedrooms/Number of rooms

Take advantage of your knowledge of data to create the right features

Feature Engineering – Date Time Features

- Number of days since a reference date
- Isolate minute, hour, day, month, day of year as separate features
- Is it morning, weekend, holiday, Black Friday, Christmas, free from work, school day?
- Handle the dates in user local time
- Relate the date to external events (incorporate external data), e.g. weather, traffic conditions, stock value, global purchasing on that day, news on that day, etc.

Custom Feature Transformation

- There will be times when we want to write our own custom transformation for clean up or feature engineering
- Scikit Learn provides the ability to write your own transformer
- Create a class and implement three methods: fit() (returning self), transform(), and fit_transform()
- Get the last one for free by simply adding TransformerMixin as a base class

Custom Feature Transformation – Example

```
from sklearn.base import BaseEstimator, TransformerMixin

rooms_ix, bedrooms_ix, population_ix, household_ix = 3, 4, 5, 6

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room = True): # no *args or **kargs
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self  # nothing else to do
    def transform(self, X, y=None):
        rooms_per_household = X[:, rooms_ix] / X[:, household_ix]
        population_per_household = X[:, population_ix] / X[:, household_ix]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]

            return np.c_[X, rooms_per_household, population_per_household,
                         bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, population_per_household]

attr_adder = CombinedAttributesAdder(add_bedrooms_per_room=False)
housing_extra_attribs = attr_adder.transform(housing.values)
```

Feature Selection

- Filtering
  - Subset Selection
- Wrapper methods
  - Recursive Feature Elimination
  - Forward Feature Elimination
  - etc…
- Embedded methods
  - Lasso Regularization

Subset Selection

1. Let $\mathcal{M}_0$ denote the *null model*, which contains no predictors. This model simply predicts the sample mean for each observation.
2. For $k = 1, 2, \ldots p$:
   (a) Fit all $\binom{p}{k}$ models that contain exactly k predictors.
   (b) Pick the best among these $\binom{p}{k}$ models, and call it $\mathcal{M}_k$. Here *best* is defined as having the smallest RSS, or equivalently largest R^2.
3. Select a single best model from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

Forward Stepwise Selection

1. Let $\mathcal{M}_0$ denote the *null* model, which contains no predictors.

2. For $k = 0, \ldots, p - 1$:

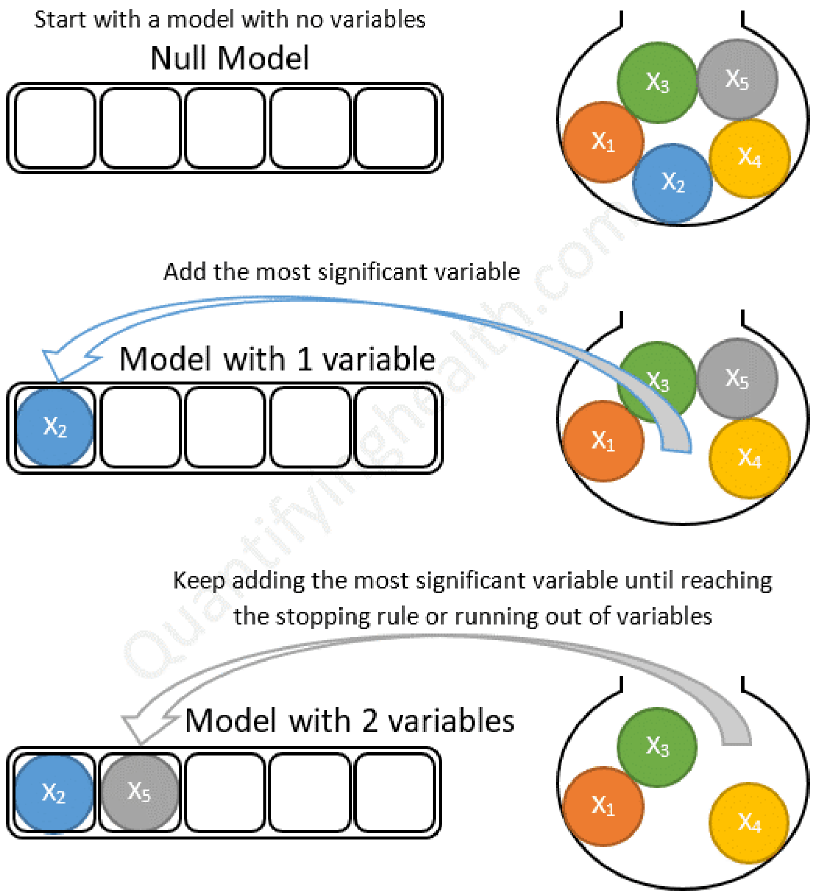
   (a) Consider all $p - k$ models that augment the predictors in $\mathcal{M}_k$ with one additional predictor.

   (b) Choose the *best* among these $p - k$ models, and call it $\mathcal{M}_{k+1}$. Here *best* is defined as having smallest RSS or highest R^2.

3. Select a single best model from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

Forward Stepwise Selection

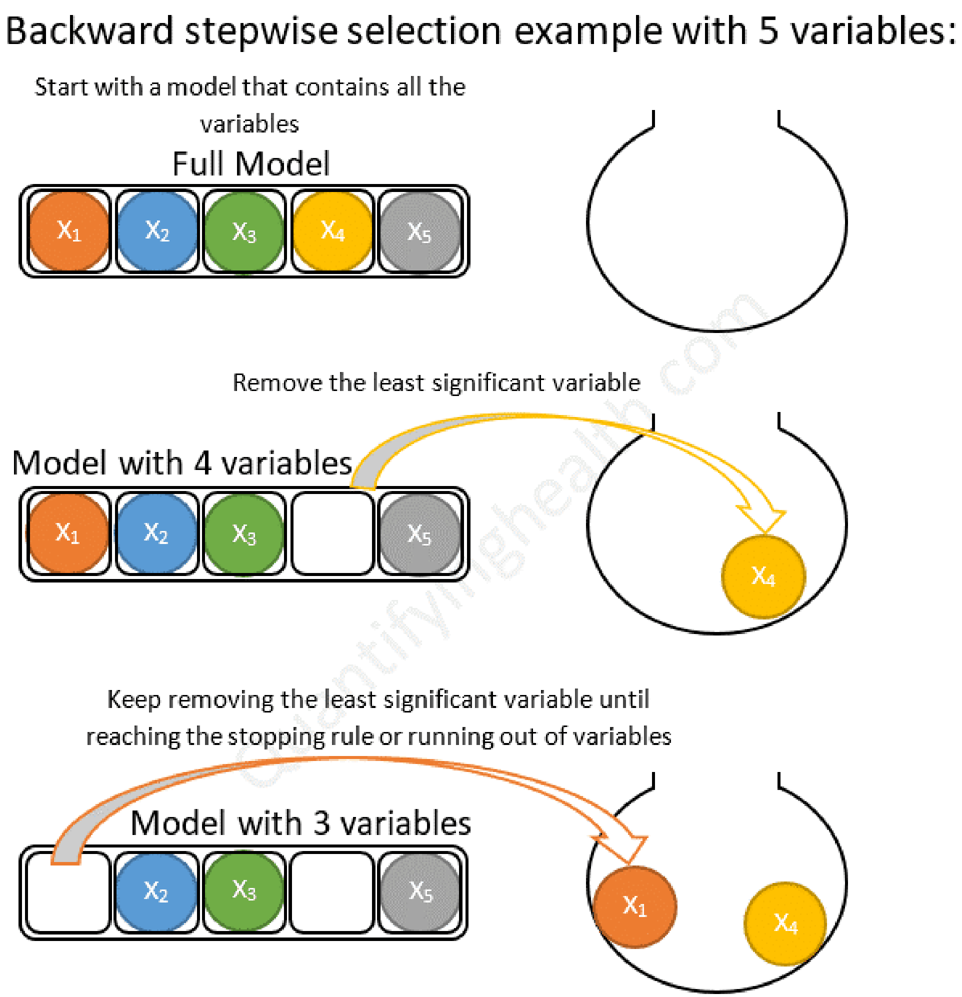
Forward stepwise selection example with 5 variables:



Backward Stepwise Selection

1. Let $\mathcal{M}_p$ denote the *full* model, which contains all p predictors.
2. For $k = p, p-1, \ldots, 1$:
   (a) Consider all k models that contain all but one of the predictors in $\mathcal{M}_k$, for a total of $k-1$ predictors.
   (b) Choose the *best* among these k models, and call it $\mathcal{M}_{k-1}$. Here *best* is defined as having smallest RSS or highest R^2.
3. Select a single best model from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

Backward Stepwise Selection



Recursive Feature Elimination (RFE)

- Works by recursively removing attributes and building a model on those attributes that remain

- It uses the model accuracy to identify which attributes (and combination of attributes) contribute the most to predicting the target attribute

- Steps
  - Train the estimator on the initial set of features
  - Prune the least important features from the list of features
  - Recursively repeat the process on the pruned list until stopping criteria is reached

Recursive Feature Elimination (RFE)

```python
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression

boston = load_boston()
X = boston["data"]
Y = boston["target"]
names = boston["feature_names"]

#use linear regression as the model
lr = LinearRegression()
#rank all features, i.e continue the elimination until the last one
rfe = RFE(lr, n_features_to_select=1)
rfe.fit(X,Y)

print "Features sorted by their rank:"
print sorted(zip(map(lambda x: round(x, 4), rfe.ranking_), names))
```

```
Features sorted by their rank:
[(1.0, 'NOX'), (2.0, 'RM'), (3.0, 'CHAS'), (4.0, 'PTRATIO'), (5.0, 'DIS'),
(6.0, 'LSTAT'), (7.0, 'RAD'), (8.0, 'CRIM'), (9.0, 'INDUS'), (10.0, 'ZN'),
(11.0, 'TAX'), (12.0, 'B'), (13.0, 'AGE')]
```

Lasso Regression

- Minimize the quantity

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j| = \text{RSS} + \lambda\sum_{j=1}^{p}|\beta_j|$$

- Lasso uses an l_1 penalty instead of an l_2 penalty. The l_1 norm of a coefficient vector β *is given by* $\|\beta\|_1 = \sum|\beta_j|$

- More on this in Week 7

Feature Selection – Summary

	Pros	Cons
Filter methods	• Usually fast • Intuitive (rely on well-known statistical dependency) • Universal (independent of the learner) • Often used as pre-processing step for other methods	• Can easily discard relevant features • Often select for highly correlated features
Wrapping methods	• Treat learner as a black box • Focus on final metric as measure of variable importance	• Can be time-consuming (especially forward feature selection, genetic algorithms)
Embedded methods	• Might be Fast • Easy to incorporate as part of training for some learners	• Approach is highly metric / learner specific

Steps Covered So Far

- Data Preprocessing
- Data Analysis
- Feature Transformation
- Feature Scaling
- Feature Engineering
- Feature Selection

Transformation Pipeline

- There are many data transformation steps that need to be executed in the right order
- Cleaning → Feature Transformation → Feature Engineering → Feature Scaling
- Fortunately, Scikit-Learn provides the Pipeline class to help with such sequences of transformations

Transformation Pipeline

- The Pipeline constructor takes a list of name/estimator pairs defining a sequence of steps
- All but the last estimator must be transformers (i.e., they must have a fit_transform() method)
- When you call the pipeline's fit() method, it calls fit_transform() sequentially on all transformers
- The output of each call as the parameter to the next call, until it reaches the final estimator, for which it just calls the fit() method

Transformation Pipeline

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

num_pipeline = Pipeline([
        ('imputer', Imputer(strategy="median")),
        ('attribs_adder', CombinedAttributesAdder()),
        ('std_scaler', StandardScaler()),
    ])

housing_num_tr = num_pipeline.fit_transform(housing_num)
```

Hyperparameters in Learning Algorithm

- Hyper-parameters are different from the typical parameters/weights for a learning algorithm
- Weights are learned during training while hyper-parameters are set before the learning process begins
- Model hyperparameters
  - Cannot be inferred while fitting training data because they refer to model selection
  - For example, number of layers of neural network
- Algorithm hyperparameters
  - Bear no influence on the performance of the model but affect the speed and quality of the learning process
  - For example, learning rate in Gradient Descent

Hyperparameter Tuning – Grid Search vs. Randomized Search

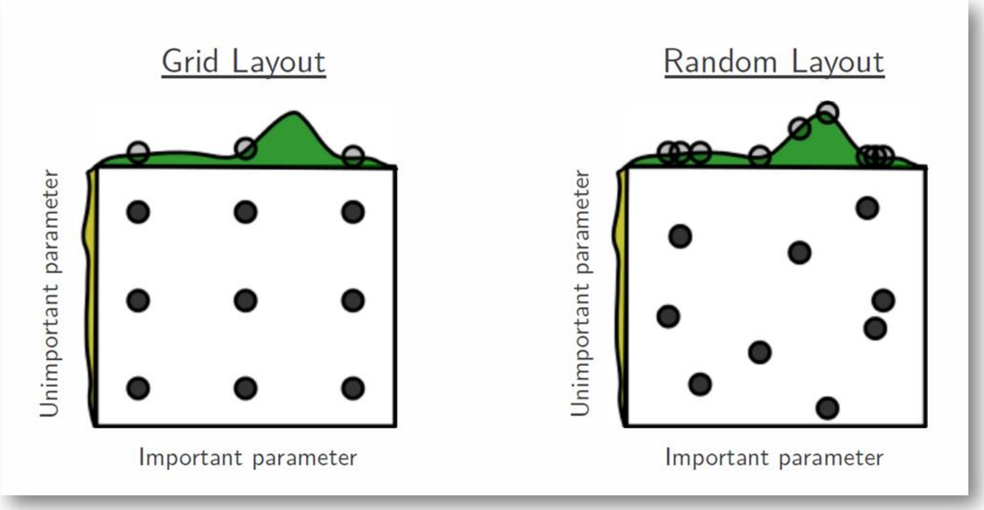
- Hyperparameter tuning is one of the trickiest parts in building the machine learning models
- Goal is to find a sweet setting for the model's hyperparameters

Grid Search

- Exhaustive search is performed for all combinations of hyperparameters

Randomized Search

- Random combinations of the hyperparameters are used to find the best solution for the built model



Validation Set for Hyperparameter Tuning

- It is important that examples from the test set are not used to make choices about the model, including its hyperparameters

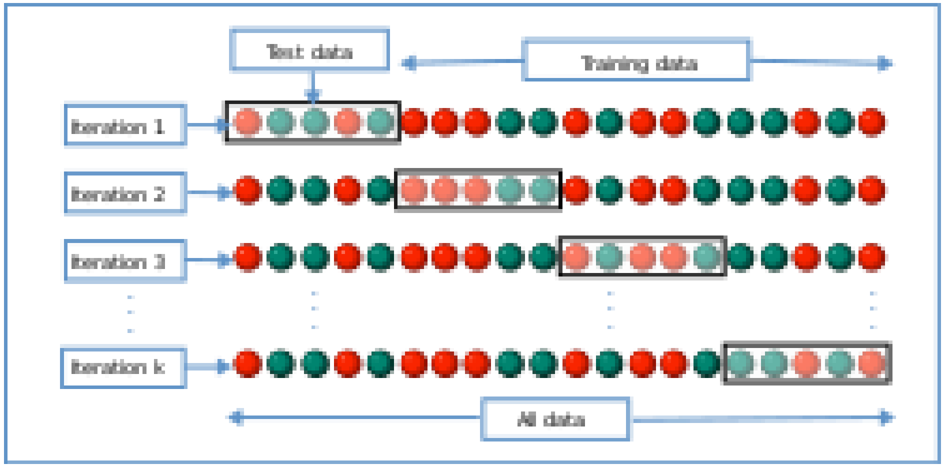
- We need a **validation set** of examples that the training algorithm does not observe

- Split the training data into two disjoint splits
  - **Training Set**: Used to learn parameters of the model (not to be confused with the larger training pool)
  - **Validation Set:** Used to guide the selection of hyperparameters

- Typical split is 80% for training and 20% for validation

Cross Validation

- When the dataset has hundreds of thousands of examples or more, splitting into fixed training and test sets works
- However, if the dataset is too small, test error will have statistical uncertainty around the estimated average test error
- **Solution**: Use all of the examples in the estimation of average test error at the cost of increased computational cost
- Use K-fold cross validation



Evaluate Performance on Test Set

- Once we are satisfied with the model, it is time to evaluate the final model on the test set

- Run your full_pipeline to transform the test data (call transform(), *not* fit_transform()!), and evaluate the final model on the test set

- You must resist the temptation to tweak the hyperparameters to make the numbers look good on the test set

- The improvements would be unlikely to generalize to new data

What's Next?

- **Prelaunch phase**
- Present your solution
- What worked and what did not, what assumptions were made, and what your system's limitations are)
- Document everything
- Create nice presentations with clear visualizations

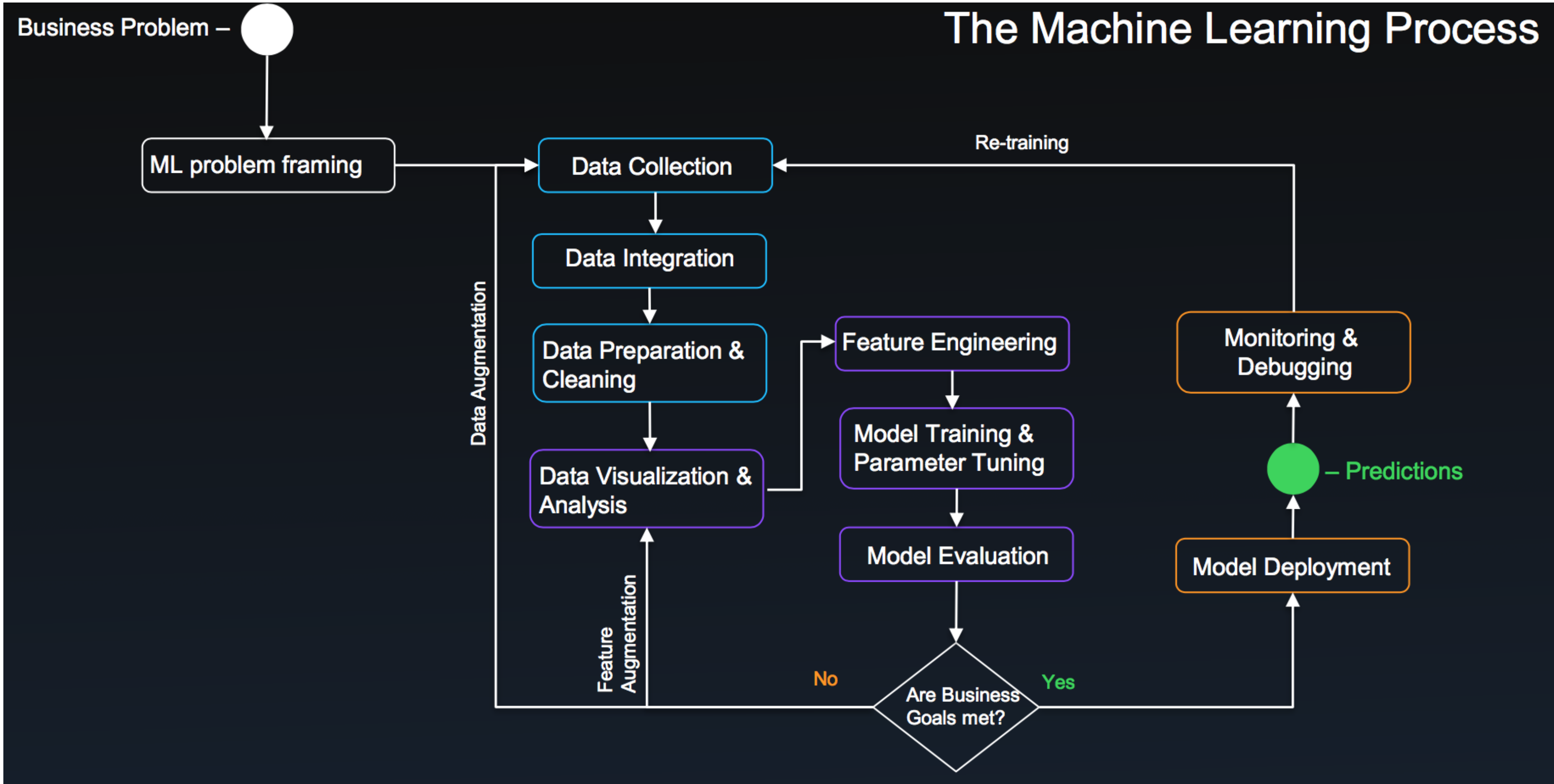
Launch, Monitor and Maintain (1)

- Plug the production input data into your system and write tests
- Write monitoring code to check your system’s live performance at regular intervals and trigger alerts when it drops
- Evaluate your system’s performance by sampling the system’s predictions. This will generally require a human analysis
- You should also make sure you evaluate the system’s input data quality

Launch, Monitor and Maintain (2)

- Monitoring the inputs is particularly important for online learning systems
- Re-train your models on a regular basis using fresh data
- Automate this process as much as possible
- If your system is an online learning system, you should make sure you save snapshots of its state at regular intervals so you can easily roll back to a previously working state

The Machine Learning Process



Resources

- *Chapter 2:* Hands-On Machine Learning with Scikit-Learn & Tensorflow

- Machine Learning Software Engineering in Practice: An Industrial Case Study

- Data Lifecycle Challenges in Production Machine Learning: A Survey

Jupyter Notebook

- *Case Study*

ON-BRAND STATEMENT

FOR GENERAL USE

> What defines the students and faculty of the University of Washington? Above all, it’s our belief in possibility and our unshakable optimism. It’s a connection to others, both near and far. It’s a hunger that pushes us to tackle challenges and pursue progress. It’s the conviction that together we can create a world of good. And it’s our determination to Be Boundless. Join the journey at **uw.edu**.