

## PROGRESSING IN YOUR DATA SCIENCE CAREER

Jonathan Balaban DAT2

### PROGRESSING IN YOUR DATA SCIENCE CAREER

## **LEARNING OBJECTIVES**

- ▶ Specify common models used within industries
- ▶ Identify use cases for common models
- ▶ Discuss next steps and additional resources for data science learning

### **OPENING**

- ▶ Let's discuss how to adapt this course to real-world problems.
- ▶ We'll talk about how to maintain and improve your analysis.
- ▶ We'll also talk about what steps can be taken to make your work "production" ready.
- Lastly, we'll focus on tools and topics in the data science ecosystem that you should explore in the future!

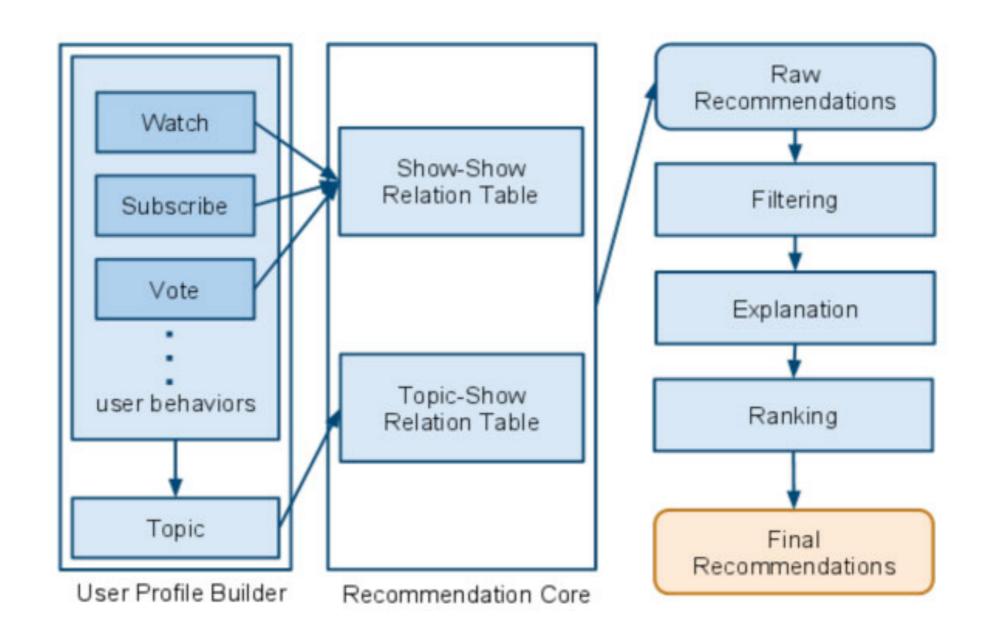
### INTRODUCTION

# MACHINE LEARNING SYSTEMS

- As you move into real world projects, it's important to remember that models and analysis are only *one part* of a larger business objective.
- ▶ Typically, the model may only answer one of *many* questions that need to be addressed.
- ▶ Even within modeling itself, there are differences between how a model runs during testing and production.

- For example, in a system that will present recommendations, we may have many modeling components that come together.
- ▶ Different aspects may categorize content, extract text features, analyze popularity, etc.
- ▶ These will all tie back into the final data product.

- ▶ Hulu's recommendation system:
  - ▶ Pulls data from multiple sources
  - ▶Builds user profiles and summaries
  - ▶ Applies a recommendation model
- There's more! Additional filters refine the model in order to ensure predictions are novel and don't overlap with previous content.



▶ Organizing and managing systems and data dependencies is essential

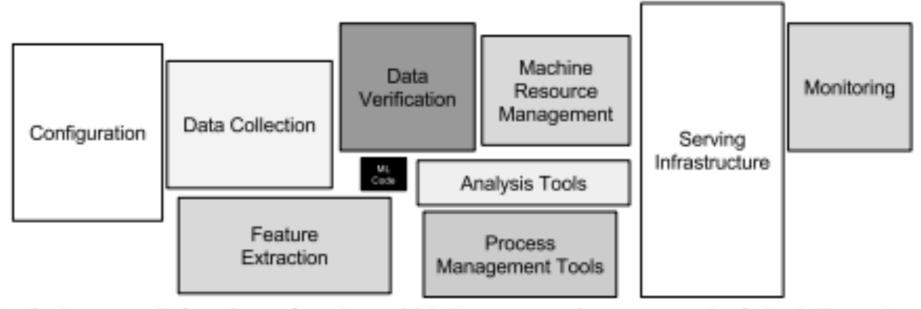
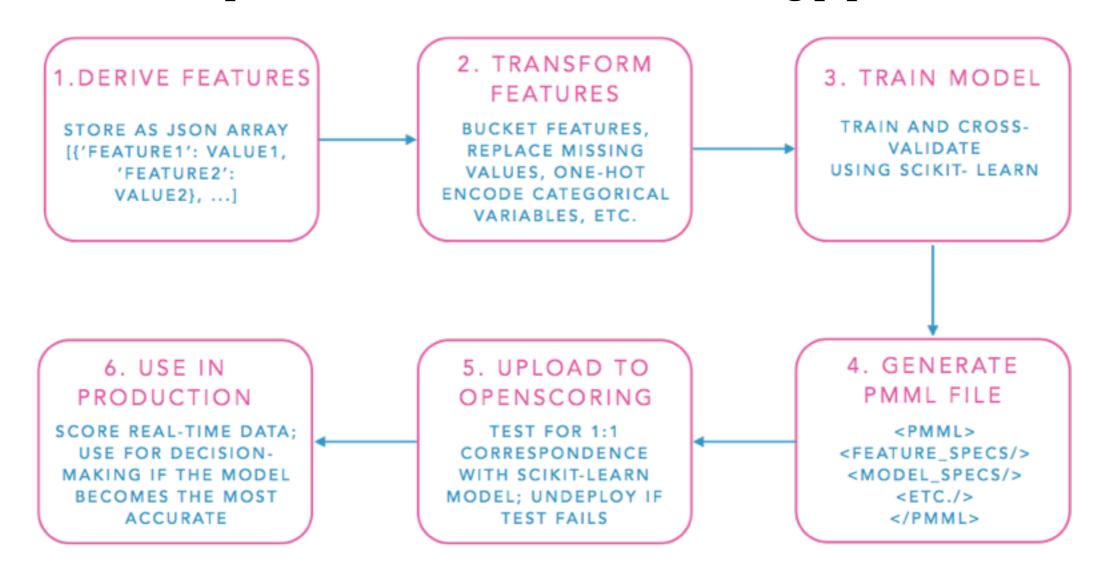


Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

- Many organizations rely on *data engineering* teams to code these common task into pipelines.
- ▶ **Data pipelines** are a series of automated data transformations that ensure the validity of your work for routine data maintenance tasks.

▶ Below is a description of the AirBnB model building pipeline.



### **MODEL MAINTENANCE**

- ▶ Our classes focused on building an initial model and analysis.
- ▶ However, once a final model is trained (and we're happy with performance), the model needs to be **maintained**!
- As new data is gathered, the model will likely need to be **retrained**.
- Over time, previously predictive features may lose value, requiring you to reinvestigate.

### **MODEL MAINTENANCE**

- ▶ Google addresses this phenomenon, describing the "Technical Debt" of machine learning systems in a paper called:
  - "Machine Learning: The High Interest Credit Card of Technical Debt"
- ▶ They focus on a few core issues that affect model maintenance:
  - ▶Code complexity
  - **▶**Evolving features
  - Monitoring and testing

- ▶ Most of the code for our class has been written in notebooks.
  - As your analysis and models develop, you are likely to revise and reuse parts of this code.
- ▶ Improving the quality of your code can simplify this process.
- This isn't always the responsibility of data scientists, but keep in mind: code clarity will lead to analysis clarity!
  - This is especially true for long term or open source projects where your code has to make sense to other people (or yourself) in the future!

- ▶ One way to write better code is to create (and follow!) a *styleguide*.
  - ▶ A *styleguide* is a clear set of rules for organizing your code.
- ▶ Columbia recently developed <u>a special styleguide just for data scientists</u>: columbia-applied-data-science.github.io/pages/lowclass-python-style-guide.html
- ▶ Some rules are pretty straightforward:
  - •Give variables, methods, and attributes descriptive names.
  - Write functions that take well-defined inputs and produce well-defined outputs.

- ▶ Another common practice is *unit testing*.
  - ▶ Unit testing involves writing short statements that *test* if a piece of code or function is working correctly.
- ▶ Typically, these tests provide a few sample inputs and outputs and then check that your code can produce the same outputs.
- According to Google, "ensuring that code has been tested is vital to ensuring that your analysis results are correct."

▶ Suppose we have the following function that calculates the area of a circle.

```
def calculate_area_of_circle(radius):
    # Use value of pi
    pi = 3.14
    area = pi * radius ** 2
    return area
```

▶ A unit test for this may look like the following.

```
def test_calculate_area_of_circle():
    # Various test cases
    assert calculate_area_of_circle(2) == 12.56
    assert calculate_area_of_circle(4) == 50.24
    assert calculate_area_of_circle(0) == 0.0
```

- On long term or big data projects, the code supporting a machine learning model can get complex.
- This "glue code" holds the model together, but it can get weighed down with bloat and feature creep over time.
- ▶ Thus, this code often needs to be *refactored* in order to trim the fat.

▶ Google describes the need to review your code, stating that:

"Only a tiny fraction of the code in many machine learning systems is actually doing "machine learning""

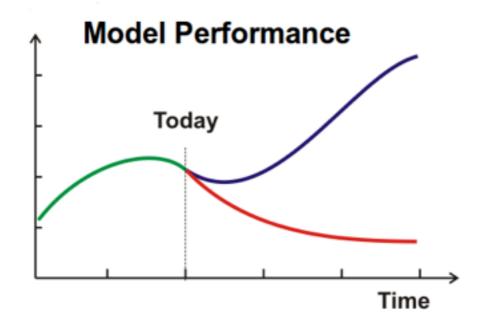
"Without care, the resulting system for preparing data in an ML-friendly format may become a jungle of scrapes, joins, and sampling steps, often with intermediate files output."

"Managing these pipelines, detecting errors, and recovering from failures are all difficult and costly."

- Creating and following a clear **styleguide** as well as **testing** and **refactoring** your code will help maintain your machine learning algorithm over time.
- ▶ Plus, reducing *technical debt* saves time and money in the long term!
- ▶ Even Google is not immune:

"In a recent cleanup effort of an important machine learning system at Google, it was possible to rip out tens of thousands of lines of unused experimental code-paths!"

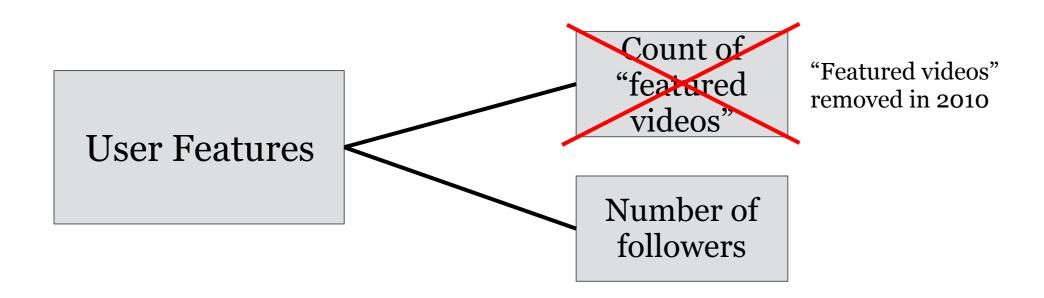
- Once your model is trained, it's important to track its performance over time.
- ▶ Many of the correlations found or features predicted may not remain true in a few months or years into the future.



- ▶ For example, our "evergreen" article prediction example looks for food mentions to predict the popularity of certain recipes.
- ▶ However, it doesn't know how to gauge trends in popular foods. Over time, it will return a smaller and smaller sample unless we readjust the model's parameters.
- As one trend takes off, the model trained on old trends may not be able to pick that up.

- Google's technical debt paper groups troublesome features into two groups: *legacy features* and *bundled features*.
  - **Legacy features**: When a feature "F" is included in a model early on, but later other features are added that make F redundant.
  - **Bundled features**: When a group of features are all bundled together, it can be hard to differentiate the features that aren't performing well from the ones that are.

- ▶ Features can also be "bundled" with commonly occurring variables, but those variable occurrences may change over time, making the features obsolete.
- For instance, we may have features for a Youtube user that are no longer tracked or relevant. These may be bundled with other "user features".



▶ From Google's paper:

"Machine learning systems often have a difficult time distinguishing the impact of correlated features.

This may not seem like a major problem: if two features are always correlated, but only one is truly causal, it may still seem okay to ascribe credit to both and rely on their observed co-occurrence.

However, if the world suddenly stops making these features co-occur, prediction behavior may change significantly."

- ▶ Changing variables is especially important for *black box models*.
- ▶ Such models rely on correlations from a wide range of features. However, in doing so, we can typically ignore one of two variables that are highly correlated.
- ▶ If these variables are no longer correlated, we may need to update this.

- ▶ Another common way for features to evolve is through *feedback loops*.
- Once you've performed an analysis and built your model, it's likely you will make decisions and take actions based on your findings.
- It's important to think about how these actions may change the data you are using for future analysis.
- ▶ Are you introducing bias to your data and model?

- ▶ For example, imagine we are investigating ways to reduce the spread of infections in hospitals:
  - We may find that whenever a doctor sees more than 5 patients in an hour, those patients have a greater risk for infection.
  - ▶ Based upon this, we implement a policy that doctors cannot see more than 5 patients in one hour.
  - If we perform our same analysis a year after this policy is enacted, the feature "saw >5 patients in an hour" won't exist!

### **MONITORING MODELS**

• Once a model is deployed and making predictions, it's important to track its performance.

### From Google:

"Unit testing of individual components and end-to-end tests of running systems are valuable, but in the face of a changing world such tests are not sufficient to provide evidence that a system is working as intended.

Live monitoring of system behavior in real time is critical."

### **MONITORING MODELS**

- One common monitoring technique is to compare your model's performance to a baseline.
- The baseline can be something simple, like a naive model that only predicts the average or most frequently occurring value.
- ▶ When monitoring a model, you can update your baseline as information becomes available.
- ▶ Your "better" model should always outperform the baseline!

### ETHICAL CONSIDERATIONS

- Another (often overlooked) aspect of managing real world data science projects are *ethical considerations*.
- It's important to understand the biases of your data and how this influenced our analysis and models.
- Two common examples are criminal justice and financial loans applications.

### ETHICAL CONSIDERATIONS

- ▶ When analyzing crime, we often want to consider what drives criminal activity and what actions might reduce it.
- ▶ However, it's important to consider how our data is collected. For example, current data is based off the current criminal justice system.
- It can be difficult to separate the biases of the current system from the correlations/predictions that you are trying to make in your model.
- If data from the current justice system overweighs specific concerns or attributes, your model will too.

### ETHICAL CONSIDERATIONS

- Similarly, data from financial lenders may be biased, as their goal is to find borrowers who are most likely to pay back in a timely fashion.
- These analyses need to be strongly regulated, so protected factors (e.g. race, gender, etc) are not considered. However, they can still leak in through proxy features.
- ▶ Proxy features are not protected per se, and are strongly correlated with specific protected features.
- ▶ For example, neighborhood zip code can be used as a proxy for race.

### **ACTIVITY: KNOWLEDGE CHECK**

### **ANSWER THE FOLLOWING QUESTIONS**



- 1. In small groups, discuss other areas of possible ethical issues in Data Science.
  - a. How might this occur when examining health data?
  - b. What about when examining educational records?

### **ACTIVITY: DATA SCIENCE IN AN ORGANIZATION**



### **DIRECTIONS (20 minutes)**

Break into groups of 4-5 students. Each group will get a company and 1-2 data products that company is building.

- 1. Brainstorm maintenance that might be performed.
  - a. When should you redo the study?
  - b. What features may change or become difficult to collect in the future?
- 2. Describe possible interventions.
  - a. Will this change the data collected in the future?
- 3. Describe ethical issues that may arise from these tasks.

- One way to improve coding and model management is to use pipelines in scikit-learn.
- ▶ Pipelines tie together all the steps you may need to prepare your dataset and make your predictions.
- ▶ Because you will need to perform all of the same transformations on your test data, encoding the *exact same steps* is important.

from sklearn.pipeline import Pipeline

▶ Previously we built a text classification model using CountVectorizer

```
import pandas as pd
import json

data = pd.read_csv("../../assets/dataset/stumbleupon.tsv", sep='\t')
data['title'] = data.boilerplate.map(lambda x: json.loads(x).get('title',
''))
data['body'] = data.boilerplate.map(lambda x: json.loads(x).get('body', ''))
titles = data['title'].fillna('')
```

• We can fit the vectorizer and transform our data.

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(max features = 1000,
                             ngram_range=(1, 2),
                             stop words='english',
                             binary=True)
# Use `fit` to learn the vocabulary of the titles
vectorizer.fit(titles)
# Use `tranform` to generate the sample X word matrix - one column per feature
(word or n-grams)
X = vectorizer.transform(titles)
```

- We used this input X, matrix of all common n-grams in the dataset, as the input to a classifier.
- ▶ We wanted to classify how evergreen a story was.

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(penalty = 'l1')
y = data['label']

from sklearn.cross_validation import cross_val_score

scores = cross_val_score(model, X, y, scoring='roc_auc')
print('CV AUC {}, Average AUC {}'.format(scores, scores.mean()))
```

- ▶ Often, we will want to combine these steps to evaluate on some future dataset.
- Therefore, we need to make sure we perform the *exact same* transformations on the data.
- ▶ Pipelines combine both **preprocessing** and **model building** into a single object, tying all the steps together.

- Similar to models and vectorizers in scikit-learn, pipelines have fit and predict or predict\_proba methods.
- ▶ However, they also make sure the proper data transformations occur.

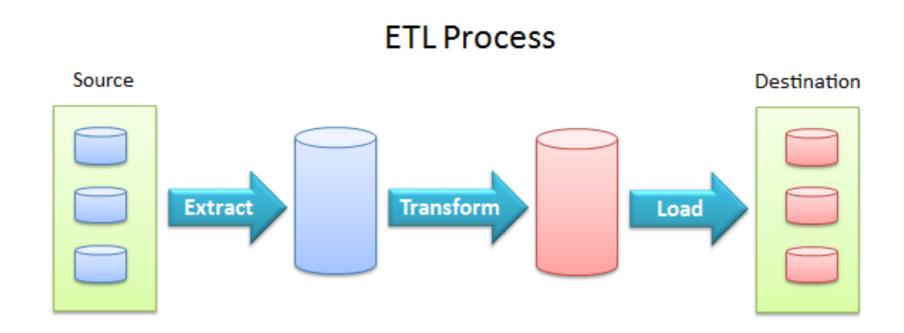
```
# Split the data into a training set
training_data = data[:6000]
X_train = training_data['title'].fillna('')
y_train = training_data['label']

# These rows are rows obtained in the future, unavailable at training time
X_new = data[6000:]['title'].fillna('')
```

```
from sklearn.pipeline import Pipeline
pipeline = Pipeline([
        ('features', vectorizer),
        ('model', model)
    ])
# Fit the full pipeline. This means we perform the steps laid out above
# First we fit the vectorizer,
# and then feed the output of that into the fit function of the model
pipeline.fit(X train, y train)
# Here again we apply the full pipeline for predictions
# The text is transformed automatically to match the features from the pipeline
pipeline.predict proba(X new)
```

- Additionally, we may want to merge many different feature sets automatically. This can be done with FeatureUnion.
- ▶ While scikit-learn pipelines help manage raw data transformation, there may be many steps occurring before this takes place in your pipeline.
- Such pipelines are often referred to as *ETL pipelines* for "Extract, Transform, Load".

In an *ETL pipeline*, the data is pulled or extracted from some source (like a database), transformed or manipulated, and then "loaded" into whatever system or analysis requires them.



- This combines many steps from the data science workflow into one repeatable process.
  - ▶Acquire Extract the data from the source
  - ▶ Parse Verify the quality of the data
  - Mine Format, clean, slice, derive columns
  - ▶ Refine (possibly) Transform the data

- ▶ Many data science teams rely on software tools to manage these ETL pipelines.
- ▶ These tools can alert you to failures and schedule jobs to run periodically, maybe daily or weekly.
  - One of the most popular Python tools for this is <u>Luigi</u>, developed by Spotify.
  - ▶ Another alternative is <u>Airflow</u> by AirBnB.

#### INTRODUCTION

# ALTERNATIVE TOOLS

- ▶ While we've mostly talked about Python in this class, there are many other languages and tools that Data Scientists might use.
- ▶ These tools have their various advantages and disadvantages.
- ▶ For example, other common programming languages for data science include:

**R** 

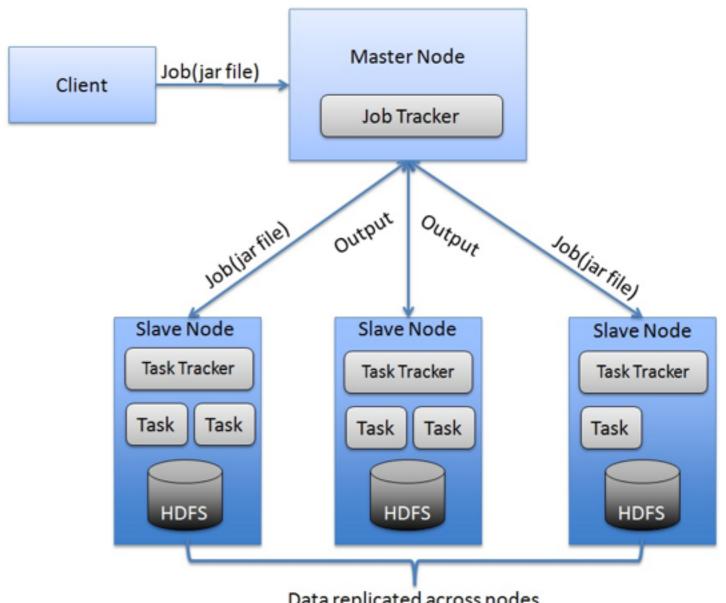
▶Java/Scala

- ▶ "R" is often used in data science and is the basis for many features found in Python data analysis.
- ▶ Pandas dataframes actually replicate the functionality of the R dataframe!
- ▶ R often contains many more specialized algorithms than Python.
- ▶ Between statsmodels and scikit-learn, Python has access to the most popular statistical algorithms. But if your problem becomes more specialized, you may require the niche algorithms available in R.

- ▶ Python's advantages over R are speed and the ability to tie into other applications (web apps, etc).
- ▶ Python code is generally faster and more efficient.
- ▶ R has tried to replicate some of this extra functionality, but it is generally more native to Python.

- ▶ Meanwhile, Java/Scala are popular for their link to the Hadoop ecosystem.
- Many larger organizations store their data in a Hadoop system and most connectors to access data are built in Java and Scala.

- ▶ What is Hadoop?
- ▶ A distributed computing system/environment.



Data replicated across nodes

- ▶ Here is a sample of the Hadoop ecosystem.
- It can be easier to interact with Hadoop systems using these languages.
- ▶ However, in general they lack the interactivity and ease of use that R and Python have.





Data Exchange Sqoop Log Collector Flume





Connectors Statistics





Oozie

Norkflow

YARN Map Reduce v2

Distributed Processing Framework



Hadoop Distributed File System



olumnar

#### **MODELING FRAMEWORKS**

- ▶ While scikit-learn is the most popular machine learning framework in Python, there are alternatives for specialized use cases.
- For example, most models in scikit-learn require datasets to be small enough to fit into memory.

#### **MODELING FRAMEWORKS**

- ▶ Other frameworks can work around this limitation.
- ▶ One example is xgboost, which provides efficient Random Forest implementations that train much faster than scikit-learn models.
- Similarly, the Vowpal Wabbit library is often used to train very large linear models, using computational tricks to operate on tens of millions of datapoints.

#### INTRODUCTION

### NEXT STEPS

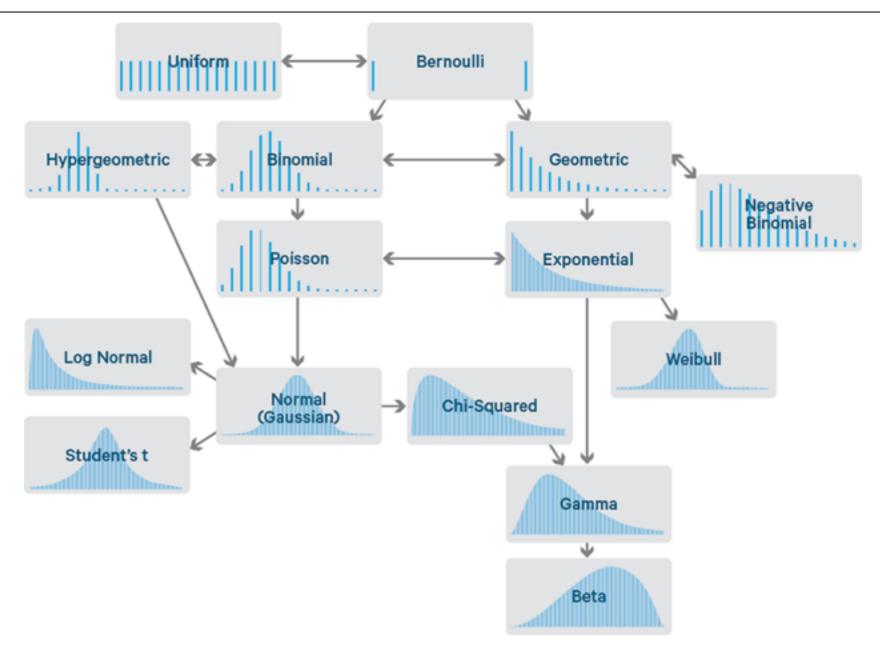
#### **NEXT STEPS**

- Most of this class has focused on statistical knowledge while practicing various methods of supervised and unsupervised learning.
- ▶ Of course, for each of these topics there are **many** alternative methods to learn! :)

#### STATISTICAL TESTING

- ▶ While you don't need to know all of them, being aware of some common statistical tests and their assumptions is useful.
- Additionally, having a clear sense of distributions (and what they look like) is important when communicating your findings.
- ▶ Being able to view a histogram and summarize it by the distribution it resembles makes it much easier to discuss your data.

### STATISTICAL TESTING: DISTRIBUTIONS

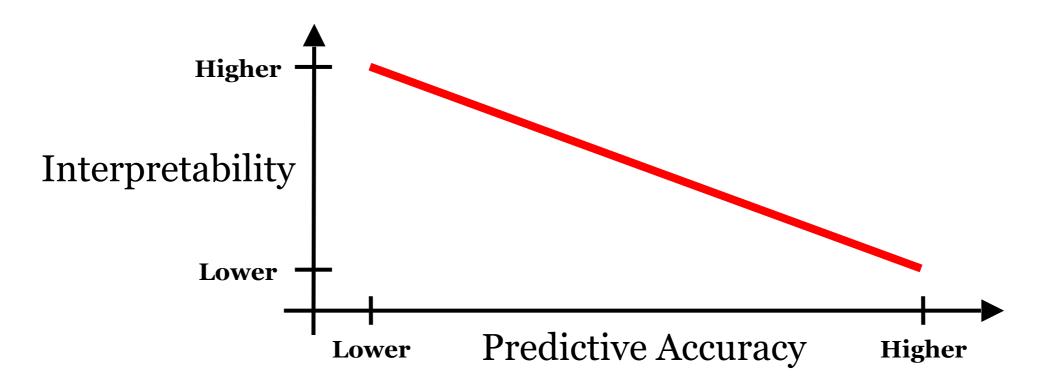


#### **VISUALIZATION**

- ▶ Visualizing data in business presentations is typically a much better way to transfer information to your audience.
- Most of the plotting for our class was done in Python, but keep in mind that these plots are often not the most visually appealing...
- Luckily, many other tools exist to build prettier plots!
- For example, you can play around with tools like plot.ly or <a href="D3.js">D3.js</a> (a javascript framework) to make your plots interactive.

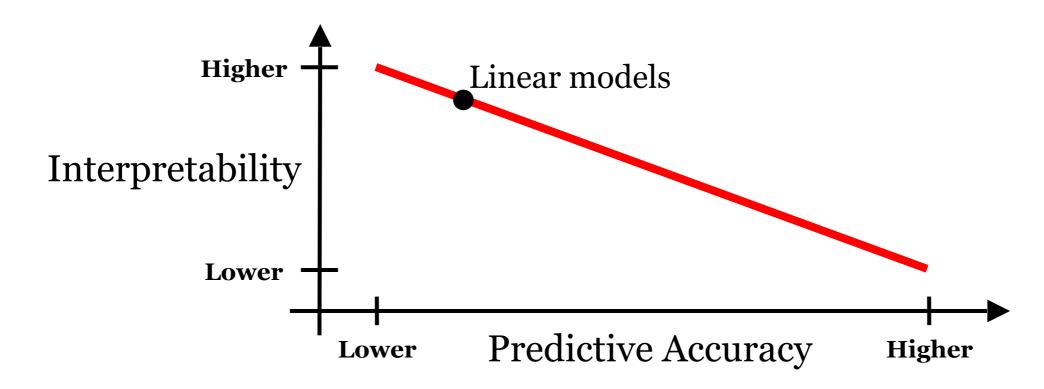
Another important point to review is that data modeling is a constant trade-off between **predictive accuracy** and **interpretability**.

### **Predictive Accuracy vs Interpretability**



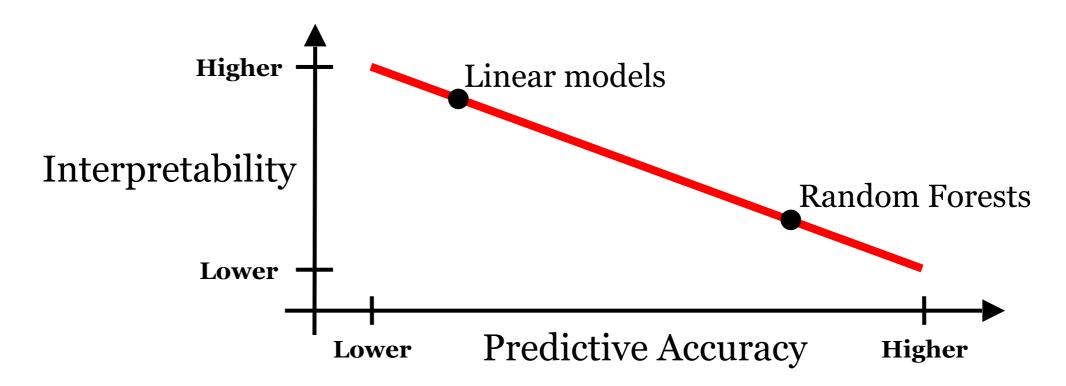
Linear models are simple, perform well, and offer a concise summary of the impact of various features through coefficients. Thus, they have a high degree of *interpretability*, but typically less predictive accuracy.

#### **Predictive Accuracy vs Interpretability**



▶ Black box models, such as Random Forests, may outperform linear models, but without as much transparency. They have a high degree of predictive accuracy, but less interpretability.

#### **Predictive Accuracy vs Interpretability**



- ▶ You should always consider whether you care more about interpretability or accuracy, and communicate your findings accordingly.
- The division between these two outcomes is very common in data science.
- Two advanced models (that you should experiment with in the future!) perfectly capture this divide. They are:
  - ▶Bayesian data analysis
  - ▶Deep learning algorithms

- ▶ Bayesian data analysis is a method of analysis that requires you to capture your expectations about the interactions of your data, then attempt to learn how strong these interactions are.
- ▶ This assumes you have some idea of how things work before you build a model and you allow this to affect your model build.

- ▶ For example, suppose you are analyzing the roll-out of a new educational policy and want to measure the impact of this policy on test scores.
- ▶ You'll need to know what else will impact those test scores and build a model that can measure the impact of this policy.
- ▶ However, you'll also want to enforce additional constraints. For example, this policy may have a related but different effect on outcomes depending on location and region.

- We may also think of other reasons that this new policy will affect subgroups differently (e.g. local resources, demographics, budgets, etc).
- ▶ We should explicitly state how these aspects further constrain our model.
- ▶ Bayesian models are typically small and their main strengths are interpretability and capturing uncertainty in the data.
- Rather than stating that X will change Y by some amount, they give a distribution or range of possible amounts and attempt to tell what will happen in all cases.

- This makes Bayesian models very useful when interpretability and defining interactions are the most important goals: they give us a clear definition of how right or wrong we are.
- ▶ For example, we may want to predict that candidate X is likely to win the election while also quantifying the degree of uncertainty.
- ▶ One tool you can use to build these models in Python is <u>pymc</u>.
- ▶ <u>Bayesian Methods for Hackers</u> is a good reference for this.

- On the other end of the spectrum, *deep learning models* are very powerful but offer little to no interpretable value.
- Deep learning models like *neural networks* are highly accurate but complex to build and understand.
- ▶ Google's has produced some interesting "art" using neural networks trained to identify certain objects.

- ▶ Deep learning models operate in a stage fashion.
  - First, they perform a dimensionality reduction to extract patterns or representations of the input data.
  - These representations are then used for the predictive task.
- Deep learning models tie these two steps together, attempting to learn the best representation for the task.

- Deep learning methods include many non-linear operations to capture complex relationships in the data.
- ▶ These models are particularly well suited for image or audio analysis.
- Some Python deep learning libraries include <u>Keras</u>, <u>lasagne</u>, and <u>Tensorflow</u>.

#### **ACTIVITY: INTERVIEWS**



#### **BREAK INTO PAIRS**

- 1. Take turns interviewing each other for a role at your current company:
  - 1. Ask detailed questions about the person's background
  - 2. How would they solve real-world problems in your industry?
  - 3. Ask about their final project and obstacles they have overcome
- 2. Provide feedback!

#### CONCLUSION

- ▶ Data science results are often incorporated into a larger final product.
- These final products including pipelines and models need to be maintained and changes over time need to be addressed.
- ▶ Maintaining complex models includes considering multiple logistical and ethical considerations.

#### CONCLUSION

- Alternative languages used in data science include R or Java/Scala (although Python has many advantages).
- ▶ Visualization skills are vital to communicate and improve your models!
- Advanced machine learning methods you should explore include Bayesian methods and deep learning.

## BEFORE NEXT CLASS

## Final Presentation

#### **LESSON**

## Q&A

#### **LESSON**

### EXIT TICKET

DON'T FORGET TO FILL OUT YOUR EXIT TICKET