Evaluating Predictions

Data Science & Machine Learning Team

09/25/2020

Andrew Wheeler, PhD

andrew.wheeler@hms.com

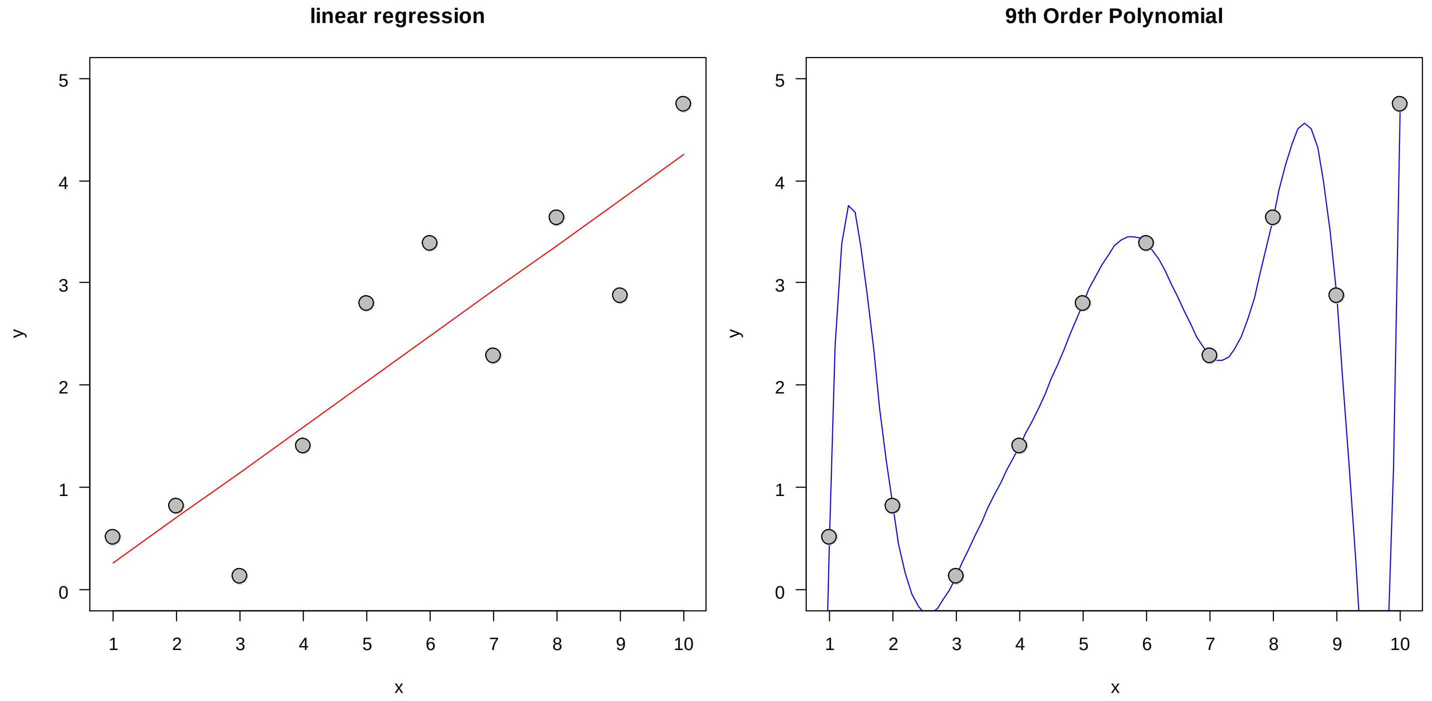
Agenda

- Train/Test approach
  - Example overfitting to in sample
- Weighing False Positives/False Negatives
- AUC and ROC Curves
  - Positive Predictive % based on prevalence
- Simple models as baseline
  - Predicting most common class
  - Mean prediction and linear regression
- Example Out of sample comparison in Python

Evaluating Models

- We need to fit a model to current data, but want to get the best predictions we can for future data.

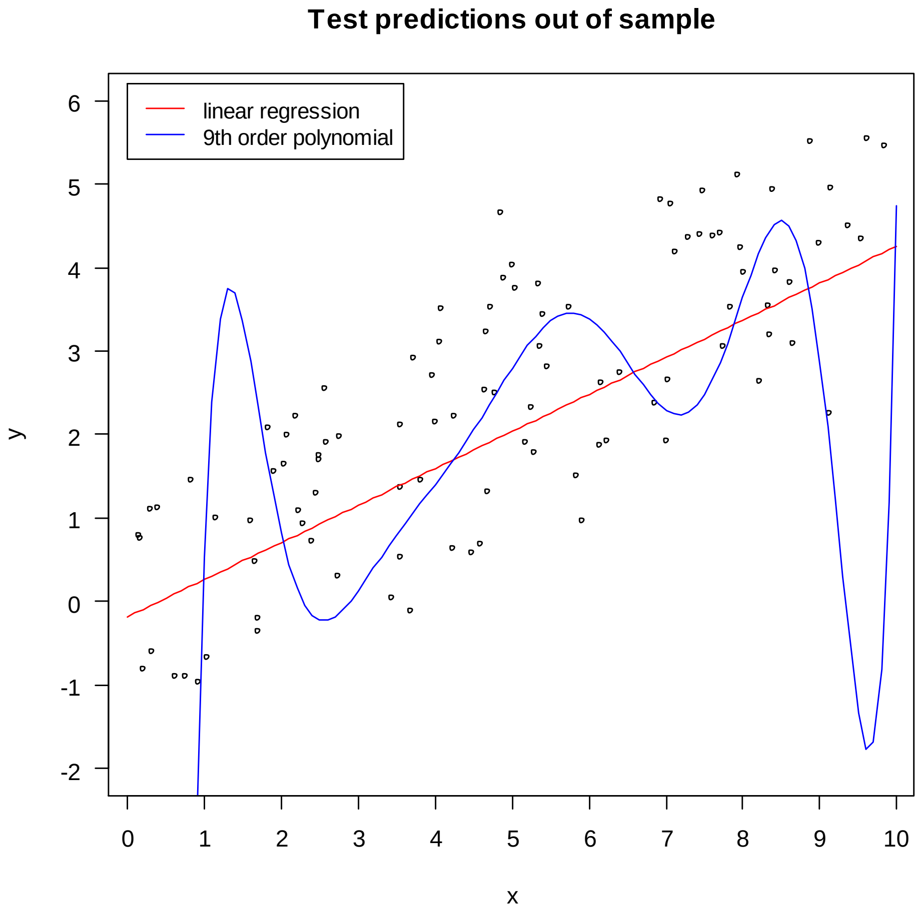
- We can perfectly predict current data, but it will overfit



The 9th order polynomial function is a perfect fit to the sample data.

Evaluating Models

- Solution: have a training dataset to fit the model, and a testing dataset to see how well the predictions do *out of sample*



The 9th order polynomial model has much larger errors on the testing dataset than the simpler linear regression

Weighing False Positives & False Negatives

- Many problems we are predicting a binary outcome (e.g. Overpayment vs Claim is Correct)

- We can then think of four different outcomes:
  - Correctly guess a claim had an overpayment (True Positive)
  - Guess claim was overpayed, but is not (False Positive)
  - Guess claim was correctly paid and it is correctly paid (True Negative)
  - Guess claim was correctly paid, but it is an overpayment (False Negative)

- These each have different costs and benefits of each outcome.
  - E.g. False Positives are wasting ours (and/or clients) time
  - False Negatives are leaving potential revenue on the table

Confusion Matrix

	Actual False	Actual True
Predicted False	True Negative (TN)	False Negative (FN)
Predicted True	False Positive (FP)	True Positive (TP)

- Hypothetical Example:
- Score 1,000 claims for probability it is an overpayment:
    - **True Positive Rate** (TP) of model is 90% (proportion of actual overpayments we capture)
    - **False Positive Rate** (FP) is 5% (proportion of cases that aren't overpayments our model incorrectly flags as overpayments)
    - **Prevalence** of match is 20% (overall proportion of overpayments, positive mix %)
    - **Accuracy** is the proportion of cases we predict correctly, (TP + TN)/Cases

Confusion Matrix

	Actual False	Actual True
Predicted False	True Negative (TN)	False Negative (FN)
Predicted True	False Positive (FP)	True Positive (TP)

$$\frac{TP}{(TP + FN)}$$

- Hypothetical Example:
- Score 1,000 claims for probability it is an overpayment:
  - **True Positive Rate (TP)** of model is 90% (proportion of actual overpayments we capture)
  - **False Positive Rate** (FP) is 5% (proportion of cases that aren't overpayments our model incorrectly flags as overpayments)
  - **Prevalence** of match is 20% (overall proportion of overpayments, positive mix %)
  - **Accuracy** is the proportion of cases we predict correctly, (TP + TN)/Cases

Confusion Matrix

	Actual False	Actual True
Predicted False	True Negative (TN)	False Negative (FN)
Predicted True	False Positive (FP)	True Positive (TP)

$$\frac{FP}{(TN + FP)}$$

- Hypothetical Example:
- Score 1,000 claims for probability it is an overpayment:
  - **True Positive Rate (TP)** of model is 90% (proportion of actual overpayments we capture)
  - **False Positive Rate** (FP) is 5% (proportion of cases that aren't overpayments our model incorrectly flags as overpayments)
  - **Prevalence** of match is 20% (overall proportion of overpayments, positive mix %)
  - **Accuracy** is the proportion of cases we predict correctly, (TP + TN)/Cases

Confusion Matrix

	Actual False	Actual True
Predicted False	True Negative (TN)	False Negative (FN)
Predicted True	False Positive (FP)	True Positive (TP)

$$\frac{FN + TP}{\text{Total Cases}}$$

- Hypothetical Example:
- Score 1,000 claims for probability it is an overpayment:
  - **True Positive Rate** **(TP)** of model is 90% (proportion of actual overpayments we capture)
  - **False Positive Rate** (FP) is 5% (proportion of cases that aren't overpayments our model incorrectly flags as overpayments)
  - **Prevalence** of match is 20% (overall proportion of overpayments, positive mix %)
  - **Accuracy** is the proportion of cases we predict correctly, (TP + TN)/Cases

Confusion Matrix

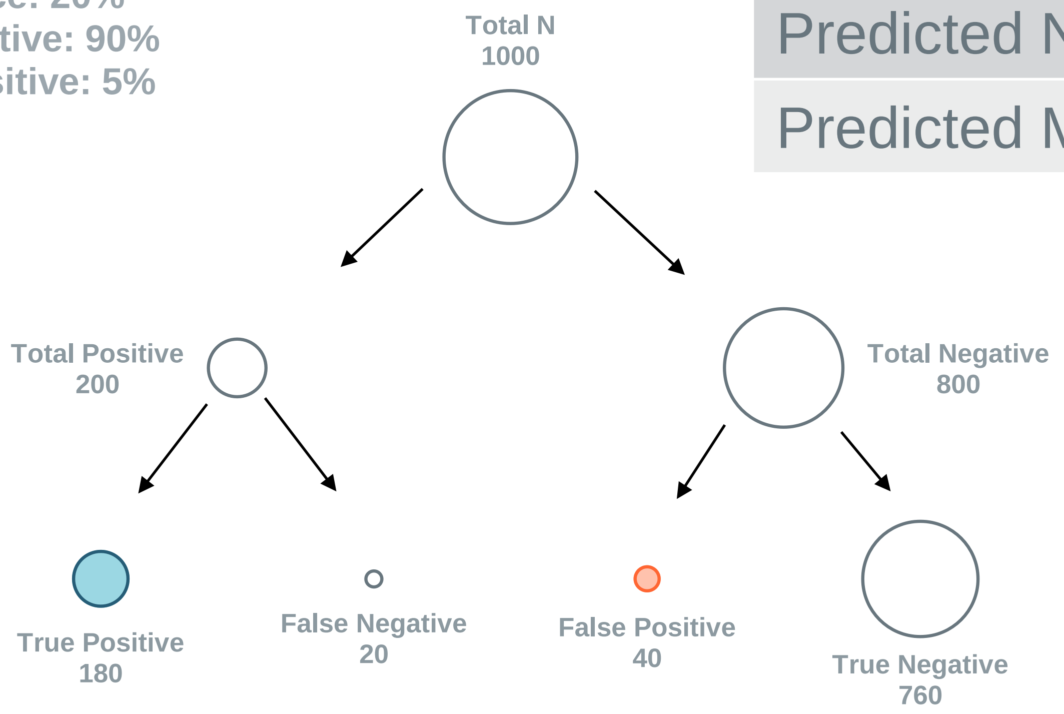
	Actual False	Actual True
Predicted False	True Negative (TN)	False Negative (FN)
Predicted True	False Positive (FP)	True Positive (TP)

<u>TP + TN</u>
Total Cases

- Hypothetical Example:
- Score 1,000 claims for probability it is an overpayment:
  - **<u>True Positive Rate</u> (TP)** of model is 90% (proportion of actual overpayments we capture)
  - **<u>False Positive Rate</u>** (FP) is 5% (proportion of cases that aren't overpayments our model incorrectly flags as overpayments)
  - **<u>Prevalence</u>** of match is 20% (overall proportion of overpayments, positive mix %)
  - **<u>Accuracy</u>** is the proportion of cases we predict correctly, (TP + TN)/Cases

Hypothetical Example

Prevalence: 20%
True Positive: 90%
False Positive: 5%



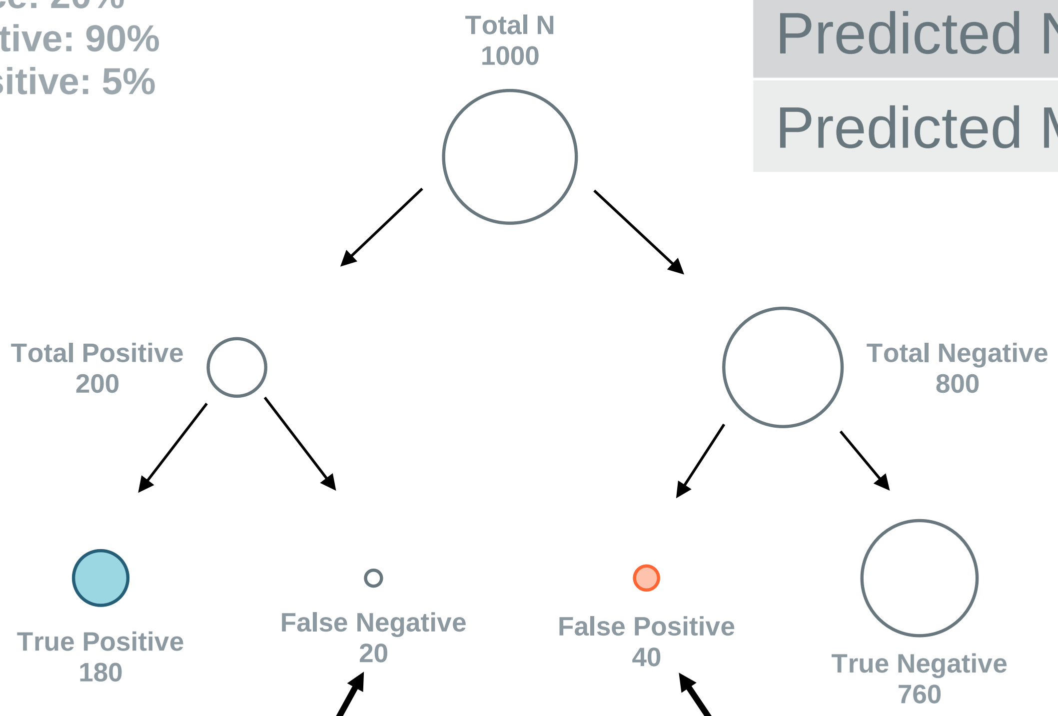
	No Match	Match
Predicted No Match	760	20
Predicted Match	40	180

Accuracy is 94%:

(760 + 180)/1000

Hypothetical Example

Prevalence: 20%
True Positive: 90%
False Positive: 5%



	No Match	Match
Predicted No Match	760	20
Predicted Match	40	180

Accuracy is 94%:

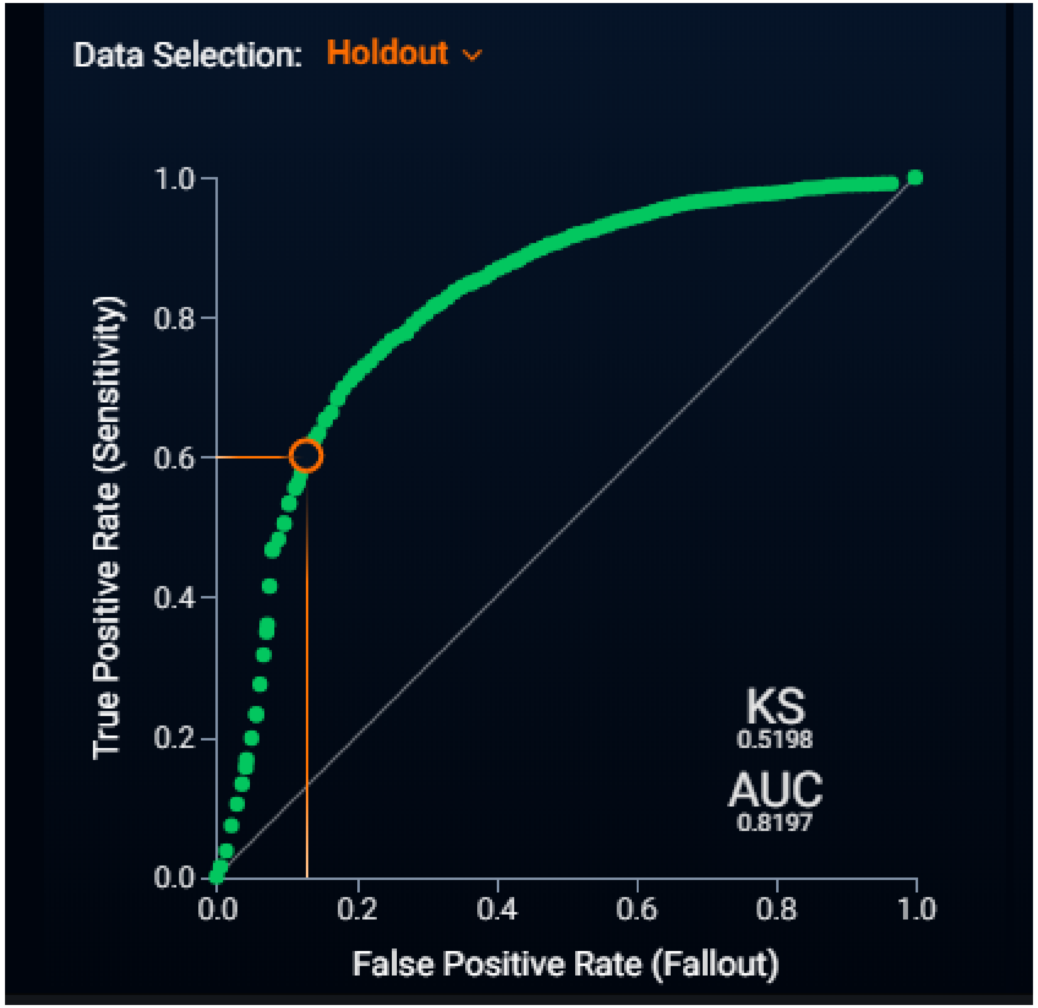
(760 + 180)/1000

Lost Potential Revenue

Wasted Time/Resources

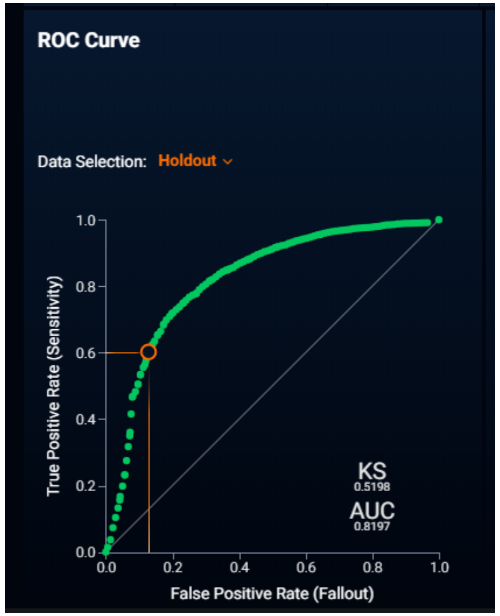
Area Under The Curve (AUC)

- If you flag more claims, you will capture more true positives, but will increase false positives
- ROC curves show this trade-off
- AUC is the area under the curve.
  - 1 is perfect
  - will get 0.5 with random guessing
- Is AUC = 0.82 good enough? Depends on costs/benefits of false positives/true positives
- For cases with extremely low positive mix % (e.g. 5% positive), there might be many more *false positives* than *true positives. In this case accuracy is NOT a good metric.*



Translating AUC to Confusion Tables

- Where to set the threshold depends on costs of false positives and benefits of true positives.
- Tradeoff: different threshold settings yield different accuracy and error rates, for the model with same AUC

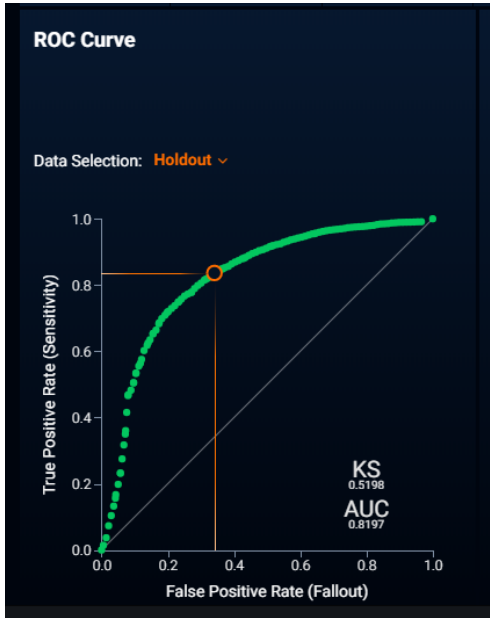


Low Thresh.	Actual False	Actual True
Predicted False	29,349 (TN)	1,962 (FN)
Predicted True	4,318 (FP)	2,969 (TP)

Lower False Positives (13%) and True Positives (60%), Accuracy is 84%

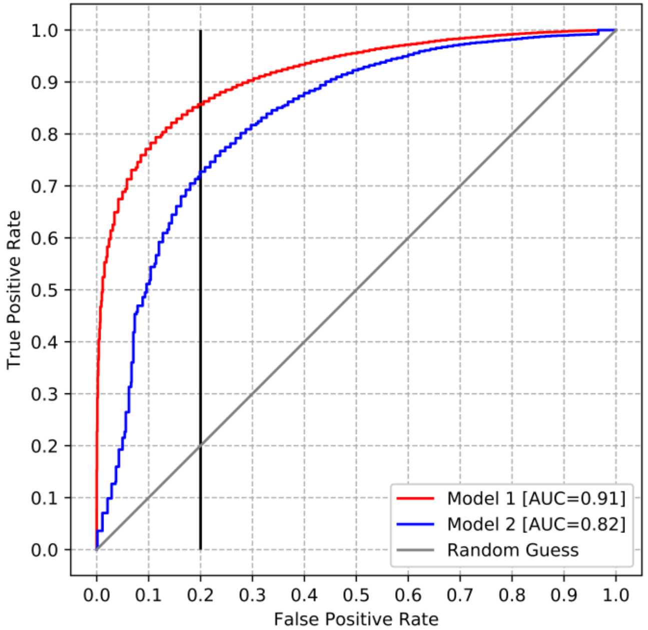
Higher False Positives (34%) and True Positives (84%), Accuracy is 68%

High Thresh.	Actual False	Actual True
Predicted False	22,193 (TN)	807 (FN)
Predicted True	11,474 (FP)	4,124 (TP)



Comparing AUC for Different Models

- Larger AUC values will capture more *true positives* for a given *false positive rate* if the line *is above* the alternative in a ROC chart.
- In most cases, a higher AUC yields higher true positive rates and accuracy given the same false positive rates



Better Model 1 (Red), False Positives (20%) and True Positives (86%), Accuracy 80%

Low Thresh.	Actual False	Actual True
Predicted False	106,943 (TN)	2,823 (FN)
Predicted True	27,727 (FP)	16,900 (TP)

Worse Model 2 (Blue), False Positives (20%) and True Positives (72%), Accuracy 79%

High Thresh.	Actual False	Actual True
Predicted False	107,898 (TN)	5,572 (FN)
Predicted True	26,772 (FP)	14,151 (TP)

Simple Models as a Baseline

- If outcome is rare, predict the most common class.
  - If outcome only happens 1% of the time, if you always guess “No” you will be right 99% of the time.
  - Probably not useful to meet business objectives
  - Need to weigh False Positives vs False Negatives to get a much better predictive model than simple model in that case
- For grouped data (e.g. diagnoses code), can simply predict mean of that group
- Good to start simple (e.g. linear regression), and see how much better more complicated models perform (e.g. random forest)
  - More complicated models need more data to train them

Example Using Python

- What we will be doing today

1) Load in data, create test and train datasets
2) Train a logistic & random forest model
3) Evaluate accuracy of those two models (test)
4) Compare AUC, and accuracy for models

- Example predicting **Failed Restaurant Inspections** in Chicago based on variables such as past number of failures, time since last inspection, garbage nearby, plus others.
- Original data can be downloaded from https://chicago.github.io/food-inspections-evaluation/ (I've limited the number of variables for simplicity.)
- Github link to follow along, https://github.com/hmsholdings/data-science-utils/tree/master/education/Intro_DataScience/Evaluating_Predictions/Analysis

Loading in Data

In [1]:
```
#Loading in the libraries we will be using
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt

#The models
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

#For evaluation
from sklearn.metrics import confusion_matrix, roc_curve, auc
from sklearn.model_selection import train_test_split

#Setting the working directory to where our data is stored
os.chdir(r'C:\Users\e009156\Documents\GitHub\data-science-utils\education\Intro_DataScience\Evaluating_Predictions\Analysis')

#Reading in the CSV data of food inspections
insp_dat = pd.read_csv('FoodInspect.csv')

#A quick view of the first few rows of data
insp_dat.head()
```

The “Inspector” variable designates different areas of Chicago.

Out[1]:

	Inspection_ID	Inspector	pastSerious	pastCritical	timeSinceLast	ageAtInspection	consumption_on_premises_incidental_activity	tobacco_retail_over_counter
0	269961	green	0	0	2.0	1	0	1
1	507211	blue	0	0	2.0	1	0	0
2	507212	blue	0	0	2.0	1	0	0
3	507216	blue	0	0	2.0	1	0	0
4	507219	blue	0	0	2.0	1	0	0

Preparing Variables for Modelling

```
In [2]: #Data Prep

        #We only have a few inspectors, so dummy coding those
        print( insp_dat['Inspector'].value_counts() )
        insp_dum = pd.get_dummies(insp_dat['Inspector'], drop_first=False)
        my_dat = pd.concat([insp_dat, insp_dum], axis=1)

        #variable we are predicting -- if restaurant failed their inspection
        dep_var = 'criticalFound'

        #Inspection ID is not needed for the predictive model
        drop_vars = ['Inspection_ID','Inspector']  #I dont want these variables in the model
        ind_vars =  list( set(my_dat) - set(drop_vars + [dep_var]) )
        print("\nIndependent Variables")
        print(ind_vars)
```

```
green     4940
orange    4068
blue      3434
yellow    3004
brown     1993
purple    1273
Name: Inspector, dtype: int64

Independent Variables
['timeSinceLast', 'temperatureMax', 'ageAtInspection', 'brown', 'consumption_on_premises_incidental_activity', 'orange', 'blu
e', 'yellow', 'pastSerious', 'green', 'heat_sanitation', 'tobacco_retail_over_counter', 'heat_garbage', 'pastCritical', 'purpl
e']
```

Since the “Inspector” variable is categorical, we need to change it to a set of numeric 0/1 (dummy) variables for modelling.

Splitting Train/Test Data & Estimating Models

In [3]:

```
#Now creating a train dataset (70% of the data, ~13,000 cases) and a test dataset (30% of the data, ~5,000 cases)
train, test = train_test_split(my_dat, test_size=0.3)

#Estimating the models on the TRAINING data

#estimating a logistic regression model
logit_model = LogisticRegression(penalty='none', solver='newton-cg',fit_intercept=False)
logit_model.fit(X = train[ind_vars], y = train[dep_var])

#estimating a random forest model
rf_model = RandomForestClassifier(n_estimators=500, max_depth=20, min_samples_leaf=30)
rf_model.fit(X = train[ind_vars], y = train[dep_var])
```

We only use the "train" data to fit the two models.

Out[3]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=20, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=30, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=500,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

Evaluating Predictions (Part 1 – Accuracy)

In [4]:
```python
#Generating Predicted Probabilities on the TEST dataset for each model
pred_probL = logit_model.predict_proba(X = test[ind_vars])[::,1]
pred_probR = rf_model.predict_proba(X = test[ind_vars])[::,1]

#Generating a confusion matrix, setting threshold to predict failed inspection at 30%
th = 0.30
con_matL = pd.DataFrame(confusion_matrix(test[dep_var], pred_probL > th),
                        columns=['Predict Pass','Predict Fail'], index=['Pass Inspect', 'Fail Inspect'])
con_matR = pd.DataFrame(confusion_matrix(test[dep_var], pred_probR > th),
                        columns=['Predict Pass','Predict Fail'], index=['Pass Inspect', 'Fail Inspect'])

#The correct guesses are on the diagonal of the confusion matrix
accuracyL = (con_matL.iloc[0,0] + con_matL.iloc[1,1] ) / len(test)
print("Accuracy Logit Model")
print("%.2f" % accuracyL)
print( con_matL )

accuracyR = (con_matR.iloc[0,0] + con_matR.iloc[1,1] ) / len(test)
print("\nAccuracy Random Forest Model")
print("%.2f" % accuracyR)
print( con_matR )
```

```
Accuracy Logit Model
0.83
              Predict Pass  Predict Fail
Pass Inspect          4450           351
Fail Inspect           609           204

Accuracy Random Forest Model
0.83
              Predict Pass  Predict Fail
Pass Inspect          4480           321
Fail Inspect           614           199
```

The overall failure rate in the dataset is 14%, so a simple model of always guessing “Pass” would be 86% accurate.

Evaluating Predictions (Part 2 – AUC)

In [5]:
```
#Evaluating the AUC of the two models, and plot the ROC curves

#Getting the ROC curve statistics
fprL, tprL, threshL = roc_curve(test[dep_var], pred_probL, pos_label=1)
fprR, tprR, threshR = roc_curve(test[dep_var], pred_probR, pos_label=1)

#Calculating the Area Under the Curve for each model
aucL = auc(fprL, tprL)
print("AUC Statistic for Logit Model")
print(round(aucL,2))

aucR = auc(fprR, tprR)
print("\nAUC Statistic for Random Forest Model")
print(round(aucR,2))
```

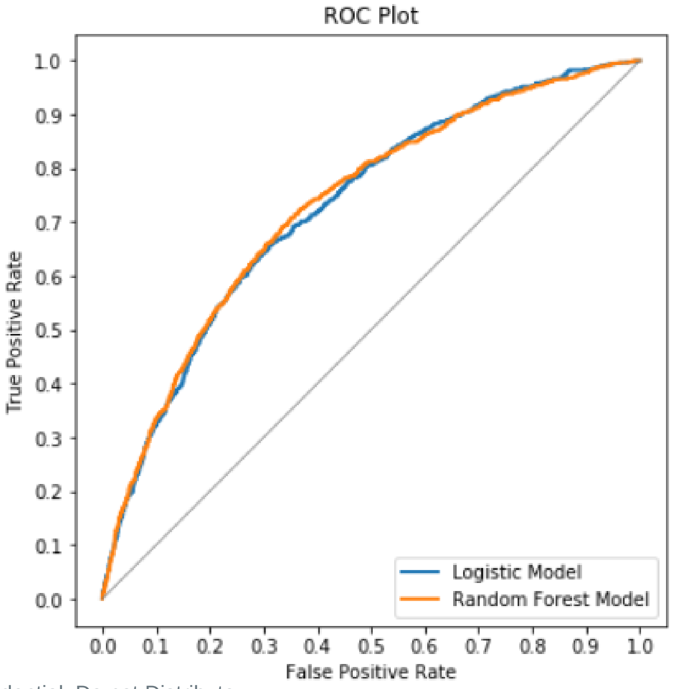
```
AUC Statistic for Logit Model
0.73

AUC Statistic for Random Forest Model
0.73
```

Both models perform very similar when comparing AUC, and both are much better than random (AUC = 0.5)

Evaluating Predictions (Part 3 – ROC Curve)

In [6]:
```
#Now making an ROC graph to illustrate
fig, ax = plt.subplots()
fig.set_size_inches(6,6)
ax.plot(fprL, tprL, drawstyle='steps-post', label='Logistic Model', linewidth=2)
ax.plot(fprR, tprR, drawstyle='steps-post', label='Random Forest Model', linewidth=2)
ax.plot([0,1], [0,1], color='grey', linewidth=0.8) #mid-reference line
ax.set_title("ROC Plot")
plt.xticks(np.arange(0,1.1,0.1))
plt.yticks(np.arange(0,1.1,0.1))
ax.legend(loc='lower right')
ax.set_xlabel('False Positive Rate')
ax.set_ylabel('True Positive Rate')
ax.set_aspect(aspect='equal')
plt.show()
```



In terms of ranking predictions, both models perform equally well.

Since the Logistic regression is simpler than the Random Forest, you may prefer that model.

Future Topics

- Show different cross-validation strategies to evaluate models and various statistics. See “validation-strategies-best-practices” within our DSML goverance docs.

- How to optimize the threshold for binary predictions using cost-benefit analysis.

- Show how to choose the best hyperparameters for Random Forest.

Questions?

Future Topics

Have requests?

Let me know!

Introduction to Data Science Course Outline

Andrew Wheeler, PhD, andrew.wheeler@hms.com

- Lesson 01: Data Science 101
- Lesson 02: Machine Learning 101
- Lesson 03: Evaluating Predictions
- Lesson 04: Intro Data Transformation in Python
- Lesson 05: Data Visualization 101
- Lesson 06: Feature Engineering
- Lesson 07: Missing Data
- Lesson 08: Big Data and Parallel Computing Intro
- Lesson 09: Dimension Reduction and Unsupervised Learning
- Lesson 10: High Cardinality (Many Categories)
- Lesson 11: Intro to Forecasting
- Lesson 12: Conducting Experiments

Evaluating Predictions

Data Science & Machine Learning Team

09/25/2020

Andrew Wheeler, PhD

andrew.wheeler@hms.com