SciComp With Py

Conditional Probability; Train/Test Data Split

Vladimir Kulyukin
Department of Computer Science
Utah State University

Outline

- Review
- Train/Test Data Splits

Review

Events and Probabilities

- A and B are events
- P(A) is the probability of event A occurring
- P(B) is the probability of event B occurring
- P(A, B) is the probability of A and B occurring
- P(A | B) is the probability of A occurring given that B has occurred

Conditional Probability

$$P(B \mid A) = P(A, B)/P(A)$$

Problem

A professor gives his students two tests. 80% of his students pass the 1st test. 70% of his students pass both tests. What percentage of students who pass the 1st test pass the 2nd test?

Solution

- A = student passes 1st test
- B = student passes 2nd test
- P(A, B) = 70%
- P(A) = 80%
- P(B | A) = P(A, B)/P(A) = 0.7/0.8 = 0.84

Train/Test Data Split

Supervised vs Unsupervised Learning

- ML algorithms fall into two broad categories: supervised and unsupervised
- Supervised algorithms work on categorized data, e.g., emails labeled as spam/ham, character images labeled with characters they contain, images with happy/angry faces, etc.
- Unsupervised algorithms work on uncategorized data, e.g., large volumes of text documents that must be clustered into categories according to some criteria

Latent Variables

- Unsupervised learning is considered anytime you *do not* know what you are looking for
- You may want to cluster movies based on their properties to find out if a specific genre is more/less popular
- You may want to cluster texts into categories to find out what words carry more meaning than others
- The things that you discover in unsupervised learning are called *latent variables*

Supervised Learning

- In supervised learning, ML algorithm is trained on labeled/categorized data to construct a predictive model
- The constructed model is used to predict answers for new/unseen values
- Example 1: predicting number of server hits next month from the numbers of server hits in the previous month
- Example 2: predicting prices of new cars from prices of sold cars

Train/Test Data Split

- How do you test how good your supervised model is?
- If you have enough data, a typical approach is to split the data into the training set and the testing set
- Various split ratios are used: 50/50, 70/30, 80/20, and 90/10
- You train your model on the training set (e.g., 70% of your data) and then test it on the training set (e.g., 30% of your data)
- Train/test is a method to minimize overfitting

Can Train/Test Fail?

- Yes, it can!
- Sample sizes may be too small
- Train and test samples may be too similar and your model may appear better than it is
- Overfitting may still happen
- Suggestions? Random sampling when splitting data into training and testing sets

K-fold Cross Validation

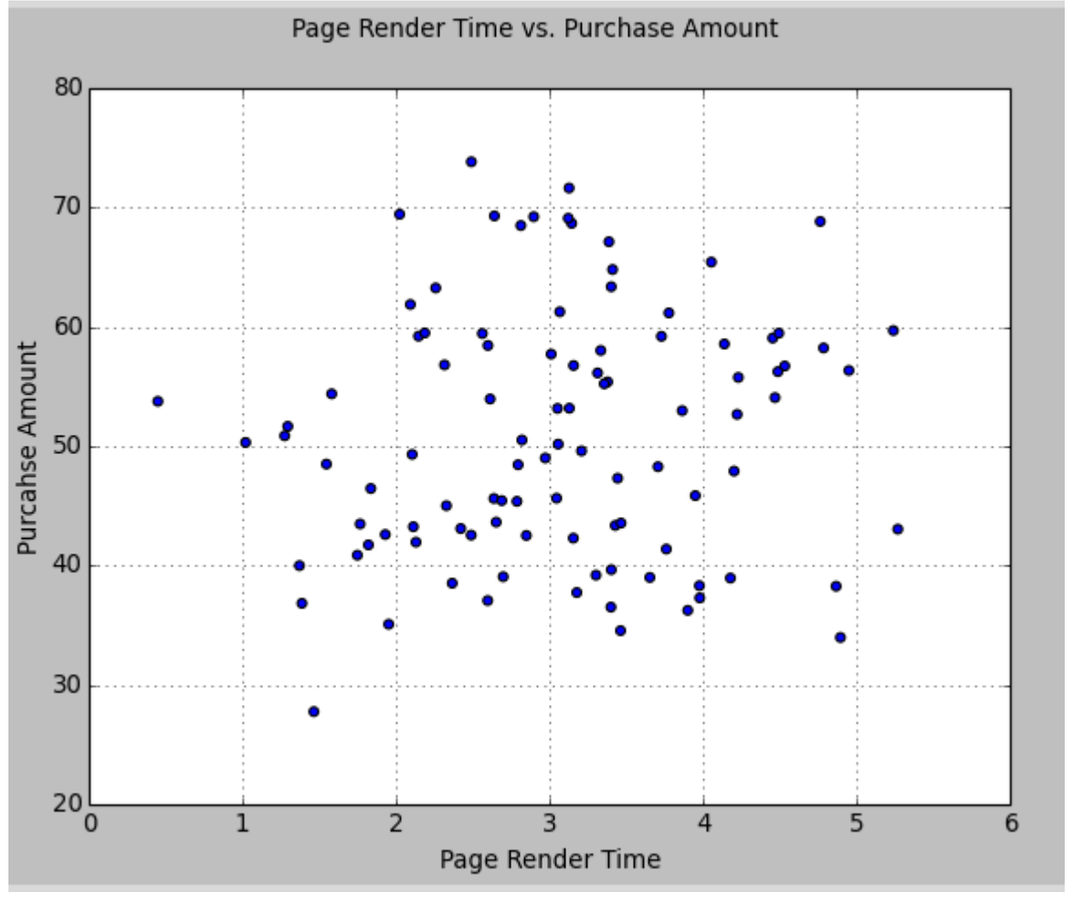
- Split your data into K randomly selected segments (aka folds)
- Reserve one fold for testing and use the other K-1 folds for training
- Train your model on the K-1 folds and test it on the test fold
- Go to step 1 and repeat
- This method minimizes overfitting and underfitting

Problem

You are a data scientist at an e-commerce company. The company is interested in finding a correlation between page rendition time (how fast a page displays to customer) and the amount of money a customer spends on that page. As the data are being collected, you decide to generate some synthetic data and use the 80/20 test/train split to fit a few polynomials over the data.

Simulated Solution

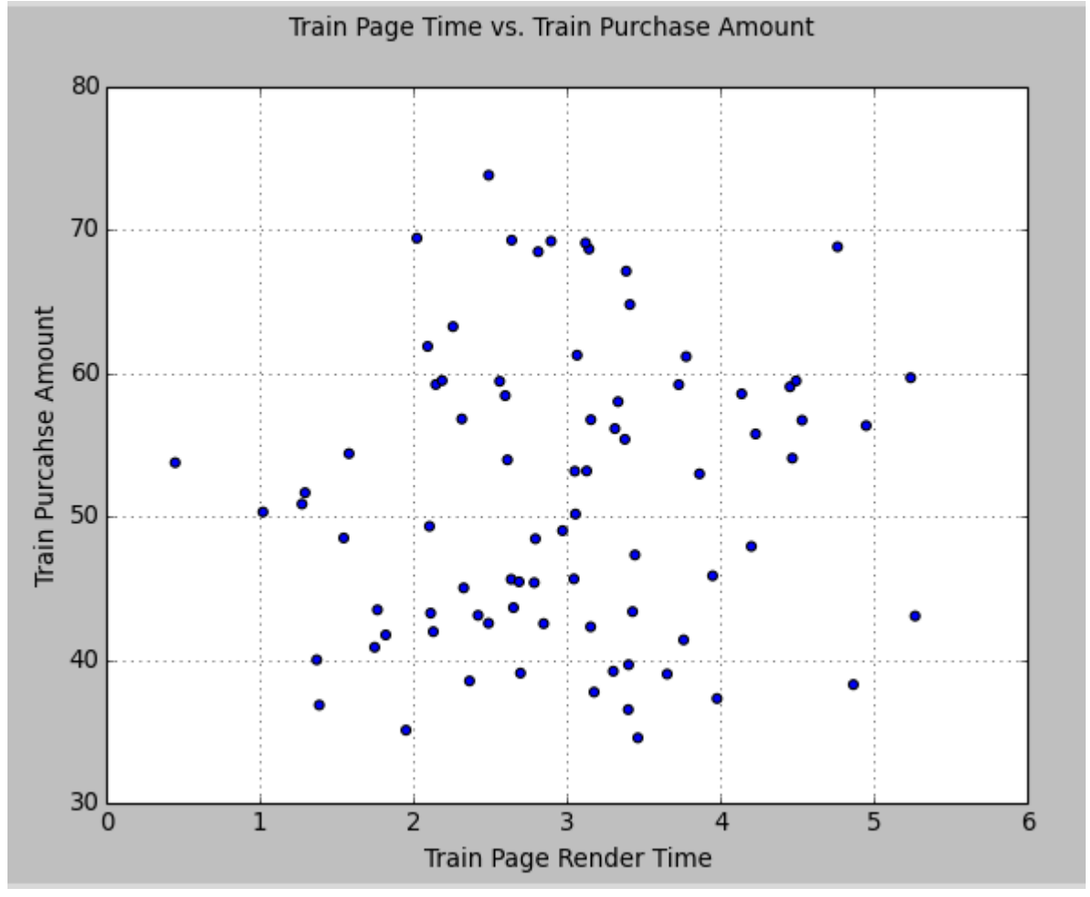
```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(0)
pageRenderTime = np.random.normal(3.0, 1.0, 100)
purchaseAmount = np.random.normal(50.0, 10.0, 100)


fig1 = plt.figure(1)
fig1.suptitle('Page Render Time vs. Purchase Amount')
plt.xlabel('Page Render Time')
plt.ylabel('Purcahse Amount')
plt.grid()
plt.scatter(pageRenderTime, purchaseAmount)
plt.show()
```



source in train_test.py

Simulated Solution: 80/20 Data Split

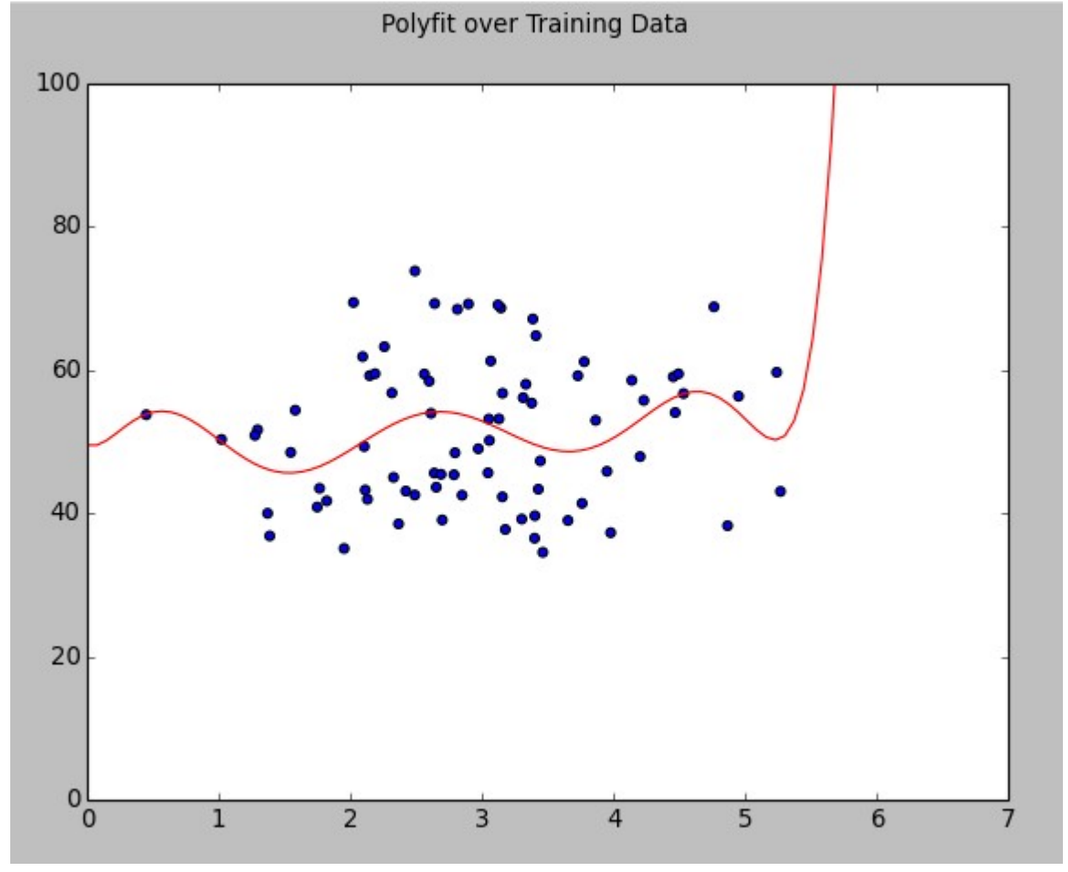
```
trainPageTime = pageRenderTime[:80]

testPageTime = pageRenderTime[80:]


trainPurchaseAmount = purchaseAmount[:80]

testPurchaseAmount = purchaseAmount[80:]


fig2 = plt.figure(2)

fig2.suptitle('Train Page Time vs. Train Purchase Amount')

plt.xlabel('Train Page Render Time')

plt.ylabel('Train Purcahse Amount')

plt.grid()

plt.scatter(trainPageTime, trainPurchaseAmount)
```



source in train_test.py

Simulated Solution: Polyfit over Training Data

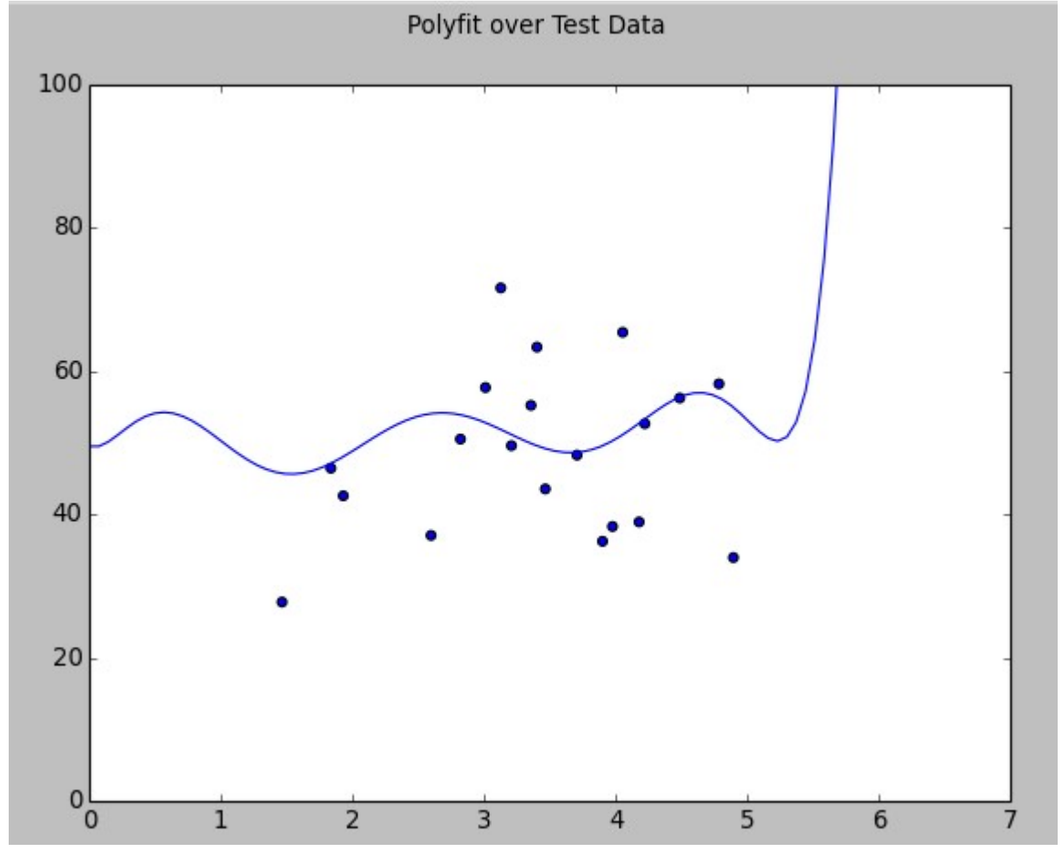
```
trainX = np.array(trainPageTime)

trainY = np.array(trainPurchaseAmount)

polyfit = np.poly1d(np.polyfit(trainX, trainY, 8))

t = np.linspace(0, 7, 100)


fig3 = plt.figure(3)

fig3.suptitle('Polyfit over Training Data')

axes = plt.axes()

axes.set_xlim([0, 7])

axes.set_ylim([0, 100])

plt.scatter(trainX, trainY)

plt.plot(t, polyfit(t), c='r')
```



source in train_test.py

Simulated Solution: Polyfit over Testing Data

```
testX = np.array(testPageTime)

testY = np.array(testPurchaseAmount)

fig4 = plt.figure(4)

fig4.suptitle('Polyfit over Test Data')

axes = plt.axes()

axes.set_xlim([0, 7])

axes.set_ylim([0, 100])

plt.scatter(testX, testY)

plt.plot(t, polyfit(t), c='b')
```



source in train_test.py

Simulated Solution: Goodness of Fit

- How do you estimate goodness of fit?
- You can use the sum of squared errors
- Another common way to evaluate the goodness of fit is to use R^2 score
- R^2 score score estimates the goodness of fit from 1.0 (perfect fit to negative infinity – the smaller it gets the worse it is)

```
>>> from sklearn.metrics import r2_score
>>> r2_score([3, 2, 1], [3, 2, 1])
1.0
>>> r2_score([4, 2, 1], [3, 2, 1])
0.7857142857142857
```

Simulated Solution: Goodness of Fit

```
from sklearn.metrics import r2_score
test_r2 = r2_score(testY, polyfit(testX))
print('Test R2  = %f' % test_r2)


train_r2 = r2_score(trainY, polyfit(trainX))
print('Train R2 = %f' % train_r2)
```

```
Test R2  = 0.019188
Train R2 = 0.075026
```

References

http://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html