Homework 5 - Fixing the pipeline from HW3

Allison Collins

Document Overview

In this file, I will run my latest version of the updated pipeline on the kaggle project funding dataset.

***A few notes and citations:

-- In this version of the pipeline, I leverage some of the "at_k" functions from Professor Ghani's repo to correctly do the threshold (which are also cited in the code), as well as the dictionary and ParameterGrid architecture.

--Feature selection: Given time takes to run all of the models, after trying on a set including additional features, I here include a (slightly smaller) subset of features *Rationale/explanation: this includes removing multiple variables within hierarchy (e.g. sub-focus areas) as well as multiple levels of geography (the most granular, noting that there could be some lack of overlap between train/test on this dimension), and additionally, we remove all of the id features as unique ids are not sensical feature given the unique status

Data cleaning and processing

We will import the pipeline file and a few other modules we might need in cleaning the data (specific to this dataset and the task at hand).

In [1]:
```
import pipeline_revised
import pandas as pd
import pydot
import numpy as np
import warnings
import matplotlib.pyplot as plt
from IPython.display import Image
from pylab import plot, show, savefig, xlim, figure, ylim, legend, boxpl
ot, setp, axes
warnings.simplefilter('ignore')
```

In [2]:
```
#Read in the data
df = pipeline_revised.read_file('projects_2012_2013.csv')
```

In [3]:
```
#Check its contents
df.head(1)
```

Out[3]:

	projectid	teacher_acctid	
0	00001ccc0e81598c4bd86bacb94d7acb	96963218e74e10c3764a5cfb153e6fea	9f3f9f2c2da7edda5648cc

1 rows × 26 columns

For this particular dataset, our target variable is time to funding, so we will need to create an additional column which looks at the time required to get full funding (e.g. the difference between date posted and date fully funded, since this is not explicitly included in the dataset).

In [4]:
```
df['datefullyfunded'] = pd.to_datetime(df.datefullyfunded)
df['date_posted'] = pd.to_datetime(df.date_posted)
df['time_to_funding'] = (df.datefullyfunded - df.date_posted).dt.days
```

In addition, we will have to create a binary outcome "label" column which assigns 1 or 0 based on whether the time to funding was over or under 60; a 1 represents projects which did NOT get funded in the first 60 days.

In [5]:
```
#We will convert the time-to-funding column to be binary as required for
some of the models
df['time_tf'] = np.where(df['time_to_funding']>=60, 1, 0)
```

For this particular dataset, we have a number of categorical columns; in this case, I will opt to drop NAs; with the information we have on hand, it doesn't make sense to impute categories such as a school's focus area or poverty level. I'm subsetting this to columns planned to include in the analysis (as explained above) -- so if there is a na in a different column, we will retain it.

In [6]:
```
df = df.dropna(axis=0,subset=['time_to_funding','total_price_including_o
ptional_support','students_reached',
                              'eligible_double_your_impact_match','pover
ty_level','resource_type','primary_focus_subject','school_metro','teache
r_prefix','school_charter','school_state'])
```

We can now look at some summary stats for the numerical columns with the cleaned dataset we are going to be working with. We can see that the average time to funding is 48 days, with a median of 41, so it is skewed upwards. The average total price is about 670 dollars, and on averge 90 students are reached.

In [7]:
```
pipeline_revised.calc_summary_stats(df,cols_to_include=['time_to_fundin
g','total_price_including_optional_support','students_reached'])
```

Out[7]:

	mean	std_dev	median	max_val	min_val
time_to_funding	48.101857	31.901565	41.00	120.00	5.0
total_price_including_optional_support	659.220428	1138.253237	511.74	164382.84	92.0
students_reached	96.012471	165.467000	31.00	12143.00	1.0

We will next split the data temporally - across 3 sets (based on the 6 month rolling window, and 60 day break

In [8]:
```
predictor_list = ['total_price_including_optional_support','students_rea
ched',
            'eligible_double_your_impact_match','poverty_level','resourc
e_type',
            'primary_focus_subject','school_metro','teacher_prefix','sch
ool_charter',
            'school_state']

#Create April dataset
x_train_a, x_test_a, y_train_a, y_test_a = pipeline_revised.temporal_spl
it(
    df,'date_posted','2013-04-30',180,60,'time_tf', predictors=predictor
_list)

#Create May dataset
x_train_m, x_test_m, y_train_m, y_test_m = pipeline_revised.temporal_spl
it(
    df,'date_posted','2013-05-31',180,60,'time_tf', predictors=predictor
_list)

#Create June dataset
x_train_j, x_test_j, y_train_j, y_test_j = pipeline_revised.temporal_spl
it(
    df,'date_posted','2013-06-30',180,60,'time_tf', predictors=predictor
_list)
```

We will now do some preprocessing of data on the train and test sets separately

In [9]:
```
#We will convert a few columns to use in our analysis - first on the tra
ining data
x_train_a = pipeline_revised.create_binary_col(x_train_a, [('eligible_do
uble_your_impact_match', {'t':1,'f':0}),('school_charter', {'t':1,'f':0
})])
x_train_a = pipeline_revised.discretize_categorical(x_train_a, ['poverty
_level','resource_type','primary_focus_subject','school_metro','teacher_
prefix','school_state'])

x_train_m = pipeline_revised.create_binary_col(x_train_m, [('eligible_do
uble_your_impact_match', {'t':1,'f':0}),('school_charter', {'t':1,'f':0
})])
x_train_m = pipeline_revised.discretize_categorical(x_train_m, ['poverty
_level','resource_type','primary_focus_subject','school_metro','teacher_
prefix','school_state'])

x_train_j = pipeline_revised.create_binary_col(x_train_j, [('eligible_do
uble_your_impact_match', {'t':1,'f':0}),('school_charter', {'t':1,'f':0
})])
x_train_j = pipeline_revised.discretize_categorical(x_train_j, ['poverty
_level','resource_type','primary_focus_subject','school_metro','teacher_
prefix','school_state'])
```

In [10]:
```
#Now we will do the same on the test data, but dropping any columns whic
h do not appear in the training

x_test_a = pipeline_revised.create_binary_col(x_test_a, [('eligible_doub
le_your_impact_match', {'t':1,'f':0}),('school_charter', {'t':1,'f':0
})])
x_test_a = pipeline_revised.discretize_categorical(x_test_a, ['poverty_l
evel','resource_type','primary_focus_subject','school_metro','teacher_pr
efix','school_state'])

x_test_m = pipeline_revised.create_binary_col(x_test_m, [('eligible_doub
le_your_impact_match', {'t':1,'f':0}),('school_charter', {'t':1,'f':0
})])
x_test_m = pipeline_revised.discretize_categorical(x_test_m, ['poverty_l
evel','resource_type','primary_focus_subject','school_metro','teacher_pr
efix','school_state'])

x_test_j = pipeline_revised.create_binary_col(x_test_j, [('eligible_doub
le_your_impact_match', {'t':1,'f':0}),('school_charter', {'t':1,'f':0
})])
x_test_j = pipeline_revised.discretize_categorical(x_test_j, ['poverty_l
evel','resource_type','primary_focus_subject','school_metro','teacher_pr
efix','school_state'])
```

In [11]:
```
#Now we will check and clean columns for values that may be in the test
 or training set only
#which would cause extraneous dummy columns

x_train_a, x_test_a = pipeline_revised.check_col_match(x_train_a, x_test
_a)
x_train_m, x_test_m = pipeline_revised.check_col_match(x_train_m, x_test
_m)
x_train_j, x_test_j = pipeline_revised.check_col_match(x_train_j, x_test
_j)
```

In [12]:
```
pipeline_revised.run_models([('April', x_train_a, x_test_a, y_train_a, y
_test_a), ('May', x_train_m, x_test_m, y_train_m, y_test_m),
                            ('June', x_train_j, x_test_j, y_train_j, y_t
est_j)],
                            models_to_run='all')
```

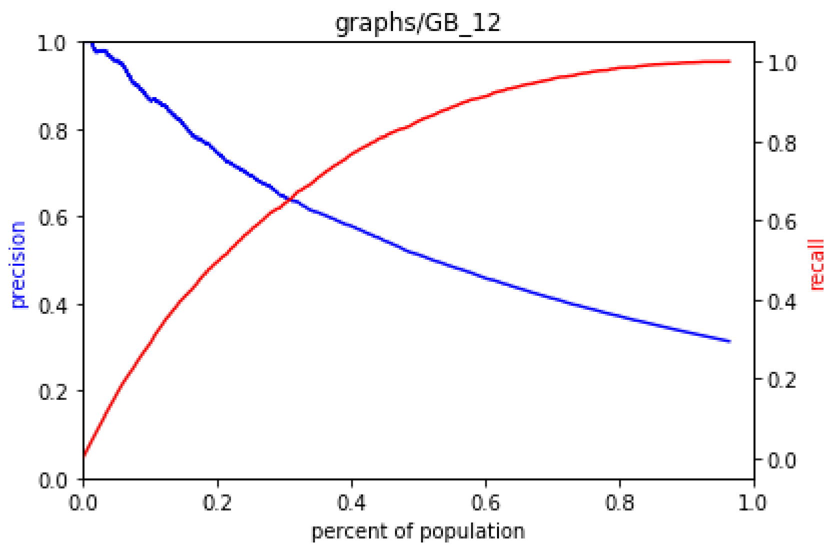
```
<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>
```

```
<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>
```

```
<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>
```

```
<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>
```

```
<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>
```



We can import the csv containing the output of the models' performances

In [14]:
```
results = pd.read_csv('model_results.csv')
results.head(1)
```

Out[14]:

	Unnamed: 0	model_type	clf	date	parameters	train_se
0	0	RF	RandomForestClassifier(bootstrap=True, class_w...	April	{'max_depth': 5, 'min_samples_split': 2, 'n_es...	

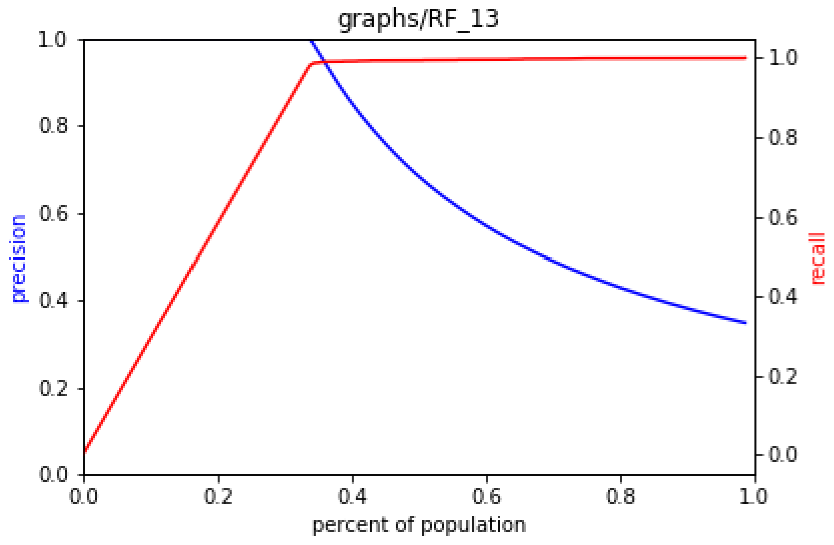
1 rows × 21 columns

Let's also look at a sample precision/recall curve (all are saved in folders in the repo)

This model has the highest AUC and also performs strongly on precision. The precision at the top 5, 10,20, 30 etc are 1 or close to 1 which is why the blue line first appears cut off.

```
In [76]: Image("graphs/RF_13.png")
```

```
Out[76]:
```

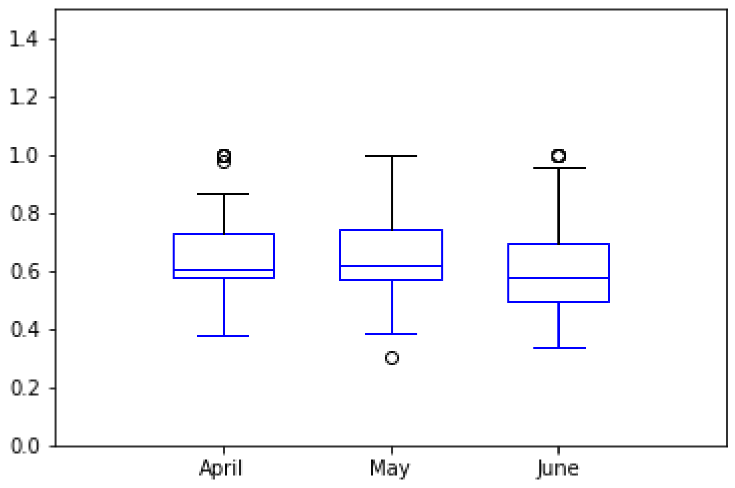


We can create a few graphs to look at performance over time -- I will set up here a few functions to avoid some repeated code

```
In [ ]: #Citation: based off of stack overflow (
        #https://stackoverflow.com/questions/16592222/matplotlib-group-boxplots)
        def setBoxColors(bp):
            setp(bp['boxes'][0], color='blue')
            setp(bp['caps'][0], color='blue')
            setp(bp['whiskers'][0], color='blue')
            setp(bp['fliers'][0], color='blue')
            setp(bp['medians'][0], color='blue')

        def createBPs(g1, g2, g3):
            bp = boxplot(g1, positions = [1], widths = 0.6)
            setBoxColors(bp)
            bp = boxplot(g2, positions = [2], widths = 0.6)
            setBoxColors(bp)
            bp = boxplot(g3, positions = [3], widths = 0.6)
            setBoxColors(bp)
```

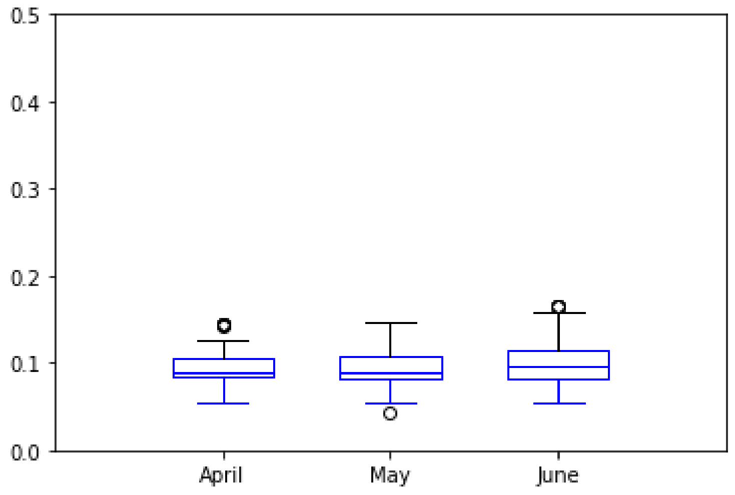
We can now look at precision at 5

In [67]:
```
# Data to plot
g1 = results[results['date']=='April']['precision_at_5']
g2 = results[results['date']=='May']['precision_at_5']
g3 = results[results['date']=='June']['precision_at_5']

fig = figure()
createBPs(g1, g2, g3)
ax = axes()
xlim(0,4)
ylim(0,1.5)
ax.set_xticklabels(['April', 'May', 'June'])
ax.set_xticks([1, 2, 3])
hB.set_visible(False)

savefig('compare/precision_at5.png')
show()
```



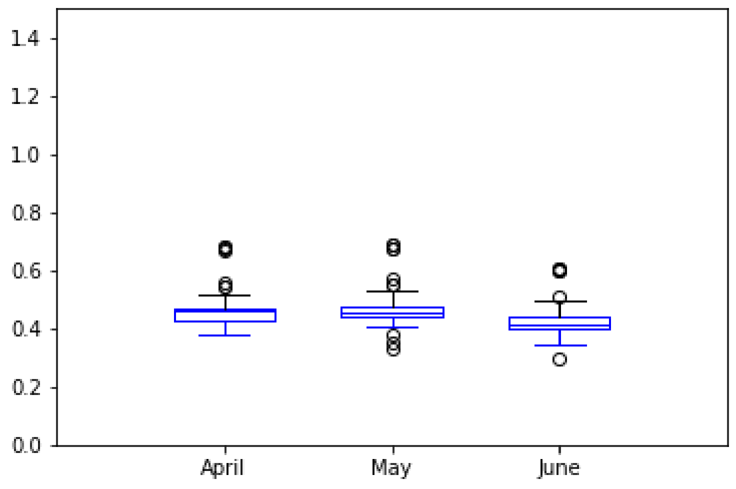
Let's also look at recall at 5

In [69]:
```
# Data to plot
g1 = results[results['date']=='April']['recall_at_5']
g2 = results[results['date']=='May']['recall_at_5']
g3 = results[results['date']=='June']['recall_at_5']

fig = figure()
ax = axes()
createBPs(g1, g2, g3)
xlim(0,4)
ylim(0,.5)
ax.set_xticklabels(['April', 'May', 'June'])
ax.set_xticks([1, 2, 3])
hB.set_visible(False)

savefig('compare/recall_at_5')
show()
```



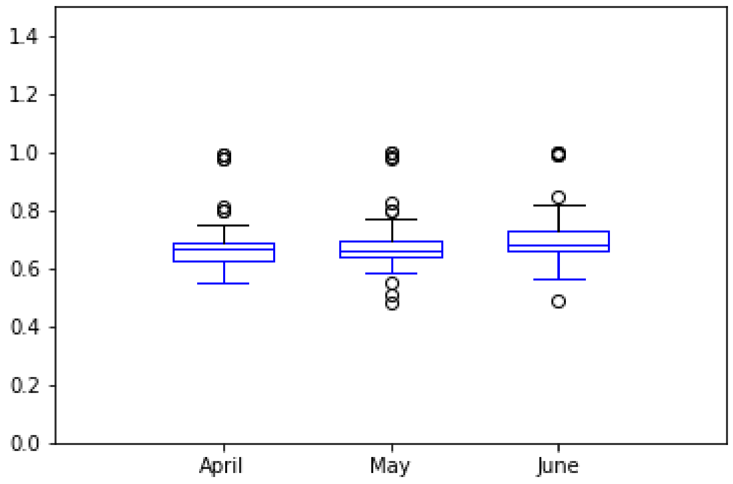
Let's look at the precision on the higher end of k, 50

```
In [73]: # Data to plot
         g1 = results[results['date']=='April']['precision_at_50']
         g2 = results[results['date']=='May']['precision_at_50']
         g3 = results[results['date']=='June']['precision_at_50']

         fig = figure()
         createBPs(g1, g2, g3)
         ax = axes()
         xlim(0,4)
         ylim(0,1.5)
         ax.set_xticklabels(['April', 'May', 'June'])
         ax.set_xticks([1, 2, 3])
         hB.set_visible(False)

         savefig('compare/precision_at50.png')
         show()
```



Let's also look at recall on the high end of k (recall at k=50)

In [72]:
```
# Data to plot
g1 = results[results['date']=='April']['recall_at_50']
g2 = results[results['date']=='May']['recall_at_50']
g3 = results[results['date']=='June']['recall_at_50']

fig = figure()
ax = axes()
createBPs(g1, g2, g3)
xlim(0,4)
ylim(0,1.5)
ax.set_xticklabels(['April', 'May', 'June'])
ax.set_xticks([1, 2, 3])
hB.set_visible(False)

savefig('compare/recall_at_50')
show()
```



Let's also look at what features were most important. We can pull in the feature list generated from running decision tree analyisis (which tends to be similar across the trees, once we get past the shallow/stump trees, though there total_price is still most significant).

Beyond total price, the others that are important include whether it is eligible for the double your impact match, the number of students reached--the full top 10 is included below.

```
In [40]: features = pd.read_csv('features/DT_50.csv')
         features.head(10)
```

Out[40]:

	Unnamed: 0	Features	Importance
0	0	total_price_including_optional_support	0.511517
1	2	eligible_double_your_impact_match_binary	0.072070
2	1	students_reached	0.059886
3	41	school_metro_rural	0.028909
4	5	poverty_level_highest poverty	0.020476
5	12	resource_type_Trips	0.015622
6	8	resource_type_Books	0.013855
7	45	teacher_prefix_Mrs.	0.013707
8	22	primary_focus_subject_Environmental Science	0.013359
9	43	school_metro_urban	0.012431

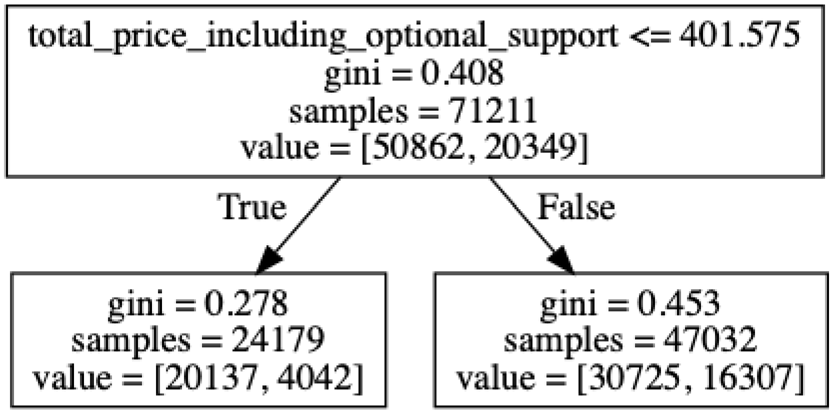
We can also see this through printouts of the tree stumps -- the first node split on is consistently the price:

```
In [57]: (graph,) = pydot.graph_from_dot_file('trees/DT_4')
```

```
In [58]: graph.write_png('trees/DT_4.png')
```

```
In [77]: Image("trees/DT_4.png")
```

Out[77]:



```
In [ ]:
```