Exploratory Flux Predictions

Nicholas Kaufman

May 3, 2015

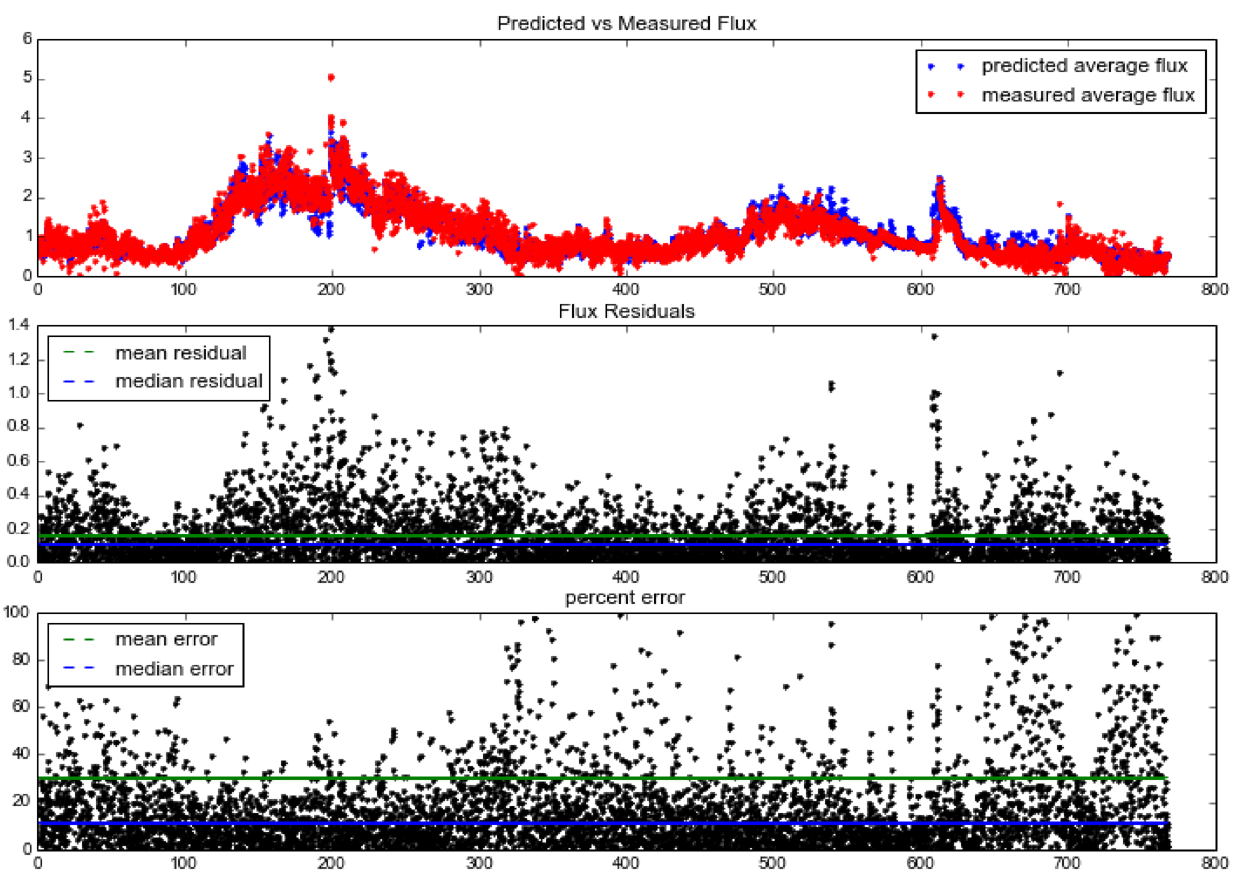
This report will detail the results of predicting average flux emissions using three different models.

Random Forests

The First Model

For the first model, time was eliminated as a training variable. All other factor variables - Air Temperature, three layers of Soil Temperature, Soil Moisture, and Light - were used.

A plot is included below showing some preliminary results.



While training, the R2 score (or the coefficient of determination) is calculated on the training data, and on a cross-validation set. Those values are given below:
R2 score on training data: 0.982474990177
R2 score on 5-folded data: [0.87108384 0.87205808 0.86948272 0.86920592 0.85897424]

Average score across folds: 0.868160959768

R2 obtains a maximum at 1, and is a measure of how well the model is performing on a particular data set, for which we can compare a known baseline.

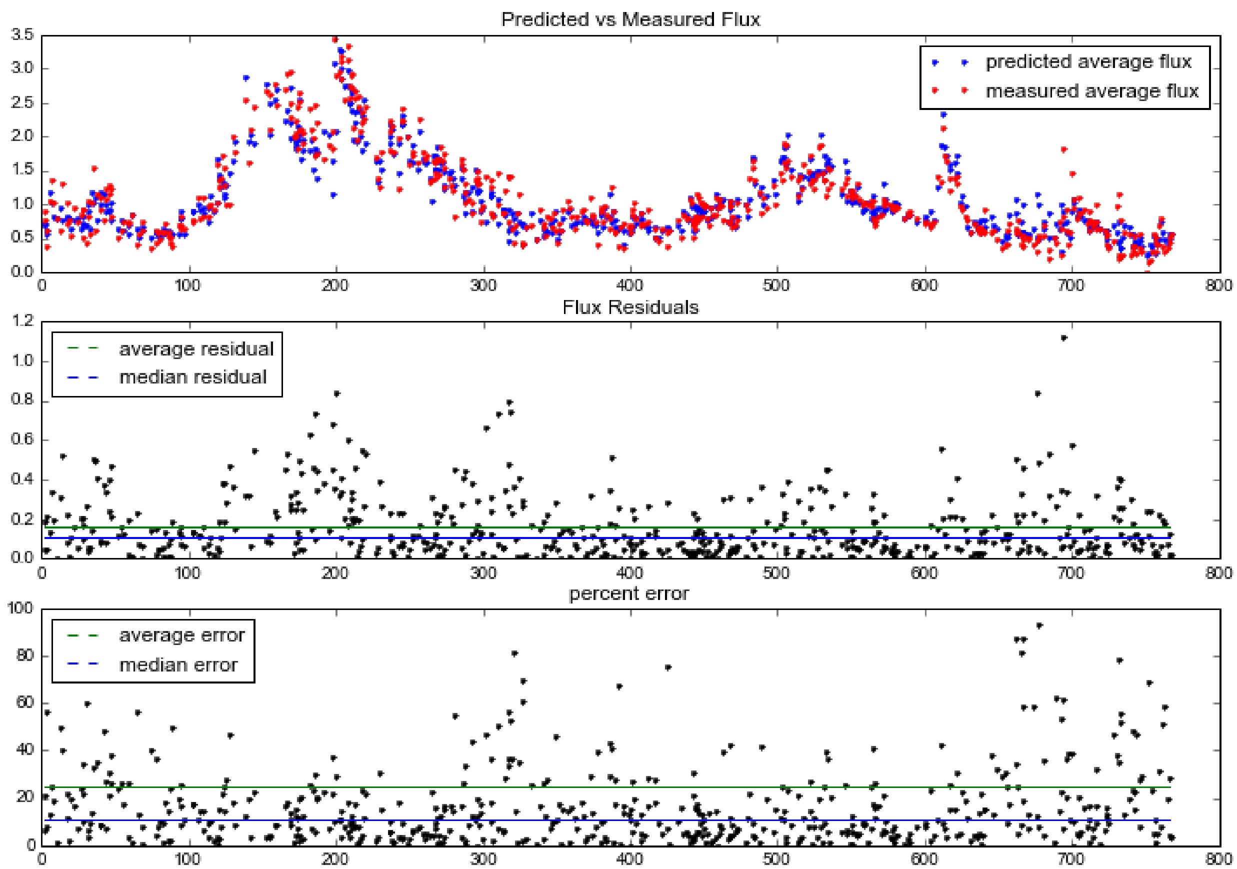
One can also compute the weights the model gives to each of the feauture variables. In this case, those are:

Featrure Importance Vector: [0.04442861 0.59248679 0.0466869 0.03521874 0.25649003 0.02468893]

The higher the number, the more important the weight. From this we see that, by a large margin, the deepest soil temperature and light are the most important features.

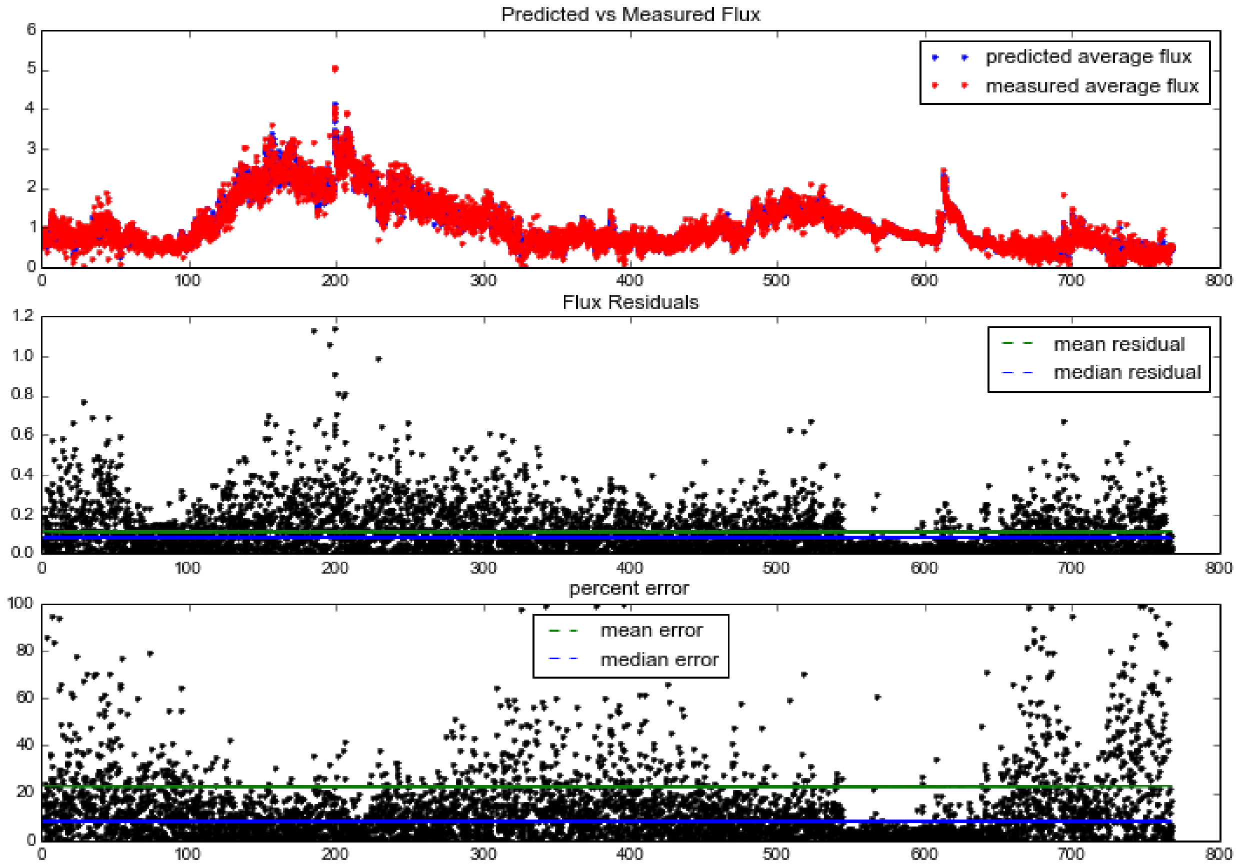
It is interesting to note that, in the predictions plot above, we can see two time distinct time periods in which the models performance is weakest. It is also worthy of mention that since this data is over the span of two years, those areas appear to correspond to roughly the same yearly time range. We can conclude that perhaps it would be useful to include the time as a variable.

In order to present a perhaps cleaner picture, we include the same plots but using only every tenth data point.



The Second Model

Here we include time as a variable, according to the discussion following the previous plot. When training the model with this additional factor variable, the predictions improve.



We notice that the predictions are significantly more accurate here. We include the calculated output as before.

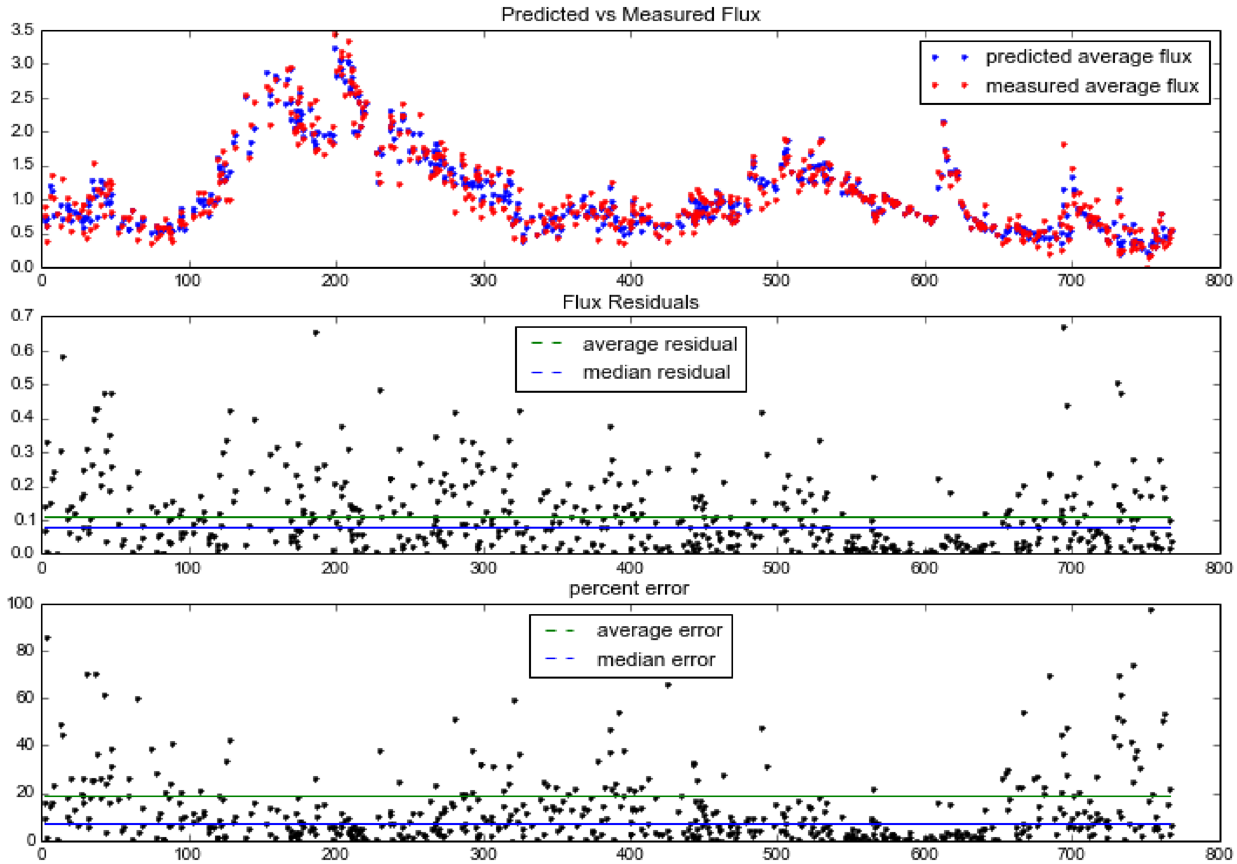
R2 score on training data: 0.990912535056 R2 score on 5-folded data: [0.9345659 0.93356225 0.93235294 0.9353663 0.92977682] Average score across folds: 0.933124842167

R2 score on testing data:

Feature Importance Vector: [0.02230462 0.50997696 0.01256326 0.01180546 0.05988339 0.00664382 0.37682248]

From this, we see that light is now no longer an important feature. Time as taken it's place by a large margin as the penultimate feature.

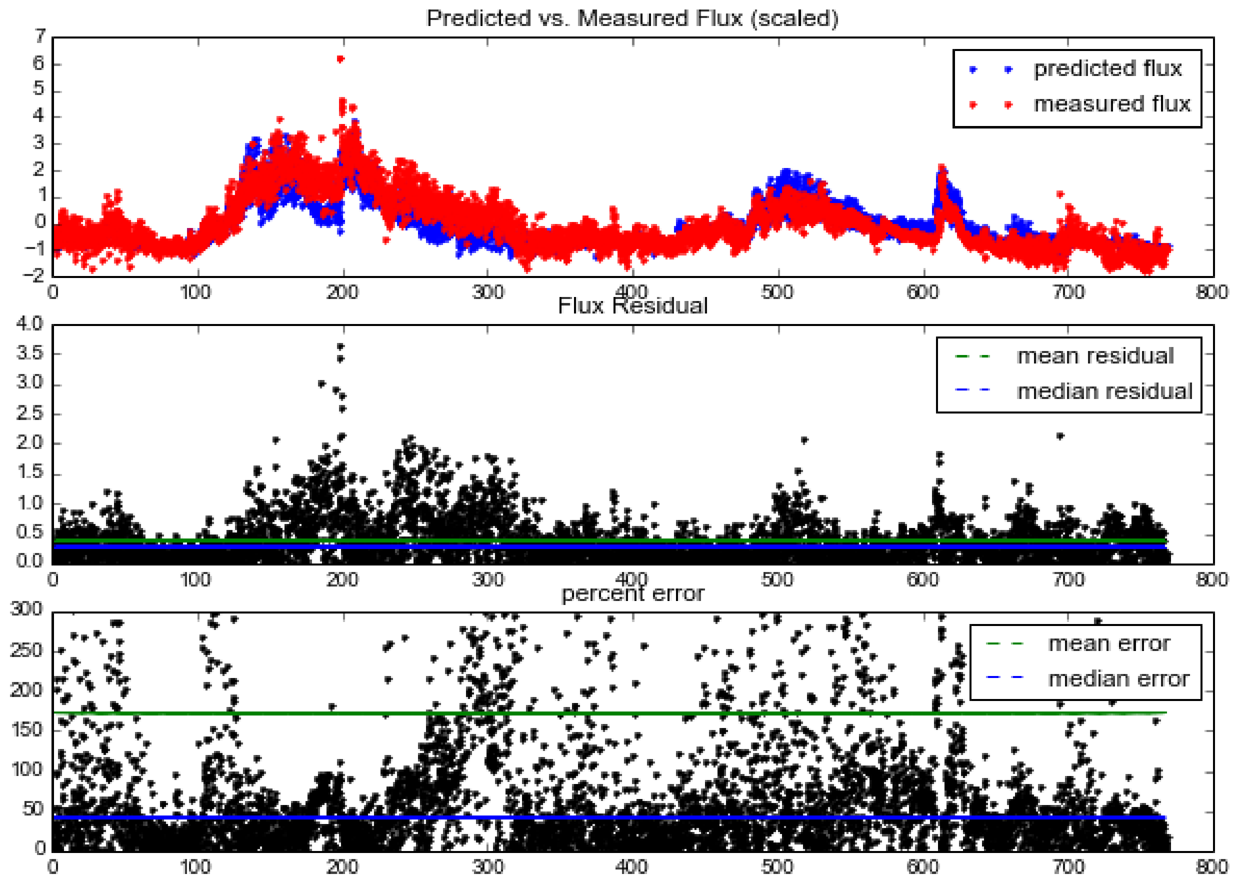
Again, we plot the results using a tenth of the data for a neater picture.



1 Support Vector Machine

1.1 The Third Model

This model's performance is quite inferior to the previous models. The results are included because they highlight the importance of time as a variable. Note that it is impossible to use time as a variable in SVM models because the model requires the data to be scaled to fit the Standard Distribution, per feature. This is obviously infeasible to do with time.



Here we can very clearly see the two regions in time in which the prediction most notably breaks down. I believe that given more feature variables, we can highlight which attributes are most significant in predicting the average flux. Further, I believe that it would be possible to use similar methodologies to predict the amount of flux for varying gasses based on a larger range of dependent variables.

2 Source Code

```
#Flux prediction
#data courtesy of Dr. Rodrigo Vargas
#Last modifed: 5/2/15

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.cross_validation import KFold, cross_val_score, train_test_split
from sklearn import ensemble
from sklearn.svm import SVR

#read in the data
raw_vars = pd.read_csv('vargas_raw_data.csv')
raw_data = np.array(raw_vars.values)

bad_ind = []
for i in enumerate(raw_vars['average_flux']):
```

```
    if (np.isnan(i[1]) or np.isnan(raw_vars['air_temp_avg'][i[0]])  \
    or np.isnan(raw_vars['deep_soil_temp'][i[0]]) \
    or np.isnan(raw_vars['mid_soil_temp'][i[0]]) \
    or np.isnan(raw_vars['shallow_soil_temp'][i[0]]) \
    or np.isnan(raw_vars['soil_moisture'][i[0]]) \
    or np.isnan(raw_vars['light'][i[0]]) or np.isnan(raw_vars['time'][i[0]])) :
        bad_ind.append(i[0])

raw_data = np.delete(raw_data, bad_ind, axis=0)

#we split the data/target into two different types:
#type 1: time is not a feature.
#need to carry time until you can after splitting for training/testing
#type 2: time is a feature.

data_type1 = raw_data[:,1:7]
target_type1 = raw_data[:,[0,7]]
data_type2 = raw_data[:,1:]
target_type2 = raw_data[:,0]

#split data up into training/testing for each type
X_t1_train, X_t1_test, y_t1_train, y_t1_test = train_test_split(data_type1,\
    target_type1, test_size=.3, random_state=42)

X_t2_train, X_t2_test, y_t2_train, y_t2_test = train_test_split(data_type2,\
    target_type2, test_size=.3, random_state=42)
#pull out the time vectors for the type 1 data, and excise it from target
time_t1_train = y_t1_train[:,1]
y_t1_train = y_t1_train[:,0]
time_t1_test = y_t1_test[:,1]
y_t1_test = y_t1_test[:,0]

#I don't know python coding procedure. where do you put all of your functions?

def scale_data(train_data,test_data,train_target,test_target):
    from sklearn import preprocessing
    scaler_X = preprocessing.StandardScaler().fit(train_data)
    scaler_y = preprocessing.StandardScaler().fit(train_target)
    train_data = scaler_X.transform(train_data)
    test_data = scaler_X.transform(test_data)
    train_target = scaler_y.transform(train_target)
    test_target = scaler_y.transform(test_target)

    return train_data, test_data, train_target, test_target

X_svm_train, X_svm_test, y_svm_train, y_svm_test = scale_data \
    (X_t1_train, X_t1_test, y_t1_train, y_t1_test)

def train_and_evaluate(clf, X, y):
    clf.fit(X,y)
    print "R2 score on training data: ", clf.score(X,y)

    kf = KFold(X.shape[0], n_folds = 5, shuffle = True, random_state = 42)
    score = cross_val_score(clf, X,y,cv=kf)
```

```
    print "R2 score on 5-folded data: ", score
    print "Average score across folds: ", np.mean(score)

rf_t1 = ensemble.RandomForestRegressor(n_estimators=75, random_state=42)
rf_t2 = ensemble.RandomForestRegressor(n_estimators=75, random_state=33)
svm_t1 = SVR()

print "training random forest 1 model..."
train_and_evaluate(rf_t1, X_t1_train, y_t1_train)

print "training random forest 2 model..."
train_and_evaluate(rf_t2, X_t2_train, y_t2_train)

print "training svm..."
train_and_evaluate(svm_t1, X_svm_train, y_svm_train)

#predict flux values using each model
rf_t1_pred = rf_t1.predict(X_t1_test)
rf_t2_pred = rf_t2.predict(X_t2_test)
svm_pred = svm_t1.predict(X_svm_test)

#compute plotting vectors, per model
rf_t1_residual = np.array(rf_t1_pred - y_t1_test)
rf_t1_error = np.array(np.abs(rf_t1_residual) / y_t1_test)
rf_t2_residual = np.array(rf_t2_pred - y_t2_test)
rf_t2_error = np.array(np.abs(rf_t2_residual) / y_t2_test)
svm_residual = np.array(svm_pred - y_svm_test)
svm_error = np.array(np.abs(svm_residual / y_svm_test))

#Now we generate plots.
f, ax = plt.subplots(figsize=(10,7), nrows=3)

ax[0].plot(time_t1_test, rf_t1_pred, 'b.', label='predicted flux')
ax[0].plot(time_t1_test, y_t1_test, 'r.', label='measured flux')
ax[0].set_title('Predicted vs Measure Flux')
ax[0].legend(loc='best')
ax[1].plot(time_t1_test, np.abs(rf_t1_residual), 'k.')
ax[1].plot(time_t1_test, np.ones(len(time_t1_test)) \
    *np.mean(np.abs(rf_t1_residual)), 'g--', label='average residual')
ax[1].plot(time_t1_test, np.ones(len(time_t1_test)) \
    *np.median(np.abs(rf_t1_residual)), 'b--', label='median residual')
ax[1].set_title('Flux Residual (abs diff of predicted and measured)')
ax[1].legend(loc='best')
ax[2].plot(time_t1_test, rf_t1_error*100, 'k.')
ax[2].plot(time_t1_test,np.ones(len(time_t1_test))*np.mean(rf_t1_error*100)\
    ,'g--', label="average error")
ax[2].plot(time_t1_test,np.ones(len(time_t1_test))*np.median(rf_t1_error*100)\
    ,'b--', label="median error")
ax[2].set_title('Percent Error in flux prediction values')
ax[2].legend(loc='best')

f.show()

g, ay = plt.subplots(figsize=(10,7), nrows=3)
```

```

ay[0].plot(X_t2_test[:,-1:], rf_t2_pred, 'b.', label='predicted flux')
ay[0].plot(X_t2_test[:,-1:], y_t2_test, 'r.', label='measured flux')
ay[0].set_title('Predicted vs Measure Flux')
ay[0].legend(loc='best')
ay[1].plot(X_t2_test[:,-1:], np.abs(rf_t2_residual), 'k.')
ay[1].plot(X_t2_test[:,-1:], np.ones(len(X_t2_test[:,-1:])) \
   *np.mean(np.abs(rf_t2_residual)), 'g--', label='average residual')
ay[1].plot(X_t2_test[:,-1:], np.ones(len(X_t2_test[:,-1:])) \
   *np.median(np.abs(rf_t2_residual)), 'b--', label='median residual')
ay[1].set_title('Flux Residual (abs diff of predicted and measured)')
ay[1].legend(loc='best')
ay[2].plot(X_t2_test[:,-1:], rf_t2_error*100, 'k.')
ay[2].plot(X_t2_test[:,-1:], np.ones(len(X_t2_test[:,-1:]))\
    *np.mean(rf_t2_error*100) ,'g--', label="average error")
ay[2].plot(X_t2_test[:,-1:], np.ones(len(X_t2_test[:,-1:]))\
    *np.median(rf_t2_error*100) ,'b--', label="median error")
ay[2].set_title('Percent Error in flux prediction values')
ay[2].legend(loc='best')

g.show()

h, az = plt.subplots(figsize=(10,7), nrows=3)

ay[0].plot(time_t1_test, svm_pred, 'b.', label='predicted flux')
ay[0].plot(time_t1_test, y_svm_test, 'r.', label='measured flux')
ay[0].set_title('Predicted vs Measure Flux')
ay[0].legend(loc='best')
ay[1].plot(time_t1_test, np.abs(svm_residual), 'k.')
ay[1].plot(time_t1_test, np.ones(len(time_t1_test)) \
   *np.mean(np.abs(svm_residual)), 'g--', label='average residual')
ay[1].plot(time_t1_test, np.ones(len(time_t1_test)) \
   *np.median(np.abs(svm_residual)), 'b--', label='median residual')
ay[1].set_title('Flux Residual (abs diff of predicted and measured)')
ay[1].legend(loc='best')
ay[2].plot(time_t1_test, svm_error*100, 'k.')
ay[2].plot(time_t1_test, np.ones(len(time_t1_test))*np.mean(svm_error*100) \
    ,'g--', label="average error")
ay[2].plot(time_t1_test, np.ones(len(time_t1_test))*np.median(svm_error*100) \
    ,'b--', label="median error")
ay[2].set_title('Percent Error in flux prediction values')
ay[2].legend(loc='best')

h.show()
```