Physics Dataset Practice Problems

Week of July 13, 2020

1 Compton Scattering

1.1 Visualize the Dataset

```
fig, axs = plt.subplots(2)
plt.suptitle('Angular Distribution - {} MV'.format(int(dataset.iloc[0,0])), fontsize=14)
axs[0].scatter(dataset.iloc[0:500, 1], dataset.iloc[0:500, -1])
axs[0].set_xlabel('Polar Angle', fontsize=14)
axs[0].set_ylabel('$\\frac{d\sigma}{d\Omega}$', fontsize=14)
axs[1].scatter(dataset.iloc[0:500, 2], dataset.iloc[0:500, -1])
axs[1].set_xlabel('Azimuthal Angle', fontsize=14)
axs[1].set_ylabel('$\\frac{d\sigma}{d\Omega}$', fontsize=14)
plt.savefig('ang_dist.png', dpi=300)
plt.show()
```

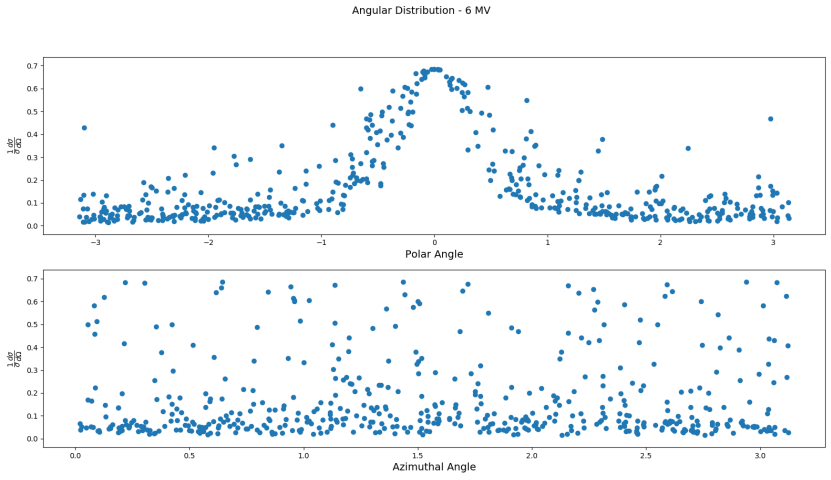
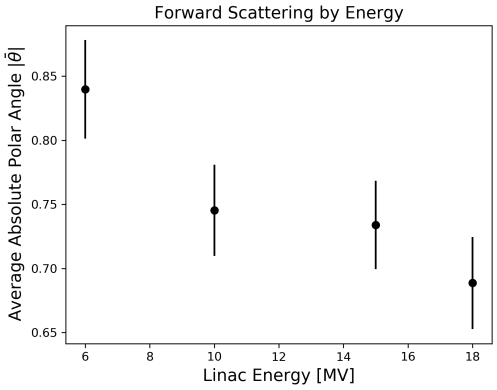
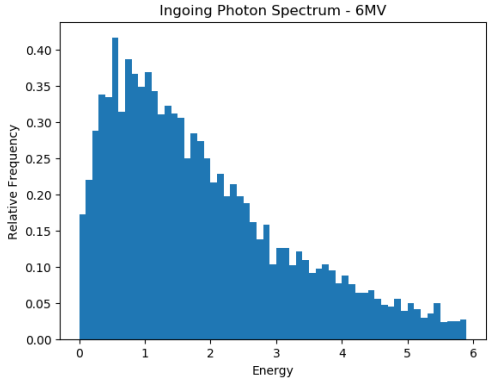


Figure 1: A 6X beam prefers Compton scattering in the forward direction.

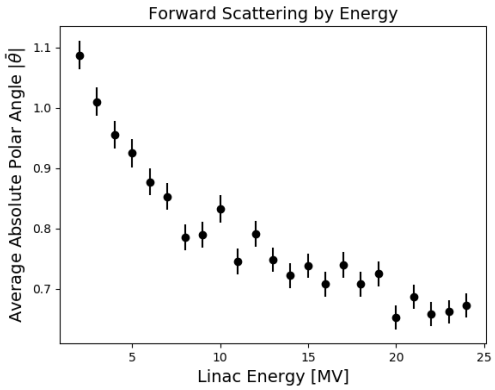
```
avgs = []
stds = []
it_idx = 0
for i in range(len(energies)):
    cs = dataset.iloc[it_idx:it_idx + particles,-1]
    t = np.abs(dataset.iloc[it_idx:it_idx + particles,1])
    avg = np.average(t, weights=cs)
    avgs.append(avg)
    stds.append(np.sqrt(np.average((t - avg)**2, weights=cs))/ np.sqrt(particles))
    it_idx += particles
fig, ax = plt.subplots()
plt.errorbar(energies, avgs, yerr=stds, fmt='ko')
plt.title('Forward Scattering by Energy', fontsize=14)
plt.xlabel('Linac Energy [MV]', fontsize=14)
plt.ylabel('Average Absolute Polar Angle $|\\bar{\\theta}|$', fontsize=14)
plt.savefig('forward_scattering.png', dpi=300)
plt.show()
```



This is a synthetic dataset, in which photon energies are sampled from an energy distribution like the one pictured below.



To further illustrate the energy dependence, we can simulate beams with nominal energies `[i for i in range(2,25)]` (in MV).



1.2 Build a Model

1.3 Training and Evaluating Models

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
np.set_printoptions(precision=2)

dataset = pd.read_csv('./compton.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values


#Preprocessing
```

```
from sklearn.model_selection import train_test_split
y = np.expand_dims(y, axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)

from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler(); sc_y = StandardScaler()
X_train = sc_X.fit_transform(X_train); y_train = sc_y.fit_transform(y_train)


#Linear Regression
from sklearn.linear_model import LinearRegression
linear_regressor = LinearRegression()
linear_regressor.fit(X_train, y_train)

#Support Vector Regression
from sklearn.svm import SVR
svr_regressor = SVR(kernel = 'rbf')
svr_regressor.fit(X_train, y_train.ravel())

#Random Forest Regression
from sklearn.ensemble import RandomForestRegressor
rf_regressor = RandomForestRegressor(n_estimators = 100)
rf_regressor.fit(X_train, y_train.ravel())

#Evaluating on Test Points
regressors = {'Linear Regression' : linear_regressor,
              'Support Vector Regression': svr_regressor,
              'Random Forest Regression': rf_regressor}
for name, regressor in regressors.items():
    y_hat = sc_y.inverse_transform(regressor.predict(sc_X.transform(X_test)))
    rmse = np.sqrt(np.mean((y_test-y_hat)**2))
    tests = np.array([[12, 0.25, 2], [8, -1, 1]])
    tests_out = sc_y.inverse_transform(regressor.predict(sc_X.transform(tests)))
    print('Model = {}'.format(name))
    print('RMSE = {}'.format(rmse))
    print('Predicted {} for point {}, {}, {}.'.format(tests_out[0], tests[0,0], tests[0,1],
    print('Predicted {} for point {}, {}, {}.'.format(tests_out[1], tests[1, 0], tests[1, 1]
    print()
```

OUTPUTS:

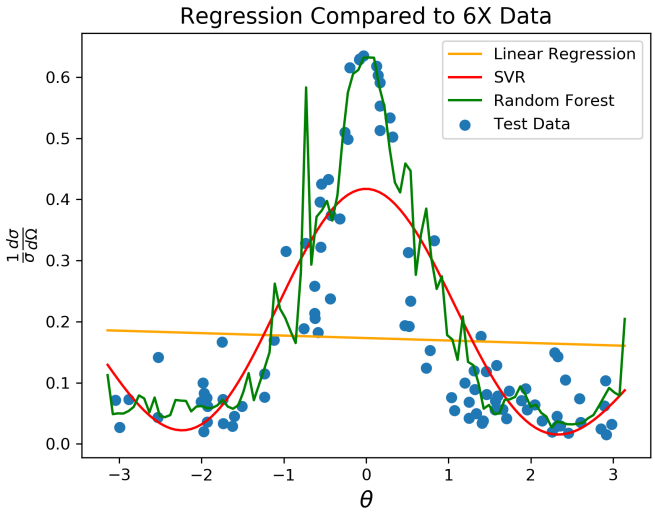
Model = Linear Regression
RMSE = 0.304653633413201
Predicted [0.22] for point 12.0, 0.25, 2.0.
Predicted [0.19] for point 8.0, -1.0, 1.0.

Model = Support Vector Regression
RMSE = 0.3414228012325779
Predicted 0.43014235682871926 for point 12.0, 0.25, 2.0.
Predicted 0.28077222529033397 for point 8.0, -1.0, 1.0.

Model = Random Forest Regression
RMSE = 0.4193699758779647
Predicted 0.7224333840265453 for point 12.0, 0.25, 2.0.
Predicted 0.22198941665800925 for point 8.0, -1.0, 1.0.

1.4 Visualizing the Predictions

```
E = [6. for i in range(100)]
T = np.linspace(-np.pi,np.pi, 100)
P = np.linspace(0, np.pi, 100)
for_plotting = sc_X.transform(np.array([E,T,P]).T)
linear_pred = sc_y.inverse_transform(linear_regressor.predict(for_plotting))
svr_pred = sc_y.inverse_transform(svr_regressor.predict(for_plotting))
rf_pred = sc_y.inverse_transform(rf_regressor.predict(for_plotting))
fig, ax = plt.subplots()
ax.plot(T, linear_pred, color='orange', label='Linear Regression')
ax.plot(T, svr_pred, color='red', label='SVR')
ax.plot(T, rf_pred, color='green', label='Random Forest')
ax.scatter(X_test[np.where(X_test[:,0] == 6.)][:,1], y_test[np.where(X_test[:,0]==6.)], labe
ax.set_xlabel('$\\theta$', fontsize=14)
ax.set_ylabel('$\\frac{1}{\\sigma}\\frac{d\\sigma}{d\\Omega}$', fontsize=14)
plt.title('Regression Compared to 6X Data', fontsize=14)
plt.legend()
plt.savefig('regression.png',dpi=300)
plt.show()
```



To visualize the scattering profile for different Energies, we can change the energy line to something like

```
E = [12.5 for i in range(100)]
```