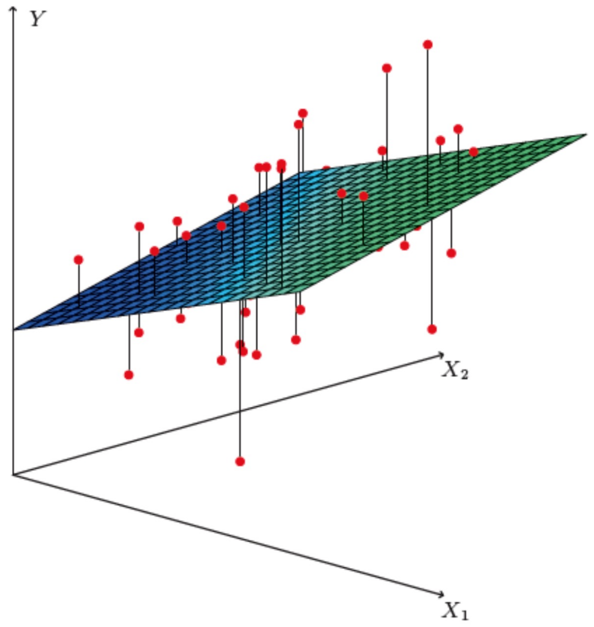
Brief intro to multiple regression

Multiple regression* can be seen as an extension of curve fitting to the case where the variable x is replaced by a multi-dimensional $\boldsymbol{x} = (x_1,\ldots,x_n)$, e.g., fitting a surface. Here suppose the data are points $(\boldsymbol{x}_i, y_i)$, $i = 1,\ldots,N$ (no error bars) and $\boldsymbol{x}$ is usually a random variable, often called the explanatory or predictor variable.



Equivalently, we can view it as an extension to classification with the discrete class label $y = 0, 1$ replaced by a continuous target y (and in this context $\boldsymbol{x}$ can also be called the feature vector).

*Note the term "multivariate" regression refers to a vector target variable y; here we treat only scalar y.

Target (fit) function and loss function

As in the case of curve fitting, we assume some parametric function of $\boldsymbol{x}$ that represents the mean of the target variable

$$E[y] = f(\mathbf{x}; \mathbf{w})$$

where $\boldsymbol{w}$ is a vector of adjustable parameters ("weights").

Suppose we have training data consisting of $(\boldsymbol{x}_i, y_i)$, $i = 1,\ldots,N$.

Use these to determine the weights by minimizing a loss function (analogous to the χ^2), e.g.,

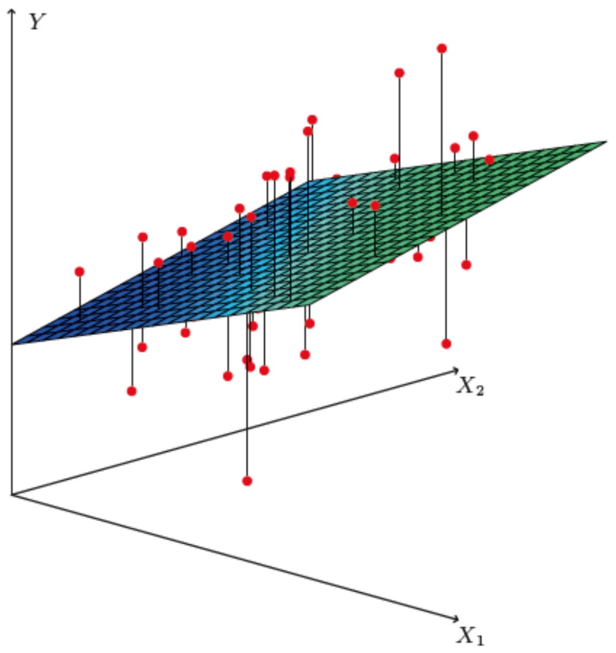
$$L(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{n} |y_i - f(\mathbf{x}_i; \mathbf{w})|^2$$

Linear regression

In linear regression, the fit function is of the form

$$f(\mathbf{x}; \mathbf{w}) = w_0 + \sum_{i=1}^{n} w_i x_i$$

i.e. the problem is equivalent to an unweighted least-squares fit of a (hyper-)plane:



Can be generalized to a nonlinear surface with higher order terms,

$$f(\mathbf{x}; \mathbf{w}) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i,j=1}^{n} w_{ij} x_i x_j + \sum_{i,j,k=1}^{n} w_{ijk} x_i x_j x_k + \ldots$$

Nonlinear regression

Other examples of nonlinear regression include:

MLP (multilayer perceptron) regression

Boosted decision tree regression

Support vector regression

For MLP regression, as with classification, regard the feature vector as the layer $k = 0$; i.e., $\varphi_i^{(0)} = x_i$.

The ith node of hidden layer k is

$$\varphi_i^{(k)} = h\left(w_{i0}^{(k)} + \sum_{j=1}^{n} w_{ij}^{(k)} \varphi_j^{(k-1)} \right)$$

where h is the activation function (tanh, relu, sigmoid,...).

MLP Regression (cont.)

For the final layer (k=K), in MLP regression (in contrast to classification), one omits the activation function, i.e.,

$$f(\mathbf{x};\mathbf{w}) = w_0^{(K)} + \sum_{j=1}^{n} w_j^{(K)} \varphi_j^{(K-1)}$$

where $\varphi_j^{(K-1)} =$ are the nodes of the last hidden layer ($k = K-1$).

For info on other types of multiple regression see, e.g.,

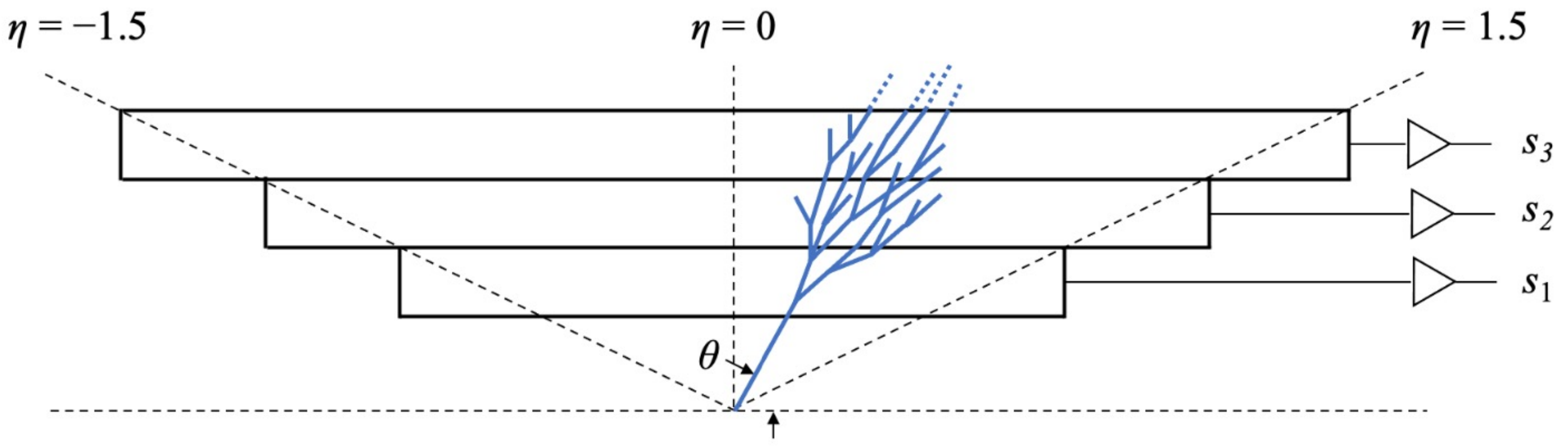
Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani, *An Introduction to Statistical Learning with Applications in R*, Springer, 2013; http://www-bcf.usc.edu/~gareth/ISL/

and the scikit-learn documentation.

Multiple regression example

Suppose particles with different energies E and angles θ (or equivalently $\eta = -\ln\tan(\theta/2)$) enter a calorimeter and create a particle showers that gives signals in three layers, s_1, s_2 and s_3, as well as an estimate of η.

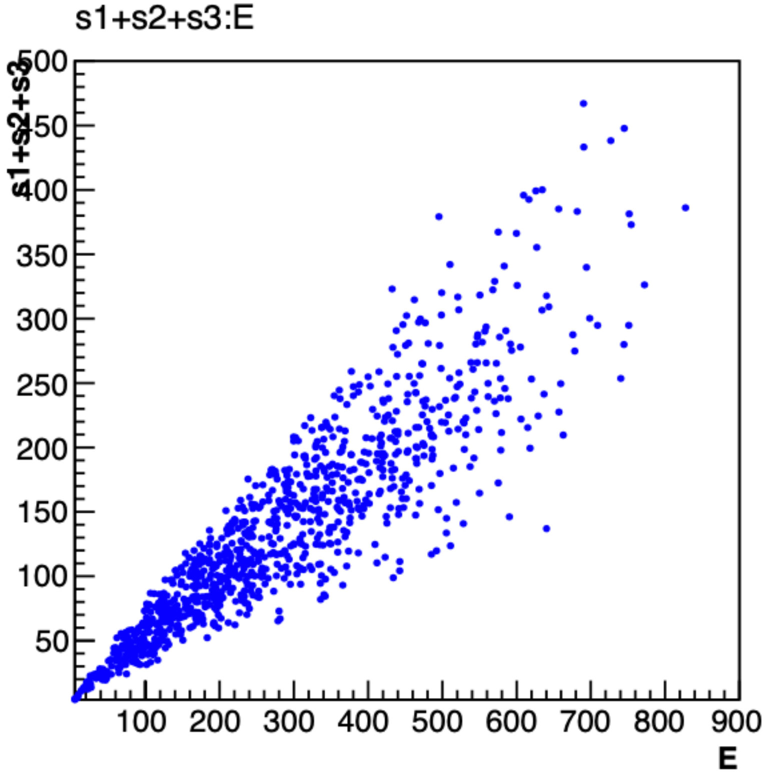
Some of the energy leaks through, with increased leakage for higher energy and higher angle θ (lower $|\eta|$).



The goal is to estimate the target $y_i = E_i$ given feature vectors $\boldsymbol{x}_i = (\eta, s_1, s_2, s_3)_i$ for $i = 1,\ldots,N$ training events.

Energy estimate from sum of signals

Naively, one could try just summing the signals: $\hat{E} = s_1 + s_2 + s_3$

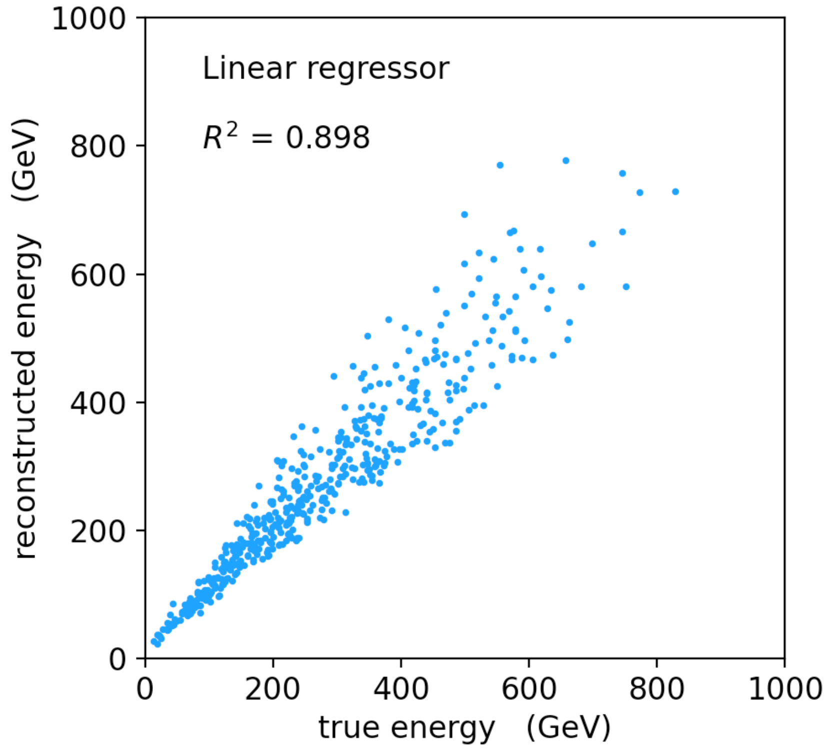


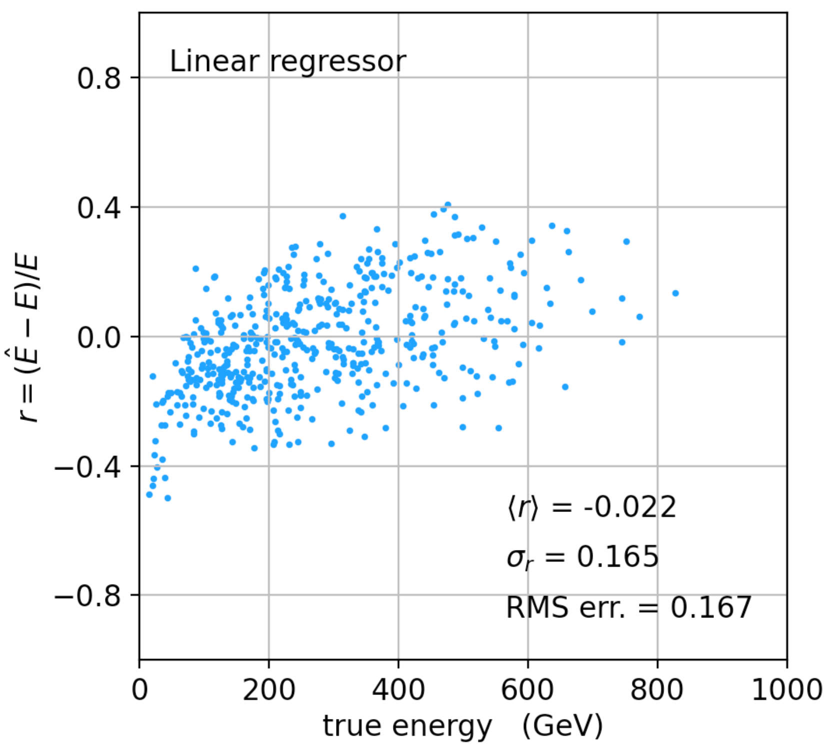
Gives very poor resolution because the particles have a distribution of energies and angles and hence differing amounts of the energy leak through undetected.

Linear regression

See MVRegressor.py, here using

```
regr = linear_model.LinearRegression()
regr.fit(X_train, y_train)
```

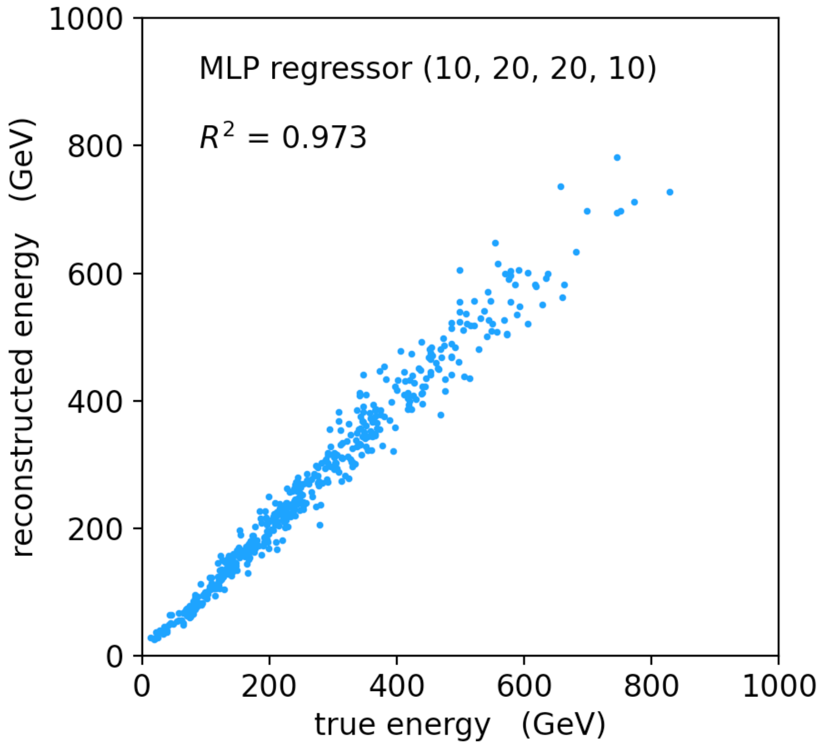


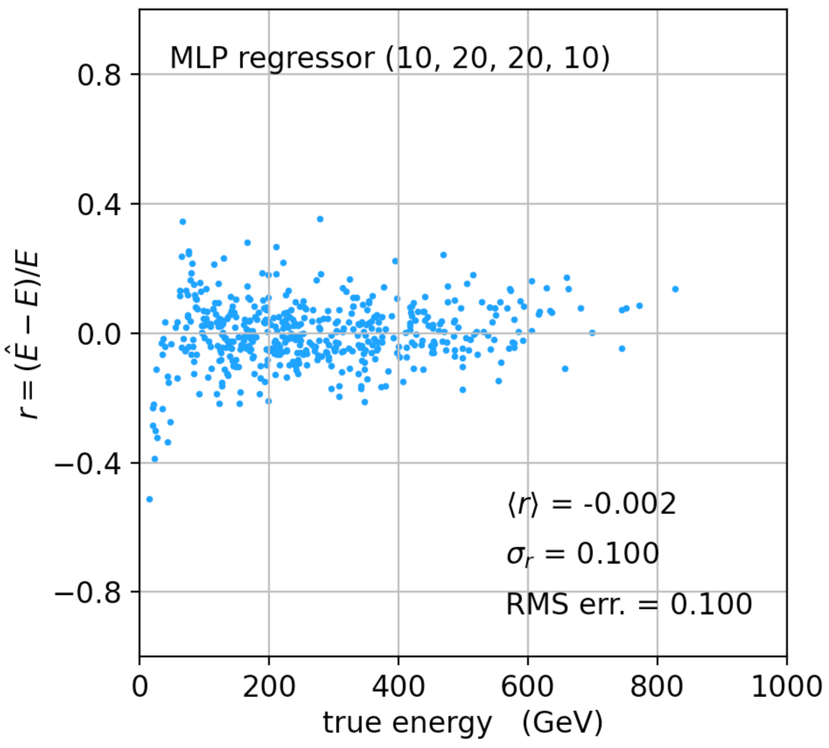


Average relative resolution 16.7%.

MLP Regression

```
regr = MLPRegressor(hidden_layer_sizes=(10,20,20,10), activation='relu'
regr.fit(X_train, y_train)
```





Better resolution (10%), here significant bias at low energies.

Refinements for multiple regression

One can try many improvements:

Scaling of predictor and target variables, e.g., standardize to zero mean and unit variance.

Use cross-validation to assess accuracy (and hence use entire sample of events for training.

Try different loss functions.

Try different regression algorithms (ridge regression, lasso, decision tree, support vector regression,...).

Some simple code using scikit-learn and a short write-up (from a year-3 project) is on the course webpage.