MACHINE LEARNING HW2 - SPAM CLASSIFICATION

LOGISTIC REGRESSION FUNCTION

```
def gradientDescent(x, y, iterations, alpha, theta, bias, tx, ty):
    decay = pd.DataFrame({'Iterations' : [],'Loss' : []})
    s = 0.0
    for i in range(iterations):
        z = np.dot(x, theta) + bias
        p = sigmoid(z)
        if i%(0.1*iterations) == 0:
            s = crossValid(tx, ty, theta, bias)
            loss_entry = pd.DataFrame([[i, s]], columns=['Iterations', 'Loss'])
            decay = decay.append(loss_entry, ignore_index=True)
        theta_gradient = np.dot((p-y),x)
        bias_gradient = np.mean(np.square(p) - 1)
        theta = theta - np.multiply(alpha, theta_gradient) / rms(theta_gradient)
        bias = bias - np.multiply(alpha, bias_gradient) / (rms(bias_gradient))
```

Fit the equation in linear regression approach, with AdaDelta

ANOTHER METHOD

Begging method is also deployed as method 2, however it does not count as another method

```
for i in range(10):
    print("Processing Model {}...".format(i+1))
    theta, bias, decay, mean, sd= startLogisticRegression(iterations, learning_rate, filepath)
    model.append((theta, bias, decay, mean, sd))
    acc.append(decay['Loss'].iloc[-1])
exportModel(model, modelpath)
```

10 model is trained separately and ensemble prediction is made according to their accuracy as weight

```
def exportPrediction(result, weight, path):
    result = np.average(result, axis = 0, weights = weight)
    result = pd.DataFrame(np.rint(result), columns=["label"], dtype=int)
    result.index.name = "id"
    result.index = result.index + 1
    result.to_csv(path_or_buf = path)
```

Begging method reduce the noise and minimize the chance of overfitting.

DISCUSSION

Method 2 should by using probabilistic model but I am sucked in covariance matrix calculations.

The linear model assumes that the probability p is a linear function of the regressors, while the logistic model assumes that the natural log of the odds p/(1-p) is a linear function of the regressors.