Fall 2020 CS4641 Homework 2

Aman Singh

October 26, 2020

1.1 Spam Detection

Naive Bayes classification uses the formula $p(t|x) = \frac{p(x|t)p(t)}{p(x)}$, where $p(t)$ is the prior and $p(x|t)$ is the likelihood. To find if "Your gift" is spam, we must calculate $p(\text{spam}|\text{"your gift"}) = p(s|yg)$, which is equal to $p(s|yg) = \frac{p(yg|s)p(s)}{p(yg)}$. Calculating the prior probabilities, we get $p(s) = \frac{2}{4} = 0.5$ and $p(yg) = \frac{0}{4} = 0$. Calculating the likelihood, we get $p(yg|s) = \frac{0}{2} = 0$. Now, if we try using the normal Bayes classification formula, we will get an undefined answer, as we would have to divide by 0. Instead, we can discard the denominator and simnply calculate whether spam or not spam has a bigger probability, using $p(s|yg) = p(\text{your}|s)p(\text{gift}|s) = p(y|s)p(g|s)$ and $p(!s|yg) = p(y|!s)p(g|!s)$. To use this, we are assuming all words are independent of each other, so we are not considering sentences at all. To avoid results of 0, we will add 1 to every probability, which should not affect the final classification. The probabilities are as follows:

$$
\begin{aligned}
p(y|s) &= \frac{0+1}{3+5} && = \frac{1}{8} \\
p(y|!s) &= \frac{0+1}{4+5} && = \frac{1}{9} \\
p(g|s) &= \frac{2+1}{3+5} && = \frac{3}{8} \\
p(g|!s) &= \frac{1+1}{4+5} && = \frac{2}{9} \\
p(yg|s) &= p(y|s)p(g|s) && = \frac{1}{8} \times \frac{3}{8} = \frac{3}{64} \\
p(yg|!s) &= p(y|!s)p(g|!s) && = \frac{1}{9} \times \frac{2}{9} = \frac{2}{81}
\end{aligned}
$$

Since $p(\text{"your gift"}|\text{spam}) > p(\text{"your gift}|\text{not spam})$, "your gift" would be classified as spam.

3.2 Feature Scaling

1. The change depends on the data and which column is changed. On average, it exacerbated the variance in the perturbed area, and made the variation proportion unusually high in the first component.

2. This change also depends on the data and which column is changed.

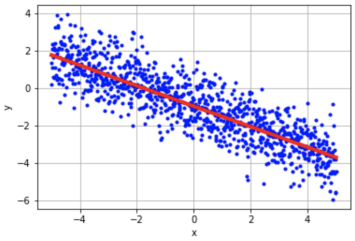
3. This does not affect the number of linearly independent components, as that is the property of linearly independent components; they are linearly independent of each other, regardless of any scalar multiplied to any column.

4.2 About RMSE

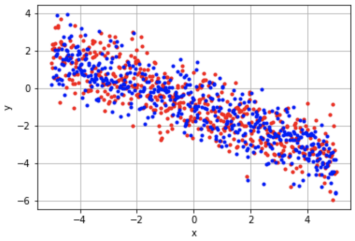
The RMSE should be as close to 0 as possible. Ideally, if pred and label are the same, RMSE would indeed be 0.

4.3 Testing: ridge regression

In [380]:
```
#helper, do not need to change
POLY_DEGREE = 5
NUM_OBS = 1000

rng = np.random.RandomState(seed=4)

true_weight = -rng.rand(POLY_DEGREE + 1, 1)
true_weight[2:, :] = 0
x_all = np.linspace(-5, 5, NUM_OBS)
reg = Regression()
x_all_feat = reg.construct_polynomial_feats(x_all, POLY_DEGREE)
y_all = np.dot(x_all_feat, true_weight) + rng.randn(x_all_feat.shape[0], 1) # in the second term, we add noise to data
# Note that here we try to produce y_all as our training data
plot_curve(x_all, y_all) # Data with noise that we are going to predict
plot_curve(x_all, np.dot(x_all_feat, true_weight), curve_type='-', color='r', lw=4) # the groundtruth information
plt.show()

indices = rng.permutation(NUM_OBS)
```

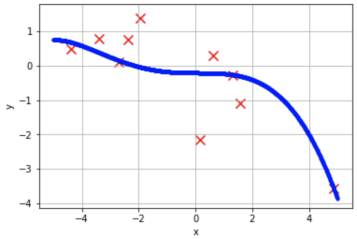


In [381]:
```
#helper, do not need to change
train_indices = indices[:NUM_OBS//2]
test_indices = indices[NUM_OBS//2:]

plot_curve(x_all[train_indices], y_all[train_indices], color='r')
plot_curve(x_all[test_indices], y_all[test_indices], color='b')
plt.show()
```



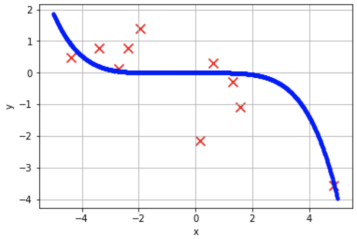
In [382]:
```
#helper, do not need to change
sub_train = train_indices[:10]
print(x_all_feat[sub_train].shape)
print(y_all[sub_train].shape)
weight = reg.ridge_fit_closed(x_all_feat[sub_train], y_all[sub_train], c_lambda=1000)

y_pred = reg.predict(x_all_feat, weight)
plot_curve(x_all, y_pred)
plt.scatter(x_all[sub_train], y_all[sub_train], s=100, c='r', marker='x')
plt.show()

y_test_pred = reg.predict(x_all_feat[test_indices], weight)
test_rmse = reg.rmse(y_test_pred, y_all[test_indices])
print('test rmse: %.4f' % test_rmse)
```

```
(10, 6)
(10, 1)
```



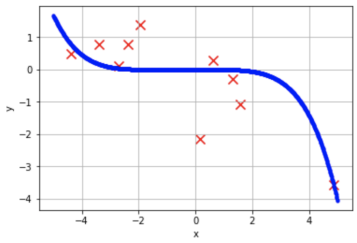

```
test rmse: 1.3589
```

In [383]:
```
#helper, do not need to change
sub_train = train_indices[:10]
weight = reg.ridge_fit_GD(x_all_feat[sub_train], y_all[sub_train], c_lambda=1000, learning_rate=1e-7)

y_pred = reg.predict(x_all_feat, weight)
plot_curve(x_all, y_pred)
plt.scatter(x_all[sub_train], y_all[sub_train], s=100, c='r', marker='x')
plt.show()

y_test_pred = reg.predict(x_all_feat[test_indices], weight)
test_rmse = reg.rmse(y_test_pred, y_all[test_indices])
print('test rmse: %.4f' % test_rmse)
```




```
test rmse: 1.5831
```

In [384]:
```
#helper, do not need to change
sub_train = train_indices[:10]
weight = reg.ridge_fit_SGD(x_all_feat[sub_train], y_all[sub_train], c_lambda=1000, learning_rate=1e-7)

y_pred = reg.predict(x_all_feat, weight)
plot_curve(x_all, y_pred)
plt.scatter(x_all[sub_train], y_all[sub_train], s=100, c='r', marker='x')
plt.show()

y_test_pred = reg.predict(x_all_feat[test_indices], weight)
test_rmse = reg.rmse(y_test_pred, y_all[test_indices])
print('test rmse: %.4f' % test_rmse)
```




```
test rmse: 1.5790
```