CS 189/289A Introduction to Machine Learning
Fall 2023 Jennifer Listgarten, Jitendra Malik **DIS3**

1 Logistic Regression

Assume that we have n i.i.d. data points $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$, where each y_i is a binary label in $\{0, 1\}$. We model the posterior probability as a Bernoulli distribution and the probability for each class is the sigmoid function, i.e., $p(y|\mathbf{x}; \mathbf{w}) = q^y(1-q)^{1-y}$, where $q = s(\mathbf{w}^\top \mathbf{x})$ and $s(\zeta) = \frac{1}{1+e^{-\zeta}}$ is the sigmoid function.

(a) Write out the likelihood and log likelihood functions.

Solution: The likelihood is:

$$L(\mathbf{w}) = \prod_{i=1}^{n} p(y = y_i|\mathbf{x}_i) = \prod_{i=1}^{n} q_i^{y_i}(1-q_i)^{1-y_i}.$$

The log likelihood is:

$$l(\mathbf{w}) = \sum_{i=1}^{n} y_i \log(q_i) + (1-y_i)\log(1-q_i)$$

(b) Show that finding maximum likelihood estimate of $\mathbf{w}$ is equivalent to the following optimization problem:

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}} \left[\sum_{i=1}^{n} (1-y_i)\mathbf{w}^\top \mathbf{x}_i + \log\Big(1 + \exp\Big\{-\mathbf{w}^\top \mathbf{x}_i\Big\}\Big) \right]$$

Solution: Now, we step through minimizing the negative log likelihood of the training data as a function of the parameters $\mathbf{w}$:

$$\begin{aligned}
\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}} -L(\mathbf{w}) &= \operatorname*{argmin}_{\mathbf{w}} \left[-\prod_{i=1}^{n} q_i^{y_i}(1-q_i)^{1-y_i} \right] \\
&= \left[\operatorname*{argmin}_{\mathbf{w}} - \sum_{i=1}^{n} y_i \log(q_i) + (1-y_i)\log(1-q_i) \right] \\
&= \left[\operatorname*{argmin}_{\mathbf{w}} - \sum_{i=1}^{n} y_i \log\left(\frac{q_i}{1-q_i}\right) + \log(1-q_i) \right]
\end{aligned}$$

Since q_i is the sigmoid function, we plug it in and simplify:

$$\begin{aligned}
\hat{\mathbf{w}} &= \underset{\mathbf{w}}{\operatorname{argmin}} \left[-\sum_{i=1}^{n} y_i \log\left(\frac{q_i}{1-q_i}\right) + \log(1-q_i) \right] \\
&= \underset{\mathbf{w}}{\operatorname{argmin}} \left[-\sum_{i=1}^{n} y_i \log\left(\frac{\frac{1}{1+\exp\{-\mathbf{w}^\top \mathbf{x}_i\}}}{1-\frac{1}{1+\exp\{-\mathbf{w}^\top \mathbf{x}_i\}}}\right) + \log\left(1 - \frac{1}{1+\exp\{-\mathbf{w}^\top \mathbf{x}_i\}}\right) \right] \\
&= \underset{\mathbf{w}}{\operatorname{argmin}} \left[-\sum_{i=1}^{n} y_i \log\left(\frac{\frac{1}{1+\exp\{-\mathbf{w}^\top \mathbf{x}_i\}}}{\frac{1+\exp\{-\mathbf{w}^\top \mathbf{x}_i\}-1}{1+\exp\{-\mathbf{w}^\top \mathbf{x}_i\}}}\right) + \log\left(\frac{1+\exp\{-\mathbf{w}^\top \mathbf{x}_i\}-1}{1+\exp\{-\mathbf{w}^\top \mathbf{x}_i\}}\right) \right] \\
&= \underset{\mathbf{w}}{\operatorname{argmin}} \left[-\sum_{i=1}^{n} y_i \log\left(\frac{1}{\exp\{-\mathbf{w}^\top \mathbf{x}_i\}}\right) + \log\left(\frac{\exp\{-\mathbf{w}^\top \mathbf{x}_i\}}{1+\exp\{-\mathbf{w}^\top \mathbf{x}_i\}}\right) \right] \\
&= \underset{\mathbf{w}}{\operatorname{argmin}} \left[-\sum_{i=1}^{n} y_i \mathbf{w}^\top \mathbf{x}_i - \mathbf{w}^\top \mathbf{x}_i - \log\left(1+\exp\{-\mathbf{w}^\top \mathbf{x}_i\}\right) \right]
\end{aligned}$$

We bring in the negative sign to get:

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} \left[\sum_{i=1}^{n} (1-y_i)\mathbf{w}^\top \mathbf{x}_i + \log\left(1+\exp\left\{-\mathbf{w}^\top \mathbf{x}_i\right\}\right) \right]$$

(c) Comment on whether it is possible to find a closed form maximum likelihood estimate of **w**, and describe an alternate approach.

Solution: Let us denote $J(\mathbf{w}) = \sum_{i=1}^{n}(1-y_i)\mathbf{w}^\top \mathbf{x}_i + \log\left(1+\exp\{-\mathbf{w}^\top \mathbf{x}_i\}\right)$. Notice that $J(\mathbf{w})$ is convex in **w**, so global minimum can be found. Note that $s'(\zeta) = s(\zeta)(1-s(\zeta))$. Now let us take the gradient of $J(\mathbf{w})$ w.r.t **w**:

$$\nabla_w J = \sum_{i=1}^{n}(1-y_i)\mathbf{x}_i - \frac{\exp\{-\mathbf{w}^\top \mathbf{x}_i\}}{1+\exp\{-\mathbf{w}^\top \mathbf{x}_i\}}\mathbf{x}_i = \sum_{i=1}^{n}(-1+s(\mathbf{w}^\top \mathbf{x}_i)-y_i+1)\mathbf{x}_i = \sum_{i=1}^{n}(s_i-y_i)\mathbf{x}_i = \mathbf{X}^\top(\mathbf{s}-\mathbf{y})$$

where, $s_i = s(\mathbf{w}^\top \mathbf{x}_i), \mathbf{s} = (s_1, \dots, s_n)^\top, \mathbf{y} = (y_1, \dots, y_n)^\top$ and $\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^\top \\ \vdots \\ \mathbf{x}_n^\top \end{bmatrix}$.

Unfortunately, we can't get a closed form estimate for **w** by setting the derivative to zero, given that the term **s** still contains **w**, and further-order derivatives will continue to carry expressions over **w**. However, the convexity of this problem allows for first-order optimization algorithms, such as gradient descent, to converge to a global minimum.

2 Gaussian Classification

Let $P(x \mid \omega_i) \sim \mathcal{N}(\mu_i, \sigma^2)$ for a two-category, one-dimensional classification problem with classes ω_1 and ω_2, $P(\omega_1) = P(\omega_2) = 1/2$, and $\mu_2 > \mu_1$.

(a) Find the optimal decision boundary and the corresponding decision rule.

Solution:

$$\begin{array}{rcll} P(\omega_1 \mid x) & = & P(\omega_2 \mid x) & \Leftrightarrow \\ P(x \mid \omega_1)\frac{P(\omega_1)}{P(x)} & = & P(x \mid \omega_2)\frac{P(\omega_2)}{P(x)} & \Leftrightarrow \\ P(x \mid \omega_1) & = & P(x \mid \omega_2) & \Leftrightarrow \\ \mathcal{N}(\mu_1, \sigma^2) & = & \mathcal{N}(\mu_2, \sigma^2) & \Leftrightarrow \\ (x-\mu_1)^2 & = & (x-\mu_2)^2 & \end{array}$$

This yields the Bayes decision boundary: $x = \frac{\mu_1+\mu_2}{2}$.

The corresponding decision rule is, given a data point $x \in \mathbb{R}$:

- if $x < \frac{\mu_1+\mu_2}{2}$, then classify x in class 1
- otherwise, classify x in class 2

Note that this is the centroid method.

(b) The probability of misclassification (error rate) is:

$$P_e = P((\text{misclassified as } \omega_1) \mid \omega_2)\, P(\omega_2) + P((\text{misclassified as } \omega_2) \mid \omega_1)\, P(\omega_1).$$

Show that the probability of misclassification (error rate) associated with this decision rule is

$$P_e = \frac{1}{\sqrt{2\pi}} \int_a^{\infty} e^{-z^2/2} dz,$$

where $a = \dfrac{\mu_2 - \mu_1}{2\sigma}$.

Solution:

$$\begin{aligned} P((\text{misclassified as } \omega_1) \mid \omega_2) &= \int_{-\infty}^{\frac{\mu_1+\mu_2}{2}} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu_2)^2}{2\sigma^2}} dx \\ &= \int_{-\infty}^{-a} \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}} dz \\ &= \frac{1}{\sqrt{2\pi}} \int_a^{+\infty} e^{-\frac{z^2}{2}} dz \\ &= P_e, \end{aligned}$$

where we have used the change of variables $z = \frac{x-\mu_2}{\sigma}$, so that $dz = \frac{1}{\sigma}dx$. We also have

$$P((\text{misclassified as } \omega_2) \mid \omega_1) = \int_{\frac{\mu_1+\mu_2}{2}}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu_1)^2}{2\sigma^2}} dx$$

$$= \int_a^{+\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}} dz$$
$$= P_e$$

Therefore:

$$P((\text{misclassified as } \omega_1) \mid \omega_2)P(\omega_2) + P((\text{misclassified as } \omega_2) \mid \omega_1)P(\omega_1) = P_e \cdot \frac{1}{2} + P_e \cdot \frac{1}{2} = P_e$$

(c) What is the limit of P_e as σ goes to 0?

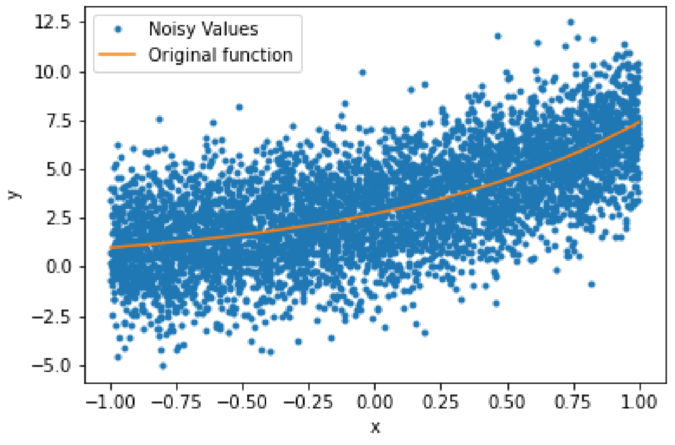
Solution: As σ goes to 0, a goes to ∞, so the integral P_e goes to 0.

3 Overview of test sets, validation, and cross-validation

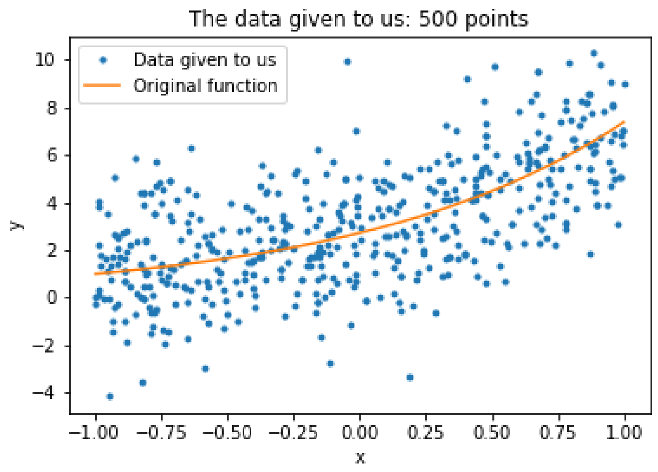
In this part, we discuss several issues having to do with test sets and the notions of validation and cross-validation. Open this notebook in datahub and discuss the questions it contains.

Solution:

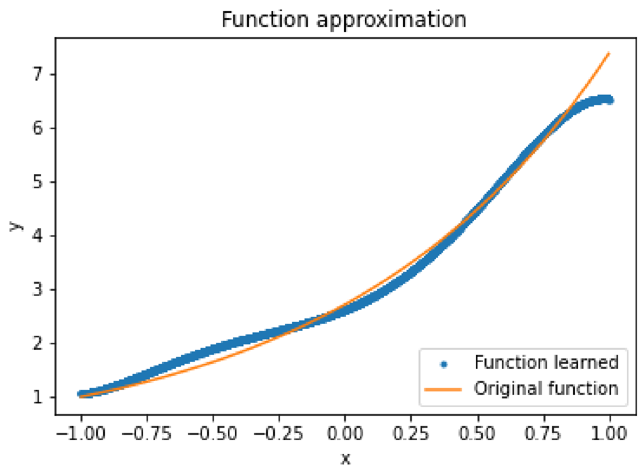
Test sets. The purpose of this notebook is to reflect about the concepts of test sets and validation. Upon executing the first cells, you will see the following plot:



The function we are trying to approximate is shown in orange, and the sampled data (with noise), in blue. Observe that the domain of the true function is the interval $[-1, 1]$. The next cell carries out a subsampling of the data. This is the data that we assume someone is giving to us in order to learn a function.



Immediately after seeing this plot, we have a cell with routines to split the data into a training and a test set, and to train a model. The model being trained in this cell is a fixed 5th-degree polynomial. Observe that there is a line of code to be added in order to carry out the correct splitting of the data into a training set and a test set. The line is `trainIndices = np.logical_not(testIndices)`.

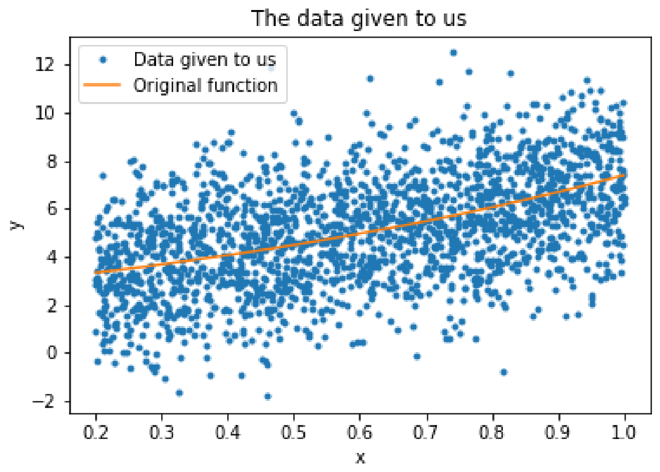


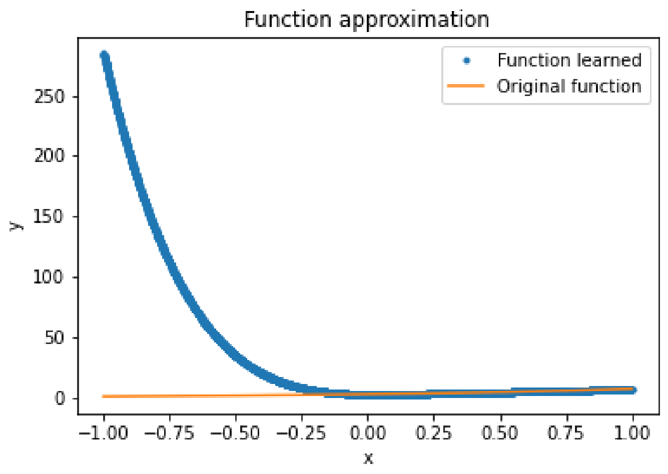
The plot we obtain immediately after the execution of this cell shows us the true function we are trying to learn, and the training and test errors. We also obtain this output from the execution of the cell:

```
Training error: 3.6511493057865483
Test error: 3.9256574783857046
True error: 0.03422772128092661
```

We observe that the test and training errors are closed to each other. Our data is noisy; this explains why in this case we see a test error bigger than the training error. We measure the true error with respect to the underlying true function, so this error is not directly affected by the noise in the training data.

Immediately after this, we now sample the data with a bias: instead of random sampling, as happened before, the data given to us for training only contains values of $x > 0.2$. The following plots show the data extracted this time, and the result of fitting our data. Regarding the code used in the notebook, observe that the first time we fit the data, we built most routines by hand; in this case, we make more significant use of existing APIs.



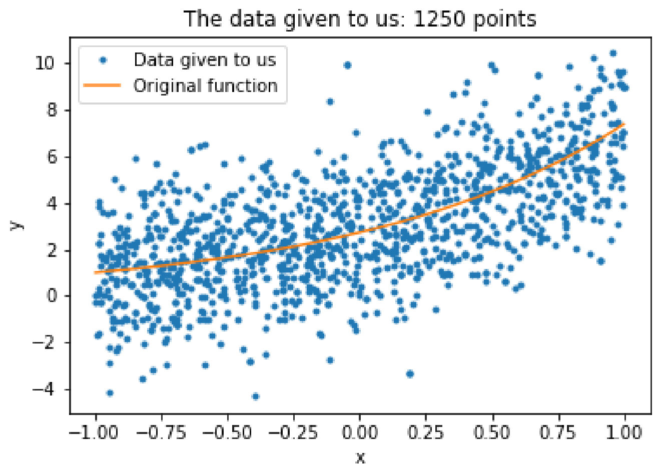


The last plot comes accompanied by this output:

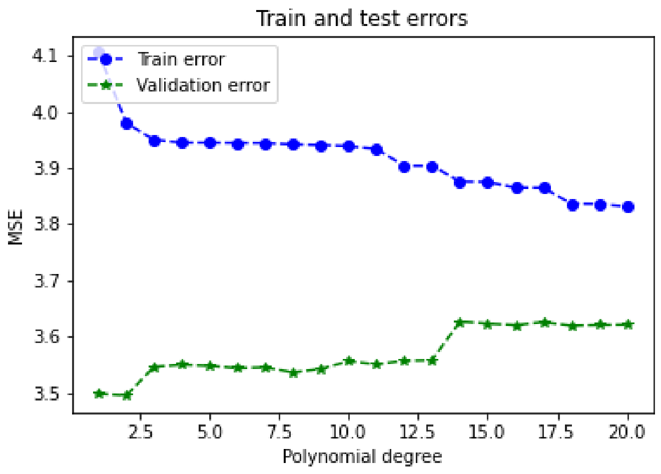
```
Training error: 3.9543414420385754
Test error: 3.588770415447887
True error: 5436.558416632976
```

We observe the training and test errors close to each other, as happened before, but now the true error is off-the-charts. The point here is that the data we used for training does not represent the entire domain of our x values. In this case, we are trying to make predictions outside the interval for which we had x values.

Validation. The next plots deal with the concept of validation. The first plot shows the data we will use during training. We use the function we introduced during our discussion of test sets. The data is randomly sampled from the entire interval $[-1,1]$



The code executed immediately after this plot does the following: it breaks the data *given to us* into training set, validation set, and test set. We train polynomials of varying degrees to our data, and we plot the training and validation errors we obtain for each:



This plot tells us that we should pick a degree equal to 2 because this yields the lowest validation error. Using the code given in the notebook, we can compute the test error for a polynomial of degree 2. We obtain 3.19.

Observe that validation is used as a methodological tool to pick a hyperparameter. Its drawback is that it prevents us to use all our training data in our optimization routines that fit our models.

Cross-validation. If we reserve k points for our validation set, we have this tradeoff: a low k means our validation set is too small and can't yield a reliable estimate of the true error; if we set

k too high, validation yields an excellent estimate of the true error, but our models will be trained on little data.

Leave-one-out cross-validation is an example of cross-validation. This technique allows us to use all data during training, but it requires us to train $\sim nm$ models, where n is the size of the training set, and m is the number of evaluations of the hyperparameters. This is unrealistic, but the idea is intriguing: we have a method that allows to train on the entire dataset while getting a handle on the generalization error. One way to address this disadvantage of leave-one-out is K-fold cross-validation. You will learn about this technique soon!