TTIC 31020: Introduction to Machine Learning
Autumn 2020

Problem Set #2

Out: October 15, 2020

Due: Monday October 26, 11:59pm

How and what to submit? Please submit your solutions electronically via Canvas. Please submit two files:

1. A PDF file with the written component of your solution including derivations, explanations, etc. You can create this PDF in any way you want, e.g.,LaTeX (recommended), Word + export as PDF, scan handwritten solutions (note: must be legible!), etc. Please name this document ⟨`firstname-lastname`⟩`-sol2.pdf`.
2. The main empirical component of the solution (Python code and the documentation of the experiments you are asked to run, including figures, for softmax classification of digit images) in a Jupyter notebook renamed to ⟨`firstname-lastname`⟩`-sol2.ipynb`.
3. There is another notebook, in which we will study bias/variance tradeoff in regression. Submit the completed notebook as ⟨`firstname-lastname`⟩`-bv.ipynb`.

Late submissions: there will be a penalty of 25 points for any solution submitted within 24 hours past the deadlne. No submissions will be accepted past then.

What is the required level of detail? When asked to derive something, please clearly state the assumptions, if any, and strive for balance: justify any non-obvious steps, but try to avoid superfluous explanations. When asked to plot something, please include in the `ipynb` file the figure as well as the code used to plot it. If multiple entities appear on a plot, make sure that they are clearly distinguishable (by color or style of lines and markers) and references in a legend or in a caption. When asked to provide a brief explanation or description, try to make your answers concise, but do not omit anything you believe is important. If there is a mathematical answer, provide is precisely (and accompany by only succinct words, if appropriate).

When submitting code (in Jupyter notebook), please make sure it's reasonably documented, runs and produces all the requested results. If discussion is required/warranted, you can include it either directly in the notebook (you may want to use the markdown style for that) or in the PDF writeup.

Collaboration policy : collaboration is allowed and encouraged, as long as you (1) write your own solution entirely on your own, (2) specify names of student(s) you collaborated with in your writeup.

1 Regularization

We will consider general L_p norm regularization for squared loss in regression:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \left\{ \sum_{i=1}^{N} \left(y_i - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i)\right)^2 + \lambda \sum_{j=1}^{d} |w_j|^p \right\}, \tag{1}$$

where $\boldsymbol{\phi}(\mathbf{x})$ is a $d+1$-dimensional feature vector, including the constant term $\phi_0 \equiv 1$ which we do not regularize, and N is the size of the training set.

Problem 1 [15 points]
Show that the objective in (1) is equivalent to

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \left\{ \sum_{i=1}^{N} \left(y_i - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i)\right)^2 \right\} \tag{2}$$
$$\text{subject to } \sum_{j=1}^{d} |w_j|^p \le \tau$$

for appropriate τ, which may depend in some way on the data and on λ. In other words, if you find $\mathbf{w}^*$ according to (1), then find $\mathbf{w}^*$ according to (2) with the appropriate τ, the two values of $\mathbf{w}^*$ will be the same.

End of problem 1

Advice: It may be helpful to consider the solution of the constrained problem in (2) using Lagrange multipliers, and perhaps review/look up the notion of Karush-Kuhn-Tucker (KKT) conditions (Wikipedia is a good starting point).

2 Softmax

In this section we will consider a discriminative model for a multi-class setup, in which the class labels take values in $\{1, \ldots, C\}$. A principled generalization of the logistic regression model to this setup is the *softmax* model. It requires that we maintain a separate parameter vector $\mathbf{w}_c$ for each class c. Under this model, the estimate for the posterior for class c, $c = 1, \ldots, C$ is

$$\widehat{p}(y = c|\mathbf{x};\, \mathbf{W}) = \operatorname{softmax}(\mathbf{w}_c \cdot \mathbf{x}) \triangleq \frac{\exp(\mathbf{w}_c \cdot \mathbf{x})}{\sum_{y=1}^{C} \exp(\mathbf{w}_y \cdot \mathbf{x})}, \tag{3}$$

where $\mathbf{W}$ is a $C \times d$ matrix, the c-th row of which is a vector $\mathbf{w}_c$ associated with class c. We will assume throughout the problem set that $\mathbf{x}$ is the feature vector associated with an input example, including the constant feature $x_0 = 1$.

Problem 2 [10 points]
Show that the softmax model as stated in (3) is *over-parameterized*, that is, show that for any settings of $\mathbf{w}_c$ for $c = 1, \ldots, C$ there is an different setting that yields exactly the same $p(y \,|\, \mathbf{x})$ for every $\mathbf{x}$. Then explain how this implies that we only need $C - 1$ trainable parameter vectors for softmax, and not C.

End of problem 2

We now turn to a practical exercise in learning the softmax model, which can be done very similarly to learning logistic regression – via (stochastic) gradient descent. We will consider the L_2 regularization

$$\mathbf{W}^* = \underset{\mathbf{W}}{\operatorname{argmax}} \left\{ \frac{1}{N} \sum_{i=1}^{N} \log \widehat{p}\,(y_i | \mathbf{x}_i; \mathbf{W}) - \lambda \|\mathbf{W}\|^2, \right\} \tag{4}$$

where $\|\mathbf{W}\|^2$ is the Frobenius norm of the matrix $\mathbf{W}$.[1]

Problem 3 [15 points]
Write down the log-loss of the L_2-regularized softmax model, and its gradients with respect to $\mathbf{W}$ (in the stochastic setting, i.e., computed over a single training example). Then, write the update equation(s) for the stochastic gradient descent, assuming learning rate η.

End of problem 3

Advice: It is helpful, both in derivation and in coding, to convert the scalar representation of the labels $y \in \{1, \ldots, C\}$ to a vector representation $\mathbf{t} \in \{0, 1\}^C$, in which if $y_i = c$ then $t_{ic} = 1$ and $t_{ij} = 0$ for all $j \neq c$. This is sometimes called "one-hot" encoding of the labels: among C elements of the 0/1 label vector, exactly one element is "hot", i.e., set to 1.

We are now ready to apply the softmax model to the problem of classifying handwritten digits. We will work with the "MNIST" data set[2]. Each example is a 28 by 28 pixel greylevel image of a handwritten digit; we will be working with a vectorized representation of the images, converted to a 784-dimensional integer vector with values between 0 (black) and 255 (white).

The data set provided to you consists of three parts

- **Training** set of 20000 examples;
- **Validation** set of 10000 examples;
- **Test** set of 10000 examples (no labels are provided).

[1] The Frobenius norm of a matrix $\mathbf{A}$ is the sum of squares of its elements, $\sum_i \sum_j A_{ij}^2$.

[2] This is a data set of handwritten digits, which has served as machine learning benchmark for many years.

The data are contained in a zipfile, which should be unzipped in the same directory the Juptyer notebook will be run from. Each set is divided roughly equally among 10 classes.

In addition to the normal training set of 20000 examples, we include a small training set of only 30 examples (3 per class), to investigate the effect data set size has on training a classifier in this domain.

We will be using a linear softmax model to classify these images. In the last problem you will implement the gradient update procedure in the previous problem in order to classify images of handwritten digits. We have provided skeleton code in the Jupyter notebook that you will have to modify in order to get the best possible prediction results. Note that the code will implement a *mini-batch* gradient descent, in which updates are based on b examples with $b \geq 1$, averaging the gradient over the mini-batch. The size of the mini-batch is one of the *hyper-parameters* of the learning procedure, along with the regularization parameter λ, stoppinc criteria, etc.

Problem 4 [45 points]

Fill in missing code for

- inference (scores and posterior probabilities from the softmax model);
- calculation of conditional log-likelihood of the labels given the data under the model;
- calculation of the gradient of the regularized objective;
- most importantly, implementaion of the (minibatch) stochastic gradient descent.

We have provided some suggested values for the "hyper-parameters" of the learning algorithm: size of the mini-batch, stopping criteria (currently just limit on number of iterations), the settings for the initial learning rate and for decreasing its value over iterations (or not). These should be a reasonable starting point, but you are encouraged to experiment with their values, and to consider adding additional variations: changing the mechanism for selection of examples in the mini-batch (e.g., how should the data be sampled?), additional stopping criteria, different schedule for dropping the learning rate, etc. We also have included the code for organizing your experiments, once the functionality above is implemented.

Feel free to guess appropriate values, or to tune them on the validation set.

Your tuning process and any design choices, and the final set of values for all the hyper-parameters chosen for the final model, should be clearly documented in the notebook; please write any comments directly in the notebook rather than in the PDF file.

Please report the following statistics for your model in the write-up: the training accuracy, the validation accuracy, and the confusion matrix. The accuracy is the fraction of examples in the set that are classified correctly. The confusion matrix in a classification experiment with C classes is a $C \times C$ matrix $\mathbf{M}$ in which M_{ij} is the number of times an example with true label i was classified as j. *Note that* $\mathbf{M}$ *is not necessarily a symmetric matrix.*

In addition, visualize the parameters of the learned model. Since these are in the same domain as the input vectors, we can visualize them as images. Specifically, ignore the bias term, and use for instance

```
plt.imshow(w[-28**:, c].reshape(28, 28))
plt.colorbar()
plt.show()
```

to show the vector $\mathbf{w}_c$ associated with class c in model $\mathbf{W}$. Try to develop and write down an intuitive explanation of what the resulting images show.

Finally, compare and contrast the behavior of training, in particular the absolute values of training and validation accuracy and the role of regularization, in the two data regimes (small vs large data set). Write your observations and conclusions in the notebook.

For the final evaluation, we have set up two Kaggle competitions to which you will be submitting your final predictions on a held-out testing set, one for your best model trained on the large training set, and one for the best model trained on the small training set. The URLs:

`https://www.kaggle.com/c/ttic-31020-hw2-mnist-fall-2020/`
`https://www.kaggle.com/c/ttic-31020-hw2-mnist-smalls-fall-2020/`

First, you will have to create a Kaggle account (with your `uchicago.edu` or `ttic.edu` email). Once you have access to the competition pages (when you have an account follow the invite links above to gain access), read through all three information pages carefully (Description, Evaluation and Rules) and accept the rules. You will now be ready to make submissions. The two competitions have the same goal and structure.

Running the notebook in its entirety will create two files, `small_submission.csv` and `large_submission.csv`, in the directory the notebook is run from. Once you have accepted the Kaggle rules, there will be an option to "Make a submission", where you can upload this CSV file. To make sure you do not overfit this held-out set, **we have limited your submissions to two per day, so start early and you will get more chances if you make mistakes.** Your up to date score will appear on the public leaderboard once you submit.

End of problem 4

3 Bias-variance decomposition

In this final problem we will do some hands-on experiments to study the effect different properties of the learning problem, the model, and the learning objective have on the bias and variance of the learning algorithm. You will need the notebook called `Bias-variance tradeoff study.ipynb`

The notebook implements the following experimental setup: We will construct data sets for a synthetic (artificial) prediction problem, $x \in \mathbb{R} \to y \in \mathbb{R}$. The true $p(x, y)$ is composed of a uniform $p(x)$ over an interval (the domain of the function, set in the code to $(-5, 5)$), and the conditional $p\,(y \mid x)$ corresponding to our standard Gaussian noise model,

$$y = F(x) + \nu \sim \mathcal{N}(0, \sigma^2)$$

where the function F is (somewhat arbitrarily) defined in the variable `fn` in the notebook (it happens to be a 4th degree polynomial).

A single experimental run will have the following form:

- Draw a training set of size N from $p(x, y)$
- For a given d and λ, fit a polynomial model of degree d to the training set using least squares ridge regression with regularization parameter λ

We will repeat such a run $M = 100$ times, for every combination of $N \in \{5, 10, 25, 50, 100\}$, $\lambda \in \{0, 1, 50\}$, and $d = 1, 2, 3, 4, 6$. *Note: this sounds like a lot, but because of the small sizes of the data, low dimensionality of the feature spaces even for* $d = 6$*, and the efficiency of the closed form solution, it should only take a few minutes on a typical, medium-strength laptop.*

We will also construct a validation (`val`) set, which we will use as an empirical proxy for the true $p(x, y)$. It will include a $T = 1000$ x points $x_1, \ldots, x_T \sim$ uniform(domain), and for each point, we will draw $K = 1000$ values of y from $p\,(y \mid x)$; we will denote the values drawn from $p\,(y \mid x_i)$ as $y_{i,1}, \ldots, y_{i,K}$.

All of this is already done in the notebook; you just need to run the code. It will produce a set of figures visualizing the set of M models fit to the set of M training sets (parse that carefully) for every combination of $N/d/\lambda$. It will also show the *average* model for each such combination, obtained by point-wise averaging: if $\widehat{f}_1, \ldots, \widehat{f}_M$ are the M models (from M training sets), then for any x,

$$\bar{f}(x) = \frac{1}{M} \sum_{m=1}^{M} \widehat{f}_m(x). \tag{5}$$

Problem 5 [15 points]

Write code that will compute for every combination of $N/d/\lambda$ the empirical expectation of the following quantities (with the expectation w.r.t. the empirical distribution of the data in `val` *and* the training sets created for that combination):

- Squared loss
- Noise variance (as defined in the lecture)
- Squared bias of $\widehat{f}(x)$ as an estimator of y
- Variance of $\widehat{f}(x)$ as an estimator of y

To clarify: empirical expectation of a quantity here means averaging over data points x and over $y|x$ in the `val` set. So, for instance, the expected squared loss for a set of models

$\widehat{f}_1, \ldots, \widehat{f}_M$ produced for a given $N/d/\lambda$ combination would be

$$\frac{1}{TKM} \sum_{t=1}^{T} \sum_{k=1}^{K} \sum_{m=1}^{M} \left(y_{i,k} - \widehat{f}_m(x_i) \right)^2 \tag{6}$$

Then, write an interpretive description of the plots generated by the provided code in conjunction with the numbers generated by your code. What trends do you see in the figures? How do these trends correspond to the trends/observations from the numerical results produced by your code? How do you explain these trends? How would these observation inform your choice of model/hyperparameters in a similar situation?

End of problem 5