Data Mining (CSC 503/SENG 474)

Assignment 2 **Due on Monday, February 24th, 11:59pm**

Instructions:

- You must complete this assignment on your own; this includes any coding/implementing, running of experiments, generating plots, analyzing results, writing up results, and working out problems. Assignments are for developing skills to make you strong. If you do the assignments well, you'll almost surely do better on the final and beyond.
- On the other hand, you can certainly have high-level discussions with classmates and can post questions to the Ed Discussion forum. You can also discuss preprocessing (data normalization) with classmates. You also are welcome to come to office hours (or otherwise somehow ask me and the TAs things if you are stuck).
- You must type up your analysis and solutions; I strongly encourage you to use LaTeX to do this. LaTeX is great both for presenting figures and math.
- Please submit your solutions via Brightspace by the due date/time indicated above. This is a hard deadline; the penalty for lateness is getting a zero on the assignment because, as senior undergraduate students or graduate students, I know that you know the importance of deadlines. (In particular, when you submit on Brightspace, be sure to check that you did in fact submit and that you submitted the correct file.) However, I will make an exception if I am notified prior to the deadline with an acceptable excuse and if you further can (soon thereafter) provide a signed note related to this excuse.

Introduction

This assignment has two components. Component 1 involves implementing and applying (or finding and applying) some of the methods you have already seen and analyzing their results on a binary classification problem; this part is for all students (both CSC 503 and SENG 474). The theme of this part is using cross-validation for hyperparameter tuning and model selection. Component 2 is bonus for all students (both CSC 503 and SENG 474). It builds on Component 1, so my intention is that you focus first on completing Component 1 well, and if more ambitious, you attempt Component 2.

1 Component 1: Experiments and Analysis

First, "implement" the following methods:

- **Support vector machines**. This is the soft-margin version of SVM (the one that works for data that aren't necessarily linearly separable). Your implementation should allow for specifying a kernel function (you'll need the linear kernel and Gaussian kernel). Please see this page for a description of the soft-margin SVM problem, along with the regularization parameter that is used: https://scikit-learn.org/stable/modules/svm.html#svc.

  The relevant part is the equation following the text "SVC solves the following primal problem". Note that effect of the regularization parameter C (a hyperparameter). Higher C means *less* regularization.

- **Neural networks**. Any number of layers is fine, as long as there is at least one hidden layer; I suggest going with 3 layers (i.e. the input layer, 1 hidden layer, and the output layer). There are a number of hyperparameters that you could play with when experimenting with neural networks. For example, one hyperparameter is the number of nodes in each hidden layer, another is the choice of nonlinearity, yet another is the regularization parameter (which modulates how much you penalize the size of the weights), and yet another is choice of optimization algorithm (common choices are stochastic gradient descent and Adam).

- **k-fold cross-validation**. Please use your implementation from Assignment 1.

I put "implement" in quotes for two reasons. First, for k-fold cross-validation, I expect you to use your implementation from Assignment 1. Next, I won't require you to (nor do I expect you to) implement SVMs and neural networks; you can instead use machine learning software that you find online. For any implementations you do, you can use whatever programming language you like. For SVMs, it is quite difficult to implement a method for learning "kernelized SVMs", i.e., SVMs that use kernel functions. I therefore very strongly recommend that you use an existing implementation. For neural networks, I also recommend using an existing implementation. SVMs (with different choices of the kernel) and neural networks are both implemented in scikit-learn. For SVMs, see Appendix C for guidance on which implementations to use from scikit-learn.

What to test on

You'll be analyzing the performance of these methods on a somewhat-recently devised classification problem, known as Fashion-MNIST. Basically, the idea is to classify 28×28 grayscale images of clothing/apparel items, where each image belongs to one of ten classes. If you would like to learn more, you can read all about it here:

https://github.com/zalandoresearch/fashion-mnist

You can obtain the dataset from the previous link. Alternatively, you can use this direct link:

https://github.com/zalandoresearch/fashion-mnist/archive/master.zip

For instructions on how to load the dataset, use this README:
https://github.com/zalandoresearch/fashion-mnist/blob/master/README.md

I recommend the section titled "**Loading data with Python (requires Numpy)**". Moreover, you should be able to use scikit-learn for this assignment. If you encounter any trouble loading the data, please see Appendix A before contacting the instructor/TAs.

As mentioned above, Fashion-MNIST corresponds to a multi-class problem with ten classes; however, this assignment is about binary classification. Therefore, once you load the dataset, use only the data points (from both the training and test sets) from classes 5 and 7 (corresponding to "Sandal" and "Sneaker" respectively). You can let class 5 from Fashion-MNIST be class 0 of your binary classification problem (and class 7 from Fashion-MNIST can be class 1 of your binary classification problem).

To make training easier, I recommend normalizing each example in one of the following ways:

- rescaling the inputs by dividing them by 255, so they lie in the range $[0, 1]$;
- normalizing each feature vector so that it has unit (Euclidean) norm.

The first technique will likely work well for most methods, but for an alternative to the first technique, see Appendix B. Also, for the SVM with Gaussian kernel, you might find getting good results is easier using the second method, especially when the number of training examples is small. You are welcome to try other preprocessing techniques. You can feel free to ask classmates about preprocessing techniques that work well or don't work well. Ed Discussion is great venue for discussing with classmates.

⋆ For Component 1, you will not actually train using the original labels. Instead, please make the classification problem more difficult as follows:

- Select a probability in the range from 0.1 to 0.3 (e.g., you might pick 0.1, or 0.2, or 0.25, etc.). Fix this value, and call it p.
- Now, independently for each example in the training set, flip its label with probability p. This is called adding "label noise". It makes the labels noisy and makes machine learning algorithms more likely to overfit.

Note that if you do any type of cross-validation, in any validation set you will be using the noisy labels. *Be sure you do not flip any of the labels in the test set.*

How to do the analysis

Now that you have implementations and the data prepared, it's time for the main part of the assignment: the experiment analysis. Your analysis will go into a file called `report.pdf` and consists of the following parts, which should be presented in separate sections in this order:[1]

First. This part is about training a linear SVM (SVM with linear kernel). The first step is to come up with a set of hyperparameter configurations (a hyperparameter configuration is a way of assigning a value to each hyperparameter). The only hyperparameter that you need to consider for a linear SVM is C. You should try at least ten values of the regularization parameter. I suggest

[1] An important note: each method has hyperparameters. When deciding which hyperparameter configurations you will use, do not use the test set (in particular, do not look at test error). However, you can use a validation set or k-fold cross-validation scores. But note that the more you use k-fold cross-validation to decide which hyperparameter configurations you will consider, the less valuable k-fold cross-validation may be when it comes to hyperparameter tuning.

using a logarithmically spaced grid for initial value $C_0 > 0$ and base $\beta > 1$, so you try C_0, βC_0, $\beta^2 C_0$, etc. The values you try ideally should capture both the regime of underfitting (too much regularization) and overfitting (too little regularization); you might use a validation set or k-fold cross-validation to choose a good set of candidate C values. However, be sure to not touch the test set. Once you have fixed your set of hyperparameter configurations, using only the training data, run k-fold cross-validation (for k being some integer between 5 and 10) to select the optimal value of C. Record this optimal value; you will use it later.

You are now allowed to use the test set. Plot how the training error and test error change as the regularization parameter C varies. Regarding this plot: (a) you can use an even larger set of C values than you used during hyperparameter tuning; (b) use the whole training set (all 12,000 examples) and the whole test set (all 2,000 examples). Discuss the results.

Second. It is time to go beyond linear classifiers. For this, you'll use kernelized SVMs with the Gaussian kernel. This kernel has a scale parameter (sometimes called a "bandwidth" parameter). For each value γ in a good range of values of the scale parameter (I suggest using a logarithmic spacing as you did with the regularization parameter), use k-fold cross-validation on the training set to select the optimal regularization parameter C_γ. Each (γ, C_γ) pair is a tuned SVM, and you have one tuned SVM for each value of γ you considered. Again using k-fold cross-validation, this time on the tuned SVMs, determine the optimal value of γ. Record this value for later use.

You are now allowed to use the test set. For each tuned SVM, train on the full training set and compute both the training error and the test error. Finally, plot the results. The x-axis will be γ, and the y-axis will be the training error (or test error) based on an SVM with parameters tuned to (γ, C_γ). Discuss the results.

How do these test errors compare to the test error of linear SVM (with regularization parameter tuned via cross-validation on the training set)?

Third. We now head into neural network training. There is considerable freedom in specifying a neural network. For example, in terms of structure, you can specify how many hidden layers there are, how many nodes are in each hidden layer, and what nonlinearity each node uses.[2] There is also a choice of how much regularization to use, how to configure the learning rate schedule, and whether or not to use early stopping. My suggestion is to (using only the training set; do not look at the test set!!) come up with a computationally manageable set of hyperparameter configurations. To avoid having to consider too many hyperparameter configurations, I suggest you consider only a small number of regularization parameter values (the default in scikit-learn should work OK), and instead focus more on networks with one or two hidden layers, vary the number of nodes in each hidden layer, and try a few different choices for the nonlinearity. For the hyperparameter configurations that you decide to use, once again use k-fold cross-validation to select the optimal tuning of the hyperparameters. Record this optimal hyperparameter configuration; you will use it later.

You are now allowed to use the test set. Come up with a few (at least two) experiments where you vary a single hyperparameter (for example, you might vary the number of nodes in a hidden layer, or you might vary the maximum number of epochs of training from some small value to a larger value). For each experiment, train on the full training set and plot both training error and test error versus the hyperparameter you are varying. Discuss the results.

Fourth. From each of the three previous parts respectively, you now have an optimally tuned linear SVM, an optimal tuned SVM with the Gaussian kernel, and an optimally tuned neural network. It's time to compare the tuned version of each method. Train each optimally-tuned

[2] In scikit-learn, you can only specify a single nonlinearity (which will be used for all hidden nodes).

method on the *entire* training set and report the method's test error. Discuss the results. Are the test errors significantly different? Think about confidence intervals (which depend on the test set size) to assess significance.

Advice if training an SVM is taking too long

You might find that training an SVM can take a long time. This is especially the case when using the Gaussian (rbf) kernel. In order to complete the assignment within a reasonable amount of time, it is OK if you reduce the size of the original training set. **If you do this, please use the same reduced-size training set throughout Component 1 of your assignment.**

For example, each of classes 5 and 7 consists of 6000 training examples (for 12000 training examples total). To help with reducing the training time, you could only use 3000 training examples for each class (for 6000 classes total). This likely will make training 4 times as fast, and so it should be good enough. If that still takes too long, you might even go as low as using only 1000 training examples for each class.

Final advice on the report.

Please make your analysis concise (yet thorough). Don't ramble, and don't make stuff up. Act as if you are a respected scientist presenting some work to your colleagues.

What to submit

In all, for the analysis portion of the assignment, you should submit (as a zip file):

- the file `report.pdf` explained above;
- a file called `README.txt` which contains instructions for running your code (for any code you wrote) or gives proper attribution to any code/datasets you got from other sources (like the Internet, for example). If you mostly used some existing code/software but needed to modify a few files, then give attribution and mention the files you modified.
- a file called `code.zip`, containing your code. Please organize this file well (embrace the idea of directories). In case you mostly used some existing code/software but needed to modify a few files, then just provide those files here, and make sure your `README.txt` mentions those files in relation to the existing code/software.
- any additional files that you need.

2 Component 2: Data augmentation (Bonus for both CSC 503 and SENG 474)

Note: for Component 2, use the original, "clean" labels (not the noisy labels) when training.

Let's see how much performance we can squeeze out of a very small training set. You will create a small training set and a small validation set. To do so, take two, small disjoint subsamples of the original training set. Be sure that each subsample is class-balanced (an equal number of examples from each class), and make sure that no example appears in both subsamples. I recommend the first sample, the small training set, have 20 examples total (so, 10 examples per class). I recommend the second sample, the small validation set, have at least 100 examples (if exactly 100 examples, then 50 examples per class). Believe it or not, even a small training sample of just 20 examples allows for decent performance.

The idea of data augmentation is to augment the training set with additional examples, where each new example is obtained via a transformation of one of the original examples. There are different types of augmentation you might consider. One common one is random rotations:

- for each original image, generate m new images, where each new image is a random rotation of the original image. Here, for each random rotation, the number of degrees is drawn uniformly at random from a range of degrees $[-r, r]$. For example, if we set $r = 30$, then all rotations will be between -30 degrees and 30 degrees.

If you want to use random rotations, I have provided Python code in `randrotate.py` that shows how to perform these rotations. For random rotations, we have a range parameter r; this can be viewed as a hyperparameter. Note that m, the number of augmented examples per original example, is also a hyperparameter. I suggest trying values in the range of 10 to 100. You might dream up a different type of data augmentation. If you do, explain why it might be useful, and ensure your method has a scalar range parameter so that you have a hyperparameter that can be varied.

Now, here is the experiment that you should run. For the SVM with Gaussian kernel using your small training set (with data augmentation, which varies as m and r vary) and small validation set (with no data augmentation), tune the following hyperparameters: the number of augmented examples per original example m, the range parameter r, the bandwidth parameter γ, and the regularization parameter C.[3] Once you tune the hyperparameters, train your tuned model on the small training set (with data augmentation) and compute the test error. Also, for comparison, tune γ and C for an SVM with Gaussian kernel based on the small training set with no data augmentation, and compute the test error. What do you observe? Did data augmentation help? Discuss.

In addition, make a plot of how, for a fixed, nontrivial value of r (picked however you like), the test error of an optimally tuned SVM with Gaussian kernel (tuned γ and C) trained *with* data augmentation varies with m. Also, make a plot of how, for a fixed, nontrivial value of m (picked however you like), the test error of an optimally tuned SVM with Gaussian kernel trained *with* data augmentation varies with r.

Report your results by augmenting `report.pdf` with an additional section.

3 How you'll be marked

For both CSC 503 and SENG 474, the total number of marks is 100.

For both grad students (CSC 503) and undergrads (SENG 474):

- the analysis (in `report.pdf`) is worth 90 marks.
- you receive 10 marks for your code for Component 1.
- you can earn up to 20 bonus marks for Component 2. If you get more than 100 marks, your extra marks can be used to compensate marks lost on Assignments 1 and 3 (undergrads) or Assignment 1 (grad students).

[3] Since you already have to tune γ and C, I suggest not trying a large number of values for r and also not trying a large number of values for m.

A Loading Fashion-MNIST

For loading the Fashion-MNIST data, I believe everything should work fine if you are using Python in a terminal. However, in case you have difficulty loading the data from a Jupyter notebook, then you can also get the data in CSV format here (you will have to get a Kaggle account to download the csv files):

https://www.kaggle.com/eliotbarr/fashion-mnist-tutorial#data

B Another type of preprocessing

For neural networks and linear SVMS in particular, you might find the below preprocessing technique (which can be applied to the original data, i.e., without any prior preprocessing) gives better results:

- For each feature separately: shift and rescale the values of the feature so that, among the training examples, the feature's sample mean is 0 and its sample variance is 1. Again, be sure to apply the same scaling to all the data; therefore, a given feature's sample mean and sample variance in the test set are not necessarily 0 and 1 respectively.

C SVMs in scikit-learn

C.1 Linear SVMs

For linear SVMs, scikit-learn has an implementation that trains using stochastic gradient descent. See the documentation at this link. This implementation will allow for much faster training than using scikit-learn's `svm.SVC`.

You will notice that scikit-learn's SGDClassifier does not use the regularization parameter C. Instead, it uses a different parameter α. So, what is the relationship between C and α? I will tell you the relationship, but first, I will tell you how you yourself can identify the relationship. To see the relationship, compare the following two expressions:

- In scikit-learn's SVM User Guide, the expression appearing immediately after the text "SVC solves the following primal problem:" (note that each ζ_i is the hinge loss on example i)
- In scikit-learn's SGD User Guide, the expression appearing immediately after the text "To find the model parameters, we minimize the regularized training error given by"

Dividing the second expression by α allows easy comparison of the two expressions, after which we see the correspondence $C = \frac{1}{\alpha n}$ and hence $\alpha = \frac{1}{nC}$.

C.2 SVMs with the Gaussian kernel

In the case of SVMs with the Gaussian kernel, I suggest you use scikit-learn's `svm.SVC`, whose documentation can be found at this link. Training may be slow, but it should be faster for some values of the regularization parameter C. Even so, be sure to try several settings for C.