ECE 6254, Spring 2024

Homework # 5

Due Sunday, March 31 at 11:59pm EST.

Suggested reading:

- *Elements of Statistical Learning* (by Hastie, Tibshirani, and Friedman): Section 4.5 (pages 129–135) discusses optimal separating hyperplanes; Sections 12.1–12.3 (pages 417–438) discuss support vector machines and kernels in more detail.

Problems:

1. **Lagrangian duality.** In this problem, we explore Lagrangian duality with a simple example. Consider the optimization problem

$$\begin{aligned} \underset{x\in\mathbb{R}}{\text{minimize}} \quad & 2x^2+4 \\ \text{subject to} \quad & (x-1)(x-5) \le 0 \end{aligned}$$

In this problem, we will denote $f(x) = 2x^2 + 4$.

   (a) Provide a simple description of the feasible set.

   (b) Plot the objective $f(x)$ and indicate on your plot the feasible set. From this plot, determine the minimizer $x^\star$ as well as the value of the objective function at the minimizer $f(x^\star)$.

   (c) Derive the Lagrangian $L(x, \lambda)$. Plot the Lagrangian for a few values of λ.

   (d) Derive and plot the dual function $L_D(\lambda)$.

   (e) State the dual problem and find (by hand) the maximizer $\lambda^\star$ as well as the value $L_D(\lambda^\star)$. Does strong duality hold (i.e., does $f(x^\star) = L_D(\lambda^\star)$) ?

2. **Unconstrained soft margin classifier, continued.** Last time, we showed how the optimal soft margin classifier was related to the *hinge loss*, $L_c(x) = \max(0, c - x)$ for some $c > 0$. Specifically, we speculated that the unconstrained optimization problem

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{n} L_c(\frac{C}{n} y_i(\mathbf{w}^\top \mathbf{x}_i + b))$$

with $c = \frac{C}{n}$ is equivalent to the optimal soft margin classifier (which we'll call the constrained problem)

$$\begin{aligned} \min_{\mathbf{w},b,\{\xi_i\}} \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i \\ \text{s.t.} \quad & y_i(\mathbf{w}^\top \mathbf{x}_i + b) \ge 1 - \xi_i \qquad i = 1,\dots,n \\ & \xi_i \ge 0 \qquad i = 1,\dots,n \end{aligned}$$

Let $\mathbf{w}^c, b^c, \{\xi_i^c\}_{i=1}^n$ be the optimal solutions to the constrained optimal soft margin problem, and let $\mathbf{w}^u, b^u$ be the optimal solutions to the unconstrained optimization problem.

In this problem, we will take this one step further and show that both problems achieve the same minimum value, i.e.,

$$\frac{1}{2}\|\boldsymbol{w}^c\|_2^2 + \frac{C}{n}\sum_{i=1}^n \xi_i^c = \frac{1}{2}\|\boldsymbol{w}^u\|_2^2 + \sum_{i=1}^n L_c\left(\frac{C}{n}y_i\left((\boldsymbol{w}_i^u)^\top \boldsymbol{x}_i + b^u\right)\right).$$

We know from last assignment that

$$\frac{1}{2}\|\boldsymbol{w}^c\|_2^2 + \frac{C}{n}\sum_{i=1}^n \xi_i^c \geq \frac{1}{2}\|\boldsymbol{w}^c\|_2^2 + \sum_{i=1}^n L_c\left(\frac{C}{n}y_i\left((\boldsymbol{w}_i^c)^\top \boldsymbol{x}_i + b^c\right)\right).$$

(a) Show that

$$\frac{1}{2}\|\boldsymbol{w}^c\|_2^2 + \sum_{i=1}^n L_c\left(\frac{C}{n}y_i\left((\boldsymbol{w}^c)^\top \boldsymbol{x}_i + b^c\right)\right) \geq \frac{1}{2}\|\boldsymbol{w}^u\|_2^2 + \sum_{i=1}^n L_c\left(\frac{C}{n}y_i\left((\boldsymbol{w}^u)^\top \boldsymbol{x}_i + b^u\right)\right).$$

(b) Defining $\xi_i^u = \max\left(0, 1 - y_i\left((\boldsymbol{w}^u)^\top \boldsymbol{x}_i + b^u\right)\right)$, we have that

$$\frac{1}{2}\|\boldsymbol{w}^u\|_2^2 + \sum_{i=1}^n L_c\left(\frac{C}{n}y_i\left((\boldsymbol{w}^u)^\top \boldsymbol{x}_i + b^u\right)\right) = \frac{1}{2}\|\boldsymbol{w}^u\|_2^2 + \frac{C}{n}\sum_{i=1}^n \xi_i^u.$$

Let $\boldsymbol{w}^e$, b^e, and $\{\xi_i^e\}_{i=1}^n$ be the solution to the *equality constrained* optimization problem

$$\min_{\mathbf{w}, b, \{\xi_i\}} \frac{1}{2}\|\mathbf{w}\|_2^2 + \frac{C}{n}\sum_{i=1}^n \xi_i$$
$$\text{s.t.} \qquad \xi_i = \max\left(0, 1 - y_i\left(\boldsymbol{w}^\top \boldsymbol{x}_i + b\right)\right) \qquad i = 1, \dots, n$$

Show that

$$\frac{1}{2}\|\boldsymbol{w}^u\|_2^2 + \frac{C}{n}\sum_{i=1}^n \xi_i^u \geq \frac{1}{2}\|\boldsymbol{w}^e\|_2^2 + \frac{C}{n}\sum_{i=1}^n \xi_i^e.$$

(c) Solutions to constrained optimization problems live in some *feasible set*. For example, we can rewrite the equality constrained optimization problem in part b.) as follows:

$$\min_{\mathbf{w}, b, \{\xi_i\} \in \mathcal{A}} \frac{1}{2}\|\mathbf{w}\|_2^2 + \frac{C}{n}\sum_{i=1}^n \xi_i$$

where $\mathcal{A} = \left\{(\mathbf{w}, b, \{\xi_i\}) : \xi_i = \max\left(0, 1 - y_i\left((\boldsymbol{w}^u)^\top \boldsymbol{x}_i + b^u\right)\right)\right\}$. Using the fact that for sets $\mathcal{A}$ and $\mathcal{B}$, if $\mathcal{B} \subset \mathcal{A}$, then

$$\min_{\boldsymbol{x} \in \mathcal{B}} f(\boldsymbol{x}) \geq \min_{\boldsymbol{x} \in \mathcal{A}} f(\boldsymbol{x}),$$

reason that

$$\frac{1}{2}\|\boldsymbol{w}^e\|_2^2 + \frac{C}{n}\sum_{i=1}^n \xi_i^e \geq \frac{1}{2}\|\boldsymbol{w}^c\|_2^2 + \frac{C}{n}\sum_{i=1}^n \xi_i^c.$$

Hint: Let the feasible set of the equality constrained optimization problem be $\mathcal{B} = \left\{(\boldsymbol{w}, b, \{\xi_i\}) : \xi_i = \max\left(0, 1 - y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b)\right)\right\}$.

(d) Show that the two optimization problems achieve the same minimum value, i.e.,

$$\frac{1}{2}\|\boldsymbol{w}^c\|_2^2 + \frac{C}{n}\sum_{i=1}^{n} \xi_i^c = \frac{1}{2}\|\boldsymbol{w}^u\|_2^2 + \sum_{i=1}^{n} L_c\left(\frac{C}{n} y_i \left((\boldsymbol{w}_i^u)^\top \boldsymbol{x}_i + b^u\right)\right).$$

3. **SVMs and image classification.** In this problem you will use SVMs to build a simple text classification system. To begin, you need to get the MNIST dataset. You can do this in Python using the commands

```
from sklearn.datasets import fetch_openml
mnist = fetch_openml("mnist_784", version=1, as_frame=False)
```

This downloads the dataset and stores it in a default location (~`/scikit_learn_data/`). You can also use the optional parameter `data_home=path` in `fetch_mldata` to direct the dataset to a custom path if you wish. The first time you run this command, the data will be downloaded from `mldata.org`, but once it has been downloaded this will simply load the dataset into the variable `mnist`.

This data set contains 70 000 handwritten digits, each of size 28×28 pixels.[1] You should begin by putting this data into a more convenient form by setting

```
X = mnist.data
y = mnist.target.astype(int) #default value is np.array of str, change to int
```

Each row of `X` corresponds to one image. You can use the following to plot the `j`th image:

```
import matplotlib.pyplot as plt
plt.title('The jth image is a {label}'.format(label=int(y[j])))
plt.imshow(X[j].reshape((28,28)), cmap='gray')
plt.show()
```

In this problem you will build a classifier to classify between the (sometimes very similar) images of the digits "4" and "9". To get just this data, you can use

```
X4 = X[y==4,:]
X9 = X[y==9,:]
```

There are a little under 7 000 examples from each class. You should begin by using this data to form three distinct datasets: 2 000 points from each class will be used for designing the classifier (this is the *training set*), 2 000 points from each class will be used for tuning parameters (this is the *validation set*), and the remainder will be used to evaluate the performance of our classifier (this is the *testing set*). In summary, you should split the dataset into three sets:

[1]You can learn more about this dataset at `http://yann.lecun.com/exdb/mnist/`.

- **Training set:** 2 000 points per class, used to fit the classifier when testing values of C.
- **Validation set:** 2 000 points per class, used to find the "best" value of C.
- **Test set:** Rest of points in a particular class, used to test the performance of the SVM classifier for a chosen value of C.

SVMs involve tuning parameters, such as C or kernel dependent parameters. We will set the values of parameters in a principled way using the so-called "holdout method". For example, to set C, you will want to decide on a finite set of "grid points" on which to test C. For each value of C, you will train an SVM on the training set, and then compute the error rate on the validation set. In the end, you can then choose the value of C that results in a classifier that gives the smallest error on the validation set. Other parameters can be set in an analogous manner.

Note: Once you have selected the value of a particular parameter, you may then retrain on all the entire training set (including the validation set), but you should **never** use the testing set until every parameter in your algorithm is set. These are used exclusively for computing the final test error.

To train the SVM , you can use the built in solver from scikit-learn. As an example, to train a linear SVM on the full dataset with a value of $C = 1$, we would use

```
from sklearn import svm
clf = svm.SVC(C=1.0,kernel='linear')
clf.fit(X,y)
```

Once you have trained the classifier, you can calculate the probability of error for the resulting classifier on the full dataset via

```
Pe = 1 - clf.score(X,y)
```

(a) In the provided file `mnist_svms.py`, complete the code to construct the training, validation, and testing sets using any method you want, including built in `sklearn` functions, like `train_test_split`.

(b) Train an SVM using the kernels $k(\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{u}^{\mathrm{T}}\boldsymbol{v} + 1)^p$, $p = 1, 2$, that is, the inhomogeneous linear and quadratic kernel. To do this, you will want to set the parameters `kernel='poly'` and `degree=1` or `degree=2`. I suggest a logarithmic grid of values to test both $C \ll 1$ and $C \gg 1$.

For each kernel, report the best value of C, the test error, and the number of data points that are support vectors (returned via `clf.support_vectors_`. Turn in your code.

(c) Now, set $C = 10$ and train an SVM using the radial basis function kernel $k(\boldsymbol{u}, \boldsymbol{v}) = e^{-\gamma\|\boldsymbol{u}-\boldsymbol{v}\|^2}$ (by setting the parameters `kernel='rbf', gamma=gamma`). Report the best value of γ, the test error, and the number of support vectors.

(d) We also want to visualize the support vectors that are the "hardest" examples to classify. In `mnist_svm.py`, there is code included to plot the 16 support vectors that violate the margin by the greatest amount. For each kernel, turn in the generated 4×4 subplot.

4. **Nonlinear regression.** Use the following code to generate training data for a nonlinear regression problem.

```
import numpy as np
np.random.seed(2022)
n = 100
xtrain = np.random.rand(n)
ytrain = np.sin(9*xtrain) + np.sqrt(1/3.0)*np.random.randn(n)
```

In the problems below, you should fit a function to this training data. In all of the methods below, there will be one or more parameters to set. You can do this manually using whatever approach you like (do not go crazy optimizing these, just tune the parameters until your estimate looks reasonable.) To quantitatively evaluate the quality of your fit, you may use

```
xtest = np.linspace(0,1,1001)
ytest = np.sin(9*xtest)
```

   (a) Use kernel ridge regression with the radial basis function kernel given by $k(x, x') = e^{-\gamma|x-x'|^2}$. You should implement this on your own using the notes from class. Report the value of γ and the value of λ you selected, and provide a plot of your function. Submit your code.

   (b) Use support vector regression with the radial basis function kernel. You can do this via

```
from sklearn.svm import SVR
reg = SVR(C=1.0, epsilon=0.1, kernel='rbf', gamma=1.0)
reg.fit(xtrain.reshape(-1,1),ytrain)
```

   To evaluate this function on `xtest` you may use `reg.predict(xtest.reshape(-1,1))` and compare the results to `ytest`.

   You now have three! parameters to fit. Play around with them (I suggest optimizing one at a time, starting from the default values given above) until you are happy with the results. Report the value of γ and also the values of C and ϵ you selected, and provide a plot of your function. Submit your code.

   (c) Comment on the differences between these two approaches. How did the bandwidth (i.e., the parameter γ) you selected differ between the two approaches (and why?) What advantages do you think one approach has versus the other?

5. **Unsupervised clustering on handwritten digits.** In this problem, we will explore how the main methods discussed in class (PCA, kernel PCA, MDS, LLE, and Isomap) perform dimensionality reduction on a (small) real-world dataset. We will work with the digits dataset, which is similar to MNIST. This dataset consists of $\sim$ 1800 images of handwritten digits. In this problem, we will work with the first 7 classes (digits 0 through 6). To load this dataset, you can use the following commands.

```
from sklearn.datasets import load_digits
digits = load_digits(n_class=7)
```

```
X, y = digits.data, digits.target
n_samples, n_features = X.shape
```

In this problem, you may use build in sklearn functions for PCA, kernel PCA, MDS, LLE, and Isomap using the following modules.

```
from sklearn import manifold, decomposition
```

For all parts, the dimension of the learned low-dimensional embedding should be 2. This will allow us to visualize the learned embeddings. A function to visualize the embeddings is included in `dimensionality_reduction.py`.

For parts (a) - (e), **include a visualization of the learned embedding**.

(a) Perform dimensionality reduction with PCA.

(b) Perform dimensionality reduction with kernel PCA, with the inhomogenous linear and quadratic kernels, that is, $k(\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{u}^\top \boldsymbol{v} + 1)^p, p = 1, 2$.

(c) Perform dimensionality reduction with MDS.

(d) Perform dimensionality reduction with LLE. Adjust parameter `n_neighbors` as needed.

(e) Perform dimensionality reduction with Isomap. Adjust parameter `n_neighbors` as needed.

Note: The for LLE and Isomap, you can set `n_neighbors` by inspection/based on how the learned embedding looks.