Introduction to Machine Learning (67577)

Exercise 3
Classification

Semester B, 2020

Submission

1. Submission is due 14.5.2020 at 23:55 via Moodle.
2. You should sumbit one file named **ex_3_FirstName_LastName.tar** containing the following files:
   - **Ex3_Answers.pdf** - a pdf file which contains all your answers to both theoretical parts and practical parts.
   - **models.py**
   - **comparison.py**
   - **mnist_data.py**

Bayes Optimal and LDA

[Relevant material - Lecture 3]

Consider binary classification with sample space $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \{\pm 1\}$. One way to model the data generation process is to assume that our samples are drawn i.i.d from an unknown **joint** distribution $\mathcal{D}$ over $\mathcal{X} \times \{\pm 1\}$. (Namely, we draw the sample and the label together from a joint distribution over $\mathcal{X} \times \{\pm 1\}$.) In this question you'll learn about two concepts that were not discussed in the lecture: The Bayes Optimal Classifier, and Linear Discriminent Analysis (LDA).

1. If we knew $\mathcal{D}$, our best predictor would have been assigning the class with the higher probability:

$$\forall \mathbf{x} \in \mathcal{X} \; h_{\mathcal{D}}(\mathbf{x}) = \begin{cases} +1 & \Pr(y = 1|\mathbf{x}) \geqslant \frac{1}{2} \\ -1 & otherwise \end{cases}$$

   where the probability is over $\mathcal{D}$. This classifier is known as the **Bayes Optimal** classifier.

   Show that

$$h_{\mathcal{D}} = \underset{y \in \{\pm 1\}}{\operatorname{argmax}} \Pr(\mathbf{x}|y) \Pr(y).$$

2. Assume that $\mathcal{X} = \mathbb{R}^d$ and that $\mathbf{x}|y \sim \mathcal{N}(\mu_y, \Sigma)$ for some mean vector $\mu_y \in \mathbb{R}^d$ and covariance matrix $\Sigma \in \mathbb{R}^{d\times d}$ (that is, the covariance matrix Σ is the same for both $y \in \{\pm 1\}$, but the expectation μ_y is different for each $y \in \{\pm 1\}$). In other words,

$$f(\mathbf{x}|y) = \frac{1}{\sqrt{(2\pi)^d \det(\Sigma)}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\mu_y)^\top \Sigma^{-1}(\mathbf{x}-\mu_y)\right\}$$

where f is the density function for the multivariate normal distribution. Show that in this case, if we knew μ_{+1}, μ_{-1} and Σ then the Bayes Optimal classifier is

$$h_{\mathcal{D}}(\mathbf{x}) = \operatorname*{argmax}_{y\in\{\pm 1\}} \delta_y(\mathbf{x}),$$

where δ_{+1} and δ_{-1} are functions $\mathbb{R}^d \to \mathbb{R}$ given by

$$\delta_y(\mathbf{x}) = \mathbf{x}^\top \Sigma^{-1}\mu_y - \frac{1}{2}\mu_y^\top \Sigma^{-1}\mu_y + \ln \Pr(y) \qquad y \in \{\pm 1\}$$

The functions $\delta_{\pm 1}$ are called **discriminant functions**: this classification rule predicts the label, based on which of the two discriminant functions is larger at the sample $\mathbf{x}$ we wish to classify.

3. In practice, we don't know μ_{+1}, μ_{-1}, Σ and $\Pr(y)$. In order to turn the above into a classifier, given a training set $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$, we need to estimate them. Write your formula for estimating μ_{+1}, μ_{-1}, Σ and $\Pr(y)$ based on S.

   When you plug these estimates into the functions $\delta_{\pm 1}$, you get a classifier known as **Linear Discriminant Analysis (LDA)**.

Spam

[Relevant material - Lecture 3]

4. You are building a spam filter - a classifier that receives an email and decides whether it's a spam message or not. What are the two kinds of errors that your classifier could make? Which of them is the error we really don't want to make? Which of the labels $\{\text{spam}, \text{not-spam}\}$ should be the **negative** label and which should be the **positive** label, if we want the false-positive error (Type-I error) to be the error we really don't want to make?

SVM- Formulation

[Relevant material - Recitation 4]

5. The canonical form of a Quadratic Program (QP) is:

$$\begin{aligned} &\operatorname*{argmin}_{\mathbf{v}\in\mathbb{R}^n} \left(\frac{1}{2}\mathbf{v}^\top Q\mathbf{v} + \mathbf{a}^\top \mathbf{v}\right) \\ &\text{s.t. } A\mathbf{v} \leqslant \mathbf{d}, \end{aligned}$$

where $Q \in \mathbb{R}^{n \times n}, A \in \mathbb{R}^{m \times n}, \mathbf{a} \in \mathbb{R}^n, \mathbf{d} \in \mathbb{R}^m$ are fixed vectors and matrices.

Write the Hard-SVM problem as a QP problem in canonical form. Specifically, using the Hard-SVM problem formulation

$$\underset{(\mathbf{w},b)}{\operatorname{argmin}} ||\mathbf{w}||^2 \text{ s.t. } \forall i, y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geqslant 1.$$

what are the values of $Q, A, \mathbf{a}, \mathbf{d}$ that express this problem as a QP in canonical form?

Why is it interesting? QP software solvers take QP in canonical form. To use a QP solver, you'll need to express the SVM problem as QP in canonical form as above.

6. In the Soft-SVM we defined the problem:

$$\arg\min_{\mathbf{w},\{\xi_i\}} \frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{m}\sum_{i=1}^{m} \xi_i \text{ s.t. } \forall_i, y_i\langle \mathbf{w}, \mathbf{x}_i \rangle \geqslant 1 - \xi_i \text{ and } \xi_i \geqslant 0$$

Show that this problem is equivalent to the problem (namely that these problem have the same solutions)

$$\arg\min_{\mathbf{w}} \frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{m}\sum_{i=1}^{m} \ell^{hinge}(y_i\langle \mathbf{w}, \mathbf{x}_i \rangle),$$

where $\ell^{hinge}(a) = \max\{0, 1 - a\}$.

Implemention and simulation-comparison of different classifiers

[Relevant material - Labs 3+4]

In the following sections we would like to empirically estimate the performance of several classification algorithms. As a baseline we use the **Perceptron** - a well known algorithm implementing the simple Half-space classifier you saw in the lecture. Here is the Perceptron algorithm:

Batch Perceptron

input: A training set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$
initialize: $\mathbf{w}^{(1)} = (0, \ldots, 0)$
  for $t = 1, 2, \ldots$
    if $(\exists\ i \text{ s.t. } y_i\langle \mathbf{w}^{(t)}, \mathbf{x}_i \rangle \leq 0)$ **then**
      $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + y_i\mathbf{x}_i$
    else
      output $\mathbf{w}^{(t)}$

7. In this question you will implement the following classifier: Half-space (using the Perceptron algorithm), LDA from Question 1 above, SVM, logistic regression and decision tree. For each model you'll create a python class and implement the following methods in it:

   (a) $\mathsf{fit}(X, \mathbf{y})$ - Given a training set as $X \in \mathbb{R}^{d \times m}$ and $y \in \{\pm 1\}^m$, this method learns the parameters of the model and stores the trained model (namely, the variables that define hypothesis chosen) in `self.model`. The method returns nothing.

(b) predict(X) - Given an unlabeled test set $X \in \mathbb{R}^{d \times m'}$, predicts the label of each sample. Returns a vector of predicted labels $y \in \{\pm 1\}^{m'}$.

(c) score(X, y) - Given an unlabeled test set $X \in \mathbb{R}^{d \times m'}$ and the true labels $y \in \{\pm 1\}^{m'}$ of this test set, returns a dictionary with the following fields:

- `num_samples`: number of samples in the test set
- `error`: error (misclassification) rate
- `accuracy`: accuracy
- `FPR`: false positive rate
- `TPR`: true positive rate
- `precision`: precision
- `recall`: recall

Please submit a file called models.py with all following classes:

(a) Perceptron - Implement a half-space classifier using the perceptron algorithm. Please do not use any package other than numpy. Please replace the "for" loop that appears in line 3 of the algoritm's description above with a "while" one, which stops when the classifier fits correctly all the training set. Also, if more than 1 option exists in the "if" section (forth line of the algorithm above), choose one of your choice. **Note:** The algorithm above (and the half-space classifier you saw in class) assume the homogeneous case $b = 0$. To use your code as a classifier in the non-homogeneous case, you can add an entry with value 1 to each vector $\mathbf{x}$ in both training and test - a trick you saw in the linear regression lecture.

(b) LDA - Implement the LDA classifier from Question 1. Please do not use any package other than numpy.

(c) SVM - Implement SVM. You may use sklearn. More about this function is in sklearn's SVC library. When creating an object of this type there are settings you can specify, choose C=1e10, kernel='linear':

```
from sklearn.svm import SVC
svm = SVC(C=1e10, kernel='linear')
```

These will make your SVM learn a linear classifier (i.e. without using a kernel) that is very similar to the hard-SVM we talked about in class. Once you create the object, you can use the functions fit, prediction, score of its instance.

(d) Logistic - Implement logistic regression. You may use sklearn as you saw in the Lab. To use this package declare the following:

```
from sklearn.linear_model import LogisticRegression
logistic = LogisticRegression(solver='liblinear')
```

Once you create the object, you can use the functions fit, prediction, score of its instance.

(e) DecisionTree - Implement a decision tree. You may use sklearn as you saw in the Lab. To use this package declare the following:

```
from sklearn.tree import DecisionTreeClassifier
```

You should provide the parameters for this function as you see fit (e.g., max depth of the tree).

The next questions should be implemented in a file called "comparison.py". Please submit this file along with "models.py" from above.

8. Assume a distribution that generates the data points $\mathbf{x}$ to be $\mathcal{D} = \mathcal{N}(\mathbf{0}, \boldsymbol{I}_2)$, i.e. a two-dimensional Gaussian with mean vector of zeros and a unit matrix for covariance. The true labels are determined by $f(\mathbf{x}) = \text{sign}(\left\langle \begin{pmatrix} 0.3 \\ -0.5 \end{pmatrix}, \mathbf{x} \right\rangle + 0.1)$.

   Create a function **draw_points**(m) that given an integer m returns a pair X, y where X is $2 \times m$ matrix where each column represents an i.i.d sample from the distribution above, and $y \in \{\pm 1\}^m$ is its corresponding label, according to $f(\mathbf{x})$.

   Use numpy.random.multivariate_normal to draw the points.

9. For each $m \in \{5, 10, 15, 25, 70\}$, draw m training points $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ and create the following figure:

   - The drawn data points, colored according to their labels (e.g., blue for positive labels and orange for negative ones)
   - Add the hyperplane of the **true hypothesis** (the function f)
   - Add the hyperplane of the hypothesis generated by the **perceptron**
   - Add the hyperplane of the hypothesis generated by **SVM**
   - Add a legend to explain which hyperplane is which

   Add the 5 plots (one for each m) to your PDF file.

10. We'll now add a test set to compare three of the above algorithms. We'll use the following procedure:

    - Draw training points $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ from the distribution $\mathcal{D}$ above and classify them according to a true hypothesis f (i.e. with labels $y_1, \ldots, y_m$). **Note:** The training data should always have points from two classes. So if you draw a training set where no point has $y_i = 1$ or no point has $y_i = -1$ then just draw a new dataset instead, until you get points from both types.
    - Draw k test points $\{\mathbf{z}_1, \ldots, \mathbf{z}_k\}$ from the same distribution $\mathcal{D}$ and calculate their true labels as well.
    - Train a **Perceptron** classifier, an **SVM** classifier and an **LDA** classifier on $\{(\mathbf{x}_i, y_i)\}_{i=1}^m$.
    - Calculate the accuracy of these classifiers (the fraction of test points that is classified correctly) on $\{\mathbf{z}_1, \ldots, \mathbf{z}_k\}$.

    For each $m \in \{5, 10, 15, 25, 70\}$, repeat the above procedure 500 times with $k = 10000$ and save the accuracies (or just keep the mean accuracy, remember that the accuracy is a number between 0 to 1) of each classifier. Finally, plot the mean accuracy as function of m for each of the algorithms (SVM, Perceptron and LDA). Do not forget to add a legend to your graph. Add the plot to your PDF file.

11. Which classifier did better? why do you think that happened? No need for a formal argument, just explain what are the properties of the classifiers that cause these results.

Classificaton of two digits from the MNIST Dataset

[Relevant material - Labs 3+4]

Please submit the coding of this part in a file called mnist_data.py.

MNIST is a large database of handwritten digits that is commonly used for training various image processing systems. To download this database locally, please use the following code in your file (note that at first time it may take several minutes):

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
(x_train, y_train),(x_test, y_test) = mnist.load_data()
```

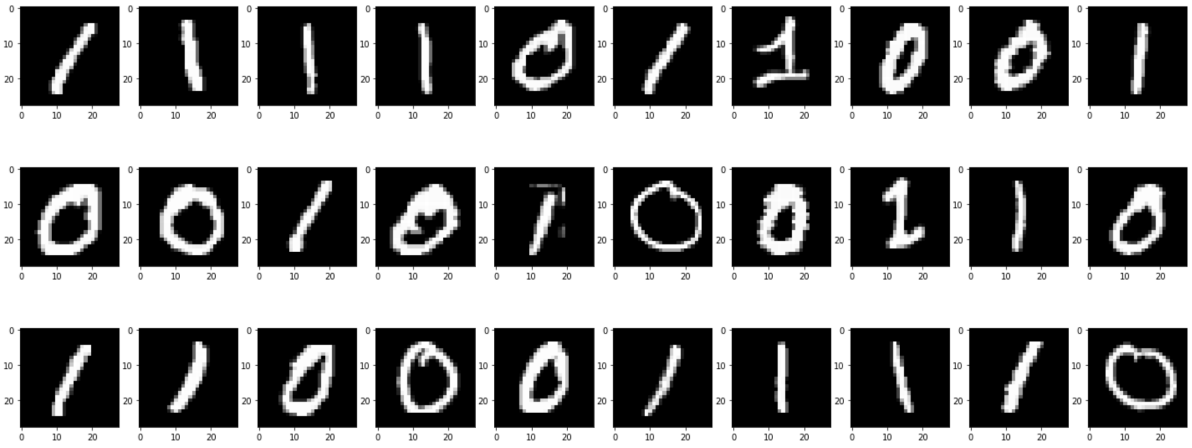
Note: If you have any problem with tensorflow, please do not fight it. Instead, there are many ways to download MNIST data, like this link, or you can download it locally into a csv file, and just read it from there, like this link.

In order to create a binary classifier we'll use only '0' and '1' digits. Therefore our database would be:

```
train_images = np.logical_or((y_train == 0), (y_train == 1))
test_images = np.logical_or((y_test == 0), (y_test == 1))
x_train, y_train = x_train[train_images], y_train[train_images]
x_test, y_test = x_test[test_images], y_test[test_images]
```

Note that the size of X is $m \times 28 \times 28$, since each sample $\mathbf{x}$ is a 28×28 pixels image.

Here is some examples from your current dataset:



Now that we have all we need we can start with the fun part!

12. (Play with the dataset) Draw 3 images of samples labeled with '0' and 3 images of samples labeled with '1'. Use the function matplotlib.pyplot.imshow (you can read about the function here).

13. Create a function rearrange_data(X) that given a data as a tensor of size $m \times 28 \times 28$, returns a new matrix of size $m \times 784$ with the same data.

14. (similar to question 10, with different models) Consider the following procedure:

    - Draw m training points $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ by choosing uniformly and at random from the train set. **Note:** The training data should always have points from two classes. So if you draw a training set where no point has $y_i = 1$ or no point has $y_i = -1$ then just draw a new dataset instead, until you get points from both types.
    - Train a **logistic regression** classifier, an **Soft-SVM** classifier, a **decision tree** and a k**-nearest neighbors** classifier on $\{(\mathbf{x}_i, y_i)\}_{i=1}^m$. You may choose any existing implementation of these algorithms as long as you note clearly in your solution which implementation you used. (Choose the regularization parameter for Soft-SVM and k for nearest neighbors to the best of your judgment - you're not expected to make the best possible choice at this stage. Although you are certainly encouraged to do your best and use side computations if you want.)
    - Calculate their accuracy (the fraction of test points that is classified correctly) on the entire test set.

    For each $m \in \{50, 100, 300, 500\}$, repeat the above procedure 50 times and save the elapsed running time and the accuracy (or just keep the mean accuracy, remember that the accuracy is a number between 0 to 1) of each classifier. Finally, plot the mean accuracy as function of m for each of the algorithms (SVM, Logistic regression, decision tree and nearest neighbors). Do not forget to add a legend to your graph. Add the plot to your PDF file. Discuss what you've seen about the difference in running time.