DSE 220: Machine learning

Lab 2 — Linear classification

2.1 Programming problems

1. *Binary Perceptron.* In this problem, you will code up the (binary) Perceptron algorithm and use it to classify the Iris data set.

   (a) Write code for two functions:

   - The first function takes as input parameters w, b of a linear classifier as well as a data point x, and returns the label for that point: $\text{sign}(w \cdot x + b)$. The label is either $+1$ or -1.
   - The second function takes as input an array of data points and an array of labels (where each label is $+1$ or -1), and runs the Perceptron algorithm to learn a linear classifier w, b. The algorithm should begin by randomly permuting the data points.

   In your writeup, give the code for these two functions.

   (b) Load in the Iris data set. You can do this by simply invoking:

   ```
   from sklearn import datasets
   iris = datasets.load_iris()
   x = iris.data
   y = iris.target
   ```

   The data has four features and three labels. Restrict it to features 1 and 3 (the second and fourth columns, sepal width and petal width) and to labels 0,1. Recode label 0 as -1, since this is what the Perceptron algorithm is expecting.

   (c) Now run the Perceptron algorithm on the data. In your writeup, show a plot with the data points (where the two labels have different colors) and the resulting decision boundary.

   (d) Now modify your code from part (a) to count the *number of updates* made by the Perceptron algorithm while it is learning. Run the algorithm 20 times and keep track of the number of updates needed each time. In your writeup, include a histogram of these values.

2. *Support vector machine.* As you did with the Perceptron, use the Iris data set, but this time use features 0 and 2, and labels 1,2.

   (a) Is this data linearly separable?

   (b) Use `sklearn.svm.SVC` to fit a support vector machine classifier to the data. You will need to invoke the option `kernel='linear'`. Try at least 10 different values of the slack parameter C. In your writeup, include a table that shows these values of C and for each of them gives the training error and the number of support vectors.

   (c) Which value of C do you think is best? For this value, include a plot of the data points and the linear decision boundary.

3. *Multiclass Perceptron.* Implement the multiclass Perceptron algorithm from class.

(a) Load in the data set `data0.txt`. This has 2-d data in four classes (coded as 0,1,2,3). Each row consists of three numbers: the two coordinates of the data points and the label.

(b) Run the multiclass Perceptron algorithm to learn a classifier. Create a plot that shows all the data points (with different colors and shapes for different labels) as well as the decision regions.

4. *Multiclass SVM.* In this problem, we'll use support vector machines to classify the MNIST data set of handwritten digits.

(a) Load in the MNIST data: a training set of 60,000 points and a separate test set of 10,000 points.

(b) Learn a linear SVM classifier using `sklearn.svm.LinearSVC`. You will need to see `loss='hinge'`. How can you choose a suitable value of C? Explain your methodology.

(c) Report the final test error. Is this data linearly separable?

2.2 Mini-project: Calibrated probability estimation

Overview

This project is about getting *calibrated probability estimates* in classification problems. Say we have a binary classification task with two labels, -1 and $+1$. On any test point x, we would like not just a prediction but also a *confidence score*. One way to formalize this is to say that we want an estimate of $\Pr(y = 1|x)$.

Logistic regression tries to do this, but the probability values it outputs cannot be interpreted too literally. Here is the property we want, that we will refer to as **calibration**:

- For any $0 < p < 1$, look at all points x on which the classifier gives a value of $\Pr(y = 1|x)$ close to p, say within the range $[p - \delta, p + \delta]$.
- Then, roughly p fraction of these points should have label $y = 1$.

How can such probability estimates be obtained? The first thing to note is that many of the classifiers we have studied (and will study) can provide a real-valued *score* $z(x)$ in addition to a predicted label, for any data point x. For instance:

- For any linear classifier with boundary $w \cdot x + b = 0$, such as a support vector machine, the score on x can be taken to be $z(x) = w \cdot x + b$.
- For a random forest, the score on x is the fraction of trees that vote for the majority label.
- For a generative model, the score on x is the conditional probability, under the model, of the predicted label given x.

Let's focus on the first example, $z(x) = w \cdot x + b$, for concreteness. Intuitively, the higher this score, the more likely it is that the label is $y = 1$. Can we convert it into a calibrated probability value? Is there a *monotonic transformation* from these scores $z(x)$ to probability estimates $\Pr(y = 1|x)$?

There are two monotonic transformations that are commonly used for this purpose: **Platt scaling** and **isotonic regression**. We will try out both of them in this assignment.

The overall idea is to divide the available labeled data into three groups:

1. A *training set*, for learning the classifier (in our case, an SVM).
2. A *calibration set*, for learning the monotonic transformation from scores to probabilities.
3. A *test set*, for evaluating the resulting probability estimates.

Step 1: Learning a classifier

We will use the `sentiment` data set described in class. Download `full_set.txt` from the course website. This contains 3000 lines, each with one sentence followed by a label (0 or 1). Start by converting this data into 3000 vectors with separate labels. Several steps are involved:

- Remove the labels from the sentences and store them separately.
- Remove digits and punctuation, and make everything lowercase.
- Remove "stop words", common words that are useless for classification. I suggest removing at least the following: 'the', 'a', 'an', 'i', 'he', 'she', 'they', 'to', 'of', 'it', 'from'.
- Choose a vocabulary (of no more than 5000 words) and convert each sentence into a bag-of-words vector of the corresponding dimension. This can be done in 1-2 lines using

  `sklearn.feature_extraction.text.CountVectorizer`

- Divide the 3000 data points into a training set of size 2200, a calibration set of size 400, and a test set of size 400. Do this randomly, but keep the classes balanced (each set should be evenly split between positives and negatives).

Now use `sklearn.svm.LinearSVC` to learn a (soft-margin) SVM, using the training set alone. You will need to set the C parameter, for which you should use cross-validation (using only the training set, of course).

Step 2: Learning a monotonic transformation

We now have a *fixed* classifier: the label for x is $\text{sign}(w \cdot x + b)$. We will think of the *score* on point x as $z(x) = w \cdot x + b$. We want to convert these real-valued scores into probabilities, using a monotonically increasing function f: we will predict $\Pr(y = 1|x) = f(z(x))$. We will investigate three choices for f.

- Option 1: *The squashing function.* This involves no learning: map score z to probability $1/(1 + e^{-z})$.
- Option 2: *Platt scaling.* We will map score z to probability $1/(1 + e^{-(az+b)})$. Here $a, b \in \mathbb{R}$ are parameters to be learned using the calibration set, using maximum-likelihood. That is, we want to find a, b to minimize

  $$\sum_{(x,y)\in\mathcal{C}} \ln(1 + \exp(-y(az(x) + b))),$$

  where $\mathcal{C}$ is the calibration set, assuming labels are ± 1. Since $z(x) \in \mathbb{R}$, this is essentially a one-dimensional logistic regression problem, with data $\{(z(x), y) : (x, y) \in \mathcal{C}\}$, and can be solved easily, in 1-2 lines, using `sklearn.linear_model.LogisticRegression`.
- Option 3: *Isotonic regression.* The idea is to learn a general monotonic function $f : \mathbb{R} \to [0, 1]$ from scores to probabilities. The data will be $\{(z(x), y) : (x, y) \in \mathcal{C}\}$, assuming the labels are $y \in \{0, 1\}$ (not ± 1). Details of isotonic regression appear below. You can learn the function using `sklearn.isotonic.IsotonicRegression`. Unfortunately, the documentation for this function is very poor, so we provide a basic template here that you can use. Suppose Z is a vector of z-values of the calibration set, and Y is a vector of corresponding $0-1$ labels. (Both are 400-dimensional vectors.) To learn the monotonic transform:

  ```
  from sklearn.isotonic import IsotonicRegression
  isotonic_clf = IsotonicRegression(out_of_bounds='clip')
  isotonic_clf.fit(Z,Y)
  ```

To get probabilities for points in the test set, let Z' be the vector of test set scores. Use:

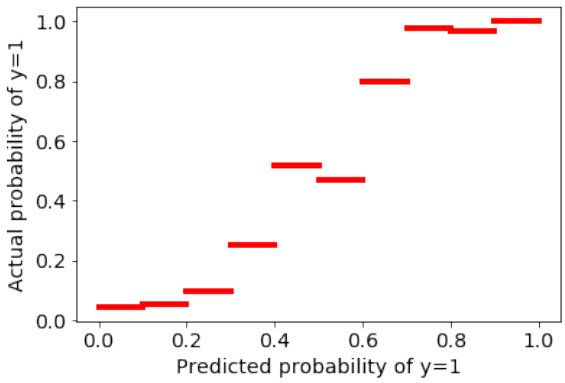
```
test_probs = isotonic_clf.transform(Z')
```

Step 3: Evaluating the predicted probabilities

We will use a qualitative evaluation strategy, constructing a *reliability diagram* using the *test set*.

- Divide $[0, 1]$ into intervals of a fixed size. For instance, the intervals (0 to 0.1), and (0.1 to 0.2), all the way to (0.9 to 1.0).
- For each interval $[a, b]$:
  - Look at all data points in the *test set* for which predicted probabilities were in $[a, b]$.
  - Look at the actual probability of $y = 1$ for these points (that is, the fraction of the true labels that were 1). Call this $p(a, b)$.
  - Draw a horizontal line from $(a, p(a, b))$ to $(b, p(a, b))$.

Here's an example:



For instance, in this case, the set of test points x for which the predicted probability $\Pr(y = 1|x)$ was in $[0.7, 0.8]$ actually almost all had label $y = 1$. Thus the predicted probability for these points was an underestimate.

What to turn in

(a) *Some details of the SVM training procedure (step 1).*

In particular, plot the different values of C that you tried, and the cross-validated error estimate you got for each. What value of C did you end up using? What is the resulting error rate of the SVM classifier on the test set?

(b) *Squashing function (option 1).* Show the reliability diagram (on the test set) for this. Is this calibrated? Why or why not?

(c) *Platt scaling (option 2).* Try doing this two ways: forcing $b = 0$ (this is the "intercept" term) or learning b. Show the reliability diagram in each case. Which is better? Why do you think this is? Is this better calibrated than option 1?

(d) *Isotonic regression (option 3).* Show the resulting reliability diagram. How does it compare to Platt scaling?

(e) What do you consider the relative merits of Platt scaling versus isotonic regression? Would you always choose one of these over the other, or does it depend on the situation? What aspects of the learning scenario (e.g. size of calibration set) would influence your choice?

(f) *Multiclass setting.* So far we have only talked about binary classification. What would calibration mean in the multiclass setting? Can you think of a way to obtain calibrated probabilities in that case?

Some information about isotonic regression

In a *line fitting* problem, we have a data set consisting of pairs $(x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^2$ and we want to draw a line through them. More precisely, we want to find parameters $a, b \in \mathbb{R}$ such that $f(x) = ax + b$ passes as close as possible to the points. We have already seen a least-squares formulation of this.

In *isotonic regression*, we allow a more general function f. It doesn't have to be a line: it just needs to be monotonically increasing, that is, $f(x) \geq f(x')$ whenever $x \geq x'$. Let's sort the data points so that $x_1 \leq x_2 \leq \cdots \leq x_n$. Monotonicity then means

$$f(x_1) \leq f(x_2) \leq \cdots \leq f(x_n).$$

In fact, we can choose any $f(x_i)$ values that meet this requirement; and we can fill in the rest of the f-curve by, say, linearly interpolating between these points.

Here is a least-squares formulation of our problem: given $x_1 \leq x_2 \leq \cdots \leq x_n$ and corresponding values $y_1, \ldots, y_n$, find $f_1, f_2, \ldots, f_n \in \mathbb{R}$ such that $f_1 \leq f_2 \leq \cdots \leq f_n$ and such that the squared loss

$$L(f) = \sum_{i=1}^{n} (y_i - f_i)^2$$

is minimized. (Here f_i corresponds to $f(x_i)$.)

1. This is a convex optimization problem (check!).

2. An elegant approach to solving this problem is the *pool adjacent violators* algorithm. It starts by simply setting $f_i = y_i$ for all i, and then repeatedly fixes any monotonicity violations: any time it finds $f_i > f_{i+1}$, it resets both of them to the average of f_i, f_{i+1} and merges points x_i and x_{i+1}.

   Here is the algorithm, given a set of x values and their corresponding $y(x)$.

   - Let S be the sorted list of x-values
   - For all x in S:
     - Set $f(x) = y(x)$
     - Assign weight $w(x) = 1$
   - While there adjacent values $x < x'$ in S with $f(x) > f(x')$:
     - Remove x' from S and set a pointer from it to x
     - Let $f(x) = \frac{w(x)f(x) + w(x')f(x')}{w(x) + w(x')}$
     - Let $w(x) = w(x) + w(x')$

   At the end, each of the original x-points either lies in the list S, in which case it receives value $f(x)$, or leads to some $\tilde{x}$ in list S by following pointers, in which case it receives value $f(\tilde{x})$.