DSE 220: Machine learning

Lab 3 — Ensemble methods and informative projections

3.1 Programming problems

1. *A toy 2-d data set for decision trees and boosting.* Obtain the data set `data1.txt` from the course webpage. Each line has a two-d data point followed by a label (0 or 1).

   (a) Plot this data to see what it looks like.

   (b) Now use `sklearn.tree.DecisionTreeClassifier` to fit a decision tree to the data. What stopping criterion did you use? Display the tree using `graphviz`.

   (c) Finally, fit boosted decision stumps to this data using `sklearn.ensemble.AdaBoostClassifier`. Use a relatively small number of stumps, and display each of them. Give a table showing how accuracy on the training data improves as each successive stump is added.

2. *Credit card fraud data.* Download the data set at

   `https://www.kaggle.com/mlg-ulb/creditcardfraud`.

   This data set has details of 284,807 credit card transactions, some of which are fraudulent. Each transaction is represented by 28 features (scrambled using PCA as a primitive kind of anonymization), and has a corresponding label (1 is fraudulent and 0 is legitimate).

   (a) How many of the transactions are fraudulent? Why might this be problematic when learning a classifier?

   (b) Downsample the legitimate transactions to make the data set more balanced.

   (c) Fit three kinds of classifier to the data:

   - decision tree
   - boosted decision stumps
   - random forest

   In each case, use cross-validation to estimate the confusion matrix.

3. *An experiment with PCA.* For this problem, we'll be using the *animals with attributes* data set. Go to

   `http://attributes.kyb.tuebingen.mpg.de`

   and, under "Downloads", choose the "base package" (the very first file in the list). Unzip it and look over the various text files.

   (a) This is a small data set that has information about 50 animals. The animals are listed in `classes.txt`. For each animal, the information consists of values for 85 features: does the animal have a tail, is it slow, does it have tusks, etc. The details of the features are in `predicates.txt`. The full data consists of a 50×85 matrix of real values, in `predicate-matrix-continuous.txt`. Load this real-valued array.

(b) We would like to visualize these animals in 2-d. Show how to do this with a PCA projection from $\mathbb{R}^{85}$ to $\mathbb{R}^2$. Show the position of each animal, and label them with their names.

Python notes: You will need to make the plot larger by prefacing your code with

```
from pylab import rcParams
rcParams['figure.figsize'] = 10, 10
```

(or try a different size if this doesn't seem right).

4. In lecture, we looked at the effect of projecting the MNIST data set of handwritten digits to lower-dimension: from 784 to 200, 150, 100, 50 dimensions. We found that the reconstruction error was fairly low for 150-dimensional projections, but noticeable for 50-dimensional projections.

   We now investigate these issues further.

   (a) Let $X \in \mathbb{R}^d$ have covariance matrix Σ. Suppose the eigenvalues of Σ are $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$, and suppose the corresponding eigenvectors are $u_1, u_2, \ldots, u_d$. Then it can be shown that X has an overall variance of $\lambda_1 + \cdots + \lambda_d$, and that when X is projected onto the top k eigenvectors, the residual variance (the information that gets lost) is $\lambda_{k+1} + \cdots + \lambda_d$. Therefore, for this projection, the fraction of lost information, intuitively speaking, is

   $$F(k) = \frac{\lambda_{k+1} + \cdots + \lambda_d}{\lambda_1 + \cdots + \lambda_d}$$

   Compute these fractions for the MNIST data set, for $k = 200, 150, 100, 50, 25$.

   (b) Suppose we are allowed a different projection for each digit. We would then expect that we can project to an even lower dimension while maintaining roughly the same amount of information. Test whether this is true as follows: for each digit $j = 0, 1, 2, \ldots, 9$,

   - Obtain the PCA projection to k dimensions, for $k = 200, 150, 100, 50, 25$.
   - Compute the fraction $F_j(k)$, for each such value of k.
   - Pick a random instance of the digit. Show the original digit as well as its reconstruction at each of these five values of k. (Note: the original images have pixel values in the range 0-255, but this might not be true of the reconstructions; therefore, you may need to clip the reconstructed pixels to lie within this range before displaying the image.)

   Show all the fractions $F_j(k)$ in a table. Which digit seems to be the most amenable to low-dimensional projection?

3.2 Mini-project: Word embeddings

The large number of English words can make language-based applications daunting. To cope with this, it is helpful to have a *clustering* or *embedding* of these words, so that words with similar meanings are clustered together, or have embeddings that are close to one another.

But how can we get at the meanings of words? John Firth (1957) put it thus:

You shall know a word by the company it keeps.

That is, words that tend to appear in similar contexts are likely to be related. In this assignment, you will investigate this idea by coming up with an embedding of words that is based on co-occurrence statistics.

The description here assumes you are using Python with NLTK.

- First, download the Brown corpus (using `nltk.corpus`). This is a collection of text samples from a wide range of sources, with a total of over a million words. Calling `brown.words()` returns this text in one long list, which is useful.
- Remove stopwords and punctuation, make everything lowercase, and count how often each word occurs. Use this to come up with two lists:
  - A *vocabulary* V, consisting of a few thousand (e.g., 5000) of the most commonly-occurring words.
  - A shorter list C of at most 1000 of the most commonly-occurring words, which we shall call *context words*.
- For each word $w \in V$, and each occurrence of it in the text stream, look at the surrounding window of four words (two before, two after):
  $$w_1 \;\; w_2 \;\; w \;\; w_3 \;\; w_4.$$
  Keep count of how often context words from C appear in these positions around word w. That is, for $w \in V, c \in C$, define
  $$n(w,c) = \#\text{ of times } c \text{ occurs in a window around } w.$$
  Using these counts, construct the probability distribution $\Pr(c|w)$ of context words around w (for each $w \in V$), as well as the overall distribution $\Pr(c)$ of context words. These are distributions over C.
- Represent each vocabulary item w by a $|C|$-dimensional vector $\Phi(w)$, whose c'th coordinate is:
  $$\Phi_c(w) = \max\left(0,\ \log \frac{\Pr(c|w)}{\Pr(c)}\right).$$
  This is known as the (positive) *pointwise mutual information*, and has been quite successful in work on word embeddings.
- Suppose we want a 100-dimensional representation. How would you achieve this?
- Investigate the resulting 100-dimensional embedding in two ways:
  - Cluster the vocabulary into 100 clusters. Look them over; do they seem completely random, or is there some sense to them?
  - Try finding the nearest neighbor of selected words. Do the answers make sense?

The Brown corpus is very small. Current work on word embeddings uses data sets that are several orders of magnitude larger, but the methodology is along the same lines.

What to turn in

(a) *A description of your 100-dimensional embedding.*

The description should be concise and clear, and should make it obvious exactly what steps you took to obtain your word embeddings. Below, we will denote these as $\Psi(w) \in \mathbb{R}^{100}$, for $w \in V$. Also clarify exactly how you selected the vocabulary V and the context words C.

(b) *Nearest neighbor results.*

Pick a collection of 25 words $w \in V$. For each w, return its nearest neighbor $w' \neq w$ in V. A popular distance measure to use for this is *cosine distance*:

$$1 - \frac{\Psi(w) \cdot \Psi(w')}{\|\Psi(w)\|\|\Psi(w')\|}.$$

Here are some suggestions for words you might choose:

communism, autumn, cigarette, pulmonary, mankind, africa, chicago, revolution, september, chemical, detergent, dictionary, storm, worship

Do the results make any sense?

(c) *Clustering.*

Using the vectorial representation $\Psi(\cdot)$, cluster the words in V into 100 groups. Clearly specify what algorithm and distance function you using for this, and the reasons for your choices.

Look over the resulting 100 clusters. Do any of them seem even moderately coherent? Pick out a few of the best clusters and list the words in them.