University of Colorado at Boulder ECEN 4532

Instructor: Mike Perkins

Machine Learning

Classification and Linear Regression

Lab 6
Lab report due on May 1, 2019

This is a Python only lab. Each student must turn in their own lab report and programs.

1 Introduction

In many real-world situations large amounts of data are collected with the hope it can be analyzed to develop useful predictions. Examples of situations you might confront as an engineer working in the field of data analytics include:

- You work for a major web retailer. Users' searches on your website are monitored, as well as any other data you can acquire about them. For example, you know what they actually purchase at your store. Can you predict what they are most likely to buy in the future? If so, you may decide to make suggestions to the user on other possible items of interest. Can you determine how well your suggestions translate into actual purchases?
- You work for a company that maintains a large distributed network. It could be a data network, a natural gas pipeline system, an electric power distribution system, etc. The network is instrumented so you have data on how it is performing at each instant. Occasionally, the network fails. Can you analyze the available sensor data to develop algorithms that predict when the network is likely to fail? Note that you may not even know why a particular set of sensor readings predicts failure...but if it does, you can use that information anyway!
- You work for a medical provider who operates a group of hospitals. You have information about who is admitted to the hospital, how long they stay, what they are admitted for, how effective their treatment was, etc. Can you find factors that predict good treatment outcomes? Can you use this knowledge to lower costs?

Problems of this sort often involve classification. As an example of binary classification, for the first case above, you may be interested in the question "is the shopper male or female" or "is the shopper younger or older than 18". Multi-class classification problems are also common, for example, you might want to predict a shopper's income quintile based on his search behavior and past purchasing record. In this case you need to assign the user to one of 5 classes.

In other situations you need to predict a continuous value based on the available data. For example, in the third case above, you might want to predict the likely cost of treatment for a patient based on the available data.

In this lab you will investigate binary classification using Support Vector Machines, linear regression for predicting a continuous quantity, and multi-class classification using neural networks. You should approach this lab as a problem of data analysis.

The answer to how to generate a prediction—or assign an instance to a class—isn't clear, and you must do research to find an appropriate method. You will need to experiment with various approaches to see what works well.

2 Background

For this lab you will use three different data sets. Specifically, you will:

- Use a Support Vector Machine (SVM) to predict who survived the sinking of the titanic. This is a binary classification problem: passengers either "survived" or "perished".
- Use linear regression to predict the water resistance encountered by a sailing yacht. Here the value being predicted is a real number.
- Use a neural network to recognize hand-written digits. This is a multi-class classification problem: any of 10 digits are possible, $\{0, 1, \ldots, 9\}$.

Note that in all of these cases we have a set of labeled outcomes. For example, in the case of the hand-written digits, we have 70,000 images of the numerals 0 through 9 written by different people and labeled with the correct answer. All three of these problems are examples of supervised machine learning—inferring an appropriate function from a large set of input/output pairs.

In order to assess the performance of a Machine Learning (ML) algorithm, it is important to divide the labeled set into two groups, a *training set* and a *test set*. The algorithm should be trained using the training set, and its performance should be tested on the test set. In practical applications, it is also important to avoid *over-training*. This occurs when the model adapts too strongly to the individual characteristics of the training set, thereby impacting its ability to generalize to instances outside of it. Various *regularization* techniques are available to address this issue, but we will not go into that in this lab.

You will need to install the Python csv module if you have not done so already (see `https://docs.python.org/3/library/csv.html` for additional information). You will also need to install the module `sklearn`.

3 Binary Classification with SVMs

The file `titanic_tsmod.csv` in Canvas contains data about the Titanic's passengers. The relevant data fields for classification are:

- Passenger class: 1, 2, or 3
- Passenger's sex: male or female
- Passenger's age: an integer between 1 and 80
- The number of siblings a passenger had on the ship, plus 1 if a spouse was on board
- The number of parents or children a passenger had on the ship
- The fare paid by the passenger
- Where the passenger embarked: one of three locations

Interestingly, John Jacob Astor IV, one of the richest people in the world at the time, died in the accident.

You can input the relevant columns of the data set using the function `get_titanic_all` in the file `input_titanic_data.py` in Canvas. This file returns a 2D table with the parameter values for each instance, as well as a vector which indicates "survived" or "perished" for each instance. Note that the data has been reformatted to make it better for an SVM. For example, instead of having a single column where class can assume 3 values (1, 2, or 3) 3 columns are used with a flag to indicate the passenger's class. You will need to look at the function provided to understand the mapping.

When doing the following problems, you will need to extract just the columns you care about into a new matrix to use for training. Note that the "fare" data may prove difficult for the SVM and you probably want to avoid using it.

Your program flow with respect to the ML aspects will look something like this:

```
.
from sklearn import svm
.
X_all, y = get_titanic_all('titanic_tsmod.csv')
# Extract columns of interest into a table X
.
```

```
clf = svm.SVC(gamma='scale', kernel='linear')
# X is the 2D table of instances, y is the label vector specifying
# the class of each instance
clf.fit(X, y)
.
# This returns a prediction for each row of X using the
# the model that was trained (fitted) above
pred_val = clf.predict( X )
```

Note that for this portion of the lab, we are NOT dividing the data set into two pieces, a training set and a test set. We will do that in a later problem. Here we are just focusing on finding the best predictor of survival we can based on the available data.

Assignment

1. Write a program to input the Titanic data set. Using your program, evaluate how well the passengers' fare class predicts survival. What percentage of the time does the SVM correctly predict the passengers' fate using just this one piece of data?
2. Which single parameter best predicts the passengers' fate? What percent of the time does the prediction succeed for each parameter? (Note that when using only a single column from `X_all`, you will need to reshape it into a 2D array like this: `X = np.reshape( X_all[:,3], (-1,1) )` in order to use it with the `sklearn` functions)
3. What seems to be a good combination of parameters to use if you are restricted to picking only 3 parameters? How well does your resulting predictor do?

4 Linear Regression

The file `yacht_data.csv` in Canvas contains data about the resistance yachts experience when they are built with different design parameters. The design parameters for predicting the resistance are:

- The longitudinal center of buoyancy: its distance forward of the stern

- The prismatic coefficient: describes how the yacht's displacement is distributed along the hull from bow to stern
- The length-to-displacement ratio
- The beam-to-draught ratio
- The length-to-beam ratio
- The Froude number: a parameter that depends on the speed/length ratio of the yacht at design speed. It captures resistance effects related to the bow wave.

For this data set, you will need to write your own input routine based on the csv module. Note that the value being predicted is in the last column of the csv file. The basic program flow is the same. After importing the correct module, you will call a "fit" function and a "predict" function.

```
.
from sklearn.linear_model import LinearRegression
.
# Import the data into a 2D table S_all and a value vector, y
.
# Compute the regression model
reg = LinearRegression().fit(S_all,y)
.
yhat = reg.predict(S_all)
```

You will characterize the performance of your predictor using the R^2 value as discussed in lecture. This will require you to compute the Explained Sum of Squares (ESS), the Total Sum of Squares (TSS), and the Residual Sum of Square (RSS).

Include these values in your report.

Assignment

1. Write a program to input the yacht data. Using all 6 columns to predict column 7, what values do you obtain for the prediction coefficients? Graph the raw data and your predictions on the same graph and put them in your report. What do you observe?
2. Write a function to compute the ESS, TSS, RSS, and the R^2 value. What is the R^2 value for the linear predictor above? What are the ESS, TSS, and RSS values?
3. Can you obtain a better prediction by using a power of each column in your predictor instead of the raw data in the column? What is the best R^2 value you can obtain in this manner, and for what power? What are the resulting prediction coefficients?
4. Now, restrict yourself to using only 3 columns. Which columns seem to be a good choice? Generate a graph showing the performance of your final predictor relative to the raw values. What is the corresponding R^2 value? Do you see any obvious (perhaps non-linear) change you can make to your prediction function to improve its performance?

5 Multi-class Classification with Neural Networks

The file `mnist_784.arff` in Canvas contains 70,000 images of hand-written digits. The images are 28x28 pixels in size.

5.1 Confusion matrix

The accuracy of the classification will be quantified using confusion matrices. In our case, the confusion matrix will measure the correct classification rate of each digit. For a given classification experiment, you will construct a 10×10 matrix whose rows represent the true digit, and whose columns are the digits as classified by your algorithm. The entry $R(i, j)$ of the matrix is the number of digits of type i classified as type j.

In the hypothetical classification example in Figure 1, where instances are being classified into one of six classes, the classification performance for the first class is:

$$\begin{bmatrix} 21 & 3 & 1 & 0 & 0 & 0 \end{bmatrix},$$

so there were 21 correct classifications (these occur along the diagonal of the matrix) out of 25 classifications total. This corresponds to a correct classification rate of 84%. The correct classification rate for each class according to this matrix is:

$$\begin{bmatrix} 0.84 & 0.88 & 0.52 & 0.92 & 0.36 & 0.40 \end{bmatrix}, \tag{1}$$

In total, out of 150 classifications, 97 were correct (the sum of the diagonal entries) for an overall success rate of 64.67%.

	class 1	class 2	class3	class 4	class 5	class 6
class 1	21	3	1	0	0	0
class 2	3	22	0	0	0	0
class 3	0	1	13	2	8	1
class 4	0	0	0	23	1	1
class 5	0	2	3	6	9	5
class 6	3	7	1	0	4	10

Figure 1: Example of a confusion matrix. The rows are the true classes, and the columns are the classes as classified by the algorithm.

5.2 Cross-validation Experiment

In this section we adopt the following terminology:

- let $\mathcal{S}_a$ be the set of all 70,000 images.
- let $\mathcal{S}$ be the set of 63,100 images obtained by using the first 6,310 images of each digit that occur in the data set.
- let $\mathcal{G}_i, i = 1, \cdots, 10$ be the ten sets of images grouped by type. Therefore:

$$\mathcal{S} = \bigcup_{i=1}^{10} \mathcal{G}_i.$$

In order to evaluate the performance of your neural network, you will perform the following five-fold cross-validation experiment 10 times (i.e. for $n = 1, 2, ..., 10$):

- Step 1

  1. randomly divide each digit set, $\mathcal{G}_i$, into 5 equally sized subsets of size 5. These sets will satisfy
  $$\mathcal{G}_i = \bigcup_{k=1}^{5} \mathcal{G}_i^k$$
  2. **for** $k = 1$ to 5
     - form the set of test digits (i.e. the *test set*)
     $$\mathcal{U}_k = \bigcup_{i=1}^{10} \{\mathcal{G}_i^k\}$$
     and the set of training digits (i.e. the *training set*)
     $$\mathcal{L}_k = \mathcal{S} \cap \mathcal{U}_k^C = \bigcup_{i=1}^{10} \bigcup_{\substack{l=1 \\ l \neq k}}^{5} \mathcal{G}_i^l$$
     - test the performance of your algorithm on the digits in $\mathcal{U}_k$ using the training digits in $\mathcal{L}_k$
     - store the confusion matrix $R^n(k)$ (here n indexes the experiment number, $n = 1, 2, \ldots, 10$)

     end for

Once you have performed the above experiment 10 times, you will have a total of 50 confusion matrices: 5 matrices for each of the 10 times you ran the experiment.

Next, do the following:

- Average the 50 confusion matrices together to obtain $\overline{R}(i, j)$
- Compute the standard deviation matrix $\sigma_R(i, j)$

The final average confusion matrix, and the associated standard deviation matrix, should be included in your report.

5.3 Writing the code

The basic coding pattern is the same as before. We input the data, train a model, and then make predictions. In this case, however, the data set is stored in a format which we must first convert to our usual numpy format. The code to do this is given below:

```
.
from sklearn.neural_network import MLPClassifier
from scipy.io import arff
.
# Load the data
print("loading the arff data...")
data = arff.loadarff('mnist_784.arff')
print("done")
.
# Convert the data into a useful numpy format
# Note that picture j is stored at location data[0][j]
# The last element of this array is the image's digit
# The images are of size 28x28
print("converting arff data to numpy format...")
rows = len(data[0])
print(f"Number of images in data set is {rows}")
e = np.dtype( data[0][0] )
cols = len( e.names ) - 1
print(f"Number of pixels in each image is {cols}")
X = np.zeros( (rows, cols), np.float )
y = np.zeros( rows, np.float )
for j in range(rows):
  tmplist = data[0][j].tolist()
  X[j] = np.asarray( tmplist[:-1], np.float)
  y[j] = np.float( tmplist[-1] )
print("done")
.
# rescale the data to the range 0 to 1
X = X / 255
# Extract the training set to get X_train and y_train
# Extract the test set to get X_test and y_test
.
```

```
# Define and train the model, and make predictions
mlp = MLPClassifier(hidden_layer_sizes=(100, 100), max_iter=400, alpha=1e-4,
                    solver='sgd', verbose=10, tol=1e-4, random_state=1)
mlp.fit(X_train, y_train)
yhat = mlp.predict(X_test)
```

In this example the neural network has two hidden layers, each of which have 100 perceptrons. There is also an input layer with 728 perceptrons and an output layer with 10 perceptrons, however the specification of these two layers is implicit; only the number and size of the hidden layers must be specified when creating the classifier. The maximum number of iterations to perform while training is set to 400, and this can be reduced to speed up the process. There should be no need to play with the other parameters, but you might find changing them interesting if you have extra time to experiment. See the documentation for details on what they do.

Assignment

1. Use the first 60,000 images as the training set and the last 10,000 for testing. Use a single hidden layer of size 10 with a maximum of 25 iterations and train a model. What percentage of the time is the output digit in the test set correctly predicted by your model? (you can use the function `mlp.score(X_test, y_test)` to compute this)
2. Compute the confusion matrix for your model and include it in your report. For each digit, which other digit is it most likely to be confused as? What percentage of the time does this particular error happen for each digit?
3. Try using hidden layer sizes of 20, 30, and 50 for the single hidden layer, with a maximum iteration count of 50. How does this affect overall performance on the test set? (no need to compute a confusion matrix)
4. Use two hidden layers of size 50 and a maximum iteration count of 50. What is the overall performance on the test set?
5. Using a single hidden layer of size 30 and a maximum iteration count of 25, write the code to perform the cross-validation experiment described above.