CMSC 25025 / STAT 37601

Machine Learning and Large Scale Data Analysis

Assignment 6

Out: Thursday, May 18, 2017
Due: Tuesday, May 30, 2015

In this final assignment, we use large scale data analysis to travel across time and space, from the poetic (exoplanets) to the prosaic (real estate). The assignment is intended to be mindful of your busy schedules at the end of the quarter!

1. *Leave-one-out cross-validation* (20 points)

   Let $\widehat{m}_n$ be a kernel regression estimator, using either a smoothing kernel or a Mercer kernel, for data $(x_1, y_1), \ldots, (x_n, y_n)$.

   (a) Show that the estimator can be written as

   $$\widehat{m}_n(X) = LY$$

   where $\widehat{m}_n(X) = (\widehat{m}_n(x_1), \ldots, \widehat{m}_n(x_n))^\top$ and $Y = (y_1, \ldots, y_n)^\top$, for some $n \times n$ matrix L.

   (b) Show that the leave-one-out cross-validation score $\widehat{R}(h)$ can be written as

   $$\widehat{R}(h) = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \widehat{m}_n(x_i)}{1 - L_{ii}}\right)^2$$

   where $L_{ii} = \ell_i(x_i)$ is the ith diagonal element of the smoothing matrix L.

LSDA

2. *Planet hunting* (40 points)

   In this problem you will work with the Kepler light curve dataset. Although the data contains light curves of 166,904 stars, we will only analyze a few of them. The stars are categorized under Kepler as:

   - `conf`: the star is confirmed to have planets
   - `fp`: confirmed to have no planet
   - `eb`: confirmed to be an *eclipsing binary star system*[1] which has no planet

[1] `http://en.wikipedia.org/wiki/Binary_star`

Note that some light curves such as Kepler 16 may have exoplanet transits, binary star eclipses and occultations[2]. Such a star would be categorized as `conf`. For each type, we have provided light curves of some example stars; see the plots below.

For `conf` stars, transits coincide with periodic drops in the light flux. Such dips are not observed in light curves of `fp` stars. The `eb` stars are more interesting. Their light curves will have two periodic flux drops, corresponding to the eclipses and occultations. Only about 0.05% of the light curves in the Kepler data are `conf`, and about 1% are `eb`. In this problem, you will use kernel regression to fit the light curves and use a thresholding technique to detect planets and eclipsing binary stars, inspecting the results visually.

An overview of the Kepler data processing pipeline is given in this article: `http://iopscience.iop.org/2041-8205/713/2/L87/pdf/2041-8205_713_2_L87.pdf`

To start, get the light curves:

```
lc = spark.read.json('/project/cmsc25025/light_curve/lightcurves-sample.json')
```

The light curve data contains four fields: name, time, flux and label. If you inspect the data, you can see that all the light curves have 4757 flux values that are recorded at roughly the same time points. The time gap between any two successive records is approximately 0.0204. All the light curves have 325 missing values (`NaN`) at the same places.

Process each of the light curves as follows.

(a) Fit the light curve using *Nadaraya-Watson kernel estimator*.

Several types of kernel functions that are commonly used, including:

- *Boxcar* $K(x) = \frac{1}{2}I(x)$
- *Gaussian* $K(x) = \frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}x^2}$
- *Epanechnikov* $K(x) = \frac{3}{4}(1-x^2)I(x)$
- *Tricube* $K(x) = \frac{70}{81}(1-|x|^3)I(x)$

where

$$I(x) = \begin{cases} 1 & \text{if } |x| \le 1 \text{ ,} \\ 0 & \text{otherwise.} \end{cases}$$

The kernel regression could be considered as a convolution between a weight vector and the response. It can computed efficiently using `numpy.convolve`. The following piece of code is an example using the *Epanachnikov* kernel.

```
gap = 0.0204
N = 4757
sz = (N-1)/2
u = np.arange(-gap*sz, gap*(sz+1), gap, dtype=np.float64)

def kernel_epa(u):
```

[2]`http://en.wikipedia.org/wiki/Occultation`

```
        w = (abs(u) <= 1) * (1 - u*u)
        return w[w > 0]

    h = 1 #bandwidth
    w = kernel_epa(u/h)
    const = np.convolve(w, np.ones(len(flux), dtype=np.float64), 'same')
    yhat = np.convolve(w, np.nan_to_num(flux), 'same') / const
```

The function `np.nan_to_sum` will replace the `NaN` values of `flux` by 0. The array `w` is the weight vector. The constant $\frac{3}{4}$ is skipped since it will be cancelled out in the normalization. We set a unified gap value so that w can be *reused* for each light curve. Note that since the Epanachnikov kernel has a compact support, w may be a very *sparse* vector for some values of h. In such case, we only take the nonzero entries so that the cost of computing the convolution will be reduced.

For the given kernel type that you choose, select the bandwidth h via the leave-one-out cross validation. You can cross-validate over a range of bandwidths that are selected by visually selecting different fits.

(b) Compute the residual r as $r_i = y_i - \widehat{y}_i$, where $y = (y_1, \ldots, y_n)^\top$ is the response (light curve value) and $\widehat{y}$ is the fitted value.

(c) Standardize the residual so that it has zero mean and variance one:

$$r_i \longleftarrow \frac{r_i - \sum_i^n r_i/n}{\sigma},$$

where σ is estimated using the *median absolute deviation* (MAD):

$$\widehat{\sigma} = 1.4826\,\mathrm{MAD}(\mathbf{X}) = \mathrm{MAD}(\mathbf{X})/\Phi^{-1}(3/4)$$
$$\mathrm{MAD}(\mathbf{X}) = \mathrm{median}(|\mathbf{X} - \mathrm{median}(\mathbf{X})|).$$

(d) Compute the *universal threshold* $\beta = \sqrt{2\log n}$. Threshold the residual, where we set r_i to zero if it is greater than or equal to $-\beta$:

$$\widetilde{r}_i = \begin{cases} r_i & \text{if } r_i < -\beta \\ 0 & \text{otherwise.} \end{cases}$$

For each star, give plots of the fitted function (light curve), and the residuals with the threshold shown. Comment on the results, and whether or not the fits suggest a planet or binary star system. Describe your procedures and findings in the notebook, including various plots of your regression fits. Pass in your code as a notebook `assn6_planets.ipynb`.

LSDA

3. *House hunting* (40 points)

In this problem, you will train random forests to forecast the sale price of real estate listings. Random forests are nonparametric methods for classification and regression. As discussed in class, the method is based on the following thinking. A good predictor will have low bias and low variance. A deep decision tree has low bias, but high variance. To reduce the variance, multiple trees are fit and averaged together. By introducing randomness in the construction of the trees, the correlation between them is reduced, to facilitate the variance reduction.

The data are available here:

```
/project/cmsc25025/zillow/zillow-train.csv
/project/cmsc25025/zillow/zillow-test.csv
```

As you can see in the header of the file, there are 16 attributes:

```
ID, Lat, Long, ListPrice, SaleYear, Bathroom, Bedroom, BuiltYear,
BuildDecade, MajorRenov, FinishSqFt, LotSqFt, MSA, City, HighSchool,
SalePrice
```

You will build regression models to predict `SalePrice`.

(a) Explore the data to learn what the different predictor variables are, and what their distribution looks like. Make plots of these distributions. Some of the variables are categorical—how many values do they take? When you plot the variables, including the response `SalePrice`, does it appear that the data are "raw" or that they have been preprocessed in different ways? How?

(b) Build random forest models to predict `SalePrice` from the other covariates. You may use the Spark tools from
`https://spark.apache.org/docs/2.1.0/mllib-ensembles.html`
The main parameters to vary are `numTrees`, and `maxDepth`; these regulate the variance and bias, respectively. Another parameter is `featureSubsetStrategy`, which is the total number of variables allowed in splits; this is aimed at regulating the correlation between the trees, by introducing randomness.
The following sample code is reproduced from the documentation:

```
from pyspark.mllib.tree import RandomForest, RandomForestModel
from pyspark.mllib.util import MLUtils

# Load and parse the data file into an RDD of LabeledPoint.
data = MLUtils.loadLibSVMFile(sc, 'data/mllib/sample_libsvm_data.txt')
# Split the data into training and test sets (30% held out for testing)
(trainingData, testData) = data.randomSplit([0.7, 0.3])

# Train a RandomForest model.
#  Empty categoricalFeaturesInfo indicates all features are continuous.
#  Note: Use larger numTrees in practice.
#  Setting featureSubsetStrategy=''auto'' lets the algorithm choose.
model = RandomForest.trainRegressor(trainingData,
        categoricalFeaturesInfo={},
        numTrees=3, featureSubsetStrategy=''auto'',
        impurity='variance', maxDepth=4, maxBins=32)

# Evaluate model on test instances and compute test error
predictions = model.predict(testData.map(lambda x: x.features))
labelsAndPredictions = \
      testData.map(lambda lp: lp.label).zip(predictions)
testMSE = labelsAndPredictions.map(\
    lambda (v, p): (v - p) * (v - p)).sum() / float(testData.count())
print('Test Mean Squared Error = ' + str(testMSE))
print('Learned regression forest model:')
print(model.toDebugString())
```

Train several random forest models, using different configurations of parameters. Evaluate each on test data; in order to carry out model selection. Which setting of the parameters performs best? Comment on your findings and explain why they do or do not make sense to you.

(c) Finally, using your best regression model, predict the sale prices for the houses given in the test data we have prepared, `/project/cmsc25025/zillow/zillow-test.csv`. Note that the `SalePrice` response has been replace by `*`. Replace this character with your predicted value, keeping all of the other attributes in the csv. Submit this file as `assn6_prob3_predict.csv`. We will evaluate the predictions in terms of the absolute relative error

$$\frac{1}{n}\sum_{i=1}^{n}\frac{|Y_i - \widehat{Y}_i|}{Y_i}$$

and will give an award to the team with the smallest error rate.

Pass in your code and writeup as the notebook `assn6_houses.ipynb`.