AnHai's Group

Search this site

Home
People
Research
Publications

Current Projects
Agenda for DI Systems
Magellan
py_entitymatching
py_stringsimjoin
py_stringmatching
Data Science

Useful Stuff
Data Repository
How to Create Packages

Past Projects
Knowledge Bases/Graphs
Crowdsourcing
Schema/Ontology Matching
Silicon Valley Intermission

Courses
CS 638 DS
Fall 2016
CS 774
CS 564

Outreach & Funding
Walmart Labs
NSF IIS-Medium 2016

Miscellaneous
Wisconsin DB Group

Courses > CS 638 Fall 2016: Introduction to Data Science > Project >

CS 638 Project Stage 4

Due 11:59 pm Thur Dec 1

In this project stage you will create golden data and then use it to find the best matcher and measure the performance of the matcher.

Creating Golden Data

Recall that in Project Stage 3 you have performed blocking over two tables A and B, and obtain as a result a table C of tuple pairs. That blocking step aimed to remove as many obviously non-matching tuple pairs as possible. So hopefully table C should still contain all or most matching tuple pairs (in addition to many non-matching tuple pairs).

In this project stage you will build a matcher and then apply it to all tuple pairs in table C. To do that, first you need to create some golden data. Here are the steps to create golden data. Pls read and follow them very carefully.

- First, randomly sample 400 tuple pairs from C. Make sure you sample without replacement, otherwise you may have duplicate pairs in your sample.
- Next, label these tuple pairs to create golden data. If you see a pair for which you don't really know how to label (so the correct label is "I don't know"), you should remove that pair from the sample (to keep things simple for now).
- It is required that you will eventually have a labeled sample of at least 350 tuple pairs. We said above to sample 400 tuple pairs, because you may have to remove some of these pairs due to “I don’t know” label. When labeling, you have to label 1 for match and 0 for non-match.
- You can use any tool to label the data. Excel is commonly used in practice. But any editor that can show you the pairs will do.

Important: After labeling, check the ratio of positive vs negative examples in your sample. If this ratio is very skewed, e.g., having just 10 positive examples (that is, matching pairs) out of 350 examples, then you will have problems in later steps. In this case, come talk to me. If you have at least 50 positives and at least

50 negatives, you should be fine. I will discuss in the class how to avoid this skew problem.

Creating Features for Matching and Extracting Feature Vectors

Let the labeled sample be G. In the next step, you should create a set of features, then convert each tuple pair in G into a feature vector. You now should have a table H, where each tuple in H is a feature vector together with a golden label.

Note: for this step, using a package such as py_stringmatching may be very useful, as it helps you create various kind of string-related features.

Filling In (i.e., Imputing) Missing Values

If you have missing values in G, when you convert G into H you often will have missing values in H too. In this case the learning algorithms of scikit-learn won’t work (they won’t work on missing values).
So if you have missing values in H, at this point you should fill in the missing values.

Creating the Best Matcher

You are now ready to create the best matcher. Your goal is to create a matcher that achieves the highest possible F-1 score. (In your project report, you should be able to explain why you can’t reach higher F-1 score than what you report.) In what follows we will often use the term “accuracy” as a shorthand to refer to precision, recall, and F-1. This is an iterative process, and will take a while. In what follows we describe the high-level overview of this process:

- 1. Split the labeled set H into a development set I and an evaluation set J.

- 2. Suppose you have five learning-based classifiers that you can use to build learning-based matchers (e.g., Decision Tree, Random Forest, SVM, etc.). How do you know which one is the best? To answer this question, perform cross validation (CV) for all matchers on I and select the one judged to be the best. Let this one be X.

- 3. If X has reached desired accuracy (as measured on CV), then exit, using X as the matcher. Otherwise debug X. To do this, split the development set I into a training set U and a testing set V . Train X on the training set U, then apply X to the testing set V . Examine the false positives and false negatives that X makes on the testing set V , and debug X accordingly. We will discuss debugging in the class.

Once you have exhausted things to do with the false positives and negatives on the testing set V , you may want to do another split of I into a new training set U and a new testing set V , then repeat the above debugging process. Do this as many times as you judge necessary.

During this step, occasionally you may want to measure X’s accuracy, by performing CV for X on I. This will tell you if you have been able to improve X’s accuracy on the set I.

- 4. Once X has been debugged, the data/label/features may have changed. So you will repeat Steps 2-3. Specifically, you will perform CV again for all matchers, then select the best matcher, and so on.
  Eventually you stop when you have obtained the desired accuracy, run out of things to try, or out of time. Let the best learning-based matcher obtained at this point be Y*.
- 5. Train Y∗ on the set I, then apply Y∗ to the set J and report the accuracy of Y∗ on the set J.

We now describe these steps in detail.

Done

1. Creating Development and Evaluation Sets: Randomly split the set H into a development set I of at least 250 examples and an evaluation set J of at least 100 examples. The idea is that we will do the development to find the best matcher on I, and we will report its precision and recall on J. J will be a data set that we don’t ever look at, we just use it to compute precision and recall.

2. Selecting the Best Learning-Based Matcher: You must use the scikit-learn package and consider at least the following learning algorithms in scikit-learn:

- Decision Tree
- Random Forest
- Support Vector Machine
- Naive Bayes
- Logistic Regression

Perform CV for all of them on the set I. Then use the CV result to select a best matcher. Let this matcher be X.

3. Debugging the Selected Learning-Based Matcher: If X has reached the desired accuracy (as computed by doing CV on I) then exit, reporting X as the best matcher found. Otherwise you need to debug X.

To do so, first split the labeled data I into a training set U and a testing set V . Many splits are possible. For example, you can split 50-50.

Next, use U and V to debug X. We will discuss debugging in the class. Examples of methods that can be used are “proxy debugging” or “feature based debugging”.

Note that by “debugging”, it pretty much means you identify a problem and then try to debug the problem.
In this context, a problem will mean a false positive or a false negative. So you should examine the false positives and false negatives.

The basic idea is that you train X on U then apply it to V . Since you know the labels on V , you can easily identify the false positives and negatives on V . You should think about what causes these false positives and negatives and what you can do to fix the problems. Typical actions include: checking if the label is incorrect, is there any problem with the tuples (maybe some of their values are incorrect, e.g., the length is mistakenly put into the place of the width)? Maybe the thresholds of various parameters of learning algorithm are being set are too high, maybe a new feature has to be added, maybe a different policy for filling in the missing values should be used, and so on.

4. Repeating the Steps of Selecting the Best Learning-based Matcher: Eventually you either (a) run out of things to debug, or (b) get tired, or (c) found the accuracy to be acceptable. At this point you might have changed the data (e.g., cleaning it up), the labels, the features, and other stuff. So it may be that X is no longer the best learning-based matcher.

So you should repeat the above two steps. That is, first redo cross validation on I to select the best learning-based matcher. Let this matcher be Y . If this matcher is still X (i.e., this cross validation step selects X again as the best matcher), then you can stop. Otherwise, you may want to debug Y . And so on.

Eventually you feel that you have done everything possible with learning based matching. At this point you have a learning-based matcher that you judge to be the best. Let this matcher be Y*.

5. Computing the Accuracy of the Best Matcher: Now train Y ∗ on I then apply it to J and report the precision and recall on J.

Remember that J is the evaluation set of at least 100 examples and that you have NOT touched J at all until now.

Note: As described, your goal is to use the set I to develop a good learning-based matcher. During the development process, you may wonder what is the accuracy of a particular matcher M. To measure this accuracy, perform CV for M on the set I.

Tip 1: While the above talks about splitting the set of feature vectors H into two sets I and J, it may make it easier for you to split the original labeled set G into two sets I and J. We will discuss this in the class.

Tip 2: There are two kinds of error: one-off error and systematic error. If you detect systematic errors, you should fix them not just in I, but also in J (and in general over the entire set C). We will discuss in the class.

What to Report

You must submit the golden data file G on your team's project page.

You must also submit (on your project page) a pdf file that describes the following:

- For each of the five learning methods (Decision Tree, Random Forest, SVM, Naive Bayes, Logistic Regression), report the precision, recall, and F-1 that you obtain when you perform cross validation for the first time for these methods on I.

- Report which learning based matcher you selected after that cross validation.

- Report all debugging iterations and cross validation iterations that you performed. For each debugging
iteration, report (a) what is the matcher that you are trying to debug, and its precision/recall/F-1, (b)
what kind of problems you found, and what you did to fix them, (c) the final precision/recall/F-1 that
you reached.

For each cross validation iteration, report (a) what matchers were you trying to evaluate using the
cross validation, and (b) precision/recall/F-1 of those.

- Report the final best learning-based matcher that you selected, and its precision/recall/F-1.

It is important to note that all precision/recall/F-1 numbers asked for in the aboves are supposed to be numbers obtained via CV on the set I.

- Now report the following:

– For each of the five learning methods, train it on I, then report its precision/recall/F-1 on J.
– For the final best matcher Y∗, train it on I then report its precision/recall/F-1 on J
– List the final set of features that you are using in your feature vectors.

- Report an approximate time estimate: (a) how much did it take to label the data, and (b) to find the best
learning-based matcher.

- Discuss why you can't reach higher precision, recall, F-1.

Comments

You do not have permission to add comments.