Data Mining: Final Project

1 Introduction

For this project, we will participate in the Kaggle.com competition "House Prices: Advanced Regression Techniques". You got your first experience with Kaggle.com in hw06.

Log in to Kaggle, and then go here for the contest we'll participate in:
https://www.kaggle.com/c/house-prices-advanced-regression-techniques

Skim around on those pages a bit to get a sense for the competition and the problem domain. Under the data tab, there are various files you can download. I've already downloaded them and put them on Moodle for you, so you should already have them in the zip along with this pdf.

Open up the gettingStarted.py file if you haven't already. Run this code, and spend a little time studying it before proceeding in this document. This is all code you could write yourself with a little time. Spend some time considering what it's doing and how it is setting up an organization for your work.

For this project, it is essential that you have a good understanding of all previous assignments in this course, including hw06.

1.1 What's the Scope of this Project?

Briefly, in this project you will process the dataset, apply some machine learning algorithms to it, tune your processing and algorithms through many experiments, and write a report of your work. This is a significant project, accounting for a significant part of your final grade in the course: somewhere around 40 points.

In class we will continue to cover additional topics for the remainder of the semester. Therefore, some of your out-of-class time will be spent studying in-class material. Of course there will be many variations among people in the class and from week to week. But I might expect that on average for the rest of the semester, you'd spend about 2 hours per week working on the ongoing class material, and another 5 hours per week working on the project. This makes just a bit over 2 hours out of class for every hour in, which is a fair target.

So to summarize, the project should represent about 5 hours per week of work. There are roughly 4 weeks left before the end of the semester, including finals week. (Note that this project will take the place of the final exam, and so, is due during finals week.) So 4 weeks, times 5 hours per week of work, makes about 20 hours of work that should be reflected in the project. This is 20 hours, assuming a good amount of comfort with the course material. If you are behind in your understanding of course material up to this point, more time may be required, to catch up. Let me know how I can help!

If you're interested in taking things even further, though, you might consider working even more, as much as you can and would like. This would be a great item to add to your resume. This work may also inspire you for a very interesting senior project (if you haven't yet done the senior project) and/or may be something great to talk about in a job interview.

1.2 Using Code and Ideas You Find Online

There are a lot of good ideas, and a lot of code, online. You are welcome to read up on whatever you want and use whatever you want. Done correctly, this can be a great learning experience. If you copy code (whether a single line or many lines) from somewhere, you must provide BEGIN, END, EXPLANATION, and ADDED comments as illustrated below:

```
def printS(s):
    # BEGIN: from http://www.stackoverflow.com/fake
    # EXPLANATION: Loops through s one character at a time.
    #   -1 is the index for the last character,
    #   so [:-1] means to slice up to but not including last char
    for c in s[:-1]:
        print(c, end='-')
    print(s[-1])  # ADDED: To handle last character differently
    # END: from http://www.stackoverflow.com/fake
```

If you can't explain the code, I'm asking you not to use it. If you make adjustments to the code, indicate that with "ADDED" comments. If you didn't copy any code, but just got an idea from somewhere, just cite the website.

You must have at least 50% of the code be your own. That means that if you copy a lot, then hey, that's cool. You just need to write more code of your own to build off of that foundation you got from online.

I know what material is out there, and it's easy for me to quickly google things in your code we haven't talked about in class. If you use material you find online but do not cite it as described above, there will be severe penalties, up to and including failing the course and being reported for academic dishonesty. If you have any questions or concerns, please feel free to ask me *before you turn in your work* – no problem, we'll work it out together! This is a major issue in real-life work. For example, do you know about the several billion dollars lawsuit between Oracle and Google over copyright infringement of code?
https://en.wikipedia.org/wiki/Oracle_America,_Inc._v._Google,_Inc

1.3 Working with Other People in Class

You should do everything with your partner, of course. Other than that, please don't share any code with anyone, or look at anyone else's code from this course or previous offerings of the course. You are welcome to discuss big-picture ideas without code, though. As with the online resources discussion above, if you have questions or concerns, please talk with me *before* doing it.

2 Completing this Project with Excellence

2.1 Data Mining: An Iterative Process with Feedback Loops

As described in detail below, your work will include processing the data, applying algorithms, experimenting and tuning your approaches, and writing a report of your results. However, do not think of these as sequential steps. Rather, this is an iterative process, where lessons learned in a "later" step will lead to more enhancements in "earlier" steps.

For example, you might do some initial pre-processing of the data and apply some algorithms, only to get mediocre results. You write up these results, and in your write-up, hypothesize about why the results were not very good. The act of writing takes time, but it forces you to think in more detail about what you've done, clarifying your thinking and leading to new ideas. You implement these new ideas, experiment some more, write up some more, etc.

In the following sections, I provide more advice about the kinds of things a very well done project will include.

2.2 Pre-process the Data

This step is called "pre-processing" because it's necessary before we attempt to build a model. This is not to suggest that this work happens only once, at the beginning of the project, though. As described above, you will need to refine your pre-processing throughout the project as you obtain experimental results and try additional ideas.

In the provided code, this will happen in the transformData function (and any helper functions you write and call from there).

The first step is to study your dataset. Know it deeply! Pore over it! If you don't know all about what your data says, you can't use it effectively. Start with data_description.txt. It describes the possible values for each attribute. For each attribute, ask yourself:

- What is the type of this attribute? (Nominal, ordinal, interval, ratio)
- Do I need to convert its type to something else? If so, what?
- Do I need to normalize or standardize this attribute?
- Are there missing attributes? How many? How should I handle them? Should I replace missing values? Delete rows with missing values? Delete columns with missing values? How is a missing value indicated for this attribute? Is a value truly "missing", or can I derive it somehow from other values? Does missing mean "unknown", or "not applicable", or what, and how might this affect how I handle that "missing" value?
- Intuitively, do I expect that this attribute is very important, somewhat important, or not very important? In terms of correlation with other attributes (using the `corr` method), is this attribute highly correlated with others, meaning that some should be dropped?
- Should I transform the values in some way to more clearly capture what is important about them?
- What do I understand intuitively about this attribute that should influence how my system uses it? What do I expect to be the case about this attribute, and how can I explore whether or not that expectation is correct?

Note that your answers to the above questions may be different for each attribute. Several attributes may behave similarly, but if you think they all behave the same way, you probably have not considered them deeply enough.

You might also ponder whether it'd be useful to create additional attributes. Might a new attribute be derived from existing attributes somehow? What key ideas could be determined from the data that are not explicitly represented from the data at this time?

Plan on spending lots of time on pre-processing – quite possibly more time here than on any other step. It makes an enormous difference in the success of your model!

As you make pre-processing decisions, **be certain you keep a careful log of your preprocessing decisions**. Keep writing things up as you go, to help you organize your thoughts, and so you don't forget what you've done and why later on. You may think you'll remember all your decisions and explain them well later, but… you won't! There's just too much to do to keep it all in your head week after week.

2.3 Building a Model

With some initial pre-processing done, you're ready to feed your data into an algorithm to create a model. Using the built-in classes of scikit-learn, it's surprisingly easy to try an algorithm out. Any regression algorithm is a potentially applicable algorithm to this task, since this is a regression task (as opposed to a classification task). The provided doExperiment function illustrates the use of the LinearRegression class.

Note: in your call to cross_val_score, use the parameter:
`scoring='r2'`
This is a built-in scoring method. I'll share more information about it later.

If your code crashes when you try to apply an algorithm, it may be that some kind of pre-processing is necessary that you haven't done. For example, does your algorithm require all numerical attributes?

In addition to LinearRegression, try other algorithms. I'd encourage you to do a Google search on things like "scikit-learn regressor". This should be comparatively quick and easy to do given what you've already accomplished. You basically just call a constructor, and pass the object to cross_val_score – the same as with KNeighborsClassifier in hw05, and the same as with LinearRegression in doExperiment.

As you do this, make sure you're spending time organizing and cleaning up your code! Good design, good variable and function names, avoiding code repetition, etc. – these efforts are all time well-spent, since you're in this for the long haul.

As you begin to get results from these algorithms, review your pre-processing work again and determine if there's something you want to adjust at this time.

2.4 Tune Your Algorithms and Pre-Processing Via Experiments

To "tune" your work means to:

1. Hypothesize what adjustments may be useful:
    a. Should you pre-process some data differently?
    b. As you read the documentation about a particular algorithm in more detail, what parameters does the algorithm have? Which ones seem most important? Might it make sense to use a value other than the default for some of them?
    c. Is there a different algorithm altogether that you should try?
2. Make the adjustments and run experiments on the resulting system performance.
3. Log your results, and interpret them. Try to explain them. Return to step (1).

We did a little bit of tuning in our experiments with different values of k in k-NN. You can use that approach as a basic guide here. To take things further, you may consider trying the advice at the link below, as an example of tuning for a "gradient boosting" algorithm:
https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/
You can follow similar principles for other algorithms, and also see what else you find online.

It is very important that you document carefully, in your code, in Excel spreadsheets, and in text notes, what experiments you're trying and what results you get. Again, you may think you'll remember what's what, but very quickly things will become disorganized and confusing. You'll forget what results correspond to what system configurations, your experiments will be inadequately controlled, etc. I speak from experience, both in my own past (!), and in work with other students.

2.5 Report Your Results

As I've advised above, please be certain that you're writing throughout your project, not just at the end. The writing will help you clarify your thoughts and determine where to go next as you work. It will also help you get important ideas down while they're fresh in your mind, before you forget some details.

For full credit, your report must be written using a tool called LaTeX. I've included some tutorial materials on Moodle for you on learning to use LaTeX. This will be a great skill for you to have! In your report, you'll need to include figures, graphs, tables, equations, pseudocode, non-hardcoded references, etc. I will provide precise content requirements for the report in a separate document.

2.6 How Good Should My Model Be?

For grading purposes, it's not just about how good your best model's performance is, but about the depth of thought behind your code and the clarity of your report. So don't be too hung up on scores, but nevertheless you should find some improvement in model performance with your code.

The given code does LinearRegression with only two input attributes. It gets an R2 score of 0.56 (where numbers closer to 1 are better) in 10-fold CV. Uploading the resulting testResults.csv file to Kaggle shows a different scoring mechanism, in which lower numbers are better; by that scoring mechanism, the given code has a score of 0.26. These are pretty bad scores, as expected for such a simple approach.

In contrast, after some hard work, it's conceivable that the 10-fold CV score would be in the high 0.80's or low 0.90's. The Kaggle score could be less than 0.14 or so. Again, don't necessarily worry if your scores are not as good as these; the most important thing is the depth of thought in your code and the clarity of your report. But if your score really isn't improving much at all, it may be worthwhile for us to look together at your work so far.

3 Deadlines

There are two deadlines for this project:

1. A graded checkpoint submission due about halfway through.
2. The final products due during finals week.

Please see the Moodle page for precise dates and times of these deadlines.

3.1 The Graded Checkpoint

For the graded checkpoint, you must have the following completed:

a) Basic data pre-processing – enough to be able to run an algorithm while using at least 50% of the attributes.
b) At least one model created and results obtained, in addition to LinearRegression.
c) A Word document clearly describing your pre-processing steps, the algorithm(s) you're running, and the results you've obtained so far.

Turn in a zip of your code and your Word document to Moodle. Name the zip *lastname1-lastname2*-checkpoint.zip.

3.2 The Final Products

For the final product, please have the following completed:

a) Further improvement of your code, via:
    a. More refined data pre-processing.
    b. Multiple models created: different algorithms, different pre-processing strategies, and different parameterizations, all resulting from your tuning processes.

    If you do more work in one area, that can balance out less work in another, but do spend at least a little time in everything above.
b) Your LaTeX documents clearly describing all of this work, including your pre-processing, hypotheses, experiments, and results. Figures, graphs, and tables should be used when appropriate for capturing your work. More details of report contents will be provided in a separate document.

Turn in a zip of your code and all supporting documents. For LaTeX, please include your .tex, .pdf, figures, generated files like .out, everything. Name the zip *lastname1-lastname2*-final.zip.