Leveraging Historical Associations between Requirements and Source Code to Identify Impacted Classes - Notes

* R2RS Metrics – combo of natural language processing (NLP) techinques to measure semantic similarity in text and distribution scores to compute overall similarity
* Other metric examples:
  + Temporal locality
  + direct similarity to code
  + complexity metrics
  + code smells
* Impacted set – set of classes impacted by a new requirement
* Change impact analysis – finding the impacted set
  + Automated techniques to find initial set
    - Vector space model
    - Latent Semantic Analysis
    - Looks for classes based on history of change
  + To extend initial set
    - Dependency matrix/ graph
  + New proposed way is R2RS – uses the idea that a set of requirements which are associated with historical changes to a class are likely to exhibit semantic similarity to new requirements which impact that class
* **Classifiers** are just different machine learning algorithms used in Weka, and one of its goals is to differentiate between two or more classes.
* Finding Requirement set denoted RSC for a specific class C:
  + Set of commits associated with a requirement
  + List of source code files associated with those commits
  + List of present source code files at time of first commit of the requirement
  + Finally set of requirements associated with each class
  + Limit to 10 most recently implemented requirements
* (R2RS) Computing similarity between a new requirement R and a RSC (18)
  + Processing test of each requirement
  + Computing similarity between R and each requirement in RSC (6 methods)
    - VSM, JSD, GC, OPC, CMC, BC
  + Computing score for overall similarity between R and RSC ( 3 methods)
    - Max, average of top 5, average
* R2C- similarity of a requirement to a class (2)
  + Assumes good naming techniques
  + Uses VSM and JSD only
* Temporal Locality of Class Changes (3)
  + Simple Count percent, Linear, Logarithmic
* Complexity via CKJM (11)
* Code smells
* information gain ratio (IGR) is preferred over information gain because it is better at ranking metrics with a large number of distinct values
* used 80 – 20 for training to test
  + over sampled minority case - used a higher proportion of impacted classes otherwise would have gotten no classes are impacted since impacted classes is such a small ratio

**Data mining**

**Chapter 1**

* About looking for patterns
* *data mining* - the data is stored electronically and the search is automated—or at least augmented—by computer
* book is about techniques for finding and describing structural patterns in data. Most of the techniques are *machine learning*
* structural patterns – example decision trees
* Things learn when they change their behavior in a way that makes them perform better in the future
* *classification rules*: They predict the classification of the example in terms of whether to play or not
* *association rules* - disregard the classification and just look for any rules that strongly associate different attribute values
* association rules can “predict” any of the attributes, not just a specified class, and can even predict more than one thing
* the process of determining the weights is called *regression*
* *regression equation -* The classic way of dealing with continuous prediction is to write the outcome as a linear sum of the attribute values with appropriate weights
* These rules nicely illustrate the potential role of prior knowledge—often called *domain knowledge*—in machine learning, for in fact the only difference between the two descriptions is *leaf condition is normal* versus *leaf malformation is absent* 
  + Diagnostic rules can be generated through machine learning
* If *data* is characterized as recorded facts, then *information* is the set of patterns, or expectations, that underlie the data. You could go on to de ne *knowledge* as the accumulation of your set of expectations and *wisdom* as the value attached to knowledge

**Chapter 2**

* The input takes the form of *concepts*, *instances*, and *attributes*. We call the thing that is to be learned a *concept description*
* *classification learning*, the learning scheme is presented with a set of classified examples from which it is expected to learn a way of classifying unseen examples.
* *association learning*, any association among features is sought, not just ones that predict a particular *class* value.
* *clustering*, groups of examples that belong together are sought
* *numeric prediction*, the outcome to be predicted is not a discrete class but a numeric quantity
* we call the thing to be learned the *concept* and the output produced by a learning scheme the *concept description*.
* Association rules differ from classification rules in two ways: They can “predict” any attribute, not just the class, and they can predict more than one attribute’s value at a time.
* In database terms, you take two relations and join them together to make one, a process of flattening that is technically called *denormalization*
* Numeric attributes, sometimes called *continuous* attributes, measure numbers—either real or integer valued
* Nominal attributes take on values in a prespecified, finite set of possibilities and are sometimes called *categorical*
* Nominal quantities have values that are distinct symbols. The values themselves serve just as labels or names
* Ordinal quantities are ones that make it possible to rank-order the categories. However, although there is a notion of *ordering*, there is no notion of *distance*
* special case of the nominal scale is the *dichotomy or boolean*, which has only two members— often designated as *true* and *false* or *yes* and *no*
* Partial orderings—that is, generalization or specialization relations—frequently occur in practical situations. Information of this kind is often referred to as *metadata*, data about data
* idea of companywide database integration is known as *data warehousing*

**Chapter 5**

* The classifier predicts the class of each instance: If it is correct, that is counted as a *success*; if not, it is an *error*
* measure a classifier’s performance in terms of the *error rate.* The error rate is just the proportion of errors made over a whole set of instances, and it measures the overall performance of the classifier.
* The error rate on the training data is called the *resubstitution error* because it is calculated by resubstituting the training instances into a classifier that was constructed from them.
* To predict the performance of a classifier on new data, we need to assess its error rate on a dataset that played no part in the formation of the classifier. This independent dataset is called the *test set.* It is important that the test data is *not used in any way* to create the classifier.
* The training data is used by one or more learning schemes to come up with classifiers. The validation data is used to optimize parameters of those classifier, or to select a particular one. Then the test data is used to calculate the error rate of the final, optimized, method
* In statistics, a succession of independent events that either succeed or fail is called a *Bernoulli process*.
* you should ensure that the random sampling is done in a way that guarantees that each class is properly represented in both training and test sets. This procedure is called *stratification*, and we might speak of *stratified holdout*.
* A more general way to mitigate any bias caused by the particular sample chosen for holdout is to repeat the whole process, training and testing, several times with different random samples The error rates on the different iterations are averaged to yield an overall error rate. This is the *repeated holdout* method of error rate estimation.
* it is better to use more than half the data for training even at the expense of test data.
* In cross-validation, you decide on a xed number of *folds*, or partitions, of the data
* Split data into 3. Use 2/3 for training and 1/3 for testing and repeat 3 times with different combos. This is called *threefold cross-validation*, and if strati cation is adopted as well—which it often is—it is *stratified threefold cross-validation*.
* Extensive tests on numerous different datasets, with different learning techniques, have shown that 10 is about the right number of folds to get the best estimate of error
* tenfold cross-validation has become the standard method
* Leave-one-out cross-validation is simply *n*-fold cross-validation, where *n* is the number of instances in the dataset. Each instance in turn is left out, and the learning scheme is trained on all the remaining instances. It is judged by its correctness on the remaining instance—one or zero for success or failure, respectively. The results of all *n* judgments, one for each member of the dataset, are averaged, and that average represents the final error estimate.
* *0.632 bootstrap*. For this, a dataset of *n* instances is sampled *n* times, with replacement, to give another dataset of *n* instances. Because some elements in this second dataset will (almost certainly) be repeated, there must be some instances in the original dataset that have not been picked—we will use these as test instances.
* Bootstrap combines the test-set error rate with the resubstitution error on the instances in the training set
* *0 – 1 loss function*: The “loss” is either 0 if the prediction is correct or 1 if it is not.
* Cost sensitive learning - Take the cost matrix into account during the training process and ignore costs at prediction time.
* *ROC curves, receiver operating characteristic,* which are used in situations where the learner is trying to select samples of test instances that have a high proportion of positives
* ROC curves depict the performance of a classifier without regard to class distribution or error costs. They plot the true positive rate on the vertical axis against the true negative rate on the horizontal axis
* To summarize ROC curves in a single quantity, people sometimes use the area under the curve (AUC) because, roughly speaking, the larger the area the better the model

**Questions to ask Falessi**

1. Overview of my thesis goal again
   1. Predicting classification of tickets
   2. How to predict this, effects of misclassification, how much does this improve over manual predicting
2. Write java program using autoweka, does it matter which data set?
   1. Cross validation? Is that automated too?
   2. Just follow steps in documentation for now
3. Clarification on weka in the predicting change project
   1. Input – metrics and output- classifiers?
4. Stuff in plan
   1. What is a ticket? Tickets are things people post on github for changes in the program and are classified as requirement or bug etc
   2. Am I still computing metrics or is that from previous thesis goal?

**It’s Not a Bug, It’s a Feature: How Misclassification Impacts Bug Prediction – Notes**

* bugs—that is, requests for corrective code maintenance.
* *misclassified* issue reports—that is, reports classified as *bugs*, but actually referring to *non-bug issues*
* If such mix-ups occurred frequently and systematically they would introduce *bias* in data mining models threatening the external validity of any study that builds on such data: Predicting the most error-prone files, for instance, may actually yield files most prone to new features.
* it has become common- place to mine data from change and bug databases to detect where bugs have occurred in the past, or to predict where they will occur in the future.
  + Files are wrongly marked as fixed.
  + Files are wrongly marked to be error-prone.
* For our manual inspections, we used (a) the issue report itself, (b) all the attached comments and discussions, as well as (c) the code change that was applied to the source code.
* **BUG**- Issue reports documenting corrective maintenance tasks that require semantic changes to source code.
  + 1)  it reports a *NullpointerException* (*NPE*).
  + 2)  the discussion concludes that code had to be changed semantically to  perform a corrective maintenance task.
  + 3)  it fixes runtime or memory issues cause by defects (e.g endless loops).
* **RFE** - Issue reports documenting an adaptive maintenance task whose resolving patch(es) implemented new function- ality (request for enhancement; feature request).
  + 1) it requests to implement a new access/getter method.
  + 2) it requests to add new functionality.
  + 3) it requests to support new object types, specifications, or standards.
* **IMPR**- Issue reports documenting a perfective maintenance task whose resolution improved the overall handling or performance of existing functionality.
  + 1)  it discusses resource issues (time, memory) caused by non optimal  algorithms or garbage collection strategies.
  + 2)  it discusses semantics-preserving changes (typos, formatting) to code,  log messages, exception messages, or property fields.
  + 3)  it requests more or fewer log messages.
  + 4)  it requests changing the content of log messages.
  + 5)  it requests changing the type and/or the message of Exceptions to be  thrown.
  + 6)  it requests changes supporting new input or output formats (e.g. for  backward compatibility or user satisfaction).
  + 7)  it introduces concurrent versions of already existent functionalities.
  + 8)  itsuggestsupgradingorpatchingthirdpartylibrariestoovercomeissues  caused by third party libraries.
  + 9)  it requests changes that correct/synchronize an already implemented  feature according to specification/documentation.
* **DOC** - Issue reports solved by updating external (e.g. website) or code documentation (e.g. JavaDoc).
  + 1) its discussion unveils that the report was filed due to missing, ambiguous, or outdated documentation.
* **REFAC -** Issues reports resolved by refactoring source code. Typically, these reports were filed by developers.
  + 1)  it requests to move code into other packages, classes, or methods.
  + 2)  it requests to rename variables, methods, classes, packages, or configuration options.
* **OTHER** Any issue report that did not fit into any of the other categories. This includes: reports requesting a back- port (**BACKPORT**), code cleanups (**CLEANUP**), changes to specification (rather than documentation or code; **SPEC**), general development tasks (**TASK**), and issues regarding test cases (**TEST**). These subcategories are found in the public dataset accompanying this paper
  + 1)  it reports violations of JAVA contracts without causing failures (e.g.  “*equals()* but no *hashCode()*”).
  + 2)  complains about compatibility fixes (e.g. “should compile with GCJ”).
  + 3)  the task does not require changing source or documentation (like  packaging, configuration, download, etc.)
* The false positive rate represents the ratio between misclassified issue reports and all issue reports in the data set. The higher the noise rate, the higher the threat that the noise might cause bias in approaches based on these data sets.
* Misclassification makes a difference to data miners. If misclassified data is used on training set or prediction model, incorrect results are given