Learned Automatic Recognition Extraction of appointments from email

Lauren Paone
lpao@seas.upenn.edu
Advisor: Fernando Pereira

Abstract

Email has become one of the most prominent forms of communication. Many people receive an abundance of email each day making reading every email and extracting important information a very time consuming and tedious daily task. A lot of people use email as a means of scheduling appointments and notifying others of upcoming events. In this case, the recipient of the email must identify the details of the event and physically add them to a calendar or remember them. If a user does not have her calendar available as she is reading the email, she may forget to add the event.

This project involves trying to automate the extraction of important information regarding events and appointments from email. Specifically, two problems will be addressed. The first is identifying whether the email contains information about an event. Then, if there is information about an event, extracting things such as the title of event, the date, the time, and the location so that it could automatically be added to a calendar.

To perform these tasks, the program has two separate components: a classifier and an extractor. The classifier takes email as input and outputs a label indicating whether or not the email contains an event. The emails that do contain events are input into the extractor and the title, date, time, and location is output. These components are combined to create a full system that takes emails as inputs and outputs event information. This system is integrated as part of Penn’s CALO project.

Related Work

Information Extraction is a common application of natural language processing. Extracting “named entities, expressions that denote locations, people, companies, times, and monetary amounts” help interpret a block of text. A popular way to extract named entities is comparing text with “named entity recognizers” such as lists of people and geographic names (Bird et al, 2005). Extraction from email is a more recent application of natural language processing and a more complex version of the general information extraction problem. This is one that cannot be solved simply by using a series of named entity recognizers considering the variety of email and the lack of a globally consistent format. In the past few years, there have been numerous research projects regarding event extraction and other similar extraction applications. To date, no one has been successful with the extraction of events from email.

The known research most similar to mine is the project done by Julie A. Black and Nisheeth Ranjan of Stanford University. They too were working on extracting events
from email. They categorized all emails as one of three types: official meeting emails where the event information is separate from the rest of the email, personal meeting emails where the event information is in the body, and all other emails. Their classification algorithm is based on word frequencies. The extractor “uses pairs of sample documents and filled templates to induce pattern match rules that directly extract fillers for the slots in the template.” In addition, they have three specific features. They were not able to achieve their performance goals using this system (Black and Ranjan, 2004).

Some additional prior work has been done by Angelo Dalli of the University of Sheffield. Dalli’s goal was similar to mine and that of Black and Ranjan. He took a different approach to his project by summarizing emails. His project had four components: an entity recognizer, a threaded email filter, an email signature filter, and an email summarizer (Dalli, 2004).

At Carnegie Mellon University, William Cohen, Vitor Carvallo, and Tom Mitchell had a similar project about email classification that involved more natural language processing techniques. Their project was more general, not strictly focused on finding events but classifying all kinds of email. They came up with many classifications for emails based on nouns and verbs contained in the email. Examples are request, propose, commit, amend, and deliver information. These are combined with information about the nouns to come up with a classification or combination of classifications (Mitchell et al, 2004).

Trausti Kristjansson, Aron Culotta, Paul Viola, and Andrew McCallum have done work with information extraction to fill in database forms. They extracted the information from unstructured resources such as documents or email. Their goal was to create an interactive information extraction system which is “to assist the user in filling in database forms while giving the user confidence of the integrity of the data.” They used Conditional Random Fields (CRFs) which is the same technique used in my extractor (Kristjansson et al, 2004).

My project takes parts from each of these projects. This project is most similar to that of Black and Ranjan because it focuses on email and has both separate classification and extraction components. Their system however is based on pattern matching with sample documents whereas my project has a more natural language processing approach. Building specific features should be more effective than sample documents. Samples are only effective if all of the data is similar. Features can take into account different kinds of input and are will be more effective on unexpected data. Like Kristjansson, Culotta, and Viola, the extraction algorithm uses CRFs but I am applying this to email. A combination of all the techniques should, in the future, prove to be more successful.

**Technical Approach**

This program is divided into two large components: the classifier and the extractor. The diagram below depicts how the components interact. The following will explain the technical details of each component.
Both components use artificial intelligence machine learning techniques to train each system using pre-labeled emails. Each component has its own array of features that are specifically engineered to improve its accuracy. During training, each feature is assigned a weight. The weight of a feature represents the likeliness that an email or token matching this feature will positively identify with what the feature is testing for. For example, in the classifier, all of the features are designed to determine if the message contains an event. If a feature is weighted positively, that signifies that when that feature is found, the message most likely contains an event.

When an email initially enters the system, it is read by one of the mail readers based on its format and parsed. Information about the email such as the email’s ID, date, subject, body, etc. is stored separately so it can be easily accessed. The parsed email is then put through the classifier that determines whether or not the email contains information about an event. This information is the output of the classifier.

If there is no event in the email, the process is complete. If there is an event, then that email gets put into the extractor which will locate the title, date, time, and location of the event. If all of these are not present in the email, then the extractor will just return whichever are present. This completes the email’s run through the entire system. More generally, the input to the system as a whole is an email and the output is titles, dates, times, and locations regarding events in that email.

At this point, the entire system is built but is currently lacking a user interface. The goal of this project was to build the system and maximize its accuracy. The two components are being tested separately and have not yet been tested as one full system. The output of both systems is statistics that measure the accuracy of the system.

The accuracy is measured with three values: precision, recall, and fmeasure. Precision is a measure of the quality of the results, the percentage of emails that were classified as containing an event that actually do contain an event. A low precision implies there were a lot of false positives, meaning many emails that did not contain an event were labeled as containing an event. Recall is the percentage of messages that contain events that were not missed by the classifier. Precision and recall have an inversely proportional relationship. Fmeasure is a weighted average of precision and recall. It takes both into account making it the most important quantifier of the three. It gives a more accurate measurement of the system’s performance than either recall or precision alone.
When I started this project, code already existed for reply prediction. I used this code as a framework for my project. My project was to be implemented as an addition to the reply prediction project so all of my code was integrated directly into the pre-existing system. The reply prediction project already had a GUI for hand labeling emails, an email reader, a classifier for reply prediction, and an interface to machine learning algorithms that generated statistics.

The first task of this project and a principle technical challenge was to acquire the data to test and train on. Training and testing both the classifier and extractor require a huge dataset to get the most realistic results. This data cannot all come from one source. Diversity is necessary to ensure that this program will work with anyone’s email account and the program will be as robust as possible.

The first set of data I collected was from my own SEAS account which I downloaded from the server. I used the pre-existing labeler to label about five hundred of my own emails. I labeled these as having a structured event, unstructured event, or having no event. An email containing a structured event has the information formatted so that the computer should be able to easily parse it. For example, the email might contain the strings “Title:”, “Date:”, “Location:”, or other similar identifiers. An unstructured event is one that has no particular format and requires much more natural language processing for the computer to parse. For example, the email might contain the phrase “let’s meet at.”

Originally, the plan was to have the classifier output whether there was a structured event, unstructured event, or no event. Knowing ahead of time whether the event was structured could make it much easier to extract the information. As the work progressed, the focus turned to simply correctly identifying “event” or “no event” so structured and unstructured are both seen as just containing an event. At that point the extractor had not been started and if there was later evidence that distinguishing between structured and unstructured would help then it could be changed.

Training and testing on just my email would not be varied enough to get realistic results. Carnegie Mellon University had released the email corpora that they used for their research. Their emails are mostly task-related so it was very likely that their corpora would have many emails containing events. They changed all of the email addresses and removed all quoted material attachments and non-subject header files making the emails smaller and easier to use (Mitchell et al, 2004). They also included separate files containing labels for their emails. Their labels were more complex and required an interpreter. Also, the current mail reader was not compatible with the format of their emails. I built a new mail reader to read and parse all of their emails and an interpreter for the labels. From their labels, for the most part, I was able to tell whether the email would contain an event or not. If I could not determine that, I did not include the email in my tests so as to avoid potentially training on incorrect data. I was able to put all the necessary information in a format that could be used to train, test, and output statistics.
Once I had some data, I was able to build a classifier to start testing. The classifier is non-sequential, meaning it tags features to the email independently of how it tagged all previous emails. It uses a maximum entropy model which is a restricted model based on direction - it progresses forward and the algorithm does not consider past data. Not being able to go back produces limitations but since the tagging of the current email has no correlation with that of the previous email, that is all that is required (Russell and Norvig, 2003; Kristjansson et al, 2004; Lafferty et al, 2001).

There were already classifiers implemented in the project to determine if a message needs a reply or has an attachment. The “message contains event” classifier was modeled after these existing classifiers. The first trials were run with no event-specific features. The results on the basic classifier were much better than expected. The results can be found in the first rows of data in Tables 1 and 2, below. Despite the impressive results, they could be improved.

The body of the email is passed through sets of features. Each email that runs through the classifier has a linked list of features associated with it that is initialized as an empty list. If an email has a feature, that feature is added to the email’s linked list. If the email is part of the training data, the linked list of features is used to calculate the weights of all the features in the list. If the email is part of the testing data, based on the list of features and their respective weights, the classifier outputs whether this email contains an event or not.

By looking through some emails, I was able to design several features. Examples of event-specific features are does the email contain the string “Title:” or any word with a semicolon after it? Knowing the difference between structured and unstructured events was useful here. Some features were geared towards one of the types of events. Although the computer did not distinguish between the two, I used this knowledge when creating the features.

The features were programmed using Java Regular Expressions. The body of the email is passed through the features as one string and each regular expression is tested on that large string. I added three categories of features: Structured, Event Bigrams, and Event Dividers. The first is targeted at structured events. This feature includes a number of regular expressions that look for a word with a semi-colon after it. Some are specific words and others are more general regular expressions. After running a series of tests, the more specific ones proved to be too specific so I removed those. For Event Bigram features, I compiled a list of bi-grams that I found in many unstructured events and built features to specifically address them. I noticed that in some structured events that there are dividers between the event information and the rest of the email so I built features identifying those. Then I ran the same tests again with these specific features. Those results are displayed in the table below.
<table>
<thead>
<tr>
<th>Features</th>
<th>Precision</th>
<th>Recall</th>
<th>FMeasure</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>0.8793</td>
<td>0.9375</td>
<td>0.9055</td>
</tr>
<tr>
<td>Structured</td>
<td>0.8497</td>
<td>0.9458</td>
<td>0.8947</td>
</tr>
<tr>
<td>Event Bigrams</td>
<td>0.8772</td>
<td>0.9530</td>
<td>0.9123</td>
</tr>
<tr>
<td>Event Dividers</td>
<td>0.8665</td>
<td>0.9487</td>
<td>0.9039</td>
</tr>
<tr>
<td>All</td>
<td>0.8616</td>
<td>0.9579</td>
<td>0.9051</td>
</tr>
</tbody>
</table>

Table 1 Results of the classifier with my email
(Best results highlighted in bold.)

<table>
<thead>
<tr>
<th>Features</th>
<th>Precision</th>
<th>Recall</th>
<th>FMeasure</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>0.9306</td>
<td>0.8726</td>
<td>0.9001</td>
</tr>
<tr>
<td>Structured</td>
<td>0.9216</td>
<td>0.8369</td>
<td>0.8765</td>
</tr>
<tr>
<td>Event Bigrams</td>
<td>0.9262</td>
<td>0.8711</td>
<td>0.8973</td>
</tr>
<tr>
<td>Event Dividers</td>
<td>0.9221</td>
<td>0.8557</td>
<td>0.8874</td>
</tr>
<tr>
<td>All</td>
<td>0.9215</td>
<td>0.8740</td>
<td>0.8956</td>
</tr>
</tbody>
</table>

Table 2 Results of the classifier with CMU email
(Best results highlighted in bold.)

Tables 1 and 2 show the average precision, recall, and fmeasure of classifier tests run on my email and the CMU email. All tests used 80% of the emails as training data and tested with the remaining 20%. Ten runs of each test were performed and the results in the table are the average of those. The tests run on my email were completed with a total of 270 emails, 174 of which contained an event. The CMU tests were done with a total of 507 emails, of which 249 contained an event. I saved the information from these 10 total runs and used the testing data from my email as training data for the CMU email and vice versa. These results are displayed below.

<table>
<thead>
<tr>
<th>Features</th>
<th>Precision</th>
<th>Recall</th>
<th>FMeasure</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>0.6795</td>
<td>0.3518</td>
<td>0.4627</td>
</tr>
<tr>
<td>Structured</td>
<td>0.6767</td>
<td>0.3253</td>
<td>0.4379</td>
</tr>
<tr>
<td>Event Bigrams</td>
<td>0.6843</td>
<td>0.3382</td>
<td>0.4497</td>
</tr>
<tr>
<td>Event Dividers</td>
<td>0.6791</td>
<td>0.3382</td>
<td>0.4509</td>
</tr>
<tr>
<td>All</td>
<td>0.6705</td>
<td>0.3153</td>
<td>0.4279</td>
</tr>
</tbody>
</table>

Table 3 Results of CMU email trained on my email
(Best results highlighted in bold.)

<table>
<thead>
<tr>
<th>Features</th>
<th>Precision</th>
<th>Recall</th>
<th>FMeasure</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>0.8445</td>
<td>0.2155</td>
<td>0.3424</td>
</tr>
<tr>
<td>Structured</td>
<td>0.8402</td>
<td>0.2080</td>
<td>0.3334</td>
</tr>
<tr>
<td>Event Bigrams</td>
<td>0.8384</td>
<td>0.2063</td>
<td>0.3307</td>
</tr>
<tr>
<td>Event Dividers</td>
<td>0.8390</td>
<td>0.1977</td>
<td>0.3197</td>
</tr>
<tr>
<td>All</td>
<td>0.8359</td>
<td>0.2000</td>
<td>0.3222</td>
</tr>
</tbody>
</table>

Table 4 Results of my email trained on CMU email
(Best results highlighted in bold.)

As expected, the results of training on one email set and testing on another were a lot worse than training and testing on the same dataset. The content of the CMU corpora is much different than the content of my email. My emails include a lot of structured events and email sent to lists. A lot of these emails look alike and most of them contain all the event information in one email. The CMU emails, however, are emails between a group
of people setting up meetings. There are hardly any structured events and most of them are very short and only contain partial information about any event since they are responding to a question in a previous email. Training and testing on different types of email is a good test for how the classifier will handle unfamiliar data.

In both Tables 3 and 4, average precision is relatively high and average recall is low. Both sets are being run through the same set of features so anything that is an event in the CMU emails should constitute an event in my emails. Therefore, it should positively weight features found in the training data and correctly identify them when found in the testing data increasing precision. If features are present in the testing data and not in the training data then they will be missed during testing which reduces recall. In Table 4, both values are at extremes due to the CMU being so dissimilar from my emails. The CMU emails do not train the structured features well and therefore my emails matching the structured feature are missed during the tests.

In all four tests, adding features to the classifier hardly improved the results. For both my email and the CMU email being trained and tested on the same dataset, the average precision was highest when no extra features were included. The average recall, however, was the best when all of the features were implemented. Overall, the features did not have a large affect on average fmeasure which is the most important value to consider when evaluating the classifier. For the tests on the CMU emails the average fmeasure was highest when there were no extra features implemented. For my email, the Event Bigram features raised the fmeasure by 0.007. When testing and training on different datasets, the results were not improved at all.

These poor results show that the features were not effective. This is likely attributed to overfitting meaning the features are too specific which affects the weights when the machine is being trained. If there is a specific feature that only shows up few times throughout the training process but only appears in event emails, it will be weighted very heavily. Having a huge disproportional weight will affect the other weights and therefore throw off the classifier. Thus, the feature is overly fit for the email sample. This could be improved by reducing the feature’s specificity, adding more complexity to the weight function, or having a larger sample of emails to test on. If there had been more time I would have been able to better look into these options, however, since without the features the results were good I moved on to the extractor.

The objective of the extractor is to take the email that has been classified as containing event and output the title, date, time, and location if they exist in the email. The extractor has features just like the classifier and works in a similar manner but instead of tagging features to the entire body, the body is split up into words. Each word is a token. Features are tagged to individual tokens and the purpose of the extractor is to identify these tokens as being part of a title, date, time, location, or nothing.

For the extraction component, I made use of Mallet, a Java machine learning toolkit that has code for information extraction and natural language processing. The toolkit provides an implementation of Conditional Random Fields (CRFs) which are “a framework for
building probabilistic models to segment and label sequence data.” The tokens are a sequence of data that need to be labeled. Unlike the non-sequential model used for the classifier, CRFs are sequential and can take into account data and labels that come anywhere before the current token. By eliminating the limitations of the maximum entropy model, the CRFs will provide better results (Lafferty et al, 2001; Russell and Norvig, 2003).

There was no pre-existing framework code for extraction because there was no need for that in the CALO project up to this point. Luckily some graduate students at Penn were working on a project and had some extraction code. The input to their extraction code is a file with email bodies and subjects broken up into one token per line. Every token has a label next to it. The labels are as follows: B-<category> if the current token is the first word of a string of the type category, I-<category> for any word in the string after the first one, and O for a token with no classification. The categories are: Title, Date, Time, and Location. An example of the file would be:

```
et even O
   on O
   December B-Date
   21st I-Date
```

Punctuation can either be its own token or part of a larger token.

In order to have training and testing data for the extractor, I had to go through all of the emails I had collected and separate the emails containing events. Of approximately five hundred emails containing events, I went through all emails and labeled the title, date, time, and location as appropriate. To do this, I opened the email files in a text editor and surrounded the words pertaining to a particular category with an XML tag. So, using the example above, the string would be “event on <date>December 21st</date>”.

With all of the emails labeled, I developed a tokenizer to take a set of emails and generate a file of the correct format to be passed into the extractor. Originally, punctuation was on its own line but in the end it was left as part of a larger token. This makes times and dates which contain punctuation such as a ‘:’ or ‘/’ more easily identifiable.

Much like the classifier, specific features need to be built into the extractor. There were already some baseline features implemented such as testing if the token begins with a capital letter or is all digits. I added features specific to finding a title, date, time, or location. For example, a feature specific to finding a title is if the word is in quotes.

Many tests were run on the extractor. The tests were run on different numbers of iterations assuming that more iterations would yield better results. The results of these tests are in the table below.
My Features

Table 5 Fmeasures of tests run with 10, 20, 30, and 40 iterations
(Best results highlighted in bold.)

Table 6 Fmeasures for 30 iterations separated by tag

The measurements in Table 6 show that the extractor is best at identifying times and dates. This is probably true because times and dates have a particular format. Times are of the form “digit digit : digit digit” and are usually followed by “am” or “pm”. The extractor is programmed to recognize month names and their abbreviations, day names and their abbreviations and also the words “tonight”, “tomorrow”, and “today”. Also, the date may be formatted as “digit digit /digit digit / digit digit.” The extractor has much more of a problem identifying the less formatted information: titles and locations. There is no way to completely generalize what titles and locations should look like in unstructured email. Having many emails of a similar format or knowing what to look for in a sample of email is helpful. This is confirmed by the improvements on both categories with my features. This, however, is not a very robust solution.

The extractor results are a good starting point. The features are an improvement so adding more similar features would be helpful. Designing features to identify times and locations would greatly increase the results of the extractor as a whole.
Challenges

As previously stated, the primary difficulty I faced with this project was acquiring suitable data to use. I started this project with no data except what I had in my inbox. Since then I have been saving all my email and separating out the events. Even after seven months of saving email and the addition of the CMU corpus, I still do not have enough email to properly test the system. This, however, is a challenge for anyone attempting a problem of this nature. The researchers from Carnegie Mellon University also cite this as an issue. In their paper, they state, “Although email is ubiquitous, large and realistic email corpora are rarely available for research purposes.” This is due to the fact that all senders would have to provide written consent (Mitchell et al, 2004).

Although I did not have enough data for this project, I did have a considerable amount to label. When data needs to be labeled, the project is at a standstill because tests cannot be run without labeled data. Going through all of the events and labeling the title, date, time, and location is very time consuming and tedious task that leaves much room for error. In my tokenizer for the extractor, I built an error checker to identify the type of error and location of the error in the XML tags. This helped correct errors that would have been practically impossible to find without going through all emails again.

Another immediate technical challenge was familiarizing myself with the framework that was already in place. In order to update the code and use it as a model for some of my code, I had to understand many classes. Some were not well documented and some employed features of Java that I had not seen before this project. Also, getting used to someone else’s style of coding and keeping my code consistent with that style required additional effort. After studying the code, researching online, and asking the original authors countless questions, I felt more comfortable with the code.

For this project, I used the Java programming language and the Eclipse IDE on my personal computer and used CVS as the revision control software. Some of the framework code used Java 1.5 features. My computer, however, could only support Java 1.4.2. Every time I would download updated code, I would have to change all the Java 1.5 specific lines to be Java 1.4.2 compatible. Both versions of Java are similar but with so many lines of code it took a significant amount of time to get to the project to work on my computer. This delayed progress and took time away from more important problems.

Before doing this project, I had never worked with Java Regular Expressions. I had to take a couple days to do a tutorial on how to use them. To better understand their usage, I built two regular expression testers: one for the classifier features and one for the extractor features. Before being put into the code, the regular expressions are run through their respective tester.

The classifier had problems with overfitting my features. It is very difficult to design features with the perfect balance of specificity. The problem is that when one improves precision, recall usually drops and vice versa. This does not improve the classifier as a whole. I had a considerable difficulty with this and tried to play around with my features
but was not able to solve the problem and design great features. Unfortunately, time did not allow me to pursue this further.

Since some base code for the extractor was obtained, the extractor was not as problematic as I expected it to be. It was, however, very impractical to run on my computer and would have consumed my computer for days at a time. For good results, I had to run about thirty iterations on an email sample of about five hundred. I did not have access to a computer that I could run this on. Every time I made a significant update, the graduate student I was working with would download the code from CVS and run it on a server. It takes about a full day to run. So after making changes, I had to wait about a day or two to get results. With the final deadline of this project approaching, it was not convenient to have to wait so long to get results. Therefore, I had to be patient and test as much as I could before running a big test.

In a real-world implementation of this project complete with a GUI, the run time would not be a concern. The long run time is attributed to having to train and test the whole system all at once using CRFs. The downside of the flexibility of a CRF is a long run time, especially on a large sample of data.

**Conclusion**

Despite the challenges faced, I was able to achieve my goal of building both elements of the system. I addressed the two major problems involved in this project. The is now set up with another mail reader, label interpreters, a new classifier, and an extractor among other smaller previously mentioned elements of the system and the project is going in the right direction.

This project was a great opportunity to pursue my interests in computer science: machine learning and natural language processing. I have never gotten to take part in a research project at Penn so this gave me a better understanding of how researching works and how much research is going on in computer science. It was a great learning experience and it was fun progressing towards a goal that no one has been able to achieve.

The project overall was harder than I expected. In the beginning, I had a well laid out plan that did account for challenges and time to debug but even that was too ambitious. Since I was working in Java which I was very familiar with I did not expect it to be so hard to understand all of the code. To integrate a project of this magnitude into an already huge amount of code requires a lot of time. I expected the features to do better and did not learn about the possibility of overfitting until after I had tested them. When I was beginning this project, I underestimated the amount of work that has to go into every element: data finding, the classifier, the extractor, and integration.

In retrospect, I don’t think I would have done much differently given the time constraints. One thing I would do is work on the project with a partner. This project as a whole provides more than enough work for two people but most importantly, it would greatly increase the number of emails I have to test and train on. If I had more time, there are a
lot of improvements I could make to the system. If I were to continue this project long
term, I would create a corpora like the CMU corpora complete with name changes. I
would try to improve the classifier and tackle the overfitting problem. Tests could be
done to see if the extractor’s results are improved when the events are distinguished
between structured and unstructured. If that worked, the classifier could differentiate
between the two. Finally, I would love to connect the two components of the system and
have emails do a complete run.

Most likely this project will be continued next year. I plan on donating as many emails as
I can to add to the data. To protect privacy, they will probably only be non-personal
emails sent to list-serves. It is very possible that this soon could be a full functioning
system that can be integrated into an email client such as Thunderbird. Hopefully Penn
will progress in that direction.

Works Cited

1. Steven Bird, Ewan Klein, and Edward Loper. NLTK Tutorial: Introduction to
   Natural Language Processing. 2005.
   2004.
3. Angelo Dalli. Automated Email Integration with Personal Information
4. Mark Dredze, John Blitzer, and Fernando Pereira. Reply Expectation Prediction
   for Email Management. 2005.
5. Trausti Kristjansson, Aron Culotta, Paul Viola, and Andrew McCallum.
   Interactive Information Extraction with Constrained Conditional Random Fields.
   2004.
8. Tom M. Mitchell, William W. Cohen, and Vitor R. Carvalho. Learning to