Using Multiple Classifiers to Improve Intent Recognition in Human Chats

Joe Dumoulin
Next IT Corp.
Spokane Wa.

Abstract

Text classification is an important and uniquely challenging area of research. Successful techniques abound, but no technique really stands above the others for ease of training, successful classification, and maintainability. Successful classification is particularly elusive when one tries to classify text by intent. This paper, instead of trying to improve a particular classification method, explores a theory to improve text classification by combining multiple classifiers that are trained using different methods.

Introduction

Recognizing and acting on user intent is the key to any successful automated chat system. This paper discusses the theory underpinning a method for combining the results of multiple systems each designed to classify user intent. the method should produce better results by combination than each system is separately capable of doing. The paper does not show any experimental results but instead examines the theoretical foundations of classifier combination in the context of human chats and user intent.

Training and Testing of Classifiers

A classifier is a program that can automatically decide whether an input is a member of one or more class of objects. There are many excellent resources for examining the theory and practice of creating classifiers. See (Duda, Hart, and Stork 2012; Ripley 2007) for two examples. A classifier is trained to recognize classes in a particular domain and training can be manual (a person defining the program steps) or automatic (using one of many machine learning methods).

More precisely, suppose we have a domain D comprising objects which can each participate in one of K distinct classes $C \in \{c_1, c_2, ..., c_K\}$. Then we can develop a classifier over this domain as follows. Take N inputs $X = \{x_1, ...x_N\} \in D$ and a labeling operation that assigns one class to each input. For best results, N should be much greater than K and each c_i should be assigned as many inputs as possible.

Next a preprocessing function is usually used to extract *features* from the inputs. In text processing features might include calculating the number of words, characteristics of words, parts of speech, length of the longest word, etc. The preprocessing function $\rho : D \rightarrow F$ produces a vector of features for an input: $\rho(x_i) = \hat{f}$, where $\hat{f} \in F$.

Finally, training is performed, producing a function

$$\phi : F \rightarrow (C \times \mathbb{R})^M$$

which maps a feature vector to a set of classes and a score for each class. The output of phi is a vector of label/value pairs with a pair for each of the M classes. Typically training proceeds iteratively, attempting to reduce the error in ϕ to a minimum. Error is a measure of how well ϕ maps the input x_i to the labelled class c_j. Note that ϕ returns more than one class and a real number score for each input. For the remainder of this paper (unless otherwise noted) we assume that ϕ returns only the top scoring class and we ignore the rest of the return values.

Using ϕ as a classifier, we can now take a new input x not part of the training set, preprocess x to get $\hat{f}$, then find which class in C that corresponds to x. ϕ actually outputs a vector of pairs of labels and scores.: one label/score air for each class of the M classes in C. We use the scores to rank the class vector entries and select the label with the highest score as the 'correct' class label. How do we tell if ϕ is returning correct results? If ϕ does not produce correct results all the time, how can we evaluate the quality of ϕ? This brings us to the last part of classifier training: *testing*.

Testing

Most classifiers do not produce perfect results. This is clear if we think about the training set

$$\{x_1, ..., x_n\} = X \subset D.$$

The training set X is intended to be a representative subset of the Domain D that we want to classify. X is usually much smaller than D. If X is properly representative and if our training method is sufficient, then we can expect the classifier to answer correctly much of the time. We can judge the quality of the representation using a set of labelled inputs held out of the training. We call this the test data. Since we know the correct label for the test data, we can compare the

correct result with the classifier's highest ranked result and get a measure of the quality of the classifier ϕ.

The testing process is performed as follows:

1. Run the test data through the classifier,
2. Collect the results,
3. Compare the results of ϕ with the correct values.

Test Evaluation

We can compare the classifier results and the correct results in a number of ways. One basic measure is *accuracy*. Accuracy measures the percentage of test inputs that are classified correctly by ϕ. More formally, accuracy is calculated by summing all correct classifications and dividing this by the number of inputs in the test data. Let c_j be the *correct* class label for t_i and N_{test} is the number of test inputs $\{t_1, ..., t_{N_{test}}\} = T$. Next consider the indicator function I defined as:

$$I(t_i) = \begin{cases} 1 & \phi(t) = c_j \\ 0 & otherwise \end{cases}$$

Now accuracy is calculated as:

$$acc = \frac{\sum_{i=1}^{N_{test}} I(t_i)}{N_{test}}$$

Another way to use accuracy is to calculate the accuracy for each class. Let N_t^c be the number of test inputs whose correct class is c, $\{t_{1,c_j}, ...t_{n,c_j}\} = T_j$ are the test inputs whose correct class is c_j and let I_j be the indicator function as defined above for the class c_j. Then the accuracy of ϕ for each class c_j is:

$$acc_{c_j} = \frac{\sum_{t \in T_j} I_j(t)}{N_j}$$

In many cases the per-class accuracy will reveal that some classes have higher accuracy under ϕ than other classes. This is important information when we are trying to analyze error and improve classifiers (i.e., improve the accuracy of ϕ). We can extract a little more form this by looking at the closely related *confusion matrix*.

The Confusion Matrix

The idea of the confusion matrix is to represent the degree to which ϕ is 'confused' between the correct class and ϕ's selected class. If ϕ is not confused on class c_j, then ϕ maps all T_j inputs to c_j. If ϕ is confused on c_j then ϕ will map some of the inputs T_j to other classes. It is very rare for a classifier to map all the test inputs correctly so we always expect some degree of confusion in ϕ.

To represent the confusion of ϕ, first consider the inputs T_j^i that are labelled c_j but that ϕ maps to c_i. Next redefine the indicator function to be more specific:

$$I_j^i(t) = \begin{cases} 1 & \phi(t) = c_i \text{ and the class of } t \text{ is } c_j \\ 0 & otherwise \end{cases}$$

Next define the sum of each indicator function for i and j:

Test Input	True Label	ϕ Label
0	A	A
1	A	A
2	B	B
3	B	A
4	B	B
5	B	C
6	C	A
7	C	C
8	C	C
9	C	A

Table 1: Test data for a classifier ϕ used to illustrate accuracy and confusion matrix concepts

$$N_j^i = \sum_{t \in T_j^i} I_j^i(t)$$

With this background, the confusion matrix contains the sum of test inputs that ϕ understood in one class but were actually labelled as another class, as well as the correctly classified inputs as in the per-class accuracy. The matrix looks like this for a classifier ϕ defined on a domain of K classes:

$$CM_\phi = \begin{array}{c} \\ c_1 \\ c_2 \\ \vdots \\ c_K \end{array} \begin{array}{c} \begin{array}{cccc} c_1 & c_2 & \cdots & c_K \end{array} \\ \begin{pmatrix} N_1^1 & N_1^2 & \cdots & N_1^K \\ N_2^1 & N_2^2 & \cdots & N_2^K \\ \vdots & \vdots & \ddots & \vdots \\ N_K^1 & N_K^2 & \cdots & N_K^K \end{pmatrix} \end{array}$$

Example

The concepts above can be illustrated by considering a small classifier with 10 test inputs labelled with 3 classes. Table 1 shows the data used to test a simple classifier ϕ and the raw results of the test. In this case accuracy measures the ratio of correct classifications from ϕ compared to the number of test inputs. In this example

$$accuracy = 6/10 = 60\%.$$

Per class accuracy measures the ration of correct classification per inputs *per class*. This gives one value for each class, or:

$$\begin{aligned} accuracy_A &= 2/2 = 1 \\ accuracy_B &= 2/4 = 0.5 \\ accuracy_C &= 2/4 = 0.5 \end{aligned}$$

From this we can see that, though the average accuracy is 60%, the accuracy associated with class A is perfect. Further refinement can be achieved by looking at the unnormalized confusion matrix for this test data and ϕ. The confusion matrix is:

$$CM_\phi = \begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{pmatrix} 2 & 0 & 0 \\ 1 & 2 & 1 \\ 2 & 0 & 2 \end{pmatrix} \end{array}$$

The confusion matrix reveals even more details regarding the classifier ϕ. for example, the confusion matrix tells us that if, given some input, ϕ emits class A, then we can't tell whether the actual class is A,B, or C. We can estimate the likelihood of each class from the confusion matrix. The likelihood of ϕ emitting class A is $5/10 = 0.5$. The likelihood of emitting class A if the actual class is A is 1, but since ϕ confuses other classes with A frequently, we can't tell if the actual class is A. This fact is very hard to see without the confusion matrix.

The confusion matrix gives us a snapshot of the classifier ϕ with respect to the classes it thinks are relevant to the input data. Later we describe how to use the confusion matrix to determine which is the most likely class for unlabeled inputs when multiple classifiers are used. Before exploring multiple classifiers, this paper discusses the use of intent as a class for natural language chat inputs.

The Idea of 'Intent' in Human Discourse

It is generally accepted that human discourse takes place on (at least) two different levels: one is the level of symbols and their accepted arrangements. This is the domain of syntax. The other is the level of ideas and concepts. This is generally known as semantics. These are traditional areas of research for both the linguist and the Natural Language Programming (NLP) researcher. For an excellent survey of these topics and classification methods within their scope, see (Jurafsky and Martin 2009).

Another important area of research in NLP is the idea of **intent** in discourse, specifically in the domain of chat discourse. In this paper, intent is defined as *An interpretation of a statement or question that allows one to formulate the 'best' response to the statement*. What makes intent significant to NLP (specifically chat) applications is that, if a computer program can determine intent then the program can derive a meaningful response and thus can conduct a conversation with a person in a chat context.

For example, In an airline domain, where the system should answer inquiries regarding flying, some examples of intents might be:

- Number of Bags accepted,
- Max Age of Lap Child
- Procedure for Changing a Ticket

Each of these is an example of an intent. That is, one can capture the inquiries that are asking for information on these intents (i.e, "I need to change my flight.", How do I change my ticket?", etc.) and then formulate a response that describes the needed outcome.

But how can one create a classifier system that will identify intent in the way described in the example above?

Building Intent Classifiers

It turns out that there are many ways to train a system to identify intent. Information on some of these methods can be found in (Chakrabarti 2014). In order to build an intent classifier one should first recognize that intent is not a unique feature. In other words, the first step in developing these classifiers is to determine what is meant by 'intent' in the domain being studied. One easy way to identify intent is to let people classify a set of chats into classes with similar answers. The chats become the training data and each class becomes an intent. Since different people will have different ideas about how to classify chats, the class of a chat will often be determined by a vote of multiple people.

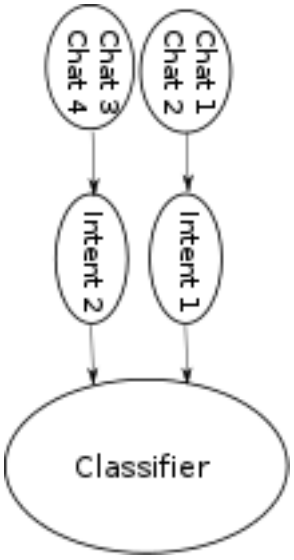

Figure 1: Training an intent classifier on labelled chats. First associated chats with intents (labelling). Then train the classifier to label new chats.

Training

Once the intent classes have been identified and the training data has been associated to intents, one can train a classifier to recognize the best intent to associate to new chats. As with the association, there are many methods for training a classifier.

One method is to directly associate a text string, a part or all of a chat, with the intent. Then any new chat that contains that string will be associated with the same intent (the assumption is that the same response can be given to both the training chats and the new chat). Another similar technique uses regular expressions to identify patterns found in chats associated with a single intent. This paper identifies classifiers trained in this way are **symbolic classifiers** Then new chats that include these patterns are going to be associated with the same intent. This is the method used by the AliceBot system (Wallace 2004) and similar systems. Other, more complex classifiers can be designed around this relatively simple manual model.

Another method uses statistical machine learning to generate a classifier from the training data. In this case, the researcher selects a learning algorithm and then treats the training data as a labeled data set. The training data is often preprocessed to produce more features and to reduce noise in the set. Then a classifier is 'trained' on the training data. Finally new chats are run through the system to let the

Classifier Accuracy	Correct Intent (%)	Incorrect Intent (%)
Classifier 1	67	33
Classifier 2	68	32

Table 2: Comparison of accuracy of two classifiers trained on identical training data and on identical intents. Classifier 1 is a statistical classifier. Classifier 2 is a symbolic classifier. The training data comprises $\sim 10,000$ inputs over ~ 800 classes.

classifier evaluate their intents. Intents can then be used in other parts of the system to generate appropriate responses. This paper identifies classifiers trained in this way as **statistical classifiers**. There are many references both for machine learning methods in NLP (Jurafsky and Martin 2009; Manning and Schütze 1999) and for intent learning (Li 2010).

Does Intent Classification Work?

The two methods described above produce classifiers that work reasonably well when implemented in a limited domain with not too many intents. We have conducted experiments to compare different types of intent classifiers. These experiments are not the topic of this paper, but the result is that, in general, all of these methods perform about equally.

Classifier performance can be measured in a number of ways. Our experiments used a measure of accuracy (as defined earlier) and measured the accuracy of classifiers by classifying a set of labelled test data that were not part of the training data.

Many classifiers created from very different methods, have comparable accuracy. Furthermore, each of these classifiers can be improved over time by increasing the amount of (labeled) training data and retraining the classifier. Table 2 shows the result of one such evaluation of classifier accuracy. In this case, the two classifiers do not recognize intent particularly well, but perhaps they are not tuned and thus are very quick to train. In any case intent classification is possible, though we will always expect to see some error in the evaluation process.

Looking at Table 2, the classifiers compared in this case produce almost equivalent accuracy and so in some sense are interchangeable. But this view of intent classification misses some important information which can be seen better in Table 3. In this table, we see the accuracy of the same classifiers compared based on how much their accuracies overlap. In the case of these two classifiers, on 60% of the test data, the intents returned by each of the classifiers are identical and correct. In 25% of the test data both classifier return the incorrect intent.

These two views of classifier test results raise a very interesting point. Table 2 shows us that using one or the other of these classifiers gives us approximately the same ability to recognize intent. But Table 3 suggests that, if we can select the classifier which gives a correct answer when available, we can increase the accuracy of our combined classifier to 75% in the optimal case. We next discuss methods to help

Classifier Overlap	Classifier 1 correct	Classifier 1 incorrect
Classifier 2 correct	60%	7%
Classifier 2 incorrect	8%	25%

Table 3: Comparison of the overlap of accuracy between the two classifiers in Table 2 trained using the same training data over the same set of intents.

determine if it is possible to improve the performance of a system by combining multiple classifiers.

Combining Classifiers

The remainder of this paper discusses some of the possibilities and considerations of combining multiple intent classifiers to produce better results than using a single classifier. When one attempts to combine the results of classifiers, a number of issues come up right away. This paper will address the two following issues, the answers to which can get us all the way to a functional classifier combiner.

One major issue with combining classifiers is deciding on the domain and training data of each classifier. In the example above, each of the classifiers is trained on the same training data and the same set of classes. we call this *classifiers with overlapping domain*. This paper will present an analysis of a method to compare two classifiers overlapping domain (Kuncheva 2004).

Another important way to train classifiers is to train each classifier on different domains. The combination of classifiers trained on different domains means that each classifier knows different sets of intents. This allows the creation of more complete systems of intent classification by combining smaller domains of knowledge.

Another important consideration is to recognize that classifiers trained using different methods will typically return very different types of score information. Scores are important in classifiers because the score provides a way to rank the classifier's results so that a 'winning' class can be determined.

For example, if a classifier that is trained on K classes receives an input, the classifier will typically return a set of classes with a score for each class. the score is typically a floating point number, allowing the classifier to order the results from most likely to least likely. Some classifiers generate a score that is very close to the posterior probability of the class, this is not true in general however. In regular expression-based text classifiers, for example, the score is often some multiple of the number of characters or tokens matched for the candidate class.

Given the variety of different scoring methods possible, how can one hope to compare results of different classifiers to provide the hoped-for improvement in intent matching?

Overlapped Classifiers with Naive Bayes Combination

In order to compare classifiers in a general way, we require a means to either normalize the scores of each classifier so that they can be compared, or to explore a way to ignore the scores but still rank the classifier results. Naive Bayes Combination is a technique to achieve the latter. The principles and theory behind Naive Bayes classification can be found in (Duda, Hart, and Stork 2012).

Consider a system of L classifiers $\{\phi_1, \phi_2, ..., \phi_L\} = \Phi$ trained on the same test data and the same set of K classes. Given an input x each ϕ_i will output a class in $\{c_1, c_2, ..., c_K\} = C$. Let

$$\Phi(x) = \begin{bmatrix}\phi_1(x) & \phi_2(x) & \cdots & \phi_L(x)\end{bmatrix}$$

be the vector of all the classes returned by the system given input x. For example, for a given x, $\Phi(x)$ might return the vector

$$\Phi(x_{example}) = \begin{bmatrix}c_2 & c_4 & c_1 & c_4 & c_4 & \cdots & c_1\end{bmatrix}.$$

What we want is to discover an expression to derive the most probable class given all the results in $\Phi(x)$ for any x. To do this, we first use the Naive Bayes assumption that each entry in the results in $\Phi(X)$ are independent of any particular class c_i. This lets us calculate the joint probability of any group of results as the product of the probabilities of each conditional probability result. This is the conditional probability assumption of Naive Bayes.

Let $P(\Phi(x)|c_i)$ be the conditional probability the $\Phi(x)$ is returned given that the correct class is c_i. Then the Naive Bayes assumption says:

$$P(\Phi(x)|c_i) = \prod_{j=1}^{L} P(\phi_j(x)|c_i)$$

The posterior conditional probability of class c_i given the classifier results is then:

$$P(c_i|\Phi(x)) = \frac{P(c_i)P(\Phi(x)|c_i)}{P(\Phi(x))}.$$

Using the conditional independence assumption describe above, we can now express the support for each class as:

$$\mu_i(x) \propto P(c_i)P(\Phi(x)|c_i) = P(c_i)\prod_{j=1}^{L} P(\phi_j(x)|c_i)$$

If we calculated the confusion matrix for each of the L classifiers during the testing phase, then we have everything we need to determine this equation for each $\mu_i(x)$.

Using The Confusion Matrix to Select a Class

In a previous section, the confusion matrix CM_k for classifier ϕ_k was defined to be the matrix which sums the number of test data which were assigned class c_j when the correct class was c_i for $1 \leq i \leq K, 1 \leq j \leq K$. This generates a $K \times K$ matrix of integers. Then $CM_k(i, j)$ is the cell containing the number of test datum where ϕ_k generated c_j but c_i was the correct class. The diagonal of CM_k has all the test results where test data was correctly classified.

Using CM_k we can estimate

$$P(\phi_k(x)|c_j) \approx CM_k(i, j)\frac{1}{N_i}$$

where N_i is the total number of test inputs whose correct classification is c_i. We can also estimate $P(c_i) \approx N_i/N$ where N is the total number of test inputs.

To arrive at a class from the K classifiers, we now only need to calculate

$$\begin{aligned}\mu_i(x) &\propto P(c_i)\prod_{j=1}^{L} P(\phi_j(x)|c_i)\\ &\approx \frac{N_i}{N}\prod_{j=1}^{L} CM_k(i, j)\frac{1}{N_i}\\ &\propto \frac{1}{N^{L-1}}\prod_{k=1}^{L} CM_k(i, j)\end{aligned}$$

for each $\mu_i, 1 \leq i \leq K$. The μ_i with the highest proportional value will determine the class.

Example: Naive Bayesian Estimation

Suppose that we have two classifiers trained over the same data and tested on 20 test inputs. There are three classes, c_1, c_2, c_3 ($K = 3$) and the number of test questions that are labeled with each class are $N_1 = 8, N_2 = 5, N_3 = 7$ respectively. The two classifiers ϕ_1, ϕ_2 have respective confusion matrices:

$$CM_1 = \begin{pmatrix}5 & 2 & 1\\ 2 & 3 & 0\\ 0 & 3 & 4\end{pmatrix}, CM_2 = \begin{pmatrix}4 & 2 & 2\\ 1 & 4 & 0\\ 1 & 1 & 5\end{pmatrix}.$$

Given an input x suppose that $\phi_1(x) = c_1$ and $\phi_2(x) = c_2$. we can use the technique described above to decide which class is correct.

$$\begin{aligned}\mu_1(x) &\propto \frac{1}{8} \times 5 \times 2 = 1.25\\ \mu_2(x) &\propto \frac{1}{5} \times 2 \times 4 = 1.6\\ \mu_3(x) &\propto \frac{1}{7} \times 0 \times 1 = 0\end{aligned}$$

in this case the class with the best support is c_2.

Smoothing

One problem with the method described above is that zero entries in the confusion matrix can cause values to go to zero. Since for large values of K the confusion matrix is sparse (most of the entries are zeros), this technique is not very robust by itself. There are a number of ways to introduce smoothing into the system (Titterington et al. 1981). We will look at one simple method for smoothing zeros by including an additive element of the inverse of the number of classes K (Kuncheva 2004).

To accomplish this we rewrite the Naive Bayes combination as:

$$P(\Phi(x)|c_i) \propto \left\{ \prod_{k=1}^{L} \frac{CM_k(i,j) + 1/K}{N_i + 1} \right\}^B$$

where B can be taken as 1 or less.

Repeating the above example with this new formula, setting $B = 1$:

$$\mu_1(x) \propto \frac{8}{20} \times \left(\frac{5 + \frac{1}{3}}{8+1} \right) \times \left(\frac{2 + \frac{1}{3}}{8+1} \right) \approx 0.55$$

$$\mu_2(x) \propto \frac{5}{20} \times \left(\frac{2 + \frac{1}{3}}{5+1} \right) \times \left(\frac{4 + \frac{1}{3}}{5+1} \right) \approx 0.42$$

$$\mu_3(x) \propto \frac{7}{20} \times \left(\frac{0 + \frac{1}{3}}{7+1} \right) \times \left(\frac{1 + \frac{1}{3}}{7+1} \right) \approx 0.02$$

Note that two things happened here. One is that the zero entry now has some weight. With sparse confusion matrices this prevents all the values from disappearing from the naive Bayes test.

The other notable result from this smoothing method is that the answer changed to class c_1! This is a common consequence of smoothing. The smoothing method used here is *Uniform Smoothing* where we borrow some probability mass from all the nonzero masses and redistribute them to remove zeros. More complex smoothing methods can be devised to have less direct affect on outcome. For examples of more complex smoothing methods in probabilistic natural language models see (Dumoulin 2012).

Conclusion

This paper has described a theoretical framework for generalized intent classifiers and a method for combining them. Currently Next IT researchers are working on building a framework for training and running a 'Panel of Experts' utilizing the mathematical framework outlined here to compare classifier results. The goal is to compare classifier outcomes with each other and with human judgement (yet another kind of classifier). These results have yet to be assembled but we expect to study the panel and use it to improve understanding of intent classifiers and their combination.

Acknowledgments

The author would like to thank Ian Beaver and Cynthia Wu for their original investigation of this topic and their work in investigating effective machine learning methods for intent classification. Thanks also to Mark Zartler and Tanya Miller for their work on advancing our understanding of symbolic classifiers. Thanks also to Next IT Corp. for its sponsorship of this fascinating research area.

References

Chakrabarti, C. 2014. *Artificial Conversations for Chatter Bots Using Knowledge Representation, Learning, and Pragmatics*. Ph.D. Dissertation, University of New Mexico.

Duda, R. O.; Hart, P. E.; and Stork, D. G. 2012. *Pattern classification*. John Wiley & Sons.

Dumoulin, J. 2012. Smoothing of ngram language models of human chats. In *Soft Computing and Intelligent Systems (SCIS) and 13th International Symposium on Advanced Intelligent Systems (ISIS), 2012 Joint 6th International Conference on*, 1–4. IEEE.

Jurafsky, D., and Martin, J. H. 2009. *Speech & Language Processing 2nd Ed.* Prentice Hall.

Kuncheva, L. I. 2004. *Combining pattern classifiers: methods and algorithms*. John Wiley & Sons.

Li, X. 2010. Understanding the semantic structure of noun phrase queries. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, 1337–1345. Association for Computational Linguistics.

Manning, C. D., and Schütze, H. 1999. *Foundations of statistical natural language processing*. MIT press.

Ripley, B. D. 2007. *Pattern recognition and neural networks*. Cambridge university press.

Titterington, D.; Murray, G.; Murray, L.; Spiegelhalter, D.; Skene, A.; Habbema, J.; and Gelpke, G. 1981. Comparison of discrimination techniques applied to a complex data set of head injured patients. *Journal of the Royal Statistical Society. Series A (General)* 145–175.

Wallace, R. 2004. From eliza to alice. *Article available at http://www. alicebot. org/articles/wallace/eliza. html*.