Classifying Restaurants with Yelp Photos

mike yu
Stanford University
myu3@stanford.edu

Emma Marriott
Stanford University
em070394@stanford.edu

Aaron Zweig
Stanford University
azweig@stanford.edu

Abstract

In this paper, we tackle a Kaggle challenge: using Yelp photos, provided by users, to label the businesses they represent. This problem involves image recognition and multilabel classification. While our approach to image recognition is extremely standard and vanilla – involving training a relatively small convolutional neural net on the dataset – we strive to try a novel approach to multilabel classification, using a single softmax classifier to pick out multiple labels. To do this, we designed and implemented a new way to use the probabilistic interpretation of a softmax vector, and a set of loss and test functions to examine our results and train the net. Unfortunately, results were not very promising, and we consider possible reasons for this, as well as some characteristics of other datasets that might be better suited to this approach.

1. Introduction

A current area of active product research in industry – at Yelp in particular – is the use of user provided images to classify businesses, to enhance their product's ability to make recommendations and position it closer to a media-based rather than text-based product.

More specifically, the challenge as presented to Kaggle includes a dataset of images, image-to-business mappings, and business-to-class mappings, where some businesses have multiple images. [1]

To do this, we will be using a single convolutional neural net – with a single classifier – to determine which of Yelp's traits each business pictured has. This, in addition to having a very real world, consumer-facing application, we attempted as multiclass labeling with just one classifier, which is a novel problem (virtually every multilabel classification solution currently involves using a binary classifier for each label, or, if there are too many labels, a binary classifier for each of several dimensions, where the dimensions describe an approximation of the label space). The dataset is available on Kaggle.

To expound a bit on how we use a single classifier on multilabel problems: we will be attempting to fit a softlabel vector produced by a softmax function (intuitively, the approximate probability distribution over classes of a picture) to the "ground truth" vector for an image, which is a vector x where x_i is $1/n$ if the image is of class i and 0 otherwise, and where n is the total number of classes in the image.

2. Background / Related Work

There is significant literature with respect to multilabel classification problems, and we examined much of it in depth. It is often used to classify textual documents, media, and genomics. Formally, the problem involves associating each training example, or image in our case, with a set of labels Y in the set of possible labels L, whereas single-label classification, which is far more common, only asks for the selection of a single label l from the set of labels L. Tsoumakas et. al in "Mining Multi-Label Data" consider two main ways to transform the problem into one solvable with single-label classifier methods. Label powerset transforms set L into the powerset of L, which becomes a set of size $2^{|L|}$ which includes every possible combination of labels a training example could have. Another approach is called binary relevance, and it is a primary mechanism in multilabel classification. BR involves training a separate classifier for each label. [2] There are numerous techniques that need to be employed when examining labelsets with very high dimensionality, making it impractical to either use the label powerset (which takes on massive dimensionality, far larger than the training set sometimes), or binary relevance, as the computing power demanded to train so many classifiers is just too much.

For the Label Powerset (LP) method, the primary solution is pruning, or simply throwing out as viable classes those that appear below a certain threshold of times in the data. Sometimes, if a lot of pruning is done, we will substituted removed classes and those examples' labels with strict subsets of the true set of labels, as long as those strict subsets are "frequent." [3]

For Binary Relevance, techniques such as PCA are often used to reduce the dimensionality of the label space (allowing researchers to train a different classifier for each dimension vs. each label) if there are more labels than feasible. [2]

In addition, current multi-label methods exploit various aspects of label structure. For instance, if some labels are hierarchical (or an item can be labeled as class *X* only if it is also class *Y*), then a tree of binary classifiers can be built, with non-root nodes (i.e. the classifier for class *X*) only being trained or use to classify if the data point is of class *Y* already. [4]

Another way that current literature looks at taking advantage of label structure is by looking at label-label correlations. For instance, Yu et. al roughly use the neighbors of a data point in high dimensional space, in conjunction with both global and local label correlations (label correlations over the whole set, as well as over the "neighborhood" of the data point) to predict labels, incorporating label-label correlations when computing the probability of a test instance *x*. [5]

Finally, a new set of testing methods and evaluation metrics have to be developed for multi-label classification problems as well – the percentage of time one guesses "the right class" is no longer an acceptable metric as it is for single-label problems. While "classification accuracy" does exist for multi-label problems, it is exceptionally strict in that guessing "the right class" involves predicting the exact set of labels that correspond to that data point in the "ground truth." Rather, there are a number of options, such as distance between sets (between the true set of labels, and the guessed set) in the form of Hamming loss, or roughly the average symmetric difference between sets. [6] There are also measurements of "precision" and "recall," or the number of true positives picked in the guessed set, divided by the number of positives overall in the ground truth, or the number of guessed labels, respectively. Finally, accuracy is the average over data points for the size of the intersection between the truth and guess, divided by the size of the union between the two sets. [7]

3. Approach

To attack this problem, we first needed to decide how to approach image feature extraction. It seemed relatively obvious that we should use a convolutional neural net, and while we started with 3 layers, with the hope that they would achieve reasonable results and be much faster to train to test new approaches. While the results could probably benefit from more layers, using 3 layers allowed us to do significantly more in terms of experimentation, both with hyperparamters, and various strategies for initialization or loss functions. Furthermore, our results have yet to be promising enough to warrant the extra time investment to do more than 3 layers – if 3 layers is not much better than random, it seems better for both results and learning to try different approaches to 3 layers than just add more layers.

A much more interesting problem to us than feature extraction, though, was the multilabel aspect of the problem. (We've done plenty of convolutional net training on the homework.) In particular, it seems weird that multi-label classification requires an asymptotic jump up in computing power from single-label; the LP method demands an exponential amount of space in the number of labels, and the BR method demands a linear number of models in the number of labels. Accordingly, we tried our hands at designing a mechanism by which we could use a single classifier to endow each training example with multiple labels.

To do this, we started with the softmax loss function. Recall that the softmax loss function returns a vector v of size $|L|$ for each image X, where each element v_i can be roughly interpreted as the probability that X is of class i. Unfortunately, this is designed for single class instances, so the vector v is a probability distribution, so the sum of all v_i is 1, and the terms are not independent.

Regardless, we thought that using this intuitive definition could be used to approximately push the softmax probabilities to our "ground truth" definition for each image X, which we proceeded to define as v^* as follows: If image X belongs to n classes, $v^*_i = n^{-1}$ if image X is of class i, and $v^*_i = 0$ otherwise.

Then, we had to design an objective function to minimize such that our neural net would try to push the softmax probabilities vector towards our definition of v^*_i for each image. To do this, we picked the Kullback-Leibler divergence, or KL divergence. This can, for discrete distributions as we have here, be roughly interpreted as the expected log-difference between two probability distributions. Formally, the KL divergence of two discrete probability distributions, P and Q, is:

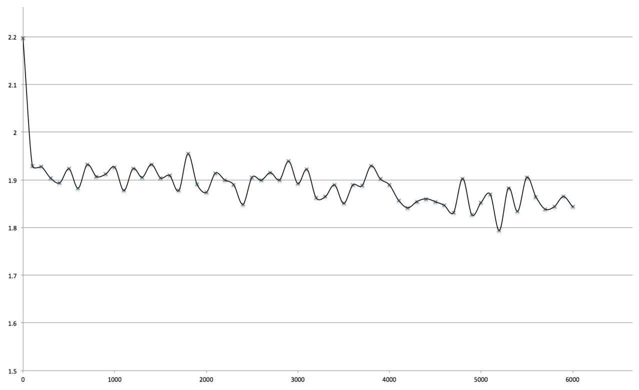
$$D_{\mathrm{KL}}(P\|Q) = \sum_i P(i)\,\log\frac{P(i)}{Q(i)}.$$

This function is continuous and differentiable, and thus the gradient is easy to backpropagate through the neural net. By minimizing this for our ground truth vectors, it's clear that we always want to increase *Q(i)* (which is v_i) for values of *i* for which *P(i)* is positive, holding all else equal. This naturally means we take away probability from those values of *i* for which *P(i)* is zero; that is, we try to push the learned probabilities up for classes which do correspond to the image, and down otherwise. Kullback-Leibler divergence does have its flaws for this use-case, with a major one being that it fails to attempt to equally distribute the "leftover" probability (that probability that is distributed to values of *i* for which *P(i)* is *0*). This is because all of those probabilities have no direct weight in the KL divergence, and this seems like it could possibly create worrying local minimum effects by allowing the algorithm to fit v^*_i in weird ways that don't necessarily make intuitive sense, such as, to, say, put all of the "leftover" probability on one class to get a marginally lower KL divergence.

After designing this approach, finding a convolutional neural net framework that might support it was nontrivial. After significant exploration, we settled on mocha.jl, a framework involving GPU implementations in Julia, which fortunately implemented most of the training for us (but none of the testing), provided we could get the data in the right format. In particular, their "softlabel" returns the softmax probabilities vector without computing a loss, and allows us to compute a custom KL-divergence loss against the ground truth we set when formatting the hdf5 database.

Our objective loss function (or KL-divergence) seemed to work reasonably well; an example from an early training instance (not with prior initialization) is graphed below

Chart 1: KL-Divergence over Training



It is slightly worrying how quickly the algorithm seems to plateau, and this seems to happen for almost any reasonable learning rate. It's also very hard to interpret this over thousands of data points – it loosely translates to an expected log difference in probability distributions, but the expectation over many datapoints of a ratio (or the log of one, as a log difference is) makes very little statistical sense, so it's very difficult to say how significant an improvement 1.9 is over 2.2.

4. Experiments

We used Yelp's dataset, as provided on Kaggle. We started with a rather naïve implementation, in which we simply used a three-layer convolutional net, which we trained from scratch, on the images cropped and compressed into 64x64. (Unfortunately, our AWS instance didn't seem to have the memory to run 256x256.) We loaded these into an hdf5 data file with the corresponding ground truth labels, v^*, as described in Section 3. We then trained the net, and then took a snapshot of the net every 500 iterations. We used these snapshots to make a pass over the validation set, get softmax probability vectors for each validation example, and predict the classes for the validation set, comparing them to the ground truth.

A major point of difficulty is converting the softmax probability vector into labels, mainly because it doesn't tell us how many labels to pick. Logically, it was clear that we should pick the top *m* labels based on probability, but it's unclear what value *m* should take. We first tried picking the smallest *m* such that the sum of the probabilities of the *m* classes chosen was at least *k* for some cutoff *k*. The first graph we printed was with *k=0.5*.

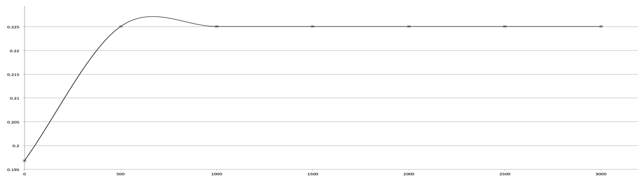
Our evaluation metric was the mean f-1 score, which measures the ratios of precision (true positives to all positives) and recall (true positives to all returned positives) in equal measure; it seems obvious that both precision and recall are key to a good model - a good model should avoid both false positives and false negatives. In particular, targeting exactly one of these properties at a time would be trivially easy; we could achieve very high precision by only choosing the highest probability label as our guess, and we could achieve perfect recall by guessing every label regardless of the probability distribution. The f-1 score is an accepted metric for evaluating high precision and high recall in a single value.

More formally, the mean f-1 score, as presented on the Kaggle challenge, is:

$$F1 = 2\frac{p \cdot r}{p + r} \quad \text{where} \quad p = \frac{tp}{tp + fp}, \quad r = \frac{tp}{tp + fn}$$

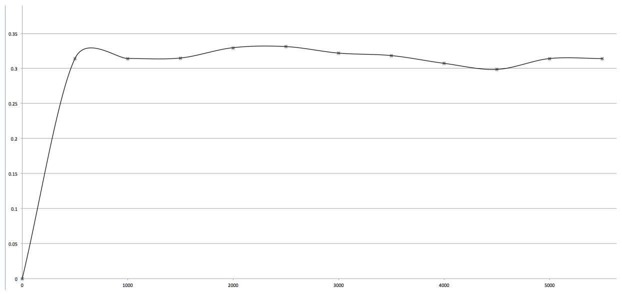
where *tp* refers to the number of true positives (or classes predicted to be labels for image *X* by both vectors *v* and *v**), *fp* to the number of false positives (or classes predicted to be labels for image *X* by vector *v* but not *v**), and *fn* to the number of false negatives (or classes predicted to be labels for image *X* by vector *v** but not by *v*). [1]

Chart 2: Mean F-1 Score Test: Cumulative Probability Method, k=0.5



As can be seen, this learned very quickly and delivered an F-1 score around .225. The F-1 score seems incredibly low (if you have as many true positives as false positives as false negatives, you will have F-1 score .5), which is extremely concerning. The first potential problem is the cutoff strategy (there also definitely is a local minimum, but even a local minimum ought to have F-1 score higher than this. We accordingly tried employing another cutoff strategy, instead picking any class with individual v_i at least *k* for some cutoff *k*. With some hyperparameter testing, we opted for $k=(9-1)^{-1}=.125$. The graph for this process follows:

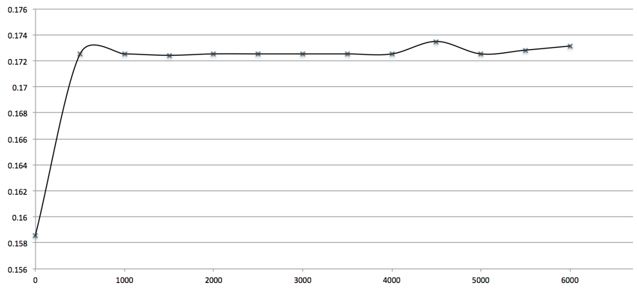
Chart 3: Mean F-1 Score Test: Individual Probability Method, k=.125



This achieves an F-1 score of around .37, which is significantly better, but still not very good.

We decided to try one more metric for choosing *m*, or the number of classes to include in our "guessed set." This metric involved taking the reciprocal of the highest probability in the softmax probability vector *v*, and taking the ceiling of this reciprocal as *m*. The intuition seems reasonable: we expect that, if the learned softmax probability vector approximates the ground truth well (that is, if *v* is a good approximation for *v**, which is the definition of our algorithm learning well based on our KL-divergence loss function), then we expect the maximal value of v^*_i to be around *1/n*, where *n* is the number of classes that apply to the image. We expect some noise, so we expect that the maximal value of v^*_i will be slightly higher than *1/n*, and so taking the reciprocal and taking the ceiling should be a good approximation for *n* if our net learns well. Unfortunately, the F-1 score was abysmal:

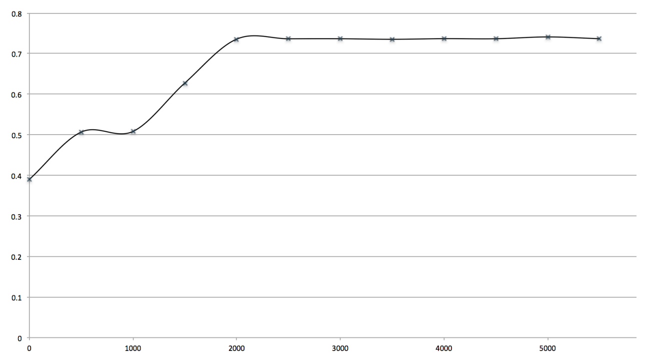
Chart 4: Mean F-1 Score Test: Ceiling of Reciprocal of Max Probability Method



This F-1 score was the worst yet, topping out just under .175. Around here, it starts to get concerning that the problem may be the learning algorithm or the overall design, and not necessarily the step where we select how many classes to guess. From here on out, we chose to continue to use the second method we tried, with individual probability cutoffs of *k=.125*.

At this point, we decided to sanity check our process (and make sure our softmax process was at least learning something approximately correct) by taking a look at the accuracy if we set *n=1*; that is, if we took only the top probability (treating softmax as a single classifier as it was originally intended) and evaluated the percentage of examples for which the maximal v_i actually referred to a class *i* that belonged to the example. This is the inverse of what is known as one-error in multi-label problems; this graph over training iterations is below:

Chart 5: Accuracy Metric: (probability that the top softmax label is a true positive)



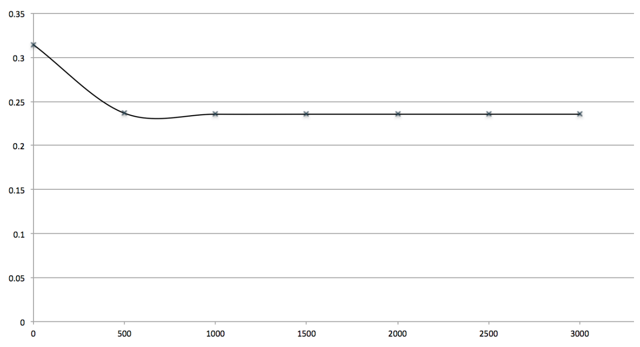
As can be seen, this seems to take longer than the F-1 score to train, which seems weird – all the metrics should train roughly in line. However, it does make it up to around .75, which is comforting – considering that we are not training a single-label classifier with conventional objectives, it doesn't seem like an unreasonable outcome that our top class is accurate around 75% of the time.

To try and solve the local minimum problem, we decided to initialize our vector with a prior on the probability vector. We did this by taking a random sample of 10,000 data points, computing the probability that an image X fit into label i for each i, and then normalizing this set of unaffiliated probabilities into a probability distribution to fit it into our softlabel vector formulation. We made this the "ground truth" for these 10,000 points, and trained the net for 10,000 iterations with these points as our "initialization."

The intuition behind this initialization was to produce a very small instance of transfer learning for our net. Rather than starting with randomly initialized weight vectors and guessing a nearly uniform probability distribution for every image, we concluded we could acquire better results in fewer iterations by initializing the weights so that the initial guess for average image was the average probability distribution.

Unfortunately, looking at the graph, it looks like the prior initialization may have put us in a position to overfit to the training set a bit. After just 3000 iterations of training on the training set from the initialized net, this is what we saw:

Chart 6: Mean F-1 Score Test: Initialized with Sample Prior



Training the priors on the training set, and then training on the training set to separate out the images in it, seems to have caused the second step to see some overfitting, and the training actually caused our net to perform worse on validation data! It was also worse than not having the initial prior, dropping to an F-1 score of around .24.

At this point, it seems possible that our softmax probability vectors aren't learning or separating as well as we thought, and therefore that the vectors v are very "noisy" representations of v^* – but that they are still viable representations. There are two possible tactics to try and remedy this problem: one is to increase the number of layers in the net, which we did to 4 (but not longer for training time constraints), and the other is to try averaging a number of guesses v at the same v^*; our dataset provides a very natural way to do this by providing multiple images for each business.

To aggregate the vectors v for a group of images that represents a single business (call the group of vectors corresponding to a group of images that represents a single business B), we tried two approaches:

1. We allowed, for all v in B, each v to cast a "vote" for each label i for which v_i exceeded a threshold k_i (similar to what we did when we picked classes with only image, except that instead of that image being labeled with that class, that image instead "votes" for that class). Then, for each class i which received more than $|B|*k$ votes for that class (the voting threshold, which is some percentage of the number of available votes), we labeled that business with that class. Consider this Approach A.
2. We computed v^B as the sum of all v in B, divided by $|B|$ (this is analogous to computing the average v in B). We then labeled the business (and accordingly, all images in the business) with

classes i for which $v^B_i > k*max_i(v^B_i)$ for some threshold k. Consider this Approach B.

The graphs for the final F-1 scores, after 10000 iterations, picking "guessed sets" of labels with Approaches A and B, for various levels of k on training and test sets, are below:

Chart 7: Approach A with k_i = .125 for varying levels of k

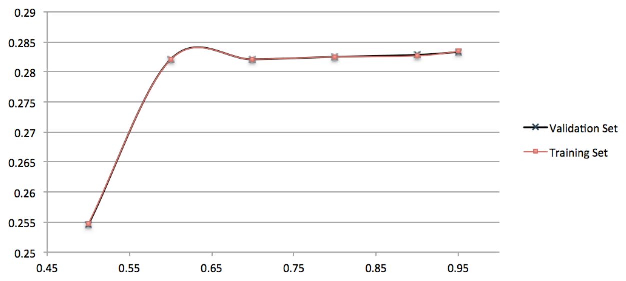
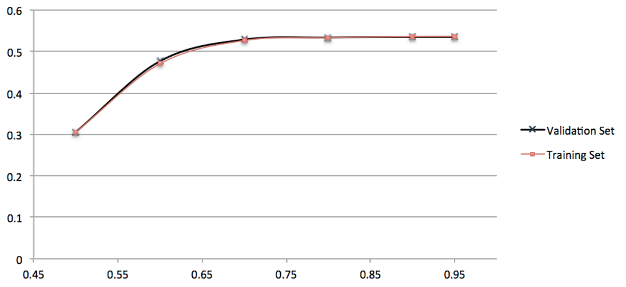


Chart 8: Approach B for varying levels of k



It can be seen that Approach B substantially improves our F-1 score (which makes sense, considering that Approach B seems to help with removing the noise significantly more than Approach A even from an intuitive standpoint – Approach A only removes noise when it is large enough to change a "vote"). We top out with Approach B's F-1 score of around .535. While this is still not great compared to Kaggle's leading .82 (which we believe comes from a decision tree – we optimized for an interesting approach vs. winning Kaggle), it does show that this approach might not be totally useless on a different dataset better optimized to this approach, and that our net did learn something, significantly beating Kaggle's random guess benchmark of around .43. [1]

Also interesting is that the accuracy on the training set is almost identical to that on the validation set; this suggests that the model, even with 4 layers, generalizes too well, and we could add more.

5. Conclusion

Overall, it seems that a single classifier approach to the multi-labeling problem, at least as we attempted it here, was relatively ineffective. This is very unfortunate and disappointing, but not that surprising – after all, it seems weird to try and push the "strength" of independent classes into a probability distribution over them, effectively making the final term entirely linearly dependent on the others. Perhaps next time a dummy class with label always false would be helpful in trying to ensure that the "class strengths" are less collinear.

A major problem is that the ground truth for, say, "takes reservations" (class 2) should be of value x for every image of a restaurant that takes reservations, where x is independent of the number of classes that image belongs to – however, this is not the case in our model. This is imposed by the requirement that the sum of the ground truth vector be exactly 1, regardless of the number of classes associated with an image; this means that every image for businesses that "take reservations" will not have the same "strength" for this class – if business X takes reservations, but is also good for lunch, good for dinner, serves alcohol, and is good for kids, it will see only *1/5* as "reservation-taking" as one that takes reservations but doesn't fit into any of those categories.

In other words, our objective function is too punishing towards learning from images with many labels. In general, it is bad that we had to normalize the "class strength" of an image X for label i for reasons unrelated to the direction relationship between X and i, but rather because of the relationships between X and other labels j.

In general, it seems that the fundamental concept we were playing with, that of using a single softmax classifier to solve a multilabel problem without expanding the label space to the label powerset, was not well tailored to this dataset at all.

We suspect that this method of using a single classifier might find more success if applied to different type of dataset. In the Yelp dataset, generally different labels were not mutually exclusive, as for example, good for kids is not necessarily correlated with good for lunch, and in some cases, there might exist significant positive correlations between labels; one imagines, for instance, that "is expensive" is positively correlated with "takes reservations." A dataset in which the labels were in direct competition, or at least strong negative correlations might exist, might be better suited to being modeled using these probability distributions.

References

[1] Yelp. Kaggle Challenge. https://www.kaggle.com/c/yelp-restaurant-photo-classification

[2] Tsoumakas, G., Katakis, I., and Vlahavas, I. Mining multilabel data. In Maimon, O. and Rokach, L. (eds.), Data Mining and Knowledge Discovery Handbook, pp. 667–685. Springer, 2010.

[3] Read, J.: A pruned problem transformation method for multi-label classification. In: Proc. 2008 New Zealand Computer Science Research Student Conference (NZCSRS 2008). (2008) 143–150

[4] Cesa-Bianchi, N., Gentile, C., Zaniboni, L.: Incremental algorithms for hierarchical classifi- cation. Journal of Machine Learning Research 7 (2006) 31–54

[5] Yu, Y. Pedrycz, N. Miao, D.: Multi-label classification by exploiting label correlations, Expert Syst. Appl. 41 (2014) 2989–3004.

[6] Schapire, R.E. Singer, Y.: Boostexter: a boosting-based system for text categorization. Machine Learning 39 (2000) 135–168

[7] Godbole, S., Sarawagi, S.: Discriminative methods for multi-labeled classification. In: Proceedings of the 8th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2004). (2004) 22–30