A Maximum Entropy Classifier for U.S. Census Data

Jonathan Bisila & Daniel Lewitz

March 15th, 2017

Introduction

For our final project, we built a maximum entropy classifier to predict an individual's income given some set of features about them. We expected that by choosing relevant features, we would be able to train a model that can predict better than a baseline. We obtained a dataset here of census data, with each instance recording demographic characteristic about an individual, such as age, education, marital status, etc, and a label of whether that person made \$50k/year. We aimed to train using generative iterative scaling on our training data, which had about $30,000$ instances, then test our model on testing data.

Given the large amount of data available for each instance, we had to decide which data we wanted to use, as well as how we wanted to encode it. One issue in identifying which features are good predictors comes when we attempt to test our model by having it classify a vector by whatever class has greater probability. Before training, when weight vectors are all 1, all instances are assigned the same label. If this label is $\leq 50k$, then the model is already correct approximately .75 of the time, so in many cases, additional training either does not improve or only marginally improves the model with respect to its overall classification rate.

Methods

Maximum Entropy and GIS

A maximum entropy model is a means of classifying data given some set of features. This model is trained by taking in a series of feature vectors along with the true binary labels. In our case, the feature vectors were generated by our program, and the labels were $\leq 50k$, referred to as class 0 and $>$ 50k, or class 1. In general, feature vectors can take on any values. In our model, feature vectors are composed of 0s and 1s, depending on whether that feature is active.

From the training data, it develops two weight vectors w_0 and w_1, with w_i corresponding to class i. To predict the label of a feature vector $\phi(x)$, the model predicts that its probability distribution is $p(c_i|x) \propto e^{w_i \cdot \phi(x)}$. Our model predicts x to be in the class that has greater probability.

For developing the correct weights, we used a technique known as generative iterative scaling. (GIS) For GIS, we need to first calculate the empirical probability that any one of our training instances is of label c_j, and has active feature f_i. This is accomplished by iterating over the training instances and a list of the true labels.

Now, to determine the accuracy of our weights, we calculate $model(i,j)$, the model's prediction about the probability that any training instance is from class j, and has feature i. Since the model does not know the true labels, this is determined by taking the probability distribution over the classes given the current weights. From this, we obtain $model(i,j) = \frac{1}{N}\sum_n p(c_j|x^{(n)})\phi(x^{(n)})_i$, where $x^{(n)}$ is a training instance, N is the number of training instances, and $\phi(x^{(n)})_i$ is the value of feature i.

To perform GIS, we begin with weight vectors w_0, w_1, both initialized to vectors of length F and composed of all 1s, where F is the number of possible features. We can calculate the empiricals $emp(i,j)$ for every feature i and class j. Since these empiricals are based off the true classes of the training instances, we need only do this once. Then, in each iteration, we can calculate the models predictions $model(i,j)$, given the current set of weights. We then adjust the weights depending on the disparity between the empirical probabilities, and the model's probabilities. Specifically, we adjust:

$$w_j[i] = w_j[i] * \left(\frac{emp(i,j)}{model(i,j)})\right)^{\frac{1}{V}}$$

Where V is the sum of the entries of each feature vector. In our model, this was just the number feature categories we chose, since for each category, there is a 1 for the true feature, and 0s elsewhere.

GIS repeats this process of calculating model probabilities, and adjusting the weights accordingly. It is guaranteed that GIS will converge to a set of stable weights. We initially wrote the GIS model to run until the point where, for each of the weight vectors, the entries changed by, on average, less than 10^{-6}. However, given that we were training our model a large number of times, we modified it to stop updating after 30 iterations if the weights have not converged by this point. From observing the weights updating, we determined that 30 iterations is almost always sufficient to obtain weights close to the ideals.

After the weights converged, we tested our model on the testing data. Our primary means of testing the accuracy of a set of weights was to classify each of the testing instances as the class that had the greater probability under out model. We could then calculate the proportion of instances that were classified correctly. Starting with a list of 12 possible features, we tested many different combinations of those features. Specifically, we tested all possible models with $1, 2, 10, 11$ or 12 features. This allows us to compare the effects of adding more features.

Additionally, in some cases we computed the average error of the probabilities given by our model. We define this error as the difference between the true class and the probability on that class. Logically, we encodes that we should prefer a model the gives a .9 probability of label 1 to an instance of true label 1 then a model that gives a .55 probability of label 1, despite the fact that in both cases it will classify the instance correctly.

Training and Testing Data

For our project, we took a dataset from the UCI Machine Learning Repository. The dataset included the characteristics of each individual along with the correct label. The set included 48,842 instances, with about 32,000 of those in a training data file, and the rest in a testing file. Of the given data, there were about 4,000 instances which were missing entries in one or more of the fields. We were able to ignore these instances, as the remaining data was still sufficiently large to train and test our model. We chose to ignore two fields, fnlwgt and relationship, due to fnlwgt being a field we didn't understand well enough to generate effective ranges for and relationship in order to save computation time.

Feature Generation

To generate features, we had a createVector() function, that took in an instance of our data, and returns a list of binaries corresponding to the active feature. For each feature category, we had a set of possible binary features. For categories such as workclass, marital-status, etc, we could thus create a binary feature for each of the possible values. For instance, someone who's value for workclass is "Private" would have a 1 for the "Private" feature, and a 0 for every other one of the possibilities.

For non-binary features, such as age, capital-gain and loss, and hours-per-week, we created possible features based on ranges. For age, we split into ranges 0-9, 10-19, ... 90-99. Thus, someone who is 25 would have a 1 for $20 - 29$ feature, and 0s for every other one. For capital-gain, we had a feature for > 3674, and a feature for ≤ 3674. This number was chosen as the median of the nonzero values of the capital gains.

As an example, if choose age and race as our desired features, then someone who is 25 and is of race "Other" will have feature vector $[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]$, with the first ten entries corresponding to the possible age features, and the last five corresponding to the possible race features.

Constructing our feature vectors in this way ensures that the sum of the entries in any feature vector is exactly equal to the number of features we are considering, as for each feature category, there will be a single feature that has a 1, and the rest of the features will be 0. This simplifies the use of GIS, as normally, a slack feature is needed to ensure that the sum of the features is a constant.

In addition to encoding features about the categories given in the data, we also encoded joint-features, that is, features which represented a combination of other features. For example, if we were to have a joint feature of work-class and sex, we would have a feature set composed of all pairings of sex with work-classes.

The premise of having joint features is that in some cases, there could be multiple features, each of which has, on its own, little impact on the classification. For example, it is possible that just knowing someone's sex or age would give little information about whether they earn 50k/year, yet knowing that the are female and privately employed would.

Results

We tested our program by running it on many different possible sets of features. We created models that incorporated all possible combinations of $1, 2$, $10, 11$ and 12 features. Due to time constraints, we did not get a chance to run it on more moderate numbers of features.

For each set of features, we tested the model by running it on the set of testing data included in the UCI repository. After removing instances with missing entries, this contained approximately $14,000$ entries. We recorded the rate of instances classified correctly, and the average difference between the probability distribution given by the model and the true label.

As would be expected, we noticed that, on average, using more features for a model increases overall accuracy and decreases average error. (Fig. 1 & 2) For instance, on average, running with a single feature will classify about 76.1% of the data correctly, while running with all features will classify 84.5% of the data correctly.

One of aims in the project was to use our model as a means of determining which features are good predictors of the label. We hoped that training models that only utilized a single feature could provide insight into this, despite the fact that maximum entropy models are almost always better when utilizing more features. One challenge we noticed in analyzing the effectiveness of certain features is that in many cases, adding one or two does not actually make the accuracy of the classifications better than just classifying everyone as class 0, or $\leq 50k$. We noticed that, before weights were trained, the model would classify all instances as $\leq 50k$, as a result of tie-breaking when the probability is .5. Since about .75 of the instances were from the class $\leq 50k$, it would thus classify about .75 of the data correctly.

As shown in figure 3, only 5 of the 12 possible features produce models better than the untrained model. Of these, capital-gain performs the best, classifying .79 of the data correctly. We suspect that, given the binary nature and cutoff of capital-gain, those who have the feature capital-gain > 3674 are almost certainly in the $> 50k$ label, and thus this model will correct classify some instances as class $> 50k$ that other feature models would not.

In general, though, we suspect that the issue of single features not improving the model follows from the fact that given the high prior of being in the class $\leq 50k$, an instance from any age range is more likely than not to be in the $\leq 50k$ class, even though this probability will vary depending on the age range.

Though examining the accuracy of single-feature models provides limited insight, there is a very significant difference in average error between the trained and untrained model. (Fig. 1) In the untrained model, it would always assign a uniform distribution over the classes; thus, the error would always be .5, regardless of the true class. When the model is run on either age or marital-status alone, for instance, the overall accuracy is not improved, yet the average error goes down to .388 and .296, respectively. Marital-status in fact gives us the model with the lowest average error, despite the fact that capital-gain performs several points better that it on overall classification.

As we would expect, using two different features for the model makes it perform better than using either one alone. In some cases, using two features both of which, on their own, are no better than the .754 from classifying everything as $\leq$ 50k produces a model with a higher classification rate. For instance, on their own, both age and workclass gave models classifying at a .754 accuracy, yet a model with both an age feature and a workclass feature produces a model with approximately .772 accuracy.

In addition to making models with multiple, separate features, we also examined the effectiveness of having joint features. We created joint features for every pair of the features workclass, education, marital-status, occupation, sex and race. We compared a models utilizing one joint feature to models utilizing those same two underlying features. For example, we would compare the effectiveness of having a workclass-education model, versus have a model with both a workclass feature, and an education feature. Though these produced different results, none of them were statistically significant. We observed that in this two feature comparison the difference in accuracy between using a joint feature versus using two separate features was never greater than .0015, and difference in error never greater than .0027. Furthermore, we found that training a model on two features separately in contrast to training with the separate features in addition to a joint feature of those two features resulted in no statistically significant changes in the results. For example, classifying using ”race, education” results in an accuracy of 0.77257 whereas classifying using ”race, education, race+education” results in an accuracy of 0.77224.

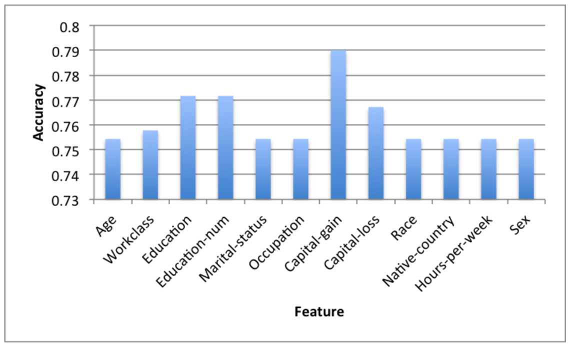

Figure 1: Accuracy of single feature models

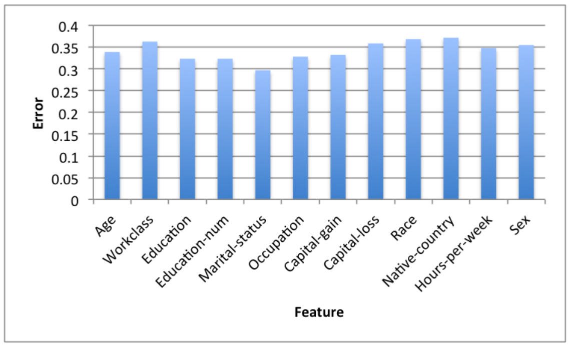

Figure 2: Error in single feature models

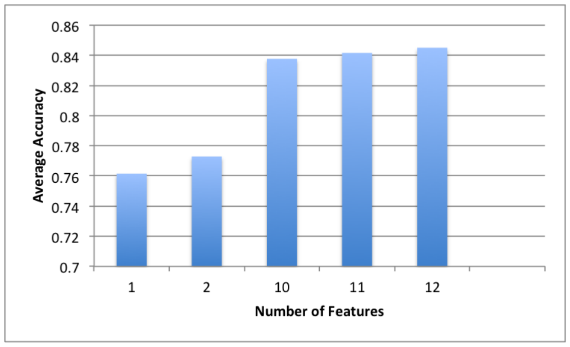

Figure 3: Number of features vs accuracy

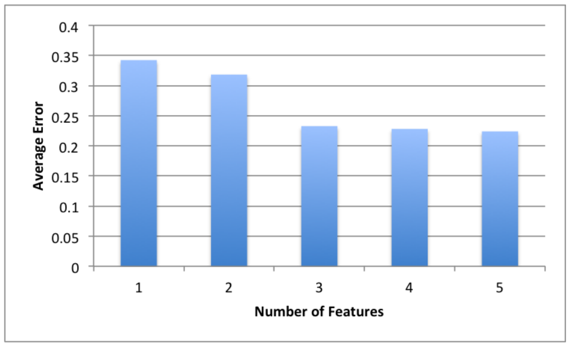

Figure 4: Number of features vs error

Discussion

For this project, we created a maximum entropy classifier to classify whether or not an individual would earn more than $50000 a year. We utilized a variety of features provided by the dataset of which we included age, workclass, education, number of years of education, marital status, occupation, capital gain, capital loss, sex, hours-per-week, race, and native-country. For this project, we focused on comparing all combinations of features for feature sets ranging from a single feature to all features. A few ways in which we might improve our solution or continue to investigate our problem in the future include creating more features, in particular, features that are class-dependent or joint features such as ”capital-gain & education”. In creating these additional features, we could investigate further the implications of joint features such as race and native-country in contrast with just including each as a separate feature in the feature vector. Alternative approaches to solving this same problem would be to use a variety of other classification techniques such as Naive Bayes, Decision Trees, or a variety of neural network inspired classification algorithms, such as perceptron, recurrent neural networks, or Bayesian networks. Similarly, our technique of maximum entropy could be and is applied to many other problems in the field of classification. If given more time for the project, it would be of interest to compare how our maximum entropy solution compares to these other classification algorithms or even just to the maximum entropy modules that have been written for libraries such as nltk, scipy, or scikit-learn.

Works Cited

- Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.
- Norvig, P. and S. Russell: Artificial Intelligence: A Modern Approach. Pearson Education, 2010.
- Stephan Dreiseitl, Lucila Ohno-Machado, Logistic regression and artificial neural network classification models: a methodology review, Journal of Biomedical Informatics, Volume 35, Issues 5–6, October 2002, Pages 352-359, ISSN 1532-0464, http://dx.doi.org/10.1016/S1532-0464(03)00034-0. (http://www.sciencedirect.com/science/article/pii/S1532046403000340)