CS229 Fall 2014 Project Final Writeup

Members: Jacob Conrad Trinidad (j3nidad), Ian Torres (itorres)

Title: Tradeshift Text Classification

Members

Members of this project are Jacob Conrad Trinidad (SUNet ID: j3nidad) and Ian Torres (SUNet ID: itorres).

Introduction

Our problem originates from the Kaggle Tradeshift Text Classification Challenge. We aim to predict the probability that a piece of text belongs in a given class. Given a feature vector that describes a text block, we must output a vector of predicted probabilities, where the i^{th} element corresponds to the probability that the i^{th} label applies to that text block. The specific meanings of the features and labels are unknown.

Dataset

The data is provided by Kaggle (found here: `http://www.kaggle.com/c/tradeshift-text-classification/data`) and is publicly available. The training set is given in train.csv, which contains 1,700,000 rows and 145 columns. Each row corresponds to a text block, and each column corresponds to a feature, which can either be boolean, numeric, or a hash value. The file trainLabels.csv contains 1,700,000 rows and 33 columns. Each row of trainLabels.csv corresponds to a text block as in train.csv, and each column corresponds to a label. If a text block belongs to a given label, it will have a 1 in the corresponding column, and 0 if not.
For instance, the beginning of an example train.csv and trainLabels.csv would look like:

train.csv	*trainLabels.csv*
id, x1, x2, x3, x4, ...	id,y1,y2,y3,y4,...
1, m268i97y, 0, NO, 105.4, 14, ...	1, 1, 0, 0, 0,...
2, j0gheu6, 1, YES, 25.631, 12, ...	2, 0, 1, 0, 1,...

The test set, given in test.csv, follows the same format as train.csv. The format of the desired output is similar to that of trainLabels.csv. For each text block, we are to write a list of probabilities that the text block belongs to each label. For instance, given a test.csv that looks like the train.csv above, we would output:

output.csv
id_label, pred
1_y1, .867
1_y2, .025
...
1_yK .061,
2_y1, .128

Features and Pre-processing

Provided in the data is a set of 145 features to describe each piece of text. What the features specifically represent is unknown, but it is given that the features include relevant data such as content, parsing, spacial,

and relational information about the text. These features come in the form of cryptographic hashes, continuous and discrete numerical values, and booleans. Through our analysis, we found there to be 50 booleans, 50 floating point numbers, 35 integers, and 10 hashes among the features. Some preprocessing was necessary to separate this data into the three categories of boolean, numeric, and hash values so that certain models could use them appropriately.

Models

We split the train.csv data set into a training set and a developer test set, with the first 70% of the data being the training set and the latter 30% of the data being the developer test set. This training and dev test set were used to test each model and determine which one would perform best. Then, upon discovering the best model, we retrained that model on the entirety of train.csv and then write the models predictions on test.csv into an output csv file.

We used four types of models in an attempt to solve the problem. In each model, we assumed that the labels are mutually independent.

Majority Algorithm

$$\hat{y}_j = \frac{1}{N_t} \sum_{i=1}^{N_t} y_{ij}$$

$N_t :=$ the number of training samples
$y_{ij} :=$ the j^{th} label of the i^{th} sample.

We used a slight variation of the majority algorithm where the predicted probability for a given label y_j is the percentage of training samples with label y_j. This model allowed us to establish a baseline score for our tests, as it did not include any information on the features.

Linear Regression

$$\hat{y}_{ij} = \theta_j^T X_i$$

$\theta_j := (X^T X)^{-1} X^T y_j$
$X :=$ matrix of samples x features
$X_i :=$ feature vector of sample i
$y_j :=$ vector of label j, where each element corresponds to the j^{th} label of each sample

This model calculated linear regression using normal equations. Due to the nature of linear regression, we only used the numerical features.

Naive Bayes with Laplace Smoothing

$$\hat{y}_{ij} = \frac{\prod_{i=1}^{N_t} p(x_i|y_j = 1)p(y_j = 1)}{\prod_{i=1}^{N_t} p(x_i|y_j = 1)p(y_j = 1) + \prod_{i=1}^{N_t} p(x_i|y_j = 0)p(y_j = 0)}$$

$N_t :=$ the number of training samples
x_i is the event that the i^{th} feature is present in the piece of text.
y_j is the event that the j^{th} label is assigned.

Due to the nature of Naive Bayes, Boolean features were the only features included in the implementation of this model.

Bag of Words
We attempted to improve our implementation of Naive Bayes by adding hash features via the usage of a bag-of-words model with binary weighting. Each unique hash that appeared at least n times in the training and test data became a boolean feature (indicated by whether or not the hash appears in a sample). The optimal value of n turned out to be $n = 17500$.

Logistic Regression

Newton's Method

$$\hat{y}_{ij} = h_\theta(x^{(i)})$$

$h_\theta(x^{(i)})$ is the sigmoid function of $\theta^T x^{(i)}$
θ is updated by $\theta \leftarrow \theta H^{-1} \nabla_\theta \ell(\theta)$
$H :=$ Hessian of the log likelihood function
$\nabla_\theta \ell(\theta) :=$ gradient of the log likelihood function

This model initially only used Newtons method to update θ. Our implementation ran until convergence. Due to the nature of logistic regression, we initially only used numerical values. We attempted to use this model due to our observations as to how well it has worked in the past.

Online Learning
The main equation is the same; however, how θ is updated changes.
$\theta_j \leftarrow \theta_j - \alpha(h_\theta(x^{(i)}) - y_{ij})$
$\alpha :=$ a constant

This model varies from the previous version of logistic regression by using online learning, stochastic gradient decent, and feature hashing. Online learning enabled us to use less memory and causes the algorithm to update the weights via stochastic gradient decent as each new sample provides more data. Feature hashing enabled us to use all of the features at once regardless of their original type by simply rehashing all of the features into buckets, enabling the algorithm to find more patterns in the data.

Adaptive Learning Rate
The main equation is the same as above; however, how α is updated changes. α now changes inversely proportional to how much θ_j has changed. An adaptive learning rate was used to speed up convergence to an optimal θ.

Results

The training set was of a size 1,190,000 samples and the test set was of a size 510,000 samples. To evaluate our model, we will be using the metric of the negative logarithm of the likelihood function averaged over N_t test samples and K labels. The equation is:
$LogLoss = \frac{1}{N_t K} \sum_{i=1}^{N_t} \sum_{j=1}^{K} [-y_{ij} log(\hat{y}_{ij}) - (1 - y_{ij}) log(1 - \hat{y}_{ij})$ where y_{ij} is the expected and $\hat{y}_{ij}$ is the predicted value of the j^{th} label on the i^{th} sample. This metric is used because its symmetric in the fact that predicting .1 for a false sample (0) has the same penalty as predicting .9 for a true sample (1). In addition, if one were to make a completely incorrect prediction, such that in the worst case one predicts 1 when the value is 0, it adds the significant penalty of infinity with the value of $log(0)$. Thus, the goal is to get our LogLoss as close to 0 as possible by getting the margin of error for each guess to be as small as possible.

Model	Training Set Score	Dev Test Set Score
Majority Algorithm	0.0943	0.1016
Linear Regression	0.0762	0.0772
Naive Bayes w/ Laplace Smoothing	0.0486	0.0498
Naive Bayes w/ Laplace Smoothing and Bag of Words	0.0478	0.0492
Logistic Regression w/ Newtons Method	0.0257	0.0291
Logistic Regression w/ Online Learning	0.0373	0.0463
Logistic Regression w/ Online Learning and Adaptive Learning Rates	0.0043	0.0098

We discovered the best model was Logistic Regression using Online Learning. We then attempted to optimize this model by finding the optimal initial learning rate α using hold-out cross validation. Here, we split the training set 70% into another training set and the remaining 30% into a cross validation set. We then trained various values of $\alpha(0.0001, 0.001, 0.01, 0.1, 1$ and then $0.07, 0.08, 0.09, 0.11, 0.12, 0.13)$ on the training set and then found their score on the cross validation set. An α value of 0.1 yielded a score of 0.00969, the lowest one across the various values tested for. We then attempted to optimize the model again by finding the optimal number of passes through the training data. We trained models based on 1,2,3,4, and up to 5 passes through the data and then found their score on the cross validation set. The model that yielded the best score was 2 online passes, as any more passes and the model appears to suffer from bias and overfit the data.

After optimizing the hyperparameters, we tested the model on Kaggle. We retrained the model on the entire train.csv file and then tested it on test.csv. After submitting the output to Kaggle, we achieved a test score of 0.0085794.

Discussion

The majority algorithm established a baseline score due to its simplistic nature. All other algorithms we implemented yielded better scores than this baseline, confirming our various improvements in our other models.

It was expected for linear regression to perform only slightly better than our baseline, due to the likely non-linearly separable features. As a result, we decided to forgo this model and attempt a new model.

Naive Bayes with Laplace Smoothing demonstrated little change between its original implementation and its usage of the bag-of-words model. This is because the optimal minimum frequency for a hash to become a feature only resulted in the addition of 3 boolean features, which may indicate that the hashes are less correlated with the labels than the other features. Changing the minimum frequency to result in the addition of more features improved the score even less. This is likely a result of the marginal gain in information due to the greater chance that a sample would have this feature on.

Heavily modifying the log regression model drastically improved the score to below 0.01. This is likely due to including all of the features as data points as well as improving the rate at which the weights are updated.

We analyzed the performance of our model by finding its precision and recall values. Since the predictions are probabilities, we can round our predictions to either 0 or 1. What we discover is that our algorithm has a precision of 97.53% and a recall of 95.91%. This demonstrates that this model has both high recall and precision.

Conclusions

Our final online learning algorithm achieved a score corresponding to the 50th percentile on the Kaggle leaderboard, and beat Tradeshift's benchmark of 0.015. As the top teams on the leaderboard used external tools and probably had access to more computing time, this was better than we expected.

The motivation for focusing on optimizing an online learning model was partly practical, as the low memory requirements enabled us to test our algorithm on our own machines, which was significantly faster than using Stanfords corn machines. For example, on the corn machines, running our initial logistic regression model using Newtons method on just 80,000 of the 1,700,000 training examples took several hours.

Future

Our final online learning model performed significantly better on the training set than the test set. Thus, a possible improvement of our online learning model would be to apply backward search or filter feature selection to reduce bias. Additionally, to improve the performance of our Naive Bayes with bag-of-words model, we could use tf-idf instead of binary weighting, which might better capture how important a hash is to a text block. To further improve our Naive Bayes model, we could discretize the numerical features so they could be included as well. Finally, besides improving the models we used, we could also attempt to implement a model using SVMs or neural networks.

References

Tradeshift Text Classification, *Kaggle*, [online] October 2014,
`https://www.kaggle.com/c/tradeshift-text-classification`, (Accessed 7 December 2014). Tradeshift Text Classification Forums, *Kaggle*, [online] October 2014,
`http://www.kaggle.com/c/tradeshift-text-classificationforums/t/10537/`, (Accessed 7 December 2014). A. Ng, Supervised Learning, Discriminative Algorithms, *Stanford*, [online] September 2014,
`http://cs229.stanford.edu/notes/cs229-notes1.pdf`, (Accessed 7 December 2014). A. Ng, Generative Algorithms, *Stanford*, [online] September 2014,
`http://cs229.stanford.edu/notes/cs229-notes2.pdf`, (Accessed 7 December 2014).