Feature Selection for Noisy Data Binary Classification

Syuzanna Sargsyan
Department of Applied Mathematics
University of Washington
Seattle, WA 98195
ssusie@uw.edu

Bethany Lusch
Department of Applied Mathematics
University of Washington
Seattle, WA 98195
herwaldt@uw.edu

1 Introduction

Nowadays, the dimensionality of collected data gets larger and larger and efficient management of this data becomes more and more challenging. The collected data usually contains irrelevant features and a high level of noise, which could result, for example, from technologies used to harvest the data. Extracting useful information from the data is an important task in machine learning, data mining and statistics. One of the methods used for this purpose is called dimensionality reduction, which could be classified into feature extraction and feature selection. The first approach constructs a new lower-dimensional feature space and projects the data onto that space, while the second approach selects a small subset of given features that will be relevant for the final goal, e.g. classification. For instance, for the classification task, this last method tries to select highly discriminant features that will be clearly different for different classes. Examples of the first approach include Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), etc., and examples of the second approach include Lasso, Fisher score, etc.

The feature selection approach itself can be categorized into "wrapper" methods and "filter" methods. If importance of features are evaluated within a learning algorithm, the method is called "wrapper". If instead, features are evaluated before applying a learning algorithm , the method is called "filter". Filter methods are developed mostly for supervised learning. We will mostly use filter methods for our project discussed below.

In this work we consider "bag-of-words" type data which consists of Reuters articles. Those articles need to be classified as related to "corporate acquisitions" or not. The data is obtained from the UCI machine learning repository:

http://archive.ics.uci.edu/ml/datasets/Dexter.

The data has only 9947 real features out of 20,000. These features represent the frequencies of occurrence of words in texts. The rest of the features are called probes (noise) and were artificially added to the data. It is also known that 2572 features out of 9947 are all zeros. We are given labels for training and validation sets, therefore we can use supervised learning techniques. There are various ways to measure errors and compare the methods, nevertheless, we will use the balanced error rate (BEF) as suggested in [1]. It is defined as:

$$BEF = \frac{1}{2}\left(\frac{\text{\# positive instances predicted wrong}}{\text{\# positive instances}} + \frac{\text{\# negative instances predicted wrong}}{\text{\# negative instances}}\right)\cdot 100\%$$

2 Support Vector Machine

Support Vector Machine (SVM) is a classification algorithm. Originally it was a binary linear classifier that constructed a hyperplane which separated two classes [2]. That hyperplane was required to maximize the margin between the two classes. However, later this algorithm has been modified so

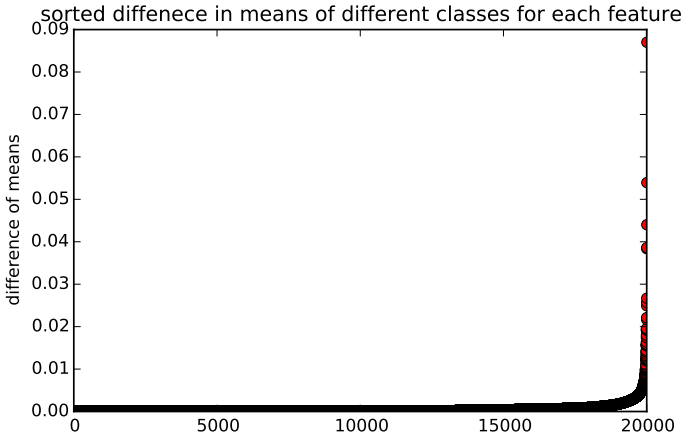

Figure 1: The difference between means of features for two classes in an increasing order.

that it can classify between more that two classes [3]. This is a widely used algorithm that has broad applications. However, one of the disadvantages of it is that it does not obtain the feature importance, i.e. it treats all features as equally important. Directly applying this algorithm to our data set may not be a wise thing to try since most of the features are irrelevant. Nonetheless, we tried this algorithm and got $\mathbf{BEF} = \mathbf{50}\%$. Here, we used the Python built-in classifier `sklearn.svm.SVC`.

3 Random Forest

Random Forest is another classification algorithm that also does random feature selection [5]. In this case, a Random Forest is a set of decision trees which are built by randomly selecting a subset of the feature set. Then, by letting those decision trees vote, prediction is done by the majority of votes. Python has a built-in Random Forest classifier: `sklearn.ensemble.RandomForestClassifier`, which we used in our code. Running the algorithm twenty times and allowing random samples to be picked from all 20,000 features, we got on average $\mathbf{15.96}\%$ error with the best result being $\mathbf{BEF} = \mathbf{10.67}\%$. However, the Python classifier has an option to limit the number of features used in each split in the tree. Making use of this option and the fact that there are only 7375 real features, we got on average $\mathbf{BEF} = \mathbf{11.4}\%$ with the best result being $\mathbf{BEF} = \mathbf{8.67}\%$.

4 Lasso

One of the most popular regression algorithms that uses feature selection is called Lasso [6]. This method uses regularization (feature selection) to avoid overfitting of training data. It penalizes the sum of absolute value of weights that describe the fitted model. As mentioned, this algorithm is used for regression, however, it can also be modified to do classification. After zeroing out some features and choosing a threshold, we could assign points to one class if the predicted value is greater than the threshold and to another class if the value is smaller. To get our results, we used `sklearn.linear_model.Lasso`, the Python built-in Lasso model. Choosing the threshold to be zero and trying different values for regularizer, the best result we got was $\mathbf{BEF} = \mathbf{9.33}\%$.

5 Linear Support Vector Classification combined with L_1 regularization

One of the Python classifiers: `sklearn.svm.LinearSVC` [7] has an option to combine L_1 regularization with classification. In particular, it uses Lasso regression method with Support Vector Machine classifier. By setting `penalty='l1'` in the mentioned function and with a suitable choice of regularizer, we got **BEF=7.67%**.

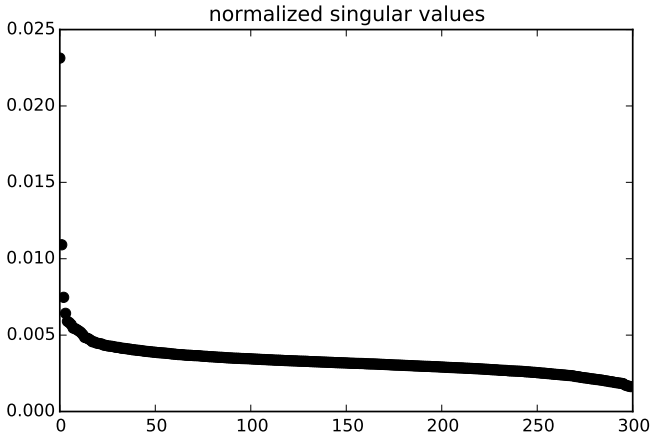

Figure 2: Singular values of the data matrix.

6 Mean of each class

So far, the methods we tried did not give us satisfactory results, therefore, we will try to make use of the additional information given, i.e. that only 9947 of features are real features. Moreover, some part of these features consists of all zeros which means they will be irrelevant for classification. The first idea we tried was the following: for each class, separately compute the mean of each feature and then compare them. If for a feature, the difference of means in two classes is small enough (smaller than some threshold), ignore that feature, otherwise consider it as an important one. After ranking the features, we applied Support Vector Machine algorithm to the data points with only the selected features and classified them. For this part, we have preprocessed the data by rescaling each feature (normalized) to be in $[0, 1]$ interval. Figure 1 shows the difference of means in ascending order. By taking the threshold to be 10^{-5}, our approach chose 7722 relevant features. We applied the SVM on this subset of features and got $\mathbf{BEF} = \mathbf{39.67}\%$, which was still bad. However, applying Random Forests on these features, we got around $\mathbf{BEF} = \mathbf{11.33}\%$. The best that Lasso gave us with these features was $\mathbf{BEF} = \mathbf{9}\%$. Changing the value for the threshold did not actually significantly improve the accuracy.

7 Singular Value Decomposition

As mentioned in Section 1, another method of doing dimensionality reduction is feature extraction. One of the most popular methods for feature extraction is the singular value decomposition (SVD), which projects the data into a smaller feature space [8]. In our case our data matrix is 300 by 20,000 dimensional so if we compute the singular value decomposition and keep all resulting features, we will have at most 300 of them. Fig. 2 shows energy in each feature computed by the SVD. As can be seen from the plot, it is hard to set a threshold value, so we kept all 300 features. Besides, in comparison to the real number of features, 300 is still very small. Applying Random Forest algorithm to the projected data, we got around $\mathbf{BEF} = \mathbf{46}\%$. Clearly, this was a bad idea since we ignored lots of relevant features of our data.

8 Fisher Score and Classification

As a final approach, we considered Fisher score (F-score) for feature selection [9], then used several classifiers to test the idea. This is one of the approaches described in [1] for solving the NIPS 2003 Feature Selection Challenge. F-score tries to select a subset of features such that the distance between the points in the same class will be as small as possible, while the distance between different classes will be as large as possible. In particular, given training data points x_k, $k = 1, ..., n$, denote

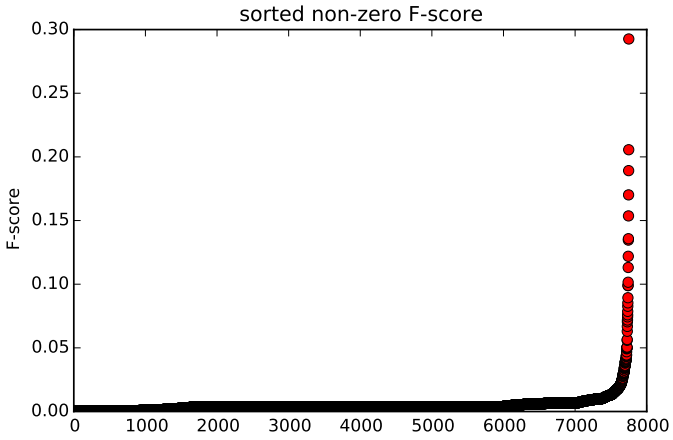

Figure 3: Non-zero F-score of features in an ascending order

the number of positive instances by n^+, number of negative instances by n^-. Furthermore, let $\bar{x}_i^+$, $\bar{x}_i^-$ and $\bar{x}_i$ denote the mean of the i-th feature for the positive, negative and whole data set respectively. Then, the F-score [1] of the i-th feature is defined as:

$$F(i) = \frac{(\bar{x}_i^+ - \bar{x}_i)^2 - (\bar{x}_i^- - \bar{x}_i)^2}{\frac{1}{n^+ - 1}\sum_{k=1}^{n^+}(\bar{x}_{k,i}^+ - \bar{x}_i^+)^2 + \frac{1}{n^+ - 1}\sum_{k=1}^{n^-}(\bar{x}_{k,i}^- - \bar{x}_i^-)^2},$$

where $\bar{x}_{k,i}^+$ ($\bar{x}_{k,i}^-$) is the i-th feature of the k-th positive (negative) instance. The numerator measures the discrimination between two classes, whereas the denominator describes the difference within each of two classes. The high F-score indicates more discrimination in the feature. Like all other methods, this method also has its disadvantages. For example, it doesn't give information about how features are related to each other.

To try this on our data, we started by computing the F-score of each feature. The plot of the sorted non-zero F-score is given in Figure 3. As a good sign, we get 7751 features that have non-zero F-score. Using all features with non-zero F-score and applying the SVM algorithm on it, we got $\mathbf{BEF} = \mathbf{14.67}\%$ error. Random Forest algorithm gave around $\mathbf{BEF} = \mathbf{12}\%$.

Conclusion

To conclude, there are various methods one could apply to this data set. There is a book [1] on methods that worked well for this specific problem. Particularly, in Chapter 12, they discuss how to combine F-score with Random Forest and SVM to get a better accuracy. Their best BEF was equal to 8% on the validation set. The winner of the competition was able to reduce the BEF to 1.8%. Thus, the methods we tried in this project gave somewhat close results to what is written in Chapter 12 of [1], which means we probably got acceptable results.

Acknowledgments

We especially thank Charles Delahunt for very helpful discussions.

References

[1] Guyon, Isabelle, et al., eds. Feature extraction: foundations and applications. Vol. 207. Springer, 2008.

[2] Cortes, C.; Vapnik, V. (1995). "Support-vector networks". Machine Learning 20 (3): 273. doi:10.1007/BF00994018

[3] Mayoraz, Eddy, and Ethem Alpaydin. "Support vector machines for multi-class classification. " Engineering Applications of Bio-Inspired Artificial Neural Networks. Springer Berlin Heidelberg, 1999. 833-842.

[4] Ho, Tin Kam (1995). Random Decision Forests (PDF). Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 1416 August 1995. pp. 278282.

[5] Breiman, Leo. "Random forests." Machine learning 45.1 (2001): 5-32.

[6] Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society. Series B (Methodological), JSTOR, pp. 267–288

[7] Documentation on LinearSVC, `http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html`

[8] Golub, Gene H., and Christian Reinsch. "Singular value decomposition and least squares solutions." Numerische mathematik 14.5 (1970): 403-420.

[9] Duda, Richard O., Peter E. Hart, and David G. Stork. Pattern classification. John Wiley Sons, 2012.