CIS 520, Machine Learning, Fall 2012: Project

Team Music Up: Qin He, Yuanhao Qiu, and Tao Lei

December 3, 2012

1. List of models used

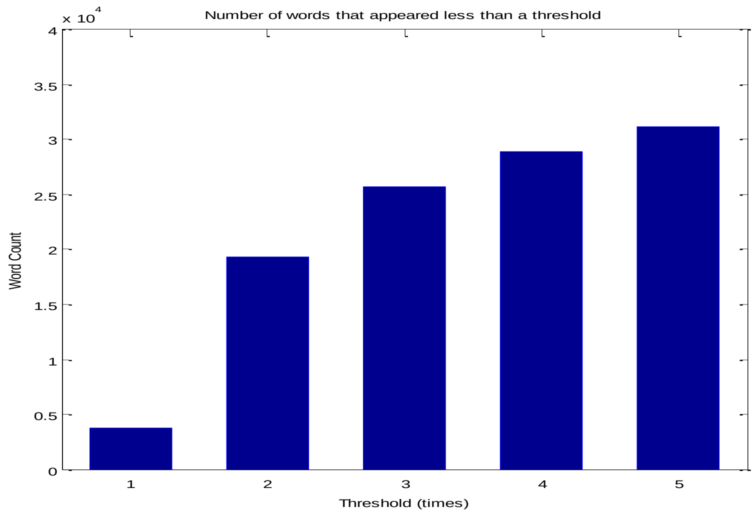
1.1 Support Vector Machine

1.1.1 For lyrics features

The very first though we had was that this project was quite similar with the *Newsgroup Post Classification* problem in Homework 6, so we decided to use Intersection Kernel and SVM upon the lyrics feature. The result was quite amazing: we hit the 2nd baseline with a rank error of 0.1452 without doing anything new.

Hoping to getter better result, we tried to improve the feature quality by inspecting the vocabulary. First we used Porter Stemmer to combine similar words, but it didn't help much. Another action we took was directly removing the words that appeared too few times in the training set from the vocabulary, and it improved the rank error by about **0.004**. The possible reason for the different effect of these two methods might be because words that barely appeared were just noisy and getting rid of them is better than keeping them there though in a neat way.

The proper threshold for vocabulary shrinkage was picked through cross validation and a check on the counts of the words. The figure below shows that more than half of the words appeared no more than 5 times (though some words such as "I" appeared as much as more than 60,000 times), thus the threshold cannot be too high otherwise the vocabulary will be too small and loss too much information. We tried on removing words appeared less than 1 to 5 times with cross validation and picked the one that gave best rank error which was **3** (i.e. words that appeared less than 3 times were ignored).



1.1.2 For audio features

Linear kernel vs. RBF kernel

Though succeeded on lyrics, intersection kernel didn't give a good performance on audio feature, which made sense because it's for sparse and non-linear feature while many of the audio features are linear. We tried both

RBF and linear kernel. The truth was that it was time consuming to train and test using RBF-kernelized SVM while RBF gave no better result than linear kernel with respect to audio feature in this case. Therefore we chose linear kernel which gave a decent rank error of 0.2182 (mean from local cross validation). Here we'd like to mention the method we utilized to tune multiple parameters (such as C and gamma) to obtain an optimized combination: **grid search** plus cross validation.

Kmod kernel

We were not quite satisfied with the rank error obtained by linear kernel and hoped to make it less than 0.2. By searching papers, we then decided to use a new kernel on audio features, which is called **kmod** [1]. The equation for kmod is:

$$kmod(x,y) = a[\exp(\frac{\gamma}{\|x-y\|^2+\sigma^2}) - 1]$$

Where ***a*** is a normalization constant equals to

$$\frac{1}{\exp(\frac{\gamma}{\sigma^2})-1}$$

Kmod kernel has two parameters, and we again tuned the parameters by grid search and cross validation and ended up with a pair of optimized ***gamma*** and ***sigma*****:** (120, 4.5). The results of Kmod-kernelized SVM from local cross validation for 100 are: Average rank error: ***0.2040***, Variance: ***0.0056***; Min rank error: ***0.1893***; Max rank error: ***0.2151***.

This method did better than linear kernel, and it may even be able to beat the second baseline since we observed that the true rank error based on quiz set is always about 25% less than the rank error based on our own test set. However, the time we found this kernel was only a couple of hours before the deadline of the code submission and we hadn't got the chance to make it better.

1.2 Neural Networks on audio features

Neural Networks did well on classification with audio features. We took the 30 audio features as the input layer and the 10 classes as the output layer. The cost function for Neural Network model is given below:

$$J(\theta)=-\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\log(h_\theta(x^{(i)}))_k + (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}(\theta_{ji}^{(l)})^2$$

Where $k=10$ is the size of output layer size, $L=3$ is the number of layers.

Totally there were three parameters we need to determine: hidden layer size, λ (used to control over fitting), and iteration time. We used Back-propagation algorithm to get the best $\theta_{ji}^{(l)}$ which could minimize $J(\theta)$. Using our cross validation system we got the best parameters: hidden layer size=40, λ = 0.2 and iteration times=1050. The average rank error of this method is 0.2063, better than linear-SVM on audio features.

1.3 Boosting

Boosting is a very nice method in binary class classification because the loss decreases exponentially, so we were thinking whether it is possible to implement a multi-class boosting. We implemented the SAMME algorithm introduced in paper [2]. SAMME is designed for multi-class boosting. The key difference between SAMME and AdaBoost is that AdaBoost requires the accuracy of weak learners should be better than 50% while SAMME requires the accuracy better than 1/K, where K is the number of classes. Unfortunately the result was not so favorable; the accuracy was only around ***56%***.

We believe that problems happened in the choosing of weak learners. The author did not mention how to choose weak learners in multi-class boosting. We tried a lot of methods to generate weak learners but couldn't make the loss decrease exponentially, although all weak learners we've chosen have accuracy better than random. As a matter of fact, the boosting converged in less than 20 rounds, we believe this was caused by the serious unbalance of the training set (there are more than half of examples labeled as genre 5 while many other classes had much fewer support examples).

1.4 Gaussian Naive Bayes on lyrics

Different from what we did in homework that used shared various Naive Bayes, for this project, we applied a non-shared variance Gaussian Naive Bayes in multi-class classification.

Frankly speaking, we didn't expect the result would be awesome for the reason that we had far more features here (40000+) than training examples (about 10000). The assumption of conditional independence in naive bayes will actually not hold since there are more features than examples. As a matter of fact, lyrics of a song are highly associated. The test result of Gaussian Naive Bayes is pretty bad as expected, the rank error on our own training and test set is **0.2998**, it may be able to beat the first baseline, but we did not submitted.

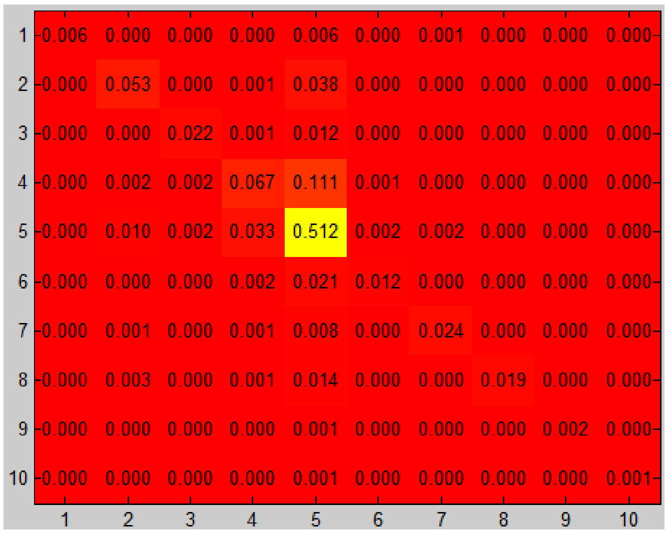
2. Key Steps

2.1 Weighted ensemble

Table 1 - Weighted Ensemble Results

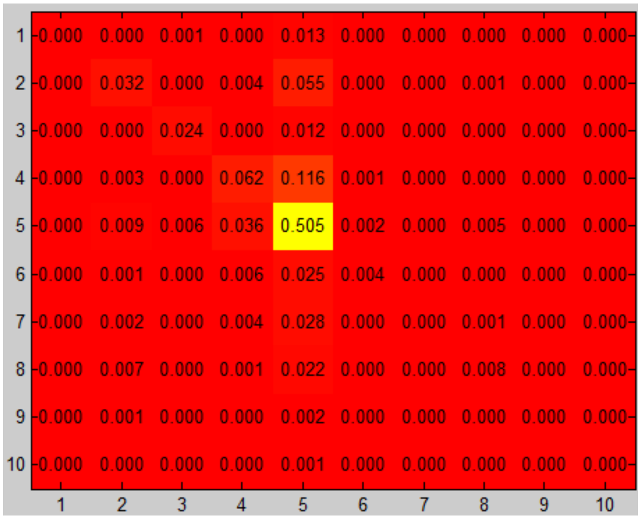
Weighted Ensemble Models	Rank Error From Local CV
SVM lyrics + NN audio	0.1456
SVM lyrics + NN audio + kmod-SVM	0.1469
SVM lyrics + NN audio + linear-SVM	0.1483

We had multiple classifiers and they were using two dramatically different types of features, therefore how to combine the results from those classifiers to obtain an optimized result become really important. **Weighted sum** should be the simplest and an effective way to ensemble results. Intuitively, the weight of prediction from different classifier is inversely proportional to their rank errors. We finally chose kernel-SVM on lyrics plus Neural Network on audio as our final model according to the rank error. *Why did not ensemble of three or more classifiers perform better?* **Confusion Matrix** helped us a lot understand the results.



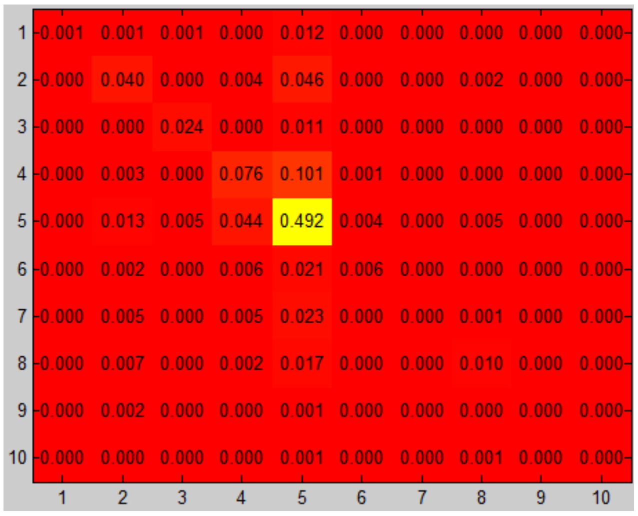
	1	2	3	4	5	6	7	8	9	10
1	0.006	0.000	0.000	0.000	0.006	0.000	0.001	0.000	0.000	0.000
2	0.000	0.053	0.000	0.001	0.038	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	0.022	0.001	0.012	0.000	0.000	0.000	0.000	0.000
4	0.000	0.002	0.002	0.067	0.111	0.001	0.000	0.000	0.000	0.000
5	0.000	0.010	0.002	0.033	0.512	0.002	0.002	0.000	0.000	0.000
6	0.000	0.000	0.000	0.002	0.021	0.012	0.000	0.000	0.000	0.000
7	0.000	0.001	0.000	0.001	0.008	0.000	0.024	0.000	0.000	0.000
8	0.000	0.003	0.000	0.001	0.014	0.000	0.000	0.019	0.000	0.000
9	0.000	0.000	0.000	0.000	0.001	0.000	0.000	0.000	0.002	0.000
10	0.000	0.000	0.000	0.000	0.001	0.000	0.000	0.000	0.000	0.001

Confusion matrix of SVM on lyrics



	1	2	3	4	5	6	7	8	9	10
1	0.000	0.000	0.001	0.000	0.013	0.000	0.000	0.000	0.000	0.000
2	0.000	0.032	0.000	0.004	0.055	0.000	0.000	0.001	0.000	0.000
3	0.000	0.000	0.024	0.000	0.012	0.000	0.000	0.000	0.000	0.000
4	0.000	0.003	0.000	0.062	0.116	0.001	0.000	0.000	0.000	0.000
5	0.000	0.009	0.006	0.036	0.505	0.002	0.000	0.005	0.000	0.000
6	0.000	0.001	0.000	0.006	0.025	0.004	0.000	0.000	0.000	0.000
7	0.000	0.002	0.000	0.004	0.028	0.000	0.000	0.001	0.000	0.000
8	0.000	0.007	0.000	0.001	0.022	0.000	0.000	0.008	0.000	0.000
9	0.000	0.001	0.000	0.000	0.002	0.000	0.000	0.000	0.000	0.000
10	0.000	0.000	0.000	0.000	0.001	0.000	0.000	0.000	0.000	0.000

Confusion matrix of SVM on audio



	1	2	3	4	5	6	7	8	9	10
1	0.001	0.001	0.001	0.000	0.012	0.000	0.000	0.000	0.000	0.000
2	0.000	0.040	0.000	0.004	0.046	0.000	0.000	0.002	0.000	0.000
3	0.000	0.000	0.024	0.000	0.011	0.000	0.000	0.000	0.000	0.000
4	0.000	0.003	0.000	0.076	0.101	0.001	0.000	0.000	0.000	0.000
5	0.000	0.013	0.005	0.044	0.492	0.004	0.000	0.005	0.000	0.000
6	0.000	0.002	0.000	0.006	0.021	0.006	0.000	0.000	0.000	0.000
7	0.000	0.005	0.000	0.005	0.023	0.000	0.000	0.001	0.000	0.000
8	0.000	0.007	0.000	0.002	0.017	0.000	0.000	0.010	0.000	0.000
9	0.000	0.002	0.000	0.000	0.001	0.000	0.000	0.000	0.000	0.000
10	0.000	0.000	0.000	0.000	0.001	0.000	0.000	0.001	0.000	0.000

Confusion matrix of Neural Network on audio

Except the obvious fact that SVM-lyrics was the best among all models, from the confusion matrix it was clear that class 5 caused the biggest confusion since it had most supporting examples and was best learned and therefore most likely to be labeled. NN-audio had better performance on class 3 and 4 than SVM-audio, especially better on class, which was very important in this project to reduce the overwhelming influence of class 5 and to make a better prediction. Based on the above, we could tell that SVM-lyrics and NN-audio complement each other and gave the best results.

2.2 Normalization

When dealing with audio features, scaling and normalization upon feature vectors are crucial from algorithm, especially for SVM. It is almost a universal rule to subtract the data set by the mean. What that really made difference was the scaling method. We tried multiple ways including dividing STD, dividing maximum of the absolute value, and scale the vector to [0, 1] or [-1, 1] using max, min and range (this is recommended by authors of LibSVM). We finally chose to scale the data into [-1, 1] based on its better performance. Dividing STD was not suitable for the audio feature because the scale for different features still varied a lot even after dividing STD, which might be due to the huge discrepancies in the nature of different audio features.

3. Future work

Non-satisfying classification using audio features was what limited our general performance so we would work more on reducing the rank error using audio feature alone and then used the improved classifier to ensemble with SVM on lyrics. As is mentioned in section 1.1 we believed that kmod-kernel should be a very promising model on audio features, we would like to further refine SVM classifier using this kernel. We should also create more informative features. Currently we were basically using features as they were. We actually tried PCA on audio features but there were no obvious principle components according to the running results so we did not think simple PCA would be a good way to create features from audio features, but we definitely should try more.

Reference:

[1] Ayat, Nedjem E., Mohamed Cheriet, and Ching Y. Suen. "KMOD-a two-parameter SVM kernel for pattern recognition." Pattern Recognition, 2002. Proceedings. 16th International Conference on. Vol. 3. IEEE, 2002.

[2] Zhu, Ji, et al. "Multi-class adaboost." Ann Arbor 1001.48109 (2006): 1612.

[3] Machine Learning, online class by Andrew Ng, Cousera

Code library used:

LibSVM
PorterStemmer