CS 221 Fall 2013: p-final

Ian Tenney
SUNet ID: iftenney

Chun-Kai (Ken) Kao
SUNet ID: kennykao

December 12, 2013

1 Overview

The overall task of our project is to develop a method to quantify song similarity. Possible applications to this system would include:

1. Identify cover songs or concert/live versions of a track
2. Identify songs in the presence of noisy backgrounds and distortion
3. Form the basis for a song-recommendation engine.

We consider tracks that are in the same "clique" based on the following criteria:

1. Track A is the cover song of track B or vice versa
2. Track A and track B are are the same song sung by different artists
3. Track A is a different version of the same song compared to track B (e.g. concert version, acoustic version...etc).

Our problem is divided into three stages. First, we need to identify relevant features that are shared by songs in a clique. Then we need to learn weights for these features to create a metric for song similarity, and finally we need to use this metric in evaluation algorithm to identify which clique a previously-unseen track belongs to.

2 Methods

2.1 Data

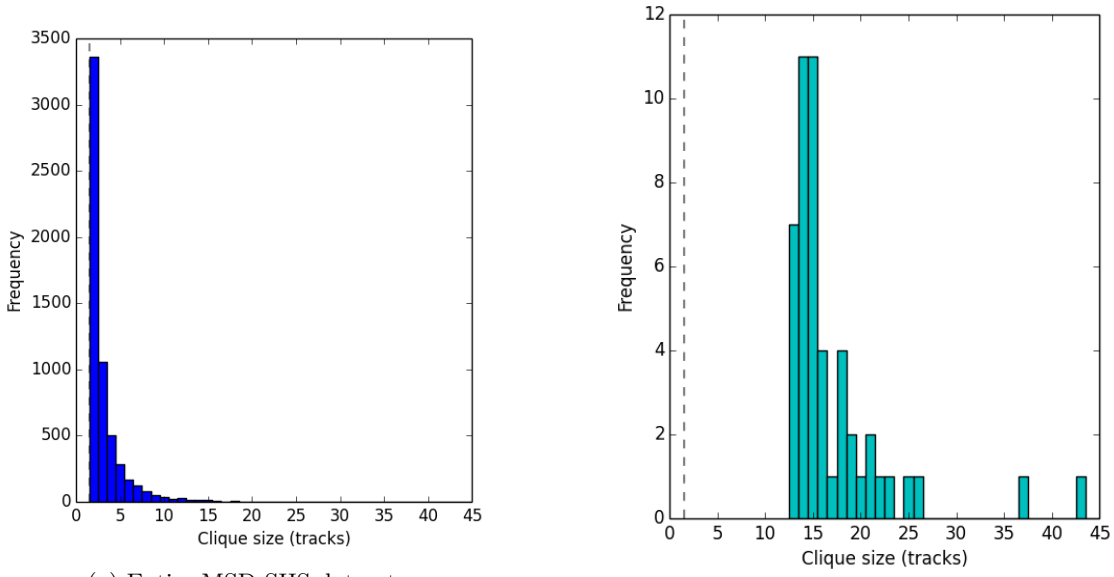
For this project, we needed to collect various songs, group them into cliques, and then classify them using the K Nearest Neighbors (KNN) algorithm.

2.1.1 Data Gathering

We use data from the Second Hand Songs (SHS) [3] subset of the Million Song Dataset (MSD) [7], which is a group of 18,000 tracks from the MSD grouped into "cliques" that are human-identified as the covers or versions of the same song (by the same or different artists). For each song, the dataset contains spectrograms and a large number of other general low-level features that can be used in our learning and retrieval scheme.

A large number of songs in this dataset are in cliques of only two or three songs, but for training a classifier it is desirable to use larger clusters. We parse the metadata for the entire dataset, then select the top 50 (or up to 300, for selected runs) cliques with the largest number of tracks.

We then sort each clique by the number of tracks it has, and take the top 50 (unless otherwise specified) cliques and their corresponding tracks for our test-run. Choosing the top 50 cliques ensures that each clique



(a) Entire MSD-SHS dataset

(b) Largest 50 cliques

Figure 1: Distribution of clique size (number of tracks).

contains at least 12 tracks, as shown in Figure 1b. We divide the tracks into two sets: training set and testing set, with a roughly 70:30 ratio.

2.1.2 Feature Extraction

Based on approaches from the literature [5, 12], we've identified a list of features to use for identifying cliques. The following list is referred to as the `combo` feature set:

- Loudness: the overall loudness estimate of the track (dB)
- Duration: the duration of the track (seconds)
- Pitches by segment: the average pitch for each track segment as defined by the Million Song Dataset
- Timbre by segment: The timbre shape by each segment
- Tempo: The overall tempo of each track (beats per minute)

This feature set was designed to combine both broad, high-level features such as loudness and tempo with the low-level pitch and timbre (MFCC-like) features. We predict that while the high-level features will go a long way toward reducing the distance between similar songs, the more specific low-level features will be provide the expressive power for our model to make exact identifications.

The MSD provides pitch, timbre, and several other features as average values across each segment of a song, where the segment boundaries are defined by The Echo Nest API [8] used to generate the dataset from raw audio and vary significantly from track to track. State-of-the-art cover song identification systems [11, 5, 1] have focused on dynamically aligning these segments for direct comparison, but these models are computationally expensive and out of the scope of our approach. As a proxy for this method, we uniformly sample 10 segments from each song, then concatenate the pitch and timbre features (each a 12-dimensional vector) at these 10 times to use in our feature vector. This gives a total dimension of 243 features for the "combo" feature set.

As described below , we achieved only limited success using this "combo" featureset. To extend the expressivity, we added a number of features described in [12] that describe the loudness and temporal characteristics of the track. These are:

- Peak loudness: 10 samples, mean, and variance

- Loudness at segment begin: 10 samples, mean, and variance
- Peak loudness onset (attack time): 10 samples, and mean
- Segment duration: mean and variance
- Beat duration variance
- Tatum duration: mean and variance
- Key
- Mode
- Danceability*
- Energy*

**Danceability and energy are scores derived from The Echo Nest API, using a proprietary algorithm but included in the MSD.* These features were chosen to complement the combo features, as well as provide statistical features more robust against intra-clique variance. In total, the combo features plus these additions give a 287-dimensional feature vector refered to hereon as `"comboPlus"`.

2.1.3 Preprocessing

The features described above Section 2.1.2 are highly heterogenous, representing different units and covering a broad numerical range. In order to convert them to a form suitable for input to our learning algorithms, we implemented three different forms of unsupervised pre-processing: a simple scaler, covariance whitening, and PCA. The simple scaler subtracts the mean and divides by the standard deviation of each feature, in order to give an approximation of all features having zero-mean and unit variance. This scales all features to a comparable domain, preventing global properties (e.g. regularization) of the learning algorithm from being dominated by features with large numerical values. Data whitening extends the scaler to divide by the full covariance matrix of all features, removing all global correlations between different features as well as normalizing values. Ideally, this gives independent features which provide a better basis for separating song cliques. Finally, PCA extends the whitening step by truncating to a reduced-dimensional feature space consisting of the components with the largest variance. In principle, this can reduce overfitting problems with our learning techniques, although in practice we found it to be less effective than regularization. All of these techniques ignore clique labels and operate on the entire training dataset before it is passed to the learning algorithm (logistic or LMNN).

2.2 Baseline System

As the baseline, we merely extracted the time-averaged timbre vector from 309 cliques, which has 1000 songs total. We chose timbre as our feature because it reflects the tone color and tone quality of the sound, which is what makes each music unique [9]. For this system, we also averaged the timbre across the entire song because each song has a different length. We then find the Euclidean distance between two tracks of the same and different cliques to see if timbre is a good feature to identify cliques.

Table 1: Statistics of baseline distance histograms.

	Same Clique	**Different Cliques**
Mean	83.40 ± 0.67	91.61 ± 0.04
Std. Dev.	± 40.77	± 40.44
Median	75.28	84.21
No. Comparisons	3696	995304

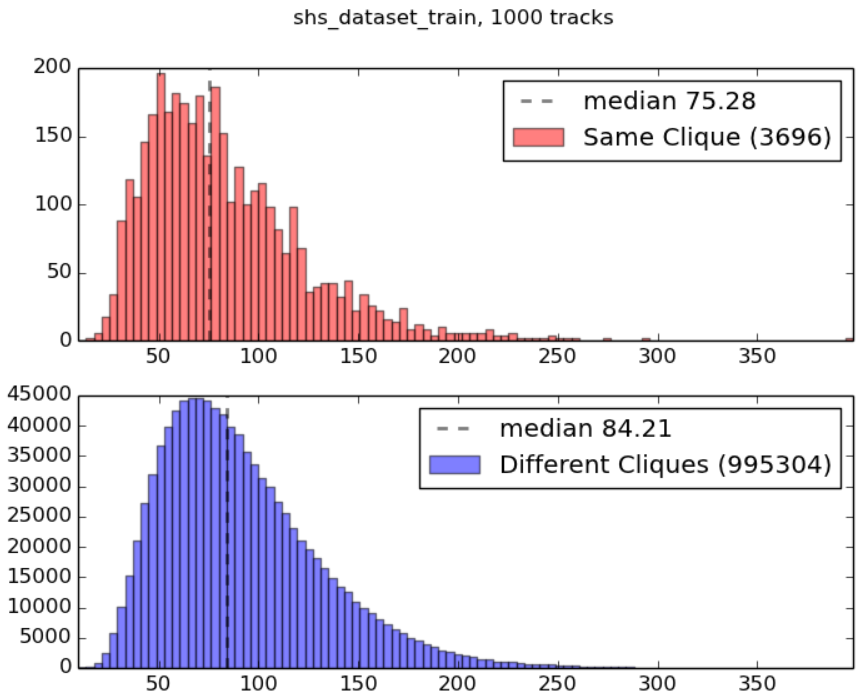


Figure 2: Distribution of timbre distance for same and different cliques.

We observe a slight difference in the two distributions, such that songs in the same clique have, on average, a smaller distance than those in different cliques (83.40 vs 91.61). However, this effect is dwarfed by the variance; the unweighted, averaged timbre features are not sufficient to distinguish cover songs from non-covers.

2.3 Training Techniques

To provide a better metric for our K Nearest Neighbors, we explored two supervised learning techniques: logistic regression, and metric learning via the LMNN algorithm.

2.3.1 Supervised Learning (Logistic Regression)

In order to learn a metric for song similarity, we transform the problem of identifying cover songs into one of learning the distinction between song pairs (s_1,s_2) that correspond to the same song versus different songs. This is then equivalent to training a binary classifier, from which parameters can be extracted and used to construct a metric.

For a set of N songs, there will be $\frac{N\cdot(N-1)}{2} = O(N^2)$ such pairings, with approximately $N \cdot C(C-1) = O(C^2 \cdot N)$ (for average clique size C) of these corresponding to same-clique pairs. This represents a highly imbalanced dataset, with a large number of negative examples. Fortunately, the large size of our dataset (18,000 songs, or our test subset of 650 songs) means that we can re-balance the dataset by undersampling the negative (not-same-clique) pairs. This is done with a random selection to generate a representative sample of equal size to the number of same-clique pairs available.

We compute pair wise features by simply taking the difference between the features of each song:

$$\phi_{pair}(s_1, s_2) = |\phi(s_1) - \phi(s_2)|$$

with the absolute value taken element-wise (so that $\phi_{pair}(s_1, s_2) = \phi_{pair}(s_2, s_1)$). Then we run logistic regression on these song pairs to optimize a weight vector, w, whose components correspond to the relative importance of each feature to distinguishing same-clique song pairs. Not unexpectedly, tempo, loudness, and duration were found to have the largest magnitudes, followed closely by some pitch class coefficients.

The accuracy of the logistic pair-wise classifier on the testing data set is between 57% - 60% with the best parameters. However, we did not find this pairwise accuracy to correlate well with the KNN test results, which are the focus of the report.

2.3.2 Metric Learning

Metric learning attempts to learn a (linear) transformation that provides a "good" embedding of the data into Euclidean space, such that distances between data points represent their similarity - in our case, the similarity of two songs. It is equivalent to learning an nxn matrix L such that

$$d(x_1, x_2)_{Mahalanobis} = \|L(x_1 - x_2)\| = \sqrt{\|(x_1 - x_2)^T M(x_1 - x_2)\|}$$

where $M = L^T \cdot L$ is known as a Mahalanobis matrix.

We chose to employ the Large Margin Nearest Neighbors (LMNN) algorithm described by [13, 2]. Instead of using a pairwise scheme, this algorithm tries to optimize a model of the K Nearest Neighbors classifier itself, making it well-suited (in theory) for our task. The KNN problem is modeled using a continuous objective derived from the hinge loss:

By learning the transformation matrix L to minimize this loss function, the algorithm attempts to minimize distance between same-label points, and to push different-label points past a minimum separation margin. This algorithm has shown strong performance for similar audio classification tasks [2], and an optimized implementation by the original author is available as a MATLAB package [14]. This package was compiled and run within a local MATLAB instance controlled by our Python codebase.

2.4 Classification (K Nearest Neighbors)

In order to classify training samples, we employ the K Nearest Neighbor classification system. The K Nearest Neighbor algorithm plots all the training samples onto an n-dimensional graph where n is the size of the feature vector. In our specific case, since we calculate weights, we multiply the feature vector by the corresponding weight for each feature and plot the product instead.

After the training data has been plotted, we then plot the testing data also scaled by the weight vector. We first specify k where k is the number of neighbors we want to investigate. The K Nearest Neighbors algorithm searches the k nearest neighbors, where "near" is defined by the Euclidean distance. After the k nearest neighbors are found, we look at the corresponding labels for each neighbor, and our prediction is the label that matches the most number of neighbors among the k neighbors.

For example, if we get 6 neighbors of a particular track, where 3 of them are part of the clique "Ain't Misbehavin", 1 of them is part of the clique "Everyday I have the Blues", and 2 of them are part of the clique "Ragazzo Mio", we go ahead and label that particular track to be "Ain't Misbehavin". We then compare the label we assign to the test data set to see if our prediction is correct. By doing this over a training tracks and testing tracks, we can calculate the accuracy our algorithm provides for both the training and test sets.

2.4.1 Cross-Validation

To ensure that we are not affected by skewed data, instead of merely splitting up the data into a roughly 70:30 ratio for training and testing sets, we create a seed. We change the value of the seed and average the results over 5 runs. This ensures that we do not get outliers, i.e. an especially compatible training/testing set pair, as was the case with one of our results during the poster session.

2.4.2 Relaxed KNN

After receiving feedback during the poster session, we realized that pinpointing a particular clique for a song out of 50+ cliques is inherently a difficult problem. We are limited by the amount of data we have (12+ songs per clique if we use 50 cliques), so expecting a classification system that can pinpoint the exact clique is a bit farfetched. Instead, we can loosen up the matching algorithm, such that given a value n, such that $n \leq k$, the top n most likely cliques among the k nearest neighbors are the labels we consider. If the actual label falls within the n labels we consider, then we consider this prediction "correct". This is a much better interpretation, as the ability to identify song cliques within 10% (top 5 closest cliques v.s. 50 total cliques) is much more reasonable.

3 Results

All results were obtained by 5 test runs with a random seed parsing our training and testing data samples. We compare the accuracy as affected by number of cliques, different feature sets, different k-values for KNN, and various pre-processing techniques. Lastly we compare the final results between our baseline system, supervised learning, and metric learning.

3.1 Accuracy by Dataset Size (number of cliques)

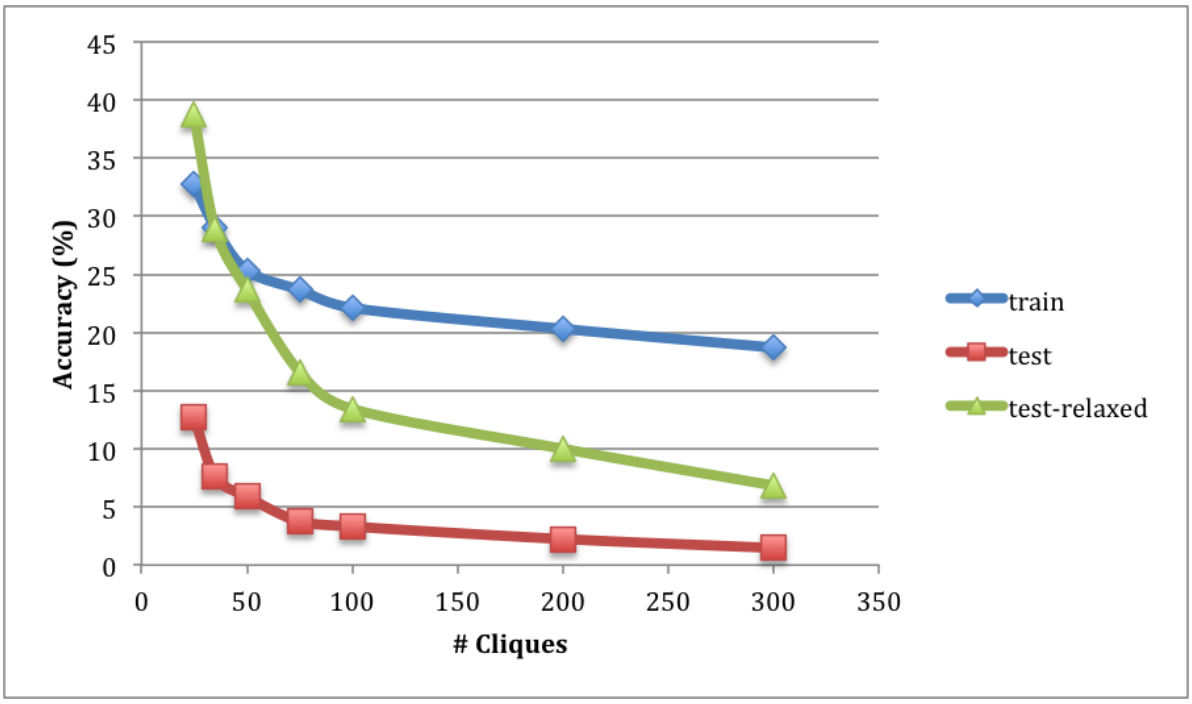


Figure 3: Effect of changing dataset size (number of cliques) on KNN relaxed accuracy (k=7, relaxation n=5), using comboPlus features, scaler preprocessing, logistic regression with L1 regularization (strength 100.0).

Based on our results, we see that as the number of cliques increase, the data becomes harder to generalize. There are multiple reasons for this. First of all, we sort the cliques by the number of songs in order to get the most sample possible. As a result, the first few cliques have a larger sample size than the last few cliques. As mentioned above, all cliques in the first 50 have 12 or more tracks, whereas cliques in the first 300 only have 4 or more tracks. As the number of tracks decrease for each clique, we are left with fewer training samples to effectively generalize. Secondly, As there may be certain features that are related across different cliques. For example, cliques of the same genre may behave similarly in our system. If we were to separate the cliques by genre and run tests by genre, we can weed out the features that make those tracks similar and focus on the differences.

In addition to varying the number of cliques, we have also relaxed the match for KNN. As mentioned in the previous section, it is much more reasonable to consider the label "correct" if the true label falls within the top 5 among the k neighbors, since pinpointing the exact clique with the limited training sample we have is inherently difficult. For the rest of our results analysis, we will look at the relaxed knn matches, with our test labeling as correct if the true label falls within the top 5 labels.

3.2 Accuracy across different feature sets

¡ At the poster session, our data seemed to suggest that combo-plus feature set works better with a smaller sample size (clique size < 100), and combo feature set works better with a larger sample size (clique size

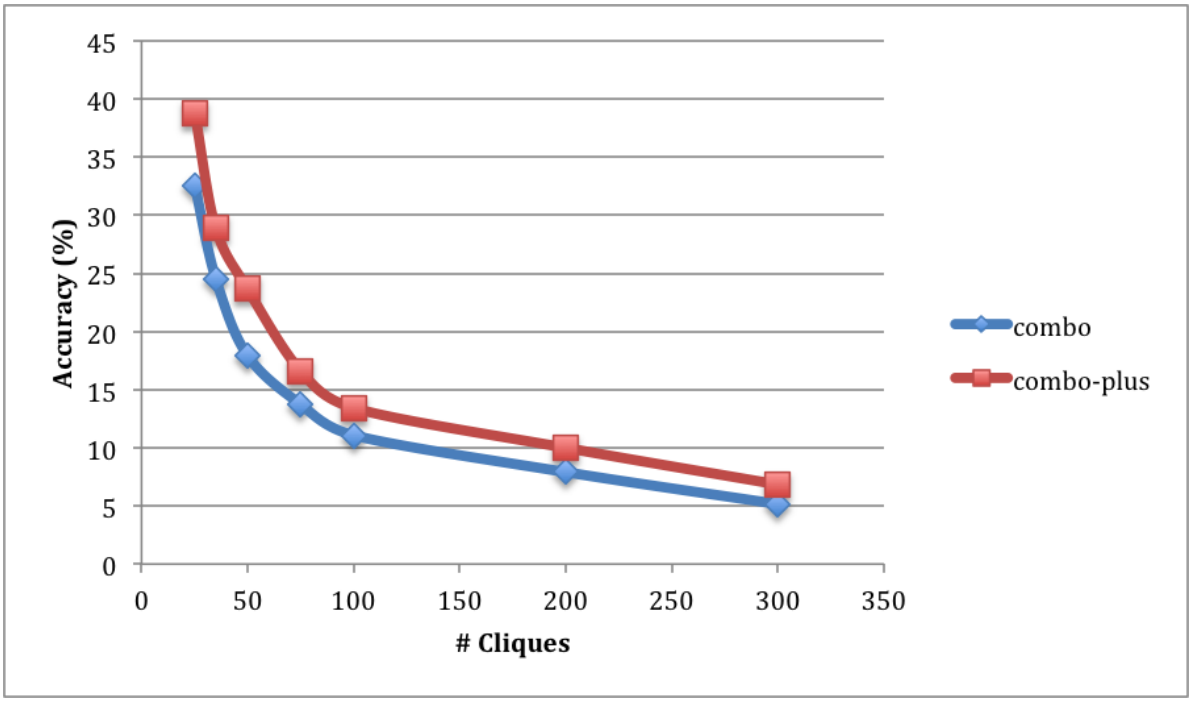


Figure 4: Effect of adding additional features (from `combo` to `comboPlus`) on KNN relaxed accuracy (k=7, relaxation n=5), using scaler preprocessing and logistic regression with L1 regularization (strength 100.0).

< 100). However, after implementing cross-validation, as the data shown above, it seems like combo-plus works better across the board. This is intuitive, as the classification of cover song is a complex problem, therefore adding a larger variety of features, such as peak, tatum, and danceability, will increase the ability for our system to generalize across tracks. Not only does combo-plus contain 46 extra dimensions, the values of those dimensions are also extracted from a wide variety of features.

3.3 Effect of changing K

As the data suggests, a larger k improves our ability to predict the label of the song. This aligns with our intuition, as we take the top 5 most likely labels among the k neighbors. Thus, if there are only 3 possible labels among 7 neighbors, we have essentially limited ourselves to 3 options. However, as k increases, we have increased the chances that there will be 5 or more different labels among the k neighbors, thus increasing the accuracy. On the other hand, the actual accuracy of the KNN test if we do not relax the knn matching, i.e. only take the most common label among the k neighbors, stays roughly the same at around 4.8% across all k's ranging from 5 to 15. This shows that based our data and training weights, the value of k does not affect the exact KNN matching results much, but does give us a boost for the relaxed KNN matching.

3.4 Effect of different preprocessing

Table 2: Average KNN-relaxed (k=7, n=5) results for different pre-processing techniques, before running logistic regression to calculate weights. All results use 50 cliques and logistic regression with L1 regularization, strength 100.0.

	Scaling	**Whitening**
Combo	17.94%	16.41%
Combo-plus	23.66%	19.39%

We find that on average, using whitening for pre-processing actually decreases our test accuracy significantly. The exact reason for this is unknown, but we suspect that the global covariance matrix does not

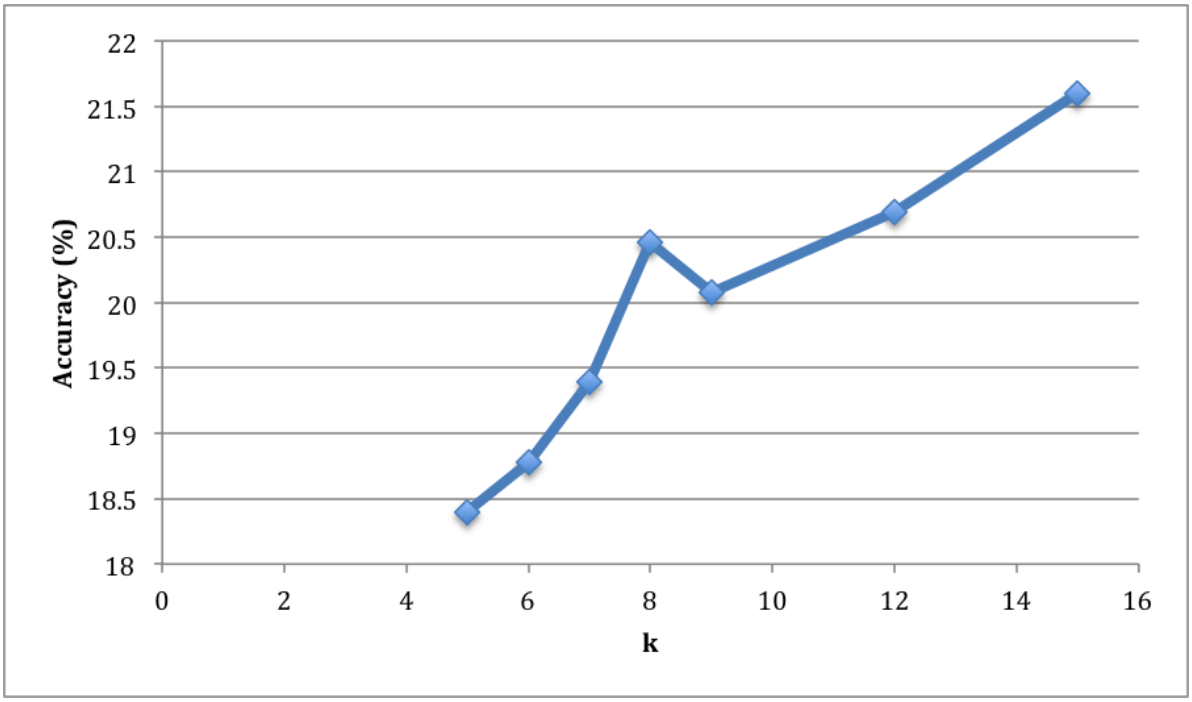


Figure 5: Effect of varying k on KNN relaxed accuracy, holding relaxation n=5 constant. Using comboPlus features, scaler preprocessing and logistic regression with L1 regularization (strength 100.0).

capture the relationships between features on a per-clique basis.

3.5 Effect of different regularization

Table 3: KNN relaxed test accuracy (k=7,n=5), using weights learned from logistic regression on 50 cliques, using scaler preprocessing and L1 regularization.

Regularization Strength	Combo	Combo-Plus
1.0	19.69%	22.52%
100.0	17.94%	23.66%

The difference in regularization for the logistic regression is very little. Although the accuracy for combo dropped by nearly 2%, while the accuracy for combo-plus increased by 1%, the difference.

For metric learning, as mu decreases, the regularization increases. This in turn improved the accuracy for our tests tremendously (from 12% to 18%). As we see from the plot, the accuracy also seems to plateau when $\mu = 0.001$ ($-\log(\mu) = 3$). This is likely because metric learning learns on a matrix, so we effectively have 287x287 = 82369 features, which increases the need for regularization.

3.6 Comparison of Metric, Supervised, and Baseline

**Random accuracy is estimated to be 10% because we have 50 possible cliques, and any of the top 5 would represent a "correct" labeling value.*

The data shows that based on a combination of picking the right features and KNN alone, we are able to increase the accuracy from an estimated 10.00% to 19.38%. Our learning methodology using logistic regression also further improved the accuracy to 23.66%, which shows that the added weights improves generalizability.

The best results from the LMNN algorithm fall short of this, and while still better than random they fail to surpass even the unweighted KNN; effectively, this means that the Mahalanobis matrix so learned

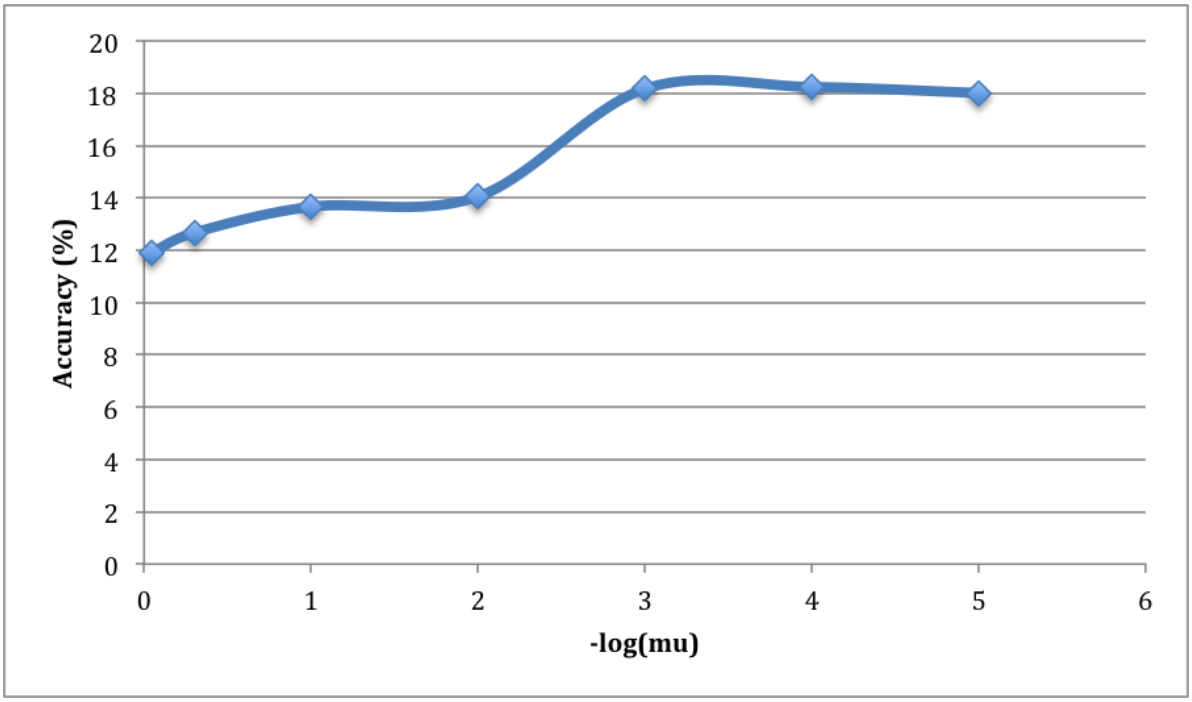


Figure 6: KNN relaxed test accuracy (k=7,n=5), using Mahalanobis matrix learned from the LMNN algorithm on 50 cliques.

Table 4: KNN relaxed test accuracy (k=7,n=5) for each learning method: no features (random), uniform weights, logistic regression, and LMNN

	Same Clique
Random*	10.00%
Uniform Weights	19.31%
Logistic Regression	23.66%
Metric Learning	19.08%

actually provides a worse embedding than the identity matrix! It is not known why this is the case, although it could be related to misuse, errors, or overfitting in the implementation of the LMNN algorithm. Further work could be done to diagnose the cause of this, but we did not have time within the scope of this project to delve into the complex implementation.

4 Conclusion

Our best results were achieved with our first approach, learning feature weights using logistic regression. Using scaling as our pre-processing method and combo-plus as our feature set. We have decided 50 cliques for our test runs because it seems to balance the randomness of identifying a track to a large set of cliques with enough cliques to make our system generalizable. We have also chosen to relax the KNN matching to 5 most likely labels because 10% of the labels seems like a reasonable threshold for our evaluation. We have chosen k=7 because preliminary test runs reveal that it gives us the best accuracy when we do a pure KNN test. However, once we relaxed the KNN matching, the KNN tests performed better with a higher k, but we did not have enough time to re-run all our tests. In the future, we hope to re-run all tests with a higher k-value to obtain more generalizable results. Nonetheless, the fact that our classifier significantly outperformed the random baseline shows that decent performance can be obtained from the "voting" scheme at the heart of the KNN algorithm, coupled with a good choice of features.

Unfortunately, we only observed a small increase in accuracy from our learning methods. As we originally

predicted, the logistic regression technique should generate feature weights that represent the importance of each feature to song similarity. Transforming the data by scaling with these feature weights embeds it into a Euclidean space where distance should be better correlated with preceived similarity. However, there are a number of reasons that this is not ideally suited to our task:

- The logistic objective tries to maximize margin, which may not be necessary for a KNN classification scheme; only a robust separation is needed
- The logistic objective function on song pairs is not exactly the L2 norm used by a KNN classifier
- The weight vector approach treats all features as independent, which is probably not true of our dataset

The LMNN algorithm should, in theory, address these problems: it tries to enforce a margin, but does not penalize non-matching neighbors once they are past it, it directly targets an L2 norm, and the off-diagonal matrix elements capture cross-correlations between different features. Nonetheless, we still achieved the best results by training with logistic regression. We do not entirely understand the reasons for this, but suspect that it could be an artifact of poor convergence and/or overfitting due to the large number of matrix elements.

5 Future Work

In summary, we implemented a training system for a cover song classifier that achieved modest success toward a very difficult multiclassification problem. Unlike many existing systems [11, 5, 6], we tried to use a learning-based approach on a large dataset, using low- and medium-level features across many songs to train a classifier to recognize music similarity. Our best results are significantly better than random choice, but we found only limited success by using learning to improve the feature weights and embedding.

We believe that some of these shortcomings are due in large part to the very nature of the problem: there may not exist a global metric that can capture the nuances of song similarity and still be amenable to a fast KNN-based classification scheme. The features that distinguish one clique from another may vary significantly across and within genres, making a local metric more appropriate. This could be implemented in a number of ways: as a set of pairwise or all-versus-one linear classifiers between cliques (a local version of our logistic approach), by using a hierarchical classifier to pre-sort songs by genre or other characteristics, or by using a nonlinear scheme, such as an SVM with an RBF kernel, for pairwise binary classification. All of these schemes come at the cost of higher complexity and slower evaluation, but may be necessary to achieve greater accuracy with a learning-based approach.

We also believe that we reached the limit of the features available to use from the MSD, at least to the extent that they could be used in a linear-metric classifier. Better results might be achievable using richer pairwise comparisons, such as aligning the beat signature of songs [5], performing spectral cross-correlations [11], or analyzing lyrics. These techniques would require a larger dataset of full audio tracks, and would be computationally much more expensive than our approach.

6 Acknowledgements

We would like to thank Will Song and Roy Frostig for their advice and guidance on this project, and Percy Liang for a great quarter of CS221.

Our code makes extensive use of the NumPy and SciPy [4] frameworks and the scikits-learn [10] machine learning library, which we relied on for optimized implementations of logistic regression, K nearest neighbors, and PCA/whitening. Additionally, we would like to thank Kilian Q Weinberger for making pubically available his reference implementation of the LMNN algorithm [14].

And finally, we'd like to thank the curators of the Million Song Dataset [7] and the Secondhand Songs Project [3] for their dataset, without which none of this project would have been possible.

References

[1] Mirex: Music information retrieval evaluation exchange, 2013.

[2] Blitzer, J., Weinberger, K. Q., and Saul, L. K. Distance metric learning for large margin nearest neighbor classification. In *Advances in neural information processing systems* (2005), pp. 1473–1480.

[3] DenisMonsieur, B. D. Z. M. D. Z. Second hand songs project, 2013.

[4] developers", S. Scientific computing tools for python, 2013.

[5] Ellis, D., and Poliner, G. E. Identifying 'cover songs' with chroma features and dynamic programming beat tracking. In *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on* (2007), vol. 4, pp. IV–1429–IV–1432.

[6] Gómez, E., and Herrera, P. The song remains the same identifying versions of the same piece using tonal descriptors.

[7] LabROSA, C. U. Million song dataset, 2012.

[8] Nest", T. E. The echo nest, 2013.

[9] Patil, K., Pressnitzer, D., Shamma, S., and Elhilali, M. Music in our ears: The biological bases of musical timbre perception. *PLoS Comput Biol 8*, 11 (11 2012), e1002759.

[10] "scikit-learn developers". scikit-learn: machine learning in python, 2013.

[11] Serrà, J., Gómez, E., Herrera, P., and Serra, X. Chroma binary similarity and local alignment applied to cover song identification. *IEEE Transactions on Audio, Speech and Language Processing 16* (08/2008 2008), 1138–1151.

[12] Slaney, M., Weinberger, K. Q., and White, W. Learning a metric for music similarity. In *ISMIR* (2008), pp. 313–318.

[13] Weinberger, K., and Saul, L. Distance metric learning for large margin nearest neighbor classification. *The Journal of Machine Learning Research 10* (2009), 207–244.

[14] Weinberger, K. Q. Lmnn, 2010.