CS 451 - Machine Learning
Aumit Leon & Mariana Echeverria

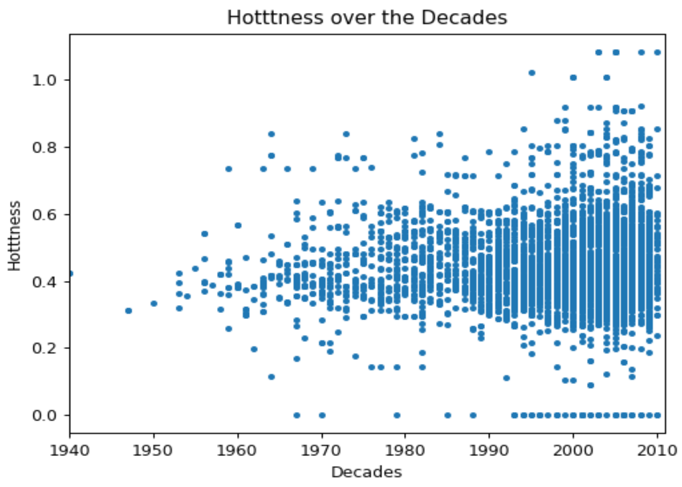
Observing the Evolution of Music Over Time Using the Million Songs Dataset

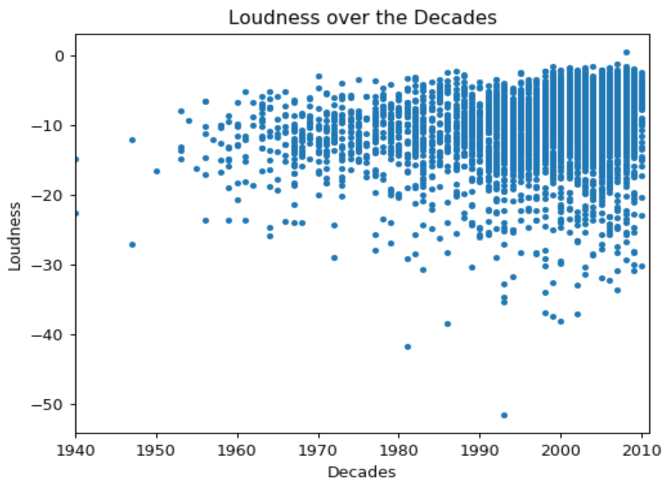
Introduction

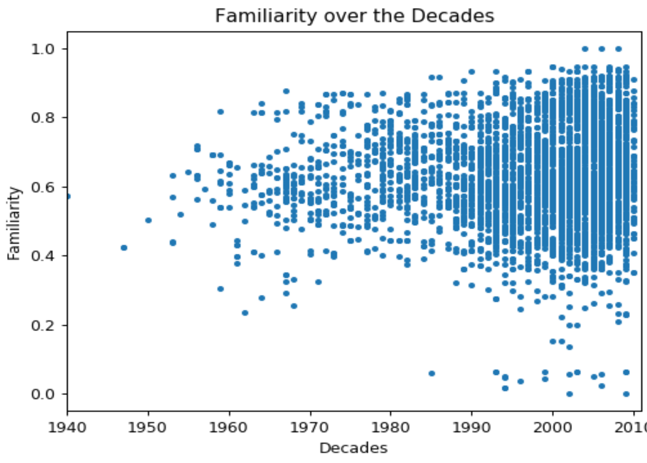
Our analysis was focused on the Million Songs Dataset (MSD), which has data on 1,000,000 songs for 44,745 unique artists, along with user supplied tags from the MusicBrainz website. Because of the size of this dataset (280 GB), for a quick taste, we ran most of our experiments on a subset of the MSD that contained 10,000 songs. We focused on predicting what year a particular track was released, as well as classifying and clustering the lyrics of the tracks we had available. For the former, we ran experiments using SVM's, Neural Nets, and a K-Nearest Neighbors algorithm; for the latter, we focused on K-Means and mini-batch K-Means algorithm as a way to cluster lyrics. By clustering lyrics and being able to accurately predict what year a track was released, we could provide some insight into the features that lead to this classification.

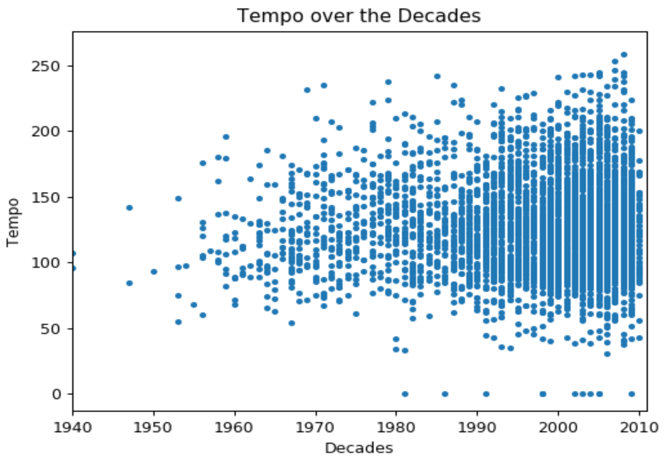
Experimental Setup

The MSD is an enormous corpus of data that requires several layers of preprocessing before the data is in a fully usable format. In order to get our data in a usable format, we first converted the data into a CSV with all the relevant features extracted for each song. The MSD was our baseline dataset, but we also focused on simplified subsets that were more suited to the task of year prediction and lyrics-clustering. We also ran some exploratory analysis on the dataset to see how various features correlated over the decades:









The above plots are just a few of the graphs we made in our exploration of the MSD, but they are emblematic of the trends across the feature space: most of the data is from the late 90's and early 2000's, and there is a small but noticeable uptick in whatever feature we are observing across the decades. These plots informed our feature extraction process-- to see more, visit our github page linked at the end of this report.

The year prediction dataset we used is a simplified subset of the MSD. This dataset has 90 attributes (features): 12 timbre average, 78 timbre covariance. The dataset extracted these features from the MSD, and contains 515,345 examples. For the sake of year prediction, we began by implementing a Support Vector Machine using scikit-learn's SVM library. We also applied the GridSearchCV algorithm using 3-fold cross validation in order to run a parameter sweep for C and Gamma. We used 3-fold cross validation as opposed to 10 fold cross validation (as the literature recommends) because of the amount of time the GridSearchCV parameter sweep was taking.

The majority of our experiments were built around Neural Nets: we ran experiments with a range of parameter values and training examples (ranging from 1,000 - 150,000) and evaluated our results using a test set and scikit-learn's score and evaluation metrics. Later on in our experimentation, we shifted away from Scikit-learn and began focusing on Keras. We made the shift because Keras provides more flexibility in the construction of deeper neural nets (with respect to parameter tuning, layer density, and output formats). We tested a variety of Neural Network architectures with various parameters and training examples and evaluated these results using Keras' evaluation metrics, focusing on training accuracy and validation accuracy, as well as the trend of various learning curves that we plotted.

After struggling with getting our accuracy higher, we consulted the literature and used a K-Nearest-Neighbor (K-NN) algorithm to classify songs by year. We used scikit-learn's K-NN library and functions to train on 200,000 examples, and then tested the performance of this model on 5,000 examples using scikit-learn's evaluation metrics (focusing on accuracy). The literature in this area provided some insight on the most effective value of k, which informed our final results. We decided which results were significant by running experiments multiple times on random subsets of the data, and consulting the literature in the field. If a certain result came up multiple times throughout experimental runs, we considered it significant.

For the clustering lyrics experiment, we used the musiXmatch (MXM) dataset, which is the official lyrics collection of the MSD. The dataset consists of lyrics for 237,662 tracks that come in a bag-of-words format, and each track is described as the word-counts for a dictionary of the top 5,000 words across the set. The data set comes in two text files — a training set and a test set. To deal with this data, we combined the train and test sets and created a list of numpy arrays. Each vector has 5,000 elements, one for each word. First, we set each entry equal to the frequency of the word at that index for a specific track. However, we got a lower cost by just setting them equal to 0 if they were absent, and 1 if they were present in the track.

We started using the K-Means code from HW5, but because of concerns about efficiency we decided to pivot to a more established library in scikit-learn. From scikit-learn, we used two algorithms: K-Means and MiniBatchKMeans. For our final clustering we used the latter because it ran significantly faster. After obtaining the cluster centers and the cluster to which each track was assigned, we used scikit-learn for Neural Nets to predict clusters based on other features, such as year, tempo, loudness, and hotttness.

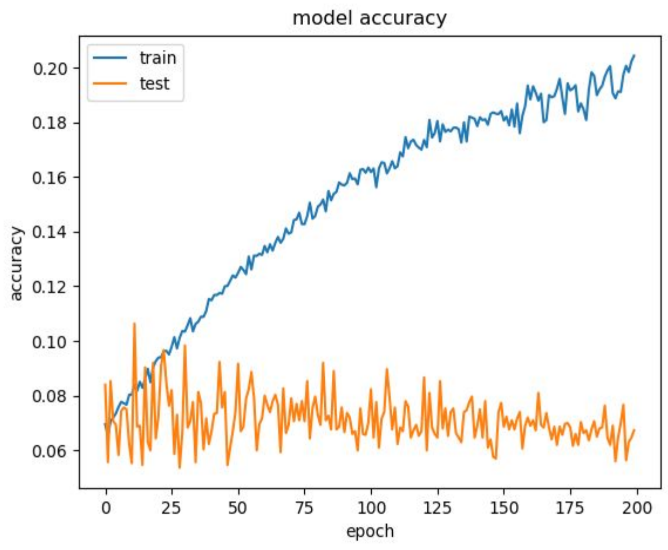
Most of our experiments were dealing with large amounts of data, so algorithms like GridSearchCV and deeper neural nets were taking a large amount of time to work with. To work around this run-time constraint, we began launching jobs on the Condor High Throughput Computing system. Condor allowed us to run multiple experiments in parallel which allowed us to iterate quickly and test our methods more efficiently.

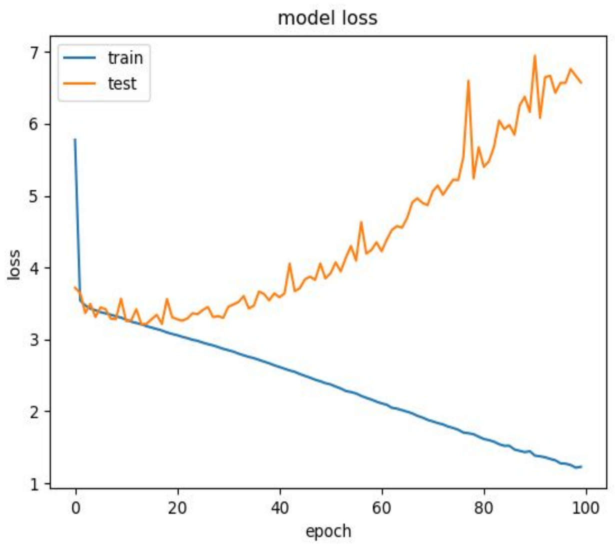
Results

<u>Year Prediction</u>

Our first attempt at year prediction came in the form of a crude Support Vector Machine implemented from scikit-learn that would classify songs into one of 89 labels (1922-2011). In order to tune parameters, we used GridSearchCV with 3 fold cross validation. We trained our model on 5,000 examples, had a cross validation set of 1,000 examples, and tested our accuracy on 1,000 examples. The highest accuracy we got from our parameter sweep was 31%, but we did not read too deeply into these results because SVM’s were not widely used in this area of research on the MSD.

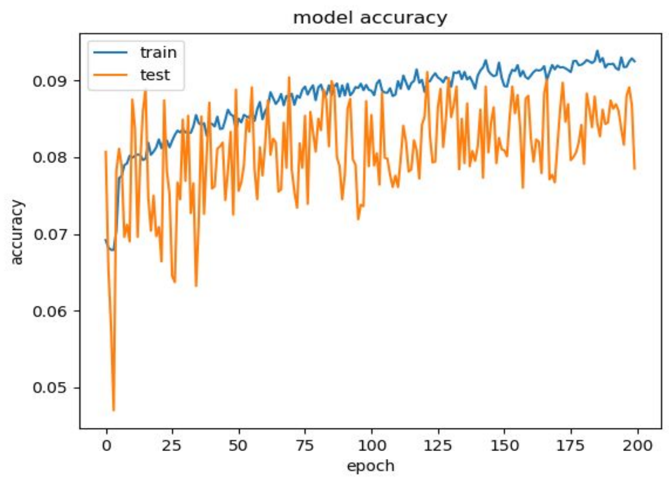
Neural Networks
We ran various experiments using Neural Networks, prototyping models in scikit-learn and then building them in Keras. The following is a discussion on the experiments we ran and the results obtained. The optimizer used for all Neural Network experiments was Stochastic Gradient Descent (SGD), and Rectified Linear Units (ReLU) functions were used for all hidden layers, while softmax functions were used for all output layers.

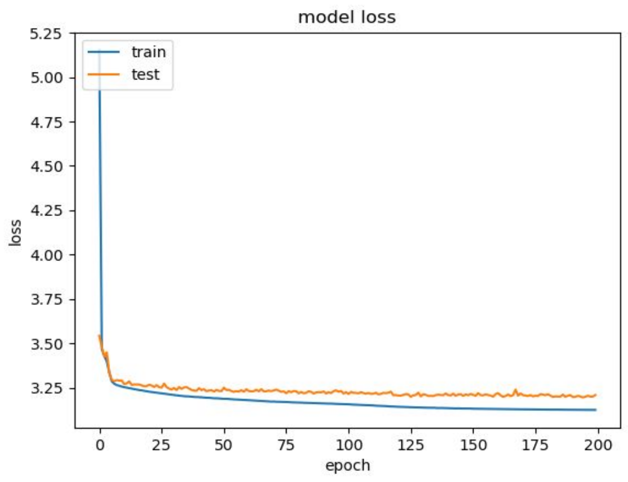




The above plots correspond to a simple Neural Network that we constructed with 5 hidden layers, 100 hidden units for each layer. This was trained on 10,000 examples with a 0.1 validation split, 200 epochs, a batch size of 32 with feature scaling.

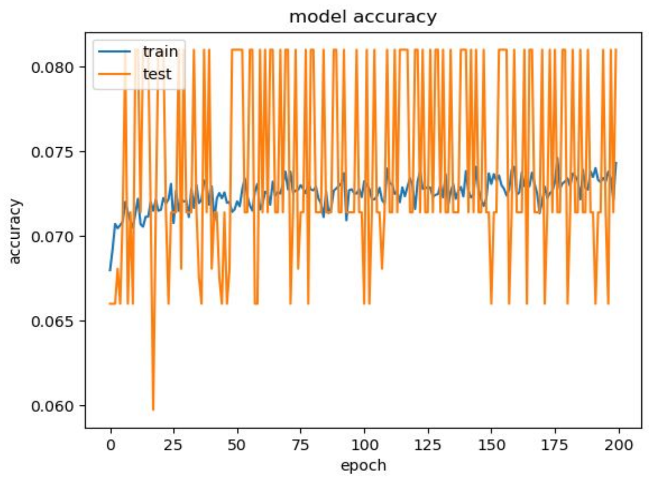
This model achieved 7% accuracy on the test set. This was one of the simpler models that we developed and tested, but it is still interesting to observe. In particular, the model accuracy told us that we had an overfitting problem-- if we wanted to address this, it might require that we focus on developing a model with more data, regularization, and a tuned architecture.

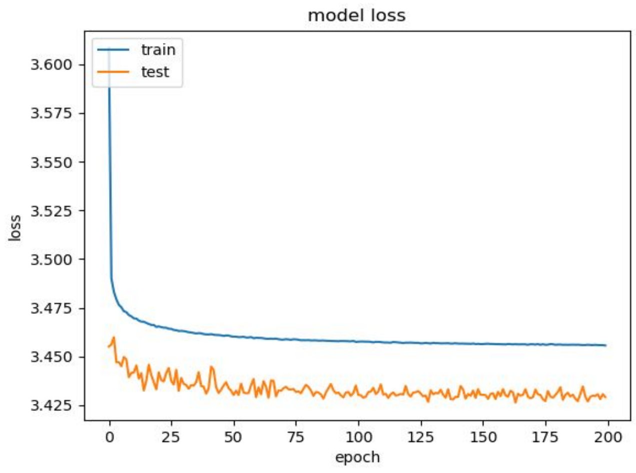




The above Neural Network had 10 hidden layers, 100 units for each unit. The model had a regularization parameter of 0.00001, and a learning rate of 0.001. After applying feature scaling, we trained this model on 10,000 examples with a 0.1 validation split for 200 epochs and a batch size of 32.

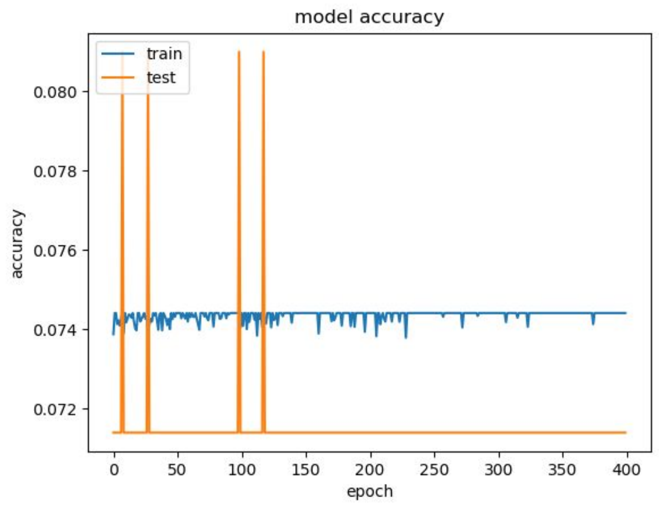
This model achieved a validation accuracy of 9.2%. Observing the model accuracy plot shows that the training and test accuracies follow the same general trend, but the test set accuracy oscillates more wildly-- possibly a function of our learning rate. The loss curve of the test set mimics that of the training set, which means the model is acting as it should.

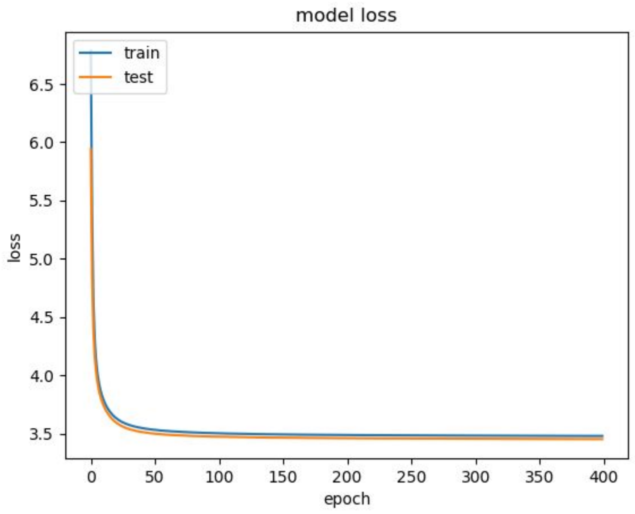




The above Neural Network had 50 hidden layers, 100 units for each layer. The regularization parameter was 0.00001, learning rate was 0.001. After applying feature scaling, we trained this model on 150,000 examples with a 0.1 validation split for 200 epochs and a batch size of 15. The decay rate for the learning rate was 1e-6. For this experiment we also tried Nesterov Momentum.

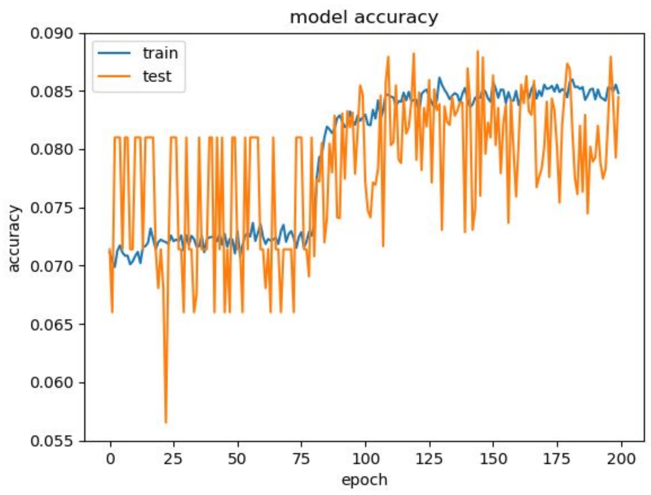
This model achieved 7.8% accuracy on the test set, which is emblematic of the accuracy levels we had achieved up till this point. From the model accuracy plot above, we can see that the accuracy of the test set follows the accuracy of the training set, although there is major oscillation which might be an issue with our learning rate. For this particular example the test loss was lower than the training loss, which was an unexpected result-- this could be a function of the Nesterov Momentum but we need to investigate further.

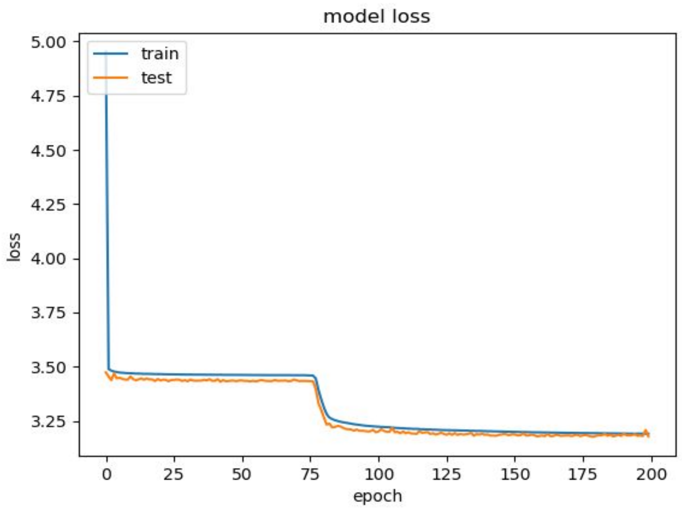




The above Neural Network had 250 hidden layers, 100 units for each layer. The regularization parameter was 0.00001, learning rate was 0.001. After applying feature scaling, we trained this model on 150,000 examples with a 0.1 validation split for 200 epochs and a batch size of 15. The decay rate for the learning rate was 1e-6. For this experiment we also tried Nesterov Momentum.

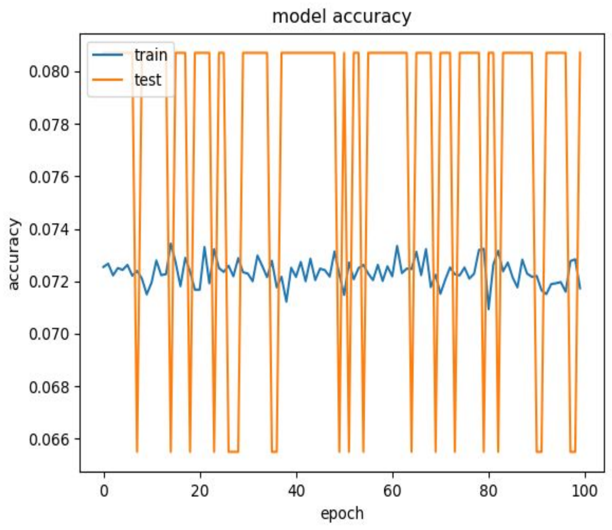
This model achieved 6.5% accuracy on the test set, which is bit smaller than the accuracies we had been getting up to this point. The accuracy plot for this experiment is rather odd, indicating major oscillations between the 1st and 5th epoch, as well as between the 100th and 150th epoch. The model loss plot shows that the loss of the training and test sets follows an almost identical trend, which told us that we were moving in the right direction in using a deeper Neural Net architecture.

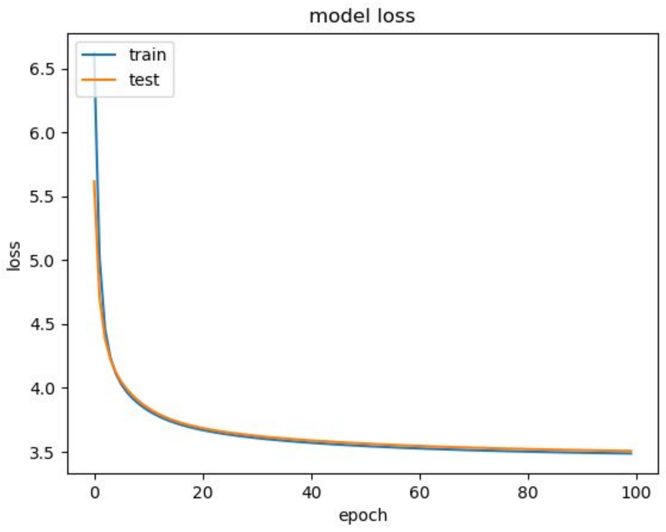




The above Neural Network had 25 hidden layers, 100 units for each layer. The regularization parameter was 0.00001, learning rate was 0.001. After applying feature scaling, we trained this model on 150,000 examples with a 0.1 validation split for 200 epochs and a batch size of 32. The decay rate for the learning rate was 1e-6. This experiment also applied Nesterov Momentum.

This model achieved 8.1% on the test set, which is a bit higher than the accuracies we have been observing up to this point. The model accuracy shows that the training and test sets are again matching each other in terms of the trend of the function. The model loss plot is emblematic of the performance of the deep net models we have been observing up to this point-- the training and test loss mirror each other.





The above Neural Network had 200 hidden layers, 10 units for each layer. The regularization parameter was 0.0001, learning rate was 0.001. After applying feature scaling, we trained this model on 100,000 examples with a 0.1 validation split for 100 epochs and a batch size of 32. The decay rate for the learning rate was 1e-6. This experiment did not apply Nesterov Momentum.

This model achieved a 6.5% accuracy on the test set-- which is on the lower end when compared to our other observations. The model accuracy plot shows the test set oscillating wildly, while the training accuracy remains relatively stagnant. The model loss plot shows that the test set and training set both decrease steadily, at an almost identical rate.

K-Nearest Neighbors

After consulting the literature in the field, we noticed that K-NN was an approach used in year prediction for the MSD. We Applied K-NN with k = 50 on 200,000 training examples, and validated against 5,000 test examples. The validation accuracy we got without feature scaling matched the accuracy we got with feature scaling: 6.6% (so feature scaling was not an effective approach). This was rather low, so we asked how far off each prediction was from a given label. The following is a table of a random sample of our prediction results.

Label	Prediction	Abs(Label-Prediction)
2005	1994	11
2005	1996	9
2005	1998	7
2005	2000	5
2005	2006	1

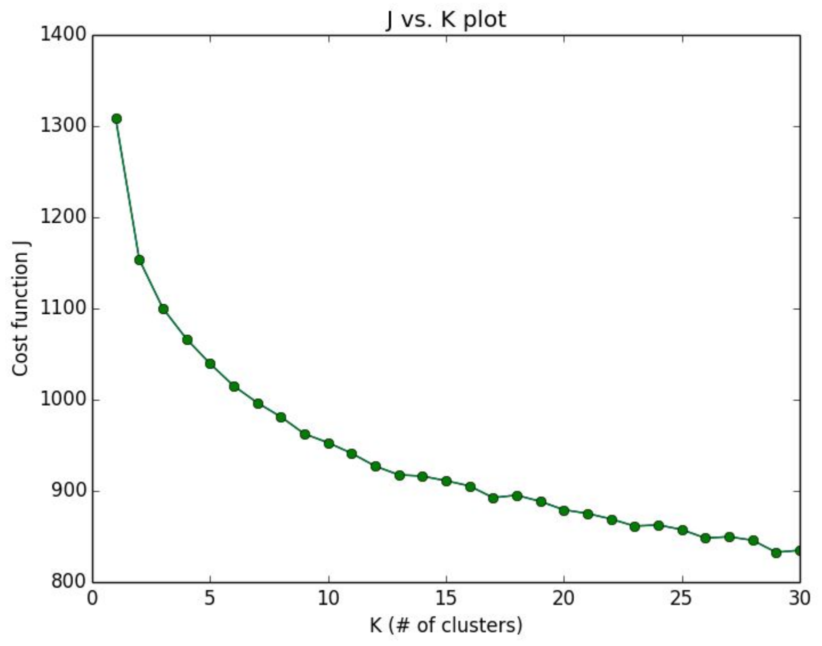
2003	1998	3
2003	2002	3
2003	2006	5
2003	2009	1
2003	2003	3

As you can tell, all of our predictions are actually rather close to the actual label. We calculated the mean absolute value and got 7.9-- this means that every prediction is on average off by about 8 years. When we adjusted our accuracy measure to account for a prediction that falls in a range of 10 years, the validation accuracy increases to 76.48%. If we applied feature scaling in our preprocessing step, there was little difference in the accuracy (0.02% change), so we can conclude that feature scaling is not an effective tactic to try.

<u>Clustering Tracks by Lyrics</u>

HW5 K-Means Code

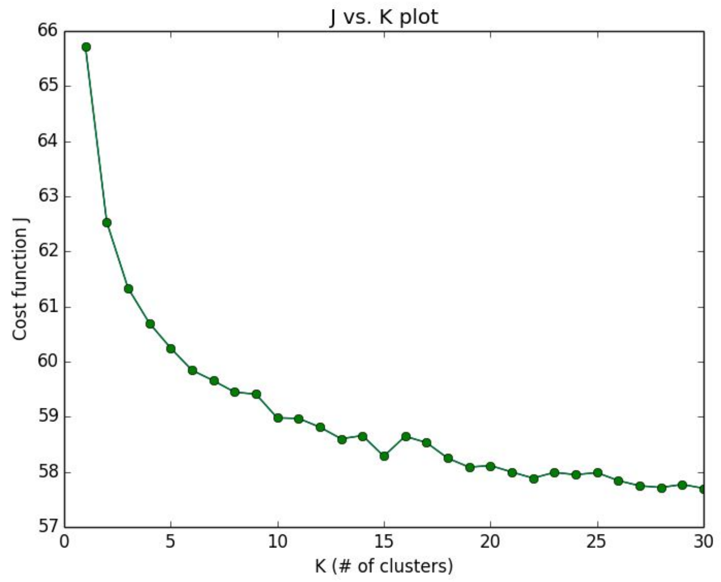
Our first attempt was to implement the K-Means code for HW5. Given that we were dealing with a large dataset, the running time was significantly high. Therefore, we decided to use 1,000 tracks. We ran our algorithm for K=1 to K=30 and kept track of the minimum costs J obtained.



The graph shown above was obtained by plotting the number of clusters K vs. the cost function J. The costs obtained are not as low as we had expected them to be. For this reason, we decided to use 0, if the word was absent, and 1, if the word was present, as entries for the vectors instead of their frequencies, which ended up giving us lower costs.

Scikit-learn K-Means

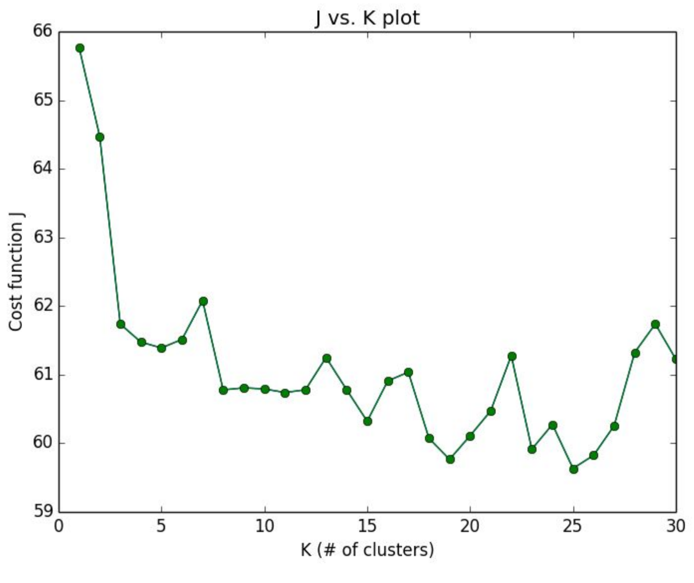
Searching for a faster algorithm, we decided to use the scikit library KMeans. With this algorithm, we were able to run all 237,662 tracks from K=1 to K=30. We gave KMeans two parameters: *n_clusters=(1-30) and random_state=0*. By setting random state equal to 0, we were able to obtain the same output every time we ran the algorithm.



The graph shown above was obtained by plotting the number of clusters K vs. the cost function J. The costs are lower than the ones obtained with HW5 code, and its runtime was significantly faster.

Scikit-learn MiniBatchKMeans

By implementing the MiniBatchKMeans, we were able to compute clusters more efficiently and determine the best K. MiniBatchKMeans does incremental updates of the centers positions using mini-batches. Scikit suggests using this mini-batch approach when dealing with a dataset bigger than 10,000 examples. We gave MiniBatchKMeans two parameters: *n_clusters=(1-30) and random_state=0*, and used the default value for the size of the batch, which is equal to 100.



As we can see in the graph above, the cost with the mini-batch version fluctuates. However, we could still observe that a reasonable value for K could be 3.

Another experiment we tried was to run this algorithm with K=10 and count how many tracks were assigned to each cluster, which gave the following results:

Cost J = 60.4381434773

Cluster	Number of Tracks
0	1
1	**155,786**
2	**36,060**
3	1
4	1
5	9
6	1
7	**45,801**
8	1
9	1

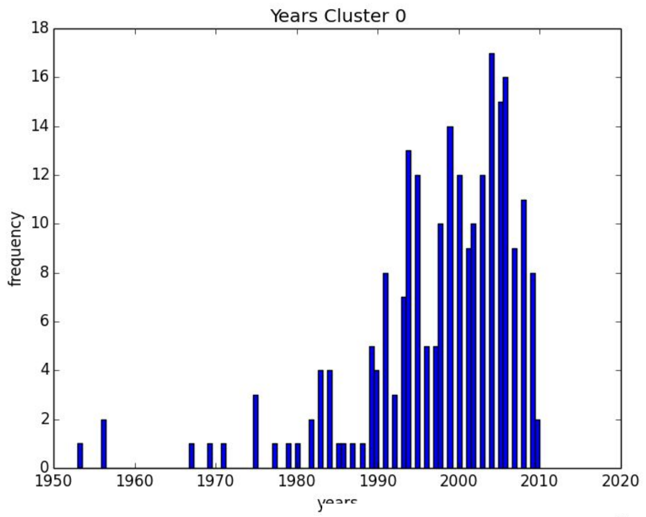
The table above shows how most of the tracks were assigned to just 3 clusters out of the 10. This would be an interesting result to investigate further in future work. This clustering also suggest that K=3 would be the ideal number of clusters to use.

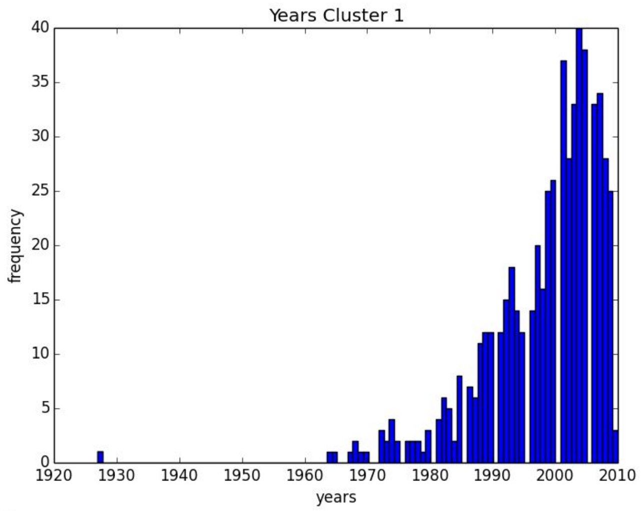
Choosing the Value of K and an Algorithm

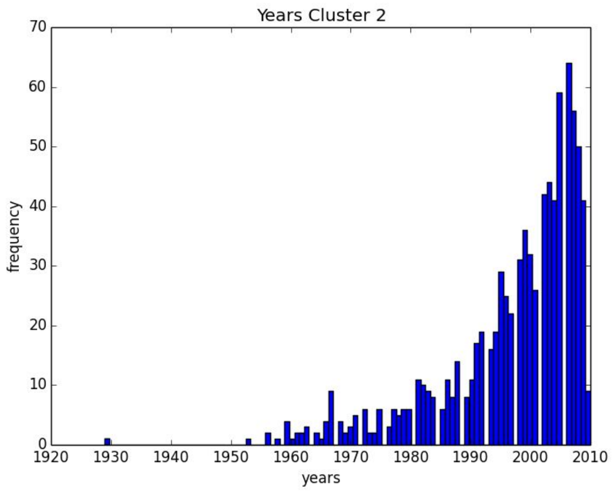
For this project, we decided to use K=3 and implement the MiniBatchKMeans. By applying the “elbow method” to the above graphs for J vs. K=1-30, we can see how from K=1 the cost J goes down rapidly, and after it reaches K=3, the cost starts going down slowly. Furthermore, the results obtained when clustering with K=10 also suggest that K=3 would be a reasonable value for the number of clusters.

Year Distribution in Each Cluster with K=3

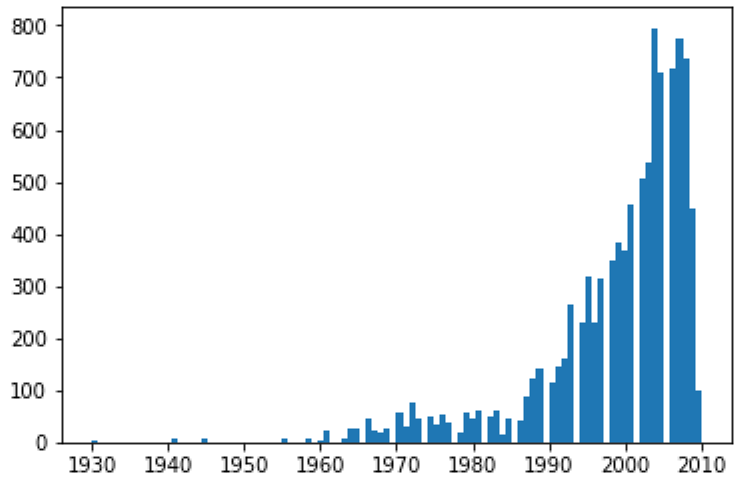
The track ID’s in the dataset allowed us to connect our clustering results with the years in which each track was released. By doing this, we plotted histograms for each of the clusters with the respective years vs. quantity of tracks. Given that not all tracks were found in the subset of songs and that some years had the value 0, our results do not contain all 237,662 tracks.







These histograms suggest that there exists a similar distribution of tracks and years in the 3 clusters. This distribution also matches the one obtained with the entire dataset, as shown in the graph below



Another observation that we could make from these histograms is that the lyrics of tracks do not play a significant role in determining the year in which the song was released.

Predicting Clusters from Features

After obtaining the clusters, we extracted features using hdf5_getters, created numpy arrays for each track, and used the cluster center indices to create labels. We divided our data in a training and test set. The Neural N
etwork had 9 hidden layers, 50 units for each layer except for the last one which had 20. The learning rate was 0.00001. After applying feature scaling, we trained this model on 2,000 examples with 351 validation examples for 200 epochs and a batch size of 100.

This model achieved 57% on the test set, which suggests that there exists a relation between the lyrics and the features we selected.

Conclusions

In summation, the Million Songs Dataset is a rich source of data for observing and analyzing trends in music over time. Our exploratory analysis highlighted that most of the data in our simplified subset was from the late 90's or the 2000's. It is reasonable for us to conclude that this trend generalizes to the entire corpus of a million songs primarily because the subsets were produced via a random sampling of the original dataset. With this in mind, it is now clear why our validation accuracies in the experiments attempting to predict year were so low-- our data is not of high enough quality to allow our models to accurately predict years. We hypothesize that if we had a gaussian distribution in our data, this would allow our models to be more accurate. While the data available to us is one of the most prominent datasets in the Music Information Retrieval (MIR) field, accurately predicting years still remains a challenge given the approaches that we took. We were, however, able to conclude that the dataset was strong enough to facilitate “good-enough” predictions-- every prediction was on average approximately 8 years off. While this model is not able to predict years exactly, it is able to predict the generation a song is in with a high accuracy, which we consider a success.

Exploring lyrics added another fascinating dimension to an already enormous dataset, and allowed us to understand the evolution of music from a fresh perspective. One of the most interesting parts of this analysis was the the fact that even though we attempted to use higher values of k for K-Means clustering, there was always 3 clusters that dominated the cluster-space. This allowed us to conclude that given the features we were extracting, there were 3 primary distinctions that could be made given the songs in the dataset. We also found it very interesting that the distribution of years within each cluster matched the overall distribution of the entire MSD-- we interpret this to mean that the lyrics in songs haven't changed as significantly as one might assume over the years.

Further analysis could look into whether or not neural nets are able to effectively lower the range in which years are being predicted (the best we have done so far is a range of 8 years), It would also be fascinating to see if an algorithm trained with a similar dataset with a gaussian distribution would be able to predict years with a higher accuracy and granularity. One of the constraints we had was a lack of storage space and computing power to train our algorithms on the entire dataset, if we had these resources available, then we could train much larger networks with higher quantities of data-- it might even be possible to say what year a song *sounds* like it could from, which is a prediction that a human could corroborate.

Link to Code

https://github.com/AumitLeon/million-songs

Literature Review

https://github.com/AumitLeon/million-songs/wiki/Literature-Review