From Classical To Hip-Hop: Can Machines Learn Genres?

Aaron Kravitz, Eliza Lupone, Ryan Diaz

***Abstract*—The ability to classify the genre of a song is an easy task for the human ear. After listening to a song for just several seconds, it is often not difficult to take note of that song's characteristics and subsequently identify this genre. However, this task is one that computers have historically not been able to solve well, at least not until the advent of more sophisticated machine learning techniques. Our team explored several of these techniques over the course of this project and successfully built a system that achieves 61.873% accuracy in classifying music genres. This paper discusses the methods we used for exploratory data analysis, feature selection, hyperparameter optimization, and eventual implementation of several algorithms for classification.**

I. Introduction

HISTORICALLY, attempts made by others to build music genre classification systems have yielded fine but not extraordinary results. Some who used the same dataset that we used for our project, which comprises one million songs each belonging to one of ten different genres, often chose to reduce the number of genres they attempted to identify, while those who decided to work with all ten genres only achieved accuracy of 40% [1].

The hardest part of music classification is working with time series data. The cutting edge of machine learning research is looking into more effective ways of extracting features from time series data, but is still a long way off. The most promising attempt is done by Goroshin et. al in their paper *Unsupervised Feature Learning from Temporal Data* [2].

The problem with temporal data is that the number of features varies per each training example. This makes learning hard, and in particular makes it hard to extract information on exactly what the algorithm is using to "learn". Algorithms that extract temporal features can be thought of as extracting a more reasonably sized set of features that can be learned from. To address the difficulties of working with temporal data, we used the idea of an acoustic fingerprint [3] to extract a constant sized number of features from each song, improving our accuracy on the testing set substantially.

We also used Bayesian Optimization, a relatively modern technique, to optimize the hyperparameters of our classification algorithms. This increased our accuracy by a few percentage points, and is substantially faster than grid search and leads to better results than random guessing.

In short, the task of developing a music genre classification algorithm is not a trivial one, and often requires substantial tuning. The best models we discuss in this paper achieve an accuracy of 61.9%.

II. The Dataset and Features

For this project we used the Million Song Genre Dataset, made available by LabROSA at Columbia University [4].

This dataset consists of one million popular songs and related metadata and audio analysis features such as song duration, tempo, time signature, timbre for each "segment", and key. In this dataset, each training example is a track corresponding one song, one release, and one artist. Among the many features that this dataset contains for each track is genre, which was of obvious importance to our project as it is the "ground truth" which we aimed to predict with our models. In particular, there are 10 genre lables, distributed according to Table I. This dataset is clearly imbalanced, making the ability to accurately classify genres other than "classic pop and rock", which makes up about 40% of the dataset, a challenge.

We changed the original dataset so as to not increase the size of the features to over 100, while adding to the ability to differentiate between genres. The original Million Song Dataset contains timbre vectors for each segment of each song, where a segment is defined as a small section of the song, usually denoting when a new note comes in. The creators of the dataset performed MFCC on the data, which is explained in greater detail in Sahidullah and Saha [5], and then performed PCA on the results of MFCC to separate the timbre of each segment into 12 principle components. The genre dataset only includes the average and variance of each of these segments, which we felt did not contain enough information to describe the data. To remedy this issue, the solution we came up with was to take the most meaningful segment of the song, using a variation on the idea of an acoustic fingerprint [3].

To achieve this, we used a variation on the idea of an acoustic fingerprint [6]. Research shows that when a person is listening to a piece of music, the brain's attention is heightened during transitions, which frequently coincide with the loudest parts of the music. As such, we decided determine the loudest point of each song in our dataset and use the corresponding timbre feature vector for that segment, an idea that is based upon work done by Vinod Menon at the Stanford School of Medicine [7]. The use of this feature improved our model accuracy to approximately 58% from around 55% originally. For further improvements, we multiplied the loudness of that particular segment and the individual timbres of that segment, which boosted accuracy to 60%.

To get a better understanding of the dataset, we first normalized and scaled the dataset, and then plotted the data using PCA in Fig. 2. We then plotted the data with the new added features, which does seem to have slightly better clustering of

classes in Fig. 3.

III. Methods

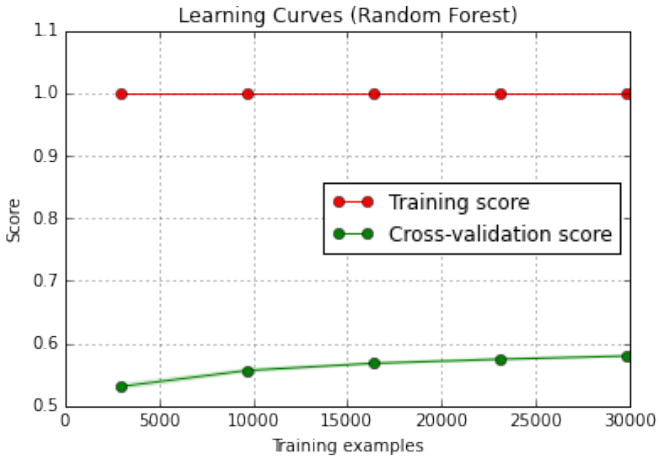

Fig. 1: Random Forest Learning Curve

A. Random Forest

After completing preliminary exploratory data analysis, we shifted our focus to various supervised learning methods, turning first to tree-based methods, which segment the feature space of a data set into distinct, non-overlapping regions using a greedy approach known as recursive binary splitting. Prediction of a test example using a classification tree is performed by determining the region to which that example belongs, and then taking the mode of the training observations in that region. A classification tree is "grown" by successively splitting the predictor space at a single cutpoint which leads to the greatest possible reduction in some chosen measure. For our project, we chose as our measure the Gini index (G), which is often thought of as a measure of region purity:

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

where K is the number of classes and $\hat{p}_{mk}$ is the proportion of training observations belonging to the kth class in the mth region. Thus, for every iteration of recursive binary splitting, the splitpoint s is chosen so as to minimize the Gini index and thus maximize the purity of each of the regions. This splitting of the feature space continues until a stopping criterion is reached (for example, if each region in the feature space contains fewer than a certain number of training observations). While one single classification tree might not have extraordinary predictive accuracy in comparison to other machine learning methods, aggregation of decision trees can substantially improve performance. For our project, we chose to use a random forest of classification trees as one of our prediction methods. Random forests are constructed by building a number of decision trees on bootstrapped observations, where each split in each tree is decided by examining a random sample of p out of n features; in this way the random forest method is an improvement upon simple bootstrap aggregation of trees as it decorrelates the trees [8]. For the purposes of our project, we used python's sklearn package to construct a random forest on our test data set. We first used Bayesian optimization, which is explained in greater detail in section 5, to tune several hyperparameters. The results of this optimization indicated that the optimal minimum number of samples required to split an internal node in a tree is 2, the optimal minimum number of samples that must be in a newly created leaf is 1, and the optimal number of trees in the forest is 300. Having tuned these parameters, we built a random forest classifier using our training set, which we then used to make genre predictions for our test set. For each split we considered $p = \sqrt{n}$ features. The random forest achieved accuracy of 61.9% and an F1 score of 57.9%. The learning curve for this method is shown in Fig. 1, and while our results seem to be an improvement upon those of previous years, the random forest suffers from high variance. That is to say, the model captures too much of the variance and noise in the training set, and as a result the generalization error suffers.

TABLE I: MSD Genre Dataset Statstics

Genre	Counts	Percent
classic pop and rock	23,895	40.09%
classical	1,874	3.14%
dance and electronica	4,935	8.28%
folk	13,192	22.13%
hip-hop	434	0.73%
jazz and blues	4,334	7.27%
pop	2,103	2.71%
metal	1,617	3.53%
punk	3,200	5.37%
soul and reggae	4,016	6.74%
Total:	59,600	100%

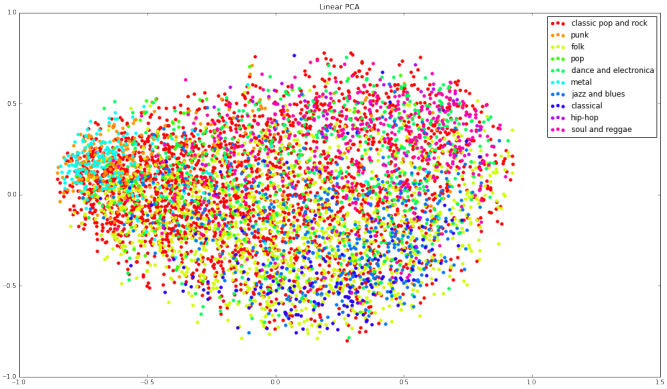

Fig. 2: PCA With Original Features

B. Gaussian Discriminant Analysis

In addition to tree-based methods, we also explored two different types of Gaussian discriminant analysis. First, we performed linear discriminant analysis on our training set using interactive polynomial features of degree 2. Before running the analysis, we created the interactive features by computing the (element-wise) product of every pair of distinct

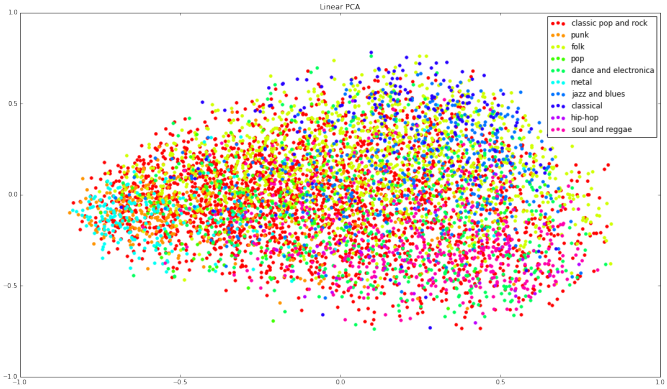

Fig. 3: PCA With Added Features

features. That is to say, for every pair of features n_1, n_2 where $n_1 \neq n_2$, we added $n_1 * n_2$ to our feature space. After completing this pre-processing, we used python's sklearn package to perform linear discriminant analysis. This method assumes that our observations $x_1, x_2, \ldots x_n$ are drawn from a multivariate Gaussian distribution with class-specific mean vectors and a common covariance matrix. Thus, the classifier assumes that an observation from the jth class, for example, is drawn from a distribution $N(\mu_j, \Sigma)$. The assumption that each class has the same covariance matrix Σ forces the decision boundaries in the feature space generated by linear discriminant analysis to be linear. The diagnostic results of this analysis are summarized in Table II.

TABLE II: LDA Diagnostics

Genre	Precision	Recall	F1-score	Support
Classic pop and rock	0.61	0.69	0.65	4752
Punk	0.49	0.53	0.51	617
Folk	0.67	0.59	0.63	2680
Pop	0.27	0.23	0.25	327
Dance and electronica	0.61	0.49	0.54	1029
Metal	0.60	0.70	0.65	415
Jazz and blues	0.60	0.46	0.52	846
Classical	0.66	0.71	0.69	367
Hip-hop	0.22	0.53	0.32	98
Soul and reggae	0.42	0.37	0.39	789
Average / Total	**0.59**	**0.59**	**0.59**	**11920**

C. Bayesian Optimization

Bayesian optimization tries to address the global optimization problem

$$x^* = \arg\max_{x \in \mathcal{X}} f(x)$$

where f(x) is a black box function, meaning that we have knowledge of the input and output of the function, but have no idea how the function works. Bayesian optimization is a way of solving this optimization problem effectively.

To use Bayesian optimization, we first decide on a prior distribution for the unknown functions. This essentially captures our belief about the family of functions that the unknown function comes from. The then decide on an acquisition function, which is used to select the next point to test [9].

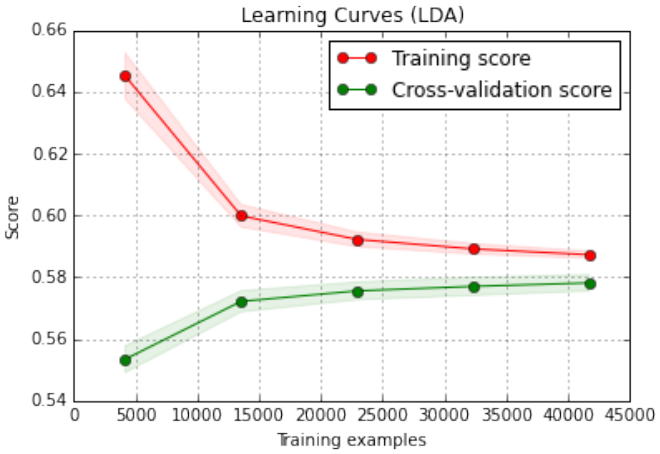

Fig. 4: Linear Discriminant Analysis Learning Curve

1) Gaussian Processes: As the prior to our Bayesian optimization problem, we used a Gaussian Process (GP). Gaussian Processes are defined by a multivariate Gaussian distribution over a finite set of points [9]. As more points are observed, the prior is updated, as in 5. What makes Gaussian processes useful for Bayesian optimization is that we can compute the marginal and conditional distributions in closed form. For more information on Gaussian Processes, consult Rasmussen and Williams [10].

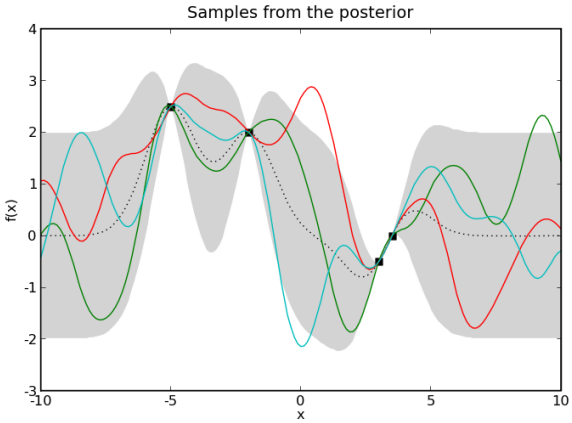

Fig. 5: Gaussian Process Prior

2) Acquisition Function: In order to optimize our function, we want to test points that have the highest chance of being an argmax. Our function $f(x)$ is drawn from a Gaussian process prior, meaning $y_n \sim \mathcal{N}(f(x_n), \nu)$. The acquisition function is the function the chooses the next point to try. With a Gaussian process, there are a few acquisition functions to choose from, but the one we used was expected improvement.

Expected improvement chooses the point that maximizes the expected improvement over our current best point [11]. With a Gaussian process, this has a closed form

$$\begin{aligned} a(x|\mathcal{X}_{1:t}) &= \mathbb{E}\left(\max\{0, f_{t+1}(x) - f_t(x^*)\}|\mathcal{X}_{1:t}\right) \\ &= \sigma\left(-u\Phi(-u) + \phi(u)\right) \end{aligned}$$

Classifier	Precision	Recall	F1
Bernoulli Naive Bayes	0.54	0.57	0.54
Multinomial Naive Bayes	0.53	0.50	0.50
Latent Dirichlet Allocation (10 topics)	0.36	0.54	0.41
Latent Dirichlet Allocation (100 topics)	0.36	0.54	0.41

TABLE III: Results on 10500 training size, 4500 test size

$$\text{s.t } u = \frac{y_{\max} - \mu}{\sigma}$$

We will not go into the details of the closed form here, but see *Snoek et al.* for details [9].

3) Bayesian Optimization of Algorithms: We used Bayesian optimization to tune the Random Forests algorithm and the SVMs. SVMs did not perform very well on the data originally, but after tuning the hyperparameters, the F1-score increased from around .45 to .51. This is a non-negligible increase that required performing only 25 iterations of Bayesian optimization in a couple of hours, as opposed to the hundreds of iterations required by grid search, which did not even finish running on the dataset provided after 12 hours. For random forests, the accuracy of the classifier increased from 59.1% to 61.9% using Bayesian optimization. This is an increase of about 3% percent, with minimal work from the implementation side. Bayesian optimization took about an hour to find a reasonable optimum, while grid search again did not complete in a reasonable amount of time. The paramater space examined by each of these algorithms contained $60,000$ different configurations of hyperparameters, so the fact that Bayesian optimization was significantly more efficient is not a surprise.

D. Lyrical Modeling

The Million Song Dataset supplied a subset of data of $210,519$ songs with lyrics in a bag-of-word (term frequency counts) format. From this data set we were able to extract those songs contained in the Million Song dataset to perform supervised learning on songs with lyrical information.

1) Naive Bayes: Using the naive bayes algorithm on the terms contained in each document we were able to construct a generative model to predict the document's genre. We preprocessed the data by removing common stop words. First we implemented Bernoulli Naive Bayes in which the features are $|V|$-length vectors indicating if a term appeared in the document or not. Next we utilized the term count in the data by running Multinomial Naive Bayes.

2) Latent Dirichlet Allocation: Latent Dirchlet Allocation models documents as coming from some number of topics, each word has a probability of being genereated by some topic. We hypothesized that the data could be modeled as documents containin words generated from topics relating either directly or indirectly to genre. By running Latent Dirichlet Allocation (using the Gensim python package) with varying number of topics we were able to generate a feature set for each document that was the fraction of the document accounted for by each model. Running these features through a SVM with a linear kernel we generated class predictions

IV. Results

A. Lyrics Analysis Results

The result of analysis of lyrical data was that Bernoulli Naive Bayes performed the best. Dividing songs into topics using Latent Dirichlet Allocation did not result in classification gains, regardless of the number of topics. The results could potentially be improved by using a larger data set; only $15,000$ of the $59,600$ labeled songs in the Million Song Dataset had lyrical information. Additionally, the fact that many songs across genres use similar words and the fact that the dataset was multilingual affected the accuracy of classification.

B. Sound Results

Model	**Training Error**	**Test Error**	**Test F1 Score**
Random Forests	0.03	0.38	0.58
Linear Discriminant Analysis	0.40	0.41	0.59

TABLE IV: Results on Pure Sound Data

V. Diagnostics

A. Confusion Matrix

In this section, we analyze how our primary models performed, and discuss reasons why they did not perform even better.Specifically, we will analyze why certain genres were commonly confused with others. Confusing classic pop and rock and folk was common with our models. This makes sense, as both have similar time signatures and generally use acoustic instruments. They also have similar loudness, which makes differentiation even more difficult. Punk was often confused with classic pop and rock, due to the fact that the Punk sample size was relatively small. Additionally, these genres tend to have similar time signatures and make heavy use the electric guitar. Pop again was commonly confused with classic pop and rock, which makes sense because even humans would have trouble discerning between "classic pop" and "pop". Most surprising, however, was the fact that dance and electronica was also often confused with classic pop and rock, potentially due to a lack of training examples (as was the case with Punk). Our hypothesis is further confirmed because of the precision and recall of the dance and electronica class. The precision is quite high, at around .711, while the recall is lingering around .5. The likely explanation is the class imbalances, as classic pop oand rock is the biggest class and thus has the highest prior probability. Metal actually did very well in both precision and recall, again being confused for classic pop and rock the most frequently due to class imbalances. Jazz and blues had very good precision, yet again bad recall, which again is probably due to class imbalances. It seems that classical did the best in both precision and recall, which makes sense because it generally has a very different timbre than most other genres. Hip-hop, the smallest class, does very poorly, and is not worth talking about as there is not enough data to make a comment. Soul and reggae again has very good precision, but the recall is not great. We believe this is again due to the fact that it is mostly acoustic, and because of the class imbalances.

TABLE V: Confusion Matrix for Linear Discriminant Analysis

3351	202	531	101	169	81	102	26	46	190
183	318	19	10	17	58	4	7	12	22
866	20	1522	33	10	3	85	50	12	23
145	8	40	76	15	3	3	3	1	17
271	20	41	14	487	27	25	16	34	42
73	37	4	1	11	285	3	3	0	2
205	8	99	12	52	8	415	41	4	9
38	1	24	3	12	3	7	283	2	4
18	3	1	1	12	0	0	1	37	18
313	4	43	25	64	1	12	1	44	312

TABLE VI: Confusion Matrix for Random Forest

4115	30	439	0	69	24	26	7	0	18
412	196	8	0	5	24	1	1	0	11
1256	3	1318	0	11	3	21	14	0	0
286	0	30	6	6	1	0	0	0	7
589	1	67	0	285	3	23	10	0	15
156	34	3	0	1	218	0	0	0	0
435	0	138	0	8	0	258	19	0	1
75	0	38	0	6	1	10	254	0	0
76	1	1	0	7	0	0	1	0	4
631	0	42	0	35	0	1	1	0	124

B. Learning Curves

After feature selection, both of our models performed similarly on the test set. This implies that more feature selection work could have been done. The reason for this is that with both a high bias algorithm, which underfits the data, and a high variance algorithm, we get similar results (see Fig. 4 and Fig. 1). This implies that perhaps the features are not descriptive enough.

VI. Future Work

Given more time we would work to develop or improve on several ideas. The first idea we had was to extend the fingerprint algorithm to find the top 3 sections of the song, rather than the top 1. Based on these 3, we would calculate the average and variance of the timbre vectors multiplied by the loudness of each segment. This hopefully would provide more data as to how different parts of the song interact, and is a way of keeping our feature vectors small, while adding to the acoustic fingerprint.

The second method we wanted to try was an ensemble method to combine classifiers. Some of the classifiers had high bias, such as Linear Discriminant Analysis, and some had high variance, such as Random Forests. An interesting topic that we have not had time to explore is combining these models (along with a lyrical classifier where the data exists) using an ensemble method, such as Bayesian model averaging (BMA) [12]. Unfortunately, we did not have the time to code an implementation of BMA.

Further work could also have been done with applying deep learning to this problem. A few viable methodologies might be to use techniques described in *Unsupervised Feature Learning from Temporal Data* [2] for unsupervised feature extraction, or use a supervised recurrent neural net to perform classification directly. These algorithms would have been able to take into account all of the time-series features of each song, possibly allowing for greater prediction accuracy.

VII. Conclusion

To conclude, we have constructed a relatively accurate classifier to determine the genre of a given song. Processing the acoustic features of the Million Song Dataset by identifying the most significant (loudest) segment and feeding this data into a Random Forest classifier, we were able to achieve 61.873% accuracy on our test set, after using Bayesian Optimization to tune hyperparameters. Steps such as expanding the dataset and feature set and using ensemble methods to combine classifiers have the potential to improve upon on results in the future.

References

[1] Miguel Francisco and Dong Myung Kim, "Music genre classification and variance comparison on number of genres," Stanford University, Tech. Rep., 2013. [Online]. Available: goo.gl/Nl6JzA

[2] Ross Goroshin, Joan Bruna, Arthur Szlam, Jonathan Tompson, David Eigen, and Yann LeCun, "Unsupervised feature learning from temporal data," in *Deep Learning and Representation Learning Workshop*, 2014. [Online]. Available: goo.gl/ZNVtFC

[3] "Evaluation tools for persistent association." [Online]. Available: http://mpeg.chiariglione.org/standards/mpeg-21/evaluation-tools-persistent-association

[4] "MSD genre dataset." [Online]. Available: http://labrosa.ee.columbia.edu/millionsong/sites/default/files/AdditionalFiles/msd_genre_dataset.zip

[5] M. Sahidullah and G. Saha, "Design, analysis and experimental evaluation of block based transformation in MFCC computation for speaker recognition," *Speech Communication*, vol. 54, no. 4, pp. 543–565, May 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167639311001622

[6] E. Wold, T. Blum, D. Keislar, and J. Wheaten, "Content-based classification, search, and retrieval of audio," *IEEE MultiMedia*, vol. 3, no. 3, pp. 27–36, 1996.

[7] M. Baker, "Music moves brain to pay attention, stanford study finds." [Online]. Available: http://med.stanford.edu/news/all-news/2007/07/music-moves-brain-to-pay-attention-stanford-study-finds.html

[8] G. James, D. Witten, T. Hastie, and R. Tibshirani, Eds., *An introduction to statistical learning: with applications in R*, ser. Springer texts in statistics. New York: Springer, 2013, no. 103.

[9] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 2951–2959. [Online]. Available: http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf

[10] C. E. Rasmussen, *Gaussian processes for machine learning*, ser. Adaptive computation and machine learning. Cambridge, Mass: MIT Press, 2006.

[11] H. J. Kushner, "A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise," *Journal of Fluids Engineering*, vol. 86, no. 1, pp. 97–106, Mar. 1964. [Online]. Available: http://dx.doi.org/10.1115/1.3653121

[12] J. A. Hoeting, D. Madigan, A. E. Raftery, and C. T. Volinsky, "Bayesian model averaging: A tutorial," *Statistical Science*, vol. 14, no. 4, pp. 382–401, Nov. 1999. [Online]. Available: http://www.jstor.org/stable/2676803