Music Discovery Using Active Learning

Adithya Manjunath
adithyam@seas.upenn.edu

Erik Marks
rekmarks@seas.upenn.edu

Abstract

In 2018, music streaming services are at the peak of their popularity, with millions of songs across genres and decades available at a user's fingertips — it is safe to say that services like Spotify and Apple Music have fundamentally changed the way music is consumed. However with this near-unlimited access to content comes an inevitable complication — content curation. In this paper, we explore the use cases of active learning to first discern and then recommend relevant new music to a Spotify user, which in a real-world setting could be integrated into Spotify's new user onboarding process. Our results show that there likely exists people whose taste in music, broadly speaking, can be estimated using 15-20 labeled instances.

1 Introduction

The music streaming industry is booming. By mid-2018, music streaming service users in the United States had already exceeded 268 billion streams, up 45% from 12 months previously (Nielsen, 2018). With a worldwide selection of artist catalogues spanning more than a century, millions of songs and billions of playlists, accessing music has never been easier. Although solving the accessibility problem is a great achievement, it also increased the urgency of solving another problem content curation. Any digital content distributor will consider the problem of content curation — and for Spotify (the leader in the music industry with over 3 billion playlists) (Spotify, 2018) a user's experience of finding new music in this time of near-unlimited access has, if anything, become harder than ever.

Our goal is to address this problem by augmenting Spotify's content curation capabilities by means of its Web API and active learning. By creating an experiment that in some ways could be utilized within the onboarding experience for a Spotify user, where by presenting a small sample of songs to a user, and learning from how the user responds to each song, the active learner gets closer to a point of being able to discern the users taste. We will also experiment with clustering using genres and unsupervised learning by training classifiers by cluster to see how they compare to our baseline learners.

One of the great challenges of recommendation or content curation is the subjective nature of the problem. To be useful, our classifiers and training data must be tied to specific users. Our data is sourced by querying the Spotify Web API via Spotipy, a Python client for the Spotify Web API, for data and feature on songs we have saved in our own respective libraries. The notion here being that this experiment could be further expanded into the entire universe of Spotify music for a new user as they signed up to become a Spotify user.

Previous research and attempts to programmatically aid users to discover music they like has been done in both academic and professional settings. Spotify, for instance, has tried multiple approaches from hiring editors to using collaborative filtering (Spotify, 2017), while other researchers have experimented with giving users more control of their recommendation systems by creating a user interface to allow users to control of what features to focus on (Millecamp et al., 2018) among other experiments.

2 Problem Definition

Formally, the problem is to, given a user u and a set of tracks[1] T, output the tracks $T' \subseteq T$ that the user would like. We approach this problem using linear classifiers in an active learning framework.

[1] In Spotify parlance, songs are referred to as tracks.

Spotify's Web API allows developers to access a range of information about the music available on the platform, which form the majority of our inputs. Some data encode music-theoretical values (e.g. loudness, tempo, key, and duration), while others pertain to features defined by Spotify (e.g. danceability, speechiness, acousticness, and liveness). The API also allows developers to download information pertaining to artists, specifically their popularity, number of followers and platform, and perhaps most crucially, genres. For instance, Meshuggah, a fairly popular Swedish metal band, falls under ten genres — alternative metal, death metal, djent, groove metal, jazz metal, metal, nu metal, progressive metal, swedish metal, and technical death metal. All input values were scaled to be on a 0-1 scale, given most classifiers are sensitive to differences in scale.

Our goal with using k-means clustering is to test the notion of using a cluster as a way to further categorize music — not unlike the way Spotify does with their genres. Given that there is little information as to how Spotify categorizes their content into genres — one would assume the granularity of their genre options would indicate that this is done with manual effort in some part of their process — using clusters built on the rest of our features provides us a programmatic way to further categorize our songs, and consequently give us another feature to utilize while training our classifier (namely, which cluster a song belongs to).

3 Algorithms

3.1 Learning Algorithms

The purpose of active learning is to obtain a satisfactory classifier with the smallest possible number of examples. Active learning frameworks are generally learner-agnostic, excepting data-hungry classifiers such as deep neural networks.

We assumed that the data would generally be linearly separable, and used various support vector machines as our learners. Specifically, we used the following algorithms from `scikit-learn`:

- `SVC`
- `LinearSVC`
- `SGDClassifer`

3.2 Active Learning Algorithm

An active learning algorithm, generally, takes as input an initialized[2] learner and some unseen training data, and outputs the most "informative" training instance. Most such algorithms differ in their definition of "informativeness," of which there are many kinds. Our chosen method was uncertainty sampling, which defines the most informative instance as the instance that the learner is most uncertain about. For SVMs, including all learners we used, this is the instance closest to the separating hyperplane. Our work active learning algorithm design is informed by (Settles, 2012).

Formally, our algorithm proceeds as follows:

- Given a learner L and training data D
- Obtain a random sample $D_{init} \subseteq D$, update D s.t. $D = D - D_{init}$, and initialize L s.t. $L = L(D_{init})$
- Until some termination criterion:
  - Retrieve the most uncertain instance x from D
  - $L = L(x)$
  - $D = D - \{x\}$

In a production setting, the termination criterion can only be set heuristically. For our experiments, we arbitrarily terminated the algorithm after 50 iterations.[3] Note that, for experimental purposes, we also implemented our active learner with random sampling.

4 Experiments

4.1 Expectations

In the main, we had the following expectations:

1. Our data was linearly separable
2. The accuracy of our active learners would approach that of a benchmark on all examples, in less than 50 iterations
3. Active learning with uncertainty sampling would outperform active learning with random sampling
4. Clustering would improve the accuracy of all learners, including benchmarks

[2] Using some appropriately small number of instances. We initialized our learners with 5 randomly sampled training instances.

[3] There were no dramatic performance changes after 50 iterations, and in our guiding user onboarding usecase, the user's stamina will run out long before listening to 50 tracks.

4.2 Methodology

For our baseline SVM model, evaluation will in practice require us to confirm/refute the labels provided by the classifiers on the test data. Following this, we can compute all the usual performance measures (accuracy, precision, recall etc.)

For the baseline models, we simply predicted on the full training data outright to find its accuracy. For each active learner, we then ran the complete active learning protocol. We could have run them until they reached benchmark accuracy, but found that 50 iterations was sufficient to give us a sense of how many songs Spotify would need to present to a new user to get a sense of what their music taste is like. We report the accuracy after initialization and after each iteration.

The training and test data used comes from a set of 5000+ examples that we downloaded by querying the Spotify Web API via Spotipy, a Python client for the Spotify Web API. We downloaded the personal song data of one of the authors, labeling saved songs as positive (under the assumption that users generally save songs that they like), and generating negative examples by finding, saving, and querying some number of songs that the author disliked.

For our tests with k-means clustering, we first tested cluster sizes with our given data to see at what points the change in inertia (or the within-cluster sum-of-squares) with each additional classifier were worth ignoring to pick our number of clusters. Our intuition with this was to ensure we didnt pick too high a number here given having too many clusters would not necessarily add any useful information to our set of features. We used several numbers of clusters to cluster the songs in our dataset. To make the cluster values available to our learners, we encoded them as boolean features, of which only one was true for any given feature.

In addition to benchmarks, we ran experiments using all combinations of the following options using our active learning protocol, for all classifiers:

- Clusters
  - Clustered
  - Unclustered
- Sampling Method
  - Random
  - Uncertainty

Algorithm	Unclustered	Clustered
SVC	65.8%	65.8%
LinearSVC	91.7%	91.7%
SGD	90.2%	91.9%

Table 1: Benchmark accuracies.

For all of the above, the entire training dataset was sampled. For an additional experiment using all sampling methods but only the **SGD** classifier, instead of selecting a new instance from the entire dataset every iteration, we selected one instance per cluster. When we speak of clustered and unclustered data, we indicate whether we added k-means cluster features to the data, or not. We repeated all experiments for various numbers of clusters. Prior to running our experiment suite, we designated a uniform 20% of the training data as our validation data, on which all accuracies are reported. Our experiments thus simulate a user interacting with a music recommendation algorithm.

5 Results

5.1 Preliminaries

All reported results use 4 clusters. Benchmark accuracies are summarized in Table 1. Plots are in the appendix, and show the accuracy after initialization (which is always with 5 examples) and every added training instance thereafter. Each plot has separate lines for random sampling and uncertainty sampling. Samples were taken from the entire dataset, with the except of the plot named *SGD Cluster Sampling*, which for every iteration sampled once from each cluster.

Barring **SVC**, all benchmark learners attain greater than 90% accuracies. We observed no difference in accuracy between clustered and unclustered data for **SVC** or **Linear SVC**, but note that **SGD** performed marginally better with clustered data.

5.2 Active Learning

The results for our active learning protocol are ambiguous. In the **SVC** case, there is no improvement over the rather low benchmark accuracy when using random sampling. On some runs it would beat the benchmark, but only for an iteration or two. Using uncertainty sampling, on the other hand, **SVC** oscillates wildly between good accuracy ($> 80\%$) and terrible accuracy ($< 40\%$).

Linear SVC performed the best out of all classifiers, with its accuracy stabilizing at above 85% in all cases. For uncertainty sampling, it stabilizes faster using clustered data. However, random sampling stabilizes the fastest, at approximately 13 instances for both clustered and unclustered data.

SGD exhibits significant variation between iterations for both sampling methods, and oscillates much like **SVC** in the case of uncertainty sampling. Unlike **SVC**, however, it arguably performs better using unclustered data than clustered data in that case. Finally, with cluster sampling, **SGD** reaches accuracies comparable to **Linear SVC** after 17 instances using uncertainty sampling, but this does not hold. If randomly sampling, variance is also high.

5.3 Discussion

If you consider 85% accuracy to be sufficient, we have shown that there likely exists people whose musical tastes can, in broad strokes, be approximated using 15-20 labeled instances using linear models. We believe that this is a reasonable number of songs for our hypothetical usecase. Whether these results can be replicated for other users remains to be seen.

While we achieved our primary objectives, our uncertainty sampling method is a failure. In all cases, uncertainty sampling either underperformed or did no better than random sampling. We believe our uncertainty sampling method causes overfitting by choosing the points closest to the decision boundary. Where, for the purpose of generalization, a lot of these points should be ignored so as to increase the margin, we are explicitly telling our algorithm to focus on them. Thus, we produce the severe oscillations in **SVC** and **SGD**. **Linear SVC**, being a more robust learner, can still stabilize with uncertainty sampling, but does so faster using random sampling.

As to clustering, it also appears to have been an exercise in futility, with only **SGD** demonstrating even marginal performance benefits. In retrospect, we should probably not be surprised that clusters computed using the training data do not add any information about the data. We believe that clustering can be useful in making predictions for specific genres of music, but in the general case, k-means appear to be useless. Whether other methods can help is an open question.

6 Related Work

While common practices exist, curation is hardly a solved problem, and there does not appear to exist any commonly accepted best methods.

With Spotify, users have various ways to discover music. Each user's page has a "discover" tab that leads users to playlists curated specifically to their taste — the Discover Weekly playlist being the most popular of the lot, along with personalized song and album recommendations. The discover page also looks at past user listening history to recommend music more directly — for instance, if you have spent a fair amount of time listening to a particular artist over the past month or so (say, Gesaffelstein), a user will have a "similar to Gesaffelstein" recommendation that lists songs and/or albums that are in the same vein as that specific artist. Spotify also has a "radio" option for songs and artists, where a user is directed to a list of songs that are deemed "similar" to the song or artist they have clicked the radio option for. This is in addition to "daily mixes", and recommended songs that look at a particular playlist and spits out recommendations for that given playlist that perhaps are thematically aligned with said playlist (Spotify, 2017).

In terms of how machine learning is utilized, Spotify's approach can be broadly classified into usage-based collaborative filtering algorithms and content-based recommendations.

Spotify initially began recommending music via collaborative filtering. The idea here is to find patterns from a user's past behavior to generate recommendations. The benefits of this is that a given algorithm/approach is easily scalable. Songs are represented as vectors, as are users, with a user's profile based on their interactions on Spotify. By finding music within a given cluster that is close to a user vector (or music enjoyed by other user's that have similar vectors), Spotify can capture distinct user tastes and also make recommendations that are time sensitive given a user's profile constantly changes based on their user interactions. However, a downside to using collaborative filtering alone is the over-reliance on user data, in what is known as the cold start problem — where popular songs are far easier to recommend over unpopular songs given the imbalance in listening/usage data. This defeats the very purpose of a recommendation system, given users are less likely to be satisfied with a recommendation system that recommends songs

that they are already likely to have heard before. New and unpopular songs, within this context, are likely to be ignored — given there is no usage data to analyze (Dieleman, 2014).

Spotify then began various methods of content-based recommendation via supervised and unsupervised machine learning methods — with the supervised methods focusing on learning from the inherent audio characteristics of a song (time, amplitude, loudness, frequencies, etc.), to technical music features (melody, beats, voices, instruments, etc.), and finally other, broader features (mood, genre, popularity, etc.) The unsupervised methods include using deep learning to predict listening preferences from audio signals by training regression models to predict the latent representations of songs that were obtained from a collaborative filtering model (Dieleman, 2014).

Suffice to say, Spotify already uses a variety of information sources and algorithms in their recommendation pipeline.

Other researchers have also experimented with various recommendation systems within the context of music and Spotify.

For instance, Millecamp experimented with creating a user interface that allows users to interact with a recommendation system, giving them control of what features to focus on. This was an attempt to remove the black box nature of todays recommender systems (which do little in terms of giving users control of what sort of music they are looking for) by giving users more control to help steer the recommendation process with additional input and feedback (Millecamp et al., 2018).

Furthermore, Chen focused on the notion of automatically generating playlists for users via Latent Markov Embedding, formulating their problem as a regularized maximum-likelihood embedding of Markov chains in Euclidean space. Their aim was to provide users with a coherent listening experience, and interestingly enough, their methodology did not involve using any audio features available via Spotify, instead choosing to learn a representation via example playlists (Chen et al., 2012).

7 Future Work

Various improvements and further areas of inquiry immediately present themselves.

First, more active learning sampling methods should be investigated, for instance query-by-committee (Settles, 2012). In addition, some work has been done specifically on active learning with SVMs. Other authors suggest *pool-based active learning* using SVMs, where instances are sampled according to their ability to partition the instance space (Tong and Koller, 2001). This coheres with our intuition concerning genres, and is certainly worthy of further investigation. Dasgupta also presents and algorithms with significant deviations from ours that ought to be explored (Dasgupta et al., 2008).

Whatever active learning framework is used, the actual learning algorithms must be resistant to noise. For instance, a track that, given its features, should be a sure-shot for a specific user may not be. Moreover, these reasons can be obscure to the user themselves, to say nothing of the algorithm. Naturally, the reverse happens as well. Either way, there will be a lot of noise. While **Linear SVC** appears to perform admirably, other algorithms ought to be explored as well.

Finally, various applications of the active learning protocol should be investigated. The utility of the protocol should be evaluated on ”real” users in a production setting. Allowing users to ”play” with the protocol is likely to present use cases beyond the imaginations of its authors.

8 Conclusion

Presenting a user with less than 20 examples for labeling suffices to approximate the user's taste through an active learning protocol. While, typical data in the domain may be too noisy for uncertainty sampling. The possibility of way forward is certain. We hope to continue exploring this topic in future work.

References

Shuo Chen, Josh L. Moore, Douglas Turnbull, and Thorsten Joachims. 2012. Playlist prediction via metric embedding. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 714–722, New York, NY, USA. ACM.

Sanjoy Dasgupta, Daniel J Hsu, and Claire Monteleoni. 2008. A general agnostic active learning algorithm. In *Advances in neural information processing systems*, pages 353–360.

Sander Dieleman. 2014. Recommending music on spotify with deep learning. Online; retrieved: 2018-12-19.

Martijn Millecamp, Nyi Nyi Htun, Yucheng Jin, and Katrien Verbert. 2018. Controlling spotify recommendations: Effects of personal characteristics on music recommender user interfaces. In *Proceedings of the 26th Conference on User Modeling, Adaptation and Personalization*, UMAP '18, pages 101–109, New York, NY, USA. ACM.

Nielsen. 2018. Mid-year report (u.s. 2018). Online; retrieved: 2018-12-19.

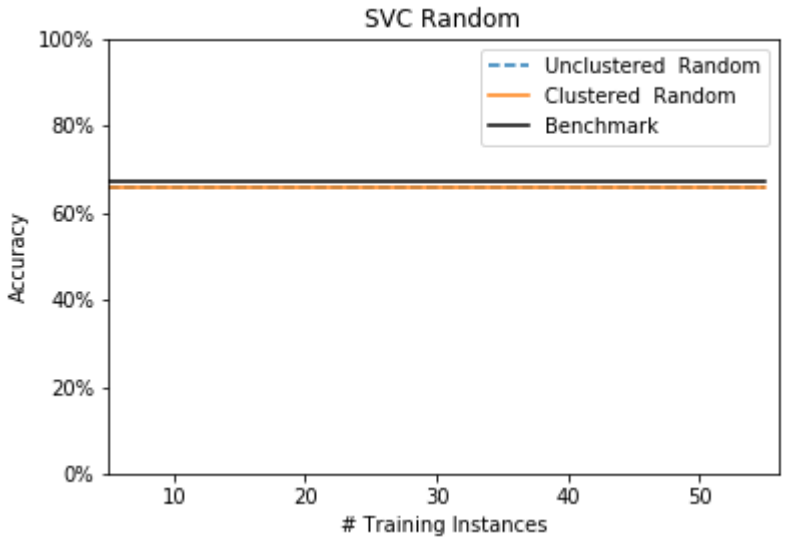
Burr Settles. 2012. Active learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–114.

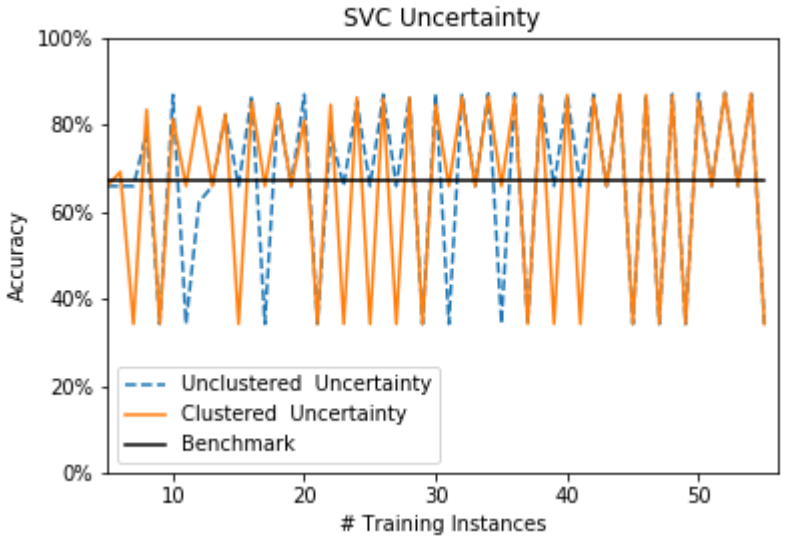
Spotify. 2017. Spotify machine learning solution for music discovery. Online; retrieved: 2018-12-19.

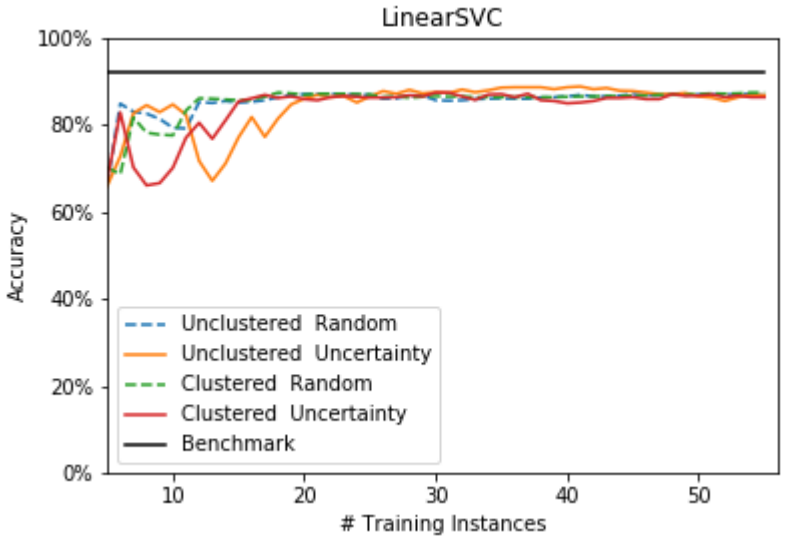
Spotify. 2018. Company info. Online; retrieved: 2018-12-19.

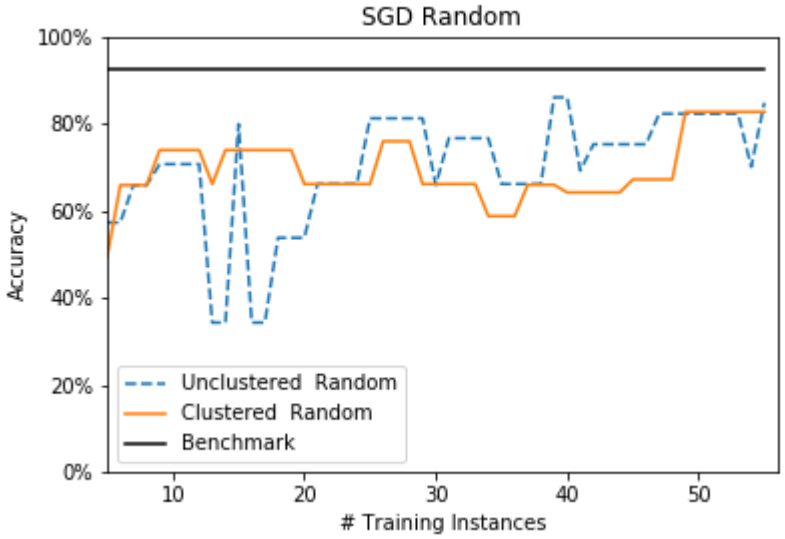
Simon Tong and Daphne Koller. 2001. Support vector machine active learning with applications to text classification. *Journal of machine learning research*, 2(Nov):45–66.

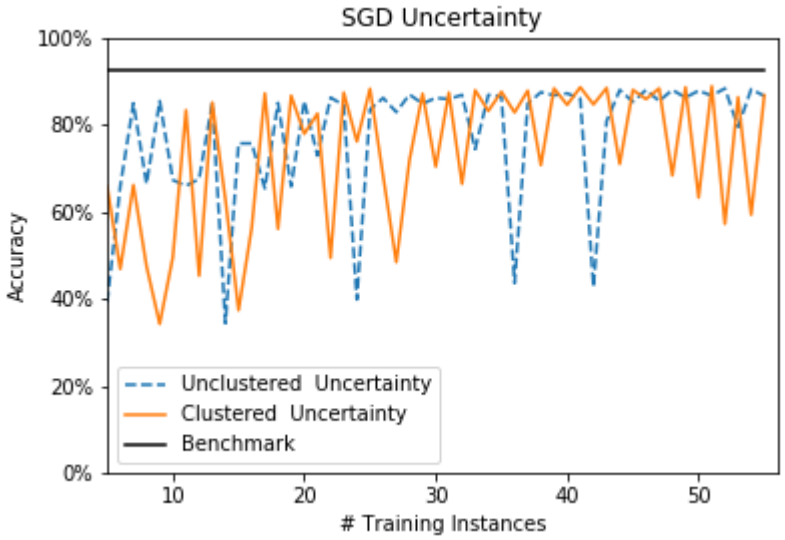
Appendix: Figures

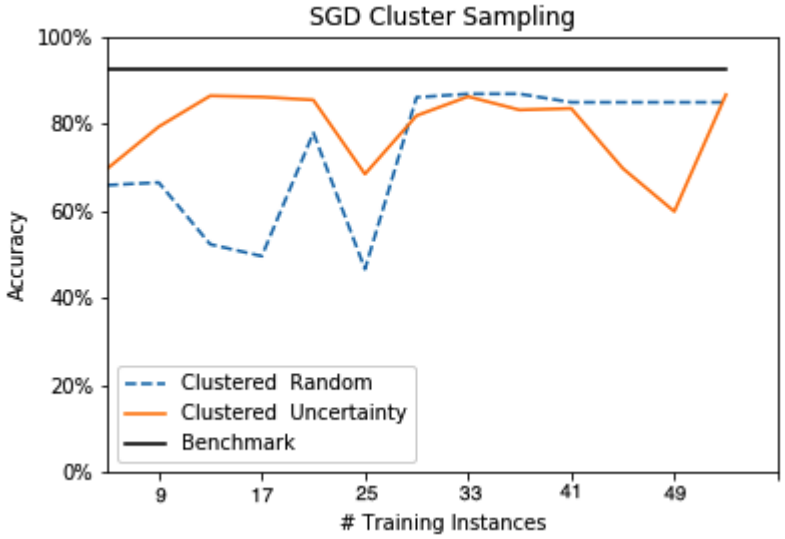





LinearSVC
100%
80%
60%
40%
20%
0%
Accuracy
10
20
30
40
50
Training Instances
Unclustered Random
Unclustered Uncertainty
Clustered Random
Clustered Uncertainty
Benchmark


SGD Random
100%
80%
60%
40%
20%
0%
Accuracy
10
20
30
40
50
Training Instances
Unclustered Random
Clustered Random
Benchmark


SGD Uncertainty
100%
80%
60%
40%
20%
0%
Accuracy
10
20
30
40
50
Training Instances
Unclustered Uncertainty
Clustered Uncertainty
Benchmark


SGD Cluster Sampling
100%
80%
60%
40%
20%
0%
Accuracy
9
17
25
33
41
49
Training Instances
Clustered Random
Clustered Uncertainty
Benchmark