Detecting Hand movements from EEG Signals[*]

Anirudhan J Rajagopalan, Michele Cerú[†]

Abstract. This project aims to detect and classify the grasp and lift hand movements of a subject from EEG (Electroencephalography) recordings. Successful identification and classification of the recordings will help in developing Brain Computer Interfaces (BCI) that can be used to restore the ability of patients to do day-to-day tasks. We are provided with $\tilde{2}4$ series of grasp and lift actions performed by twelve subjects. We run our models using the first $\tilde{1}8$ series as our training and the last $\tilde{2}$ series as our test set. We use the level1 predictions of the Kaggle contest winners as our baseline and try to compare the performance of our model against the level1 predictions of baseline. Using our simple pipeline we are able to show that we can get accuracy of 0.61 Mean Area Under the Curve (MAUC).

1. Introduction. Electroencephalography (EEG) is used to record electrical activity of the brain. It is typically noninvasive, with the electrodes placed along the scalp. In the Grasp And Lift (GAL) experiment twelve subjects were asked to perform lifting series in which the object's weight (165, 330, or 660g), surface friction (sandpaper, suede, or silk surface), or both, were changed unpredictably between trials, thus enforcing changes in fingertip force coordination.[9] The hand movement of the subject was recorded by 3D sensors which were synchronized with the EEG cap thus providing us with the exact moment at which the GAL events happen. With respecct to our goal of classifying the hand movements, we have to detect the following six events.

1. *HandStart*: Beginning of the movement.
2. *FirstDigitTouch*: Making contact with the object.
3. *BothStartLoadPhase*: Starting to load the object.
4. *LiftOff*: Holding the object up.
5. *Replace*: Replacing the object in its original position.
6. *BothReleased*: Releasing the fingers from the object.

The EEG signals recorded by 32 electrodes fixed to the scalp are recorded at a frequency of 500Hz. An added objective of this task is to make sure that we dont use any future data for doing predictions (No future data rule, as described in the data page of the challenge). This restriction is imposed to mimic the real life scenarios in which such an application can be used, wherein, we will not have access to any of the future data while making predictions in real life.[6]

This project aims to classify the hand movements of subjects by using EEG signal data. We compare the performance of LinearSVM and GaussianSVM combined with VLAD and Bag of Features (BOF) feature representations. We base our performance using the Area Under the Curve as described in the Kaggle challenge. We try to reason the performance of our model with respect to the dataset by analysing the spatial relationship and variance of the features. We then discuss about models that we might use to tackle this hard problem.

[*]Kaggle Grasp and lift competition. https://www.kaggle.com/c/grasp-and-lift-eeg-detection

[†]New York University (ajr619@nyu.edu; mc3784@nyu.edu). Questions, comments, or corrections to this document may be directed to this email address.

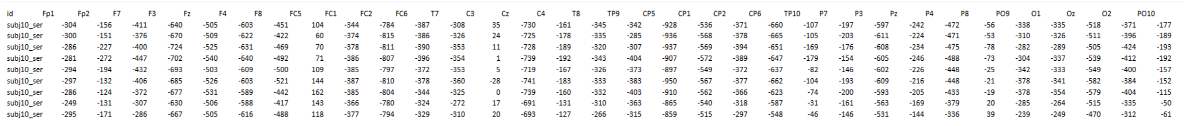
2. Dataset Description.

2.1. Sources and Format. The dataset is provided by Kaggle Inc. for their Grasp and Lift Detection challenge[6] sponsored by Way Consoritum[3]. The dataset consists of separate test and train zip files of size 153 MB and 915 MB respectively. The dataset expands to 447 MB and 3.1GB after extraction. Out of these datafiles, we cannot use the testset as they are being used for the competition submissions. The testset has only the *_data.csv files with no corresponding *_events.csv file. Since there is no *_events.csv file we will not be able to learn or evaluate the performance of our models by using the test dataset provided in the challenge website. So our effective usable dataset is only the training data which we partition into train and test set for training our models.

The dataset consists of two types of files:

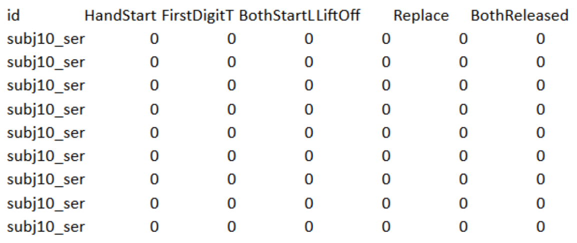
1. data.csv — CSV of series labels and values from the 32 electrode signals.
2. event.csv — CSV of series labels and event values.

The sample dataset is included below for reference.



id	Fp1	Fp2	F7	F3	Fz	F4	F8	FC5	FC1	FC2	FC6	T7	C3	Cz	C4	T8	TP9	CP5	CP1	CP2	CP6	TP10	P7	P3	Pz	P4	P8	PO9	O1	Oz	O2	PO10
subj10_ser	-304	-156	-411	-640	-505	-603	-451	104	-344	-784	-387	-308	35	-730	-161	-345	-342	-928	-536	-371	-660	-107	-197	-597	-242	-472	-56	-338	-335	-518	-371	-177
subj10_ser	-300	-151	-376	-670	-509	-622	-422	60	-374	-815	-386	-326	24	-725	-178	-335	-285	-936	-568	-378	-665	-105	-203	-611	-224	-471	-53	-310	-326	-511	-396	-189
subj10_ser	-286	-227	-400	-724	-525	-631	-469	70	-378	-811	-390	-353	11	-728	-189	-320	-307	-937	-569	-394	-651	-169	-176	-608	-234	-475	-78	-282	-289	-505	-424	-193
subj10_ser	-281	-272	-447	-702	-540	-640	-492	71	-386	-807	-396	-354	1	-739	-192	-343	-404	-907	-572	-389	-647	-179	-154	-605	-246	-488	-73	-304	-337	-539	-412	-192
subj10_ser	-294	-194	-432	-693	-503	-609	-500	109	-385	-797	-372	-353	5	-719	-167	-326	-373	-897	-549	-372	-637	-82	-146	-602	-226	-448	-25	-342	-333	-549	-400	-157
subj10_ser	-297	-132	-406	-685	-526	-603	-521	144	-387	-810	-378	-360	-28	-741	-183	-333	-383	-950	-567	-377	-662	-104	-193	-609	-216	-448	-21	-378	-341	-582	-384	-152
subj10_ser	-286	-124	-372	-677	-531	-589	-442	162	-385	-804	-344	-325	0	-739	-160	-332	-403	-910	-562	-366	-623	-74	-200	-593	-205	-433	-19	-378	-354	-579	-404	-115
subj10_ser	-249	-131	-307	-630	-506	-588	-417	143	-366	-780	-324	-272	17	-691	-131	-310	-363	-865	-540	-318	-587	-31	-161	-563	-169	-379	20	-285	-264	-515	-335	-50
subj10_ser	-295	-171	-286	-667	-505	-616	-488	118	-377	-794	-329	-310	20	-693	-127	-266	-315	-859	-515	-297	-548	-46	-146	-531	-144	-336	39	-239	-249	-470	-312	-61

Figure 1. *Sample Data format*



id	HandStart	FirstDigitT	BothStartL	LiftOff	Replace	BothReleased
subj10_ser	0	0	0	0	0	0
subj10_ser	0	0	0	0	0	0
subj10_ser	0	0	0	0	0	0
subj10_ser	0	0	0	0	0	0
subj10_ser	0	0	0	0	0	0
subj10_ser	0	0	0	0	0	0
subj10_ser	0	0	0	0	0	0
subj10_ser	0	0	0	0	0	0
subj10_ser	0	0	0	0	0	0

Figure 2. *Sample Event format*

The event dataset will have a value of 1 corresponding to the event (stimulus) that is being performed at the instant of sampling. There can be situations where there are multiple stimulus being triggered for the same EEG signal. The objective is to classify (or) label the input EEG signals to the corresponding stimulus.

The total number of samples across the complete dataset is 17985850.

The spatial relationship between each of the 32 electrodes is given by Fig. 3

2.2. Multilabeling or Multiclass classification. The dataset can have multiple stimulus at the same time as shown in figure 4. This is an important feature to consider as it forms the distinction between multiclass and multilabel classification problem.

3. Problem Statement. As described in the previous section, the objective is to classify/label the input EEG signals with their correspoinding stimulus. This is an inherently

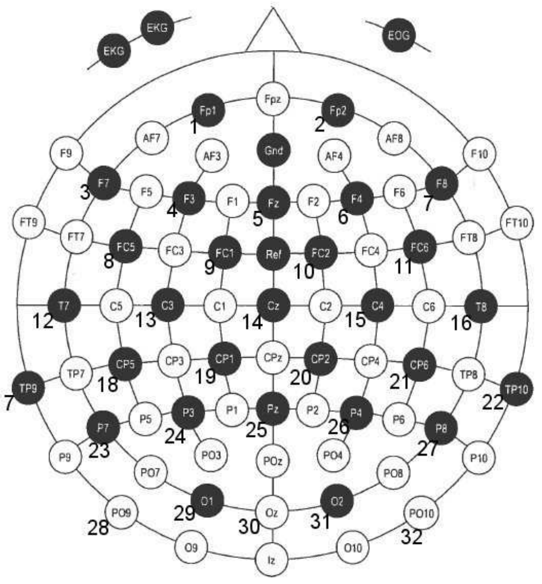
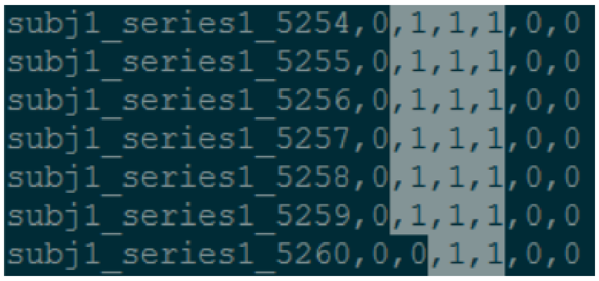


Figure 3. *Spatial relationship between 32 electrodes.*



```
subj1_series1_5254,0,1,1,1,0,0
subj1_series1_5255,0,1,1,1,0,0
subj1_series1_5256,0,1,1,1,0,0
subj1_series1_5257,0,1,1,1,0,0
subj1_series1_5258,0,1,1,1,0,0
subj1_series1_5259,0,1,1,1,0,0
subj1_series1_5260,0,0,1,1,0,0
```

Figure 4. *Multilabel or Multiclass classification problem.*

multilabel classification problem. We can reduce this to multiclass classification problem by considering the fact that the events happen in sequence. For example, BothReleased event is always followed by Replace which is always followed by LiftOff and so on and so forth. Thus the problem of finding all events corresponding to the EEG signal can be decomposed into the problem of finding the start of the occurance of a particular event.

The constraint defined above simplifies our problem into a multiclass classification problem. With proper feature selection and model selection we should be able to solve the problem effectively.

4. Baseline.

4.1. Choosing a baseline. We considered the models developed by Alexandre Barachant (a.k.a cat) and Rafal Cycon (a.k.a dog) for our baseline model (Cat & Dot solution)[2]. The solution by Cat and Dog won the first prize in the challenge and it makes logical sense to use that as our baseline. The cat and dog solution has three levels of models.

The first level of models uses

1. Covariance Matrices
2. Event Related Potentials (ERP)
3. Filter Banks

The solution uses an ensemble of multiple learning models such as Logistic Regression with LDA, Convolutional Neural Network, and Recurrent Neural Networks. This model fits our requirement to be a baseline and hence we choose the Level-1 model of *Cat and dog solution* as our baseline models. We decided to compare the performance of the baselilne with respect to our own model. The accuracy achieved by the Level-1 models are not readily available. So we tried to run the baseline to get the accuracy values.

4.2. Running the baseline. The hardware specs listed in the *Cat and Dog solution* page details the hardware used and the time taken for the entire pipeline. Considering that we are using only the Level-1 models as our baseline, we decided to try running the baseline to get a accurate idea of time and performance for Level-1 alone.

4.2.1. Attempt 1. We first attempted to run the Level-1 models in our own machines just to get an idea of how much memory and time requirements the Level-1 took. We started out by installing all the packages specified in the requirements document and then started with the preprocessing step. The preprocessing step ran successfully even in our local machine and created two numpy binary files *infos_val.npy* and *infos_test.npy*. The numpy files are to be consumed by the baseline files for building the models.

We ran the baseline using *lvl1/genAll.sh* in our local. The baseline crashed due to out of memory. Thus we decided to run all our experiments in New York University (NYU)'s High Performance Computing (HPC) clusters[5].

4.2.2. Attempt 2. The second attempt[11] involved trying to run the Level-1 code using NYU HPC clusters.

1. As used in the first attempt, we decided to treat the baseline as a blackbox and try to run the baseline code.
2. We identified out the entry point for the baseline models and decided to run the model in NYU HPC clusters.
3. We figured out the modules required to make all the python packages used in the baseline to work and loaded the modules.
4. We submitted a HPC job for running the baseline. The job requested for the exact number of CPU nodes, CPU cores, GPU resource and RAM mentioned in the Hardware specs.

The second attempt to run the baseline also ended without success as the system crased due to mismatch between required packages and the available packages in HPC.

4.2.3. Attempt 3. Since running the baseline on HPC crashed and running the models on our local resulted in 'out of memory' error, we decided to run the baseline in our local using a fraction of the sample.

1. We started out by trimming the input files from 18 series of GAL to just 6 series of GAL per subject.

2. This corresponds to just $\tilde{4}000$ lines of samples per input file and we were able to load the data in our local effectively.
3. We installed the exact version (including the minor versions) of almost all the packages specified in the baseline.
4. A few packages that were mentioned in the baseline were not available due to some mysterious reason and we had to install the previous version for these packages.
5. We ran the baseline with the subsample of the original dataset.

We still ran into a number of issues while running the code. Our guess is that the code in the *Cat and Dog Solution*'s repository has got some breaking changes after the final submission. We emailed the authors regarding the issue and even after help from the authors we were unable to get the baseline running.

4.3. Baseline Results. As discussed above, since we couldn't get the baseline running even after 3 attempts and the help of the authors and since we had spent more than 2 weeks on this problem, we decided to move ahead with building our own solution to this interesting problem.

The attempt to run the baseline gave us significant insights into the Domain, data and assumptions that can be used for solving the problem.

5. Building our own Pipeline.

5.1. Understanding the domain. The biggest challenge for building the pipeline was to understand the domain and figure out the preprocessing tasks. The components downstream the pipeline was already fixed to be VLAD[7][1] and Bag of Features. After a bit of exploring and a bit of guidance from our professor we decided to use MNE[4] for the data preprocessing. MNE provided us with a number of tools for preprocessing the data.

1. Epochs
2. Covariance measures
3. XDawn filter[12]
4. Algorithms to process the raw data and generate the Event Related Potentials.

We made good and extensive use of the codebase of *Cat and Dog* solution to understand the type of preprocessing that was carried out by them. We also went through the forum of the Kaggle challenge to gather insight about the domain and ways of processing the data.

5.2. Data Preprocessing. We used MNE to consume the raw data in csv format to find the epoch values and the Event Related Potentials. We also use MNE to get the covariance matrix on the complete training/test dataset. This information is then fed to xDawn filter with WINDOW size as 500. The value of the parameter num_components for xDawn, which determines the number of features in the processed data was varied from 2 to 4 and the resulting three dimensional array was saved in numpy binary formats.

The total run time for data preprocessing took around an hour for the dataset.

5.3. Exploring the preprocessed data. We then set out to explore the processed data to better understand the dataset.

5.3.1. Clustering of local descriptors. The total number of stimulus that has to be classified is six. Any classification task becomes easier if we find that the data is separable by

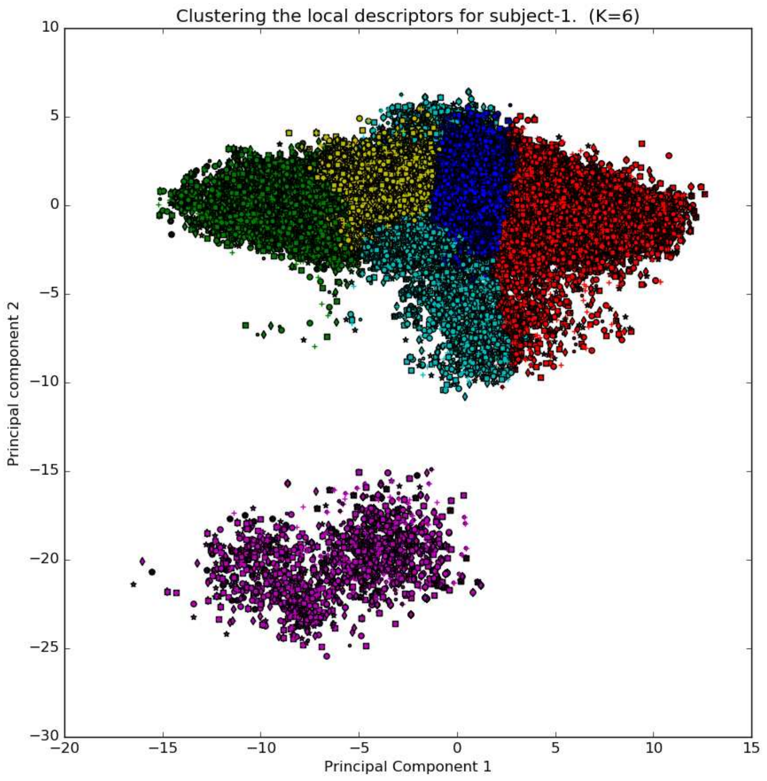


Figure 5. *Clustering the local descriptors of a single subject into 6 clusters. Different clusters are represented by different colors. The local descriptors that gives raise to a particular stimulus are denoted by different markers such as plus, star, circle, square and so on. Just a cursory look on the clusters makes it clear that there data is not separable easily.*

some degree. Since our model essentially depends on builing over the clusters formed by local descriptors, we decided to cluster the data to find if there is any amount of separatbility of data.

We ran experiments to cluster the local descriptors for one subject (Subject 1 in the dataset) using MiniBatchKMeans algorithm. We then used the cluster information and different markers to plot the cluster information on a two dimensional plot by reducing the dimensionality. The plot generated for n_components = 2 is shown as an example in Fig 5.

5.3.2. Variance of local descriptors. PCA is a technique for dimensionality reduction. It works by finding components that provide the maximum variance (It can be thought of as the direciton in which the features spread out the most). Finding the variance of the features helps us to understand how the data is spread and how much dimensions we should use to tackle the learning problem. We created a plot of the variance of a the local descriptors for a subsample of the dataset (Subject 1). The variance curve is as shown in Fig 6

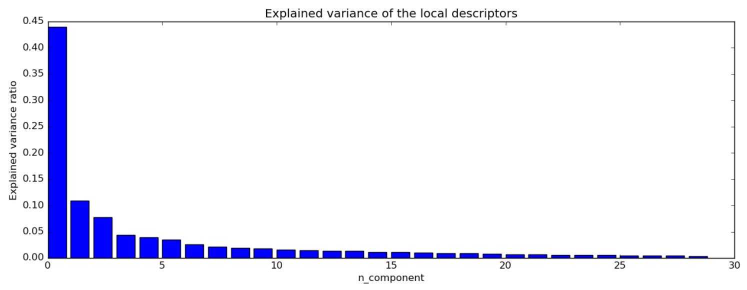


Figure 6. *Explained variance ratio of local descriptors for a subsample (Subject 1). We get variance measures of 99, 95, 90, 77, 65 and 50 for components 29, 21, 16, 7, 4, 2 respectively.*

The analysis of variance was actually performed after we got the results for BOF and Vlad models. This makes us wonder whether the model's performance might have increased if we had used PCA on the local descriptors before applying our models.

5.4. Building Bag of Words model.

5.4.1. Theorey. Bag of words feature representation is a way of representing a sample that can be separated into its local descriptors. We learn the vocabulary of the dataset using Kmeans with different cluster sizes. The vocabulary for a learning problem is the cluster centroids found by Kmeans. We process our data again to find how many local descriptors belong to a particular centroid. By belong, we denote the centroid that is closest to a particular local descriptor of a sample. The sample can then be represented as a vector of dimension equal to the cluster size K with the magnitude of each dimension equal to the number of local descriptors belonging to that particular centroid.

5.4.2. Algorithm. The following algorithm enumerates the steps for finding the Bow representation of a sample.

1. Learn the Codebook $C = \{c_1, c_2 \dots c_k\}$ of words representing the vocabulary of the domain.
2. Assign the vocabulary for every local descriptor by $c_i = NN(x)$.
3. The bow representation is given by the sum of local descriptors belonging to a particular word c_i.

Thus we get the BOW representation by the simple formula

$$B_j = \sum_{x|NN(x)=c_j;\forall j \in K} 1$$

5.4.3. Implementation. The code for this project is developed using Scikit-Learn[8]. So we wanted to build an implementation that fits well into scikit-learn's pipelines. Any component can be duck typed to fit into scikit-learn's pipeline if it has fit, transform, set_params and get_params methods. A breif explanation of the methods and its functions is discussed below.

def fit (X, Y=None) This function is used to learn the parameters from the distribution of the training set and apply the parameters on training and test set. In our BOW case, this involves learning the vocabulary of the local descriptors.

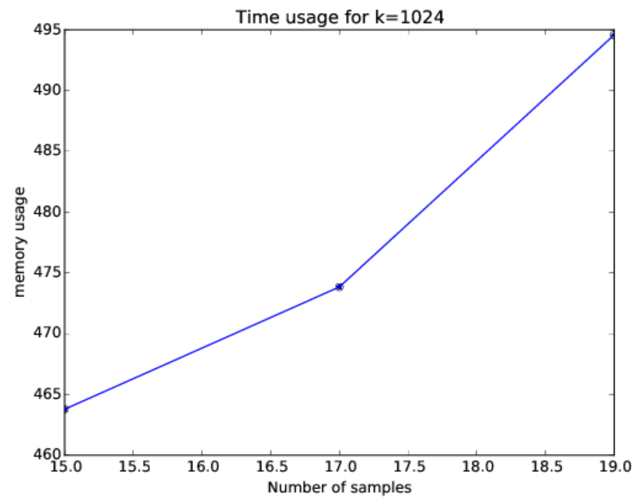


Figure 7. *Bag of Words feature representation: Runtime with fixed cluster size. (Cluster size in* $\log_2$*).*

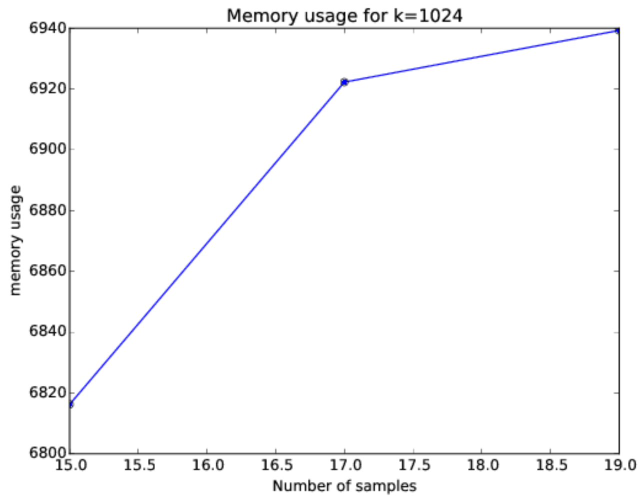


Figure 8. *Bag of Words feature representation: Memory usage with fixed cluster size. (Cluster size in* $\log_2$*)*

def transform (X) This function transforms the training/test input by using the vocabulary learnt during the fit method. In BOW implementation a sample containing a list of local descriptors is transformed into a vector of dimension K (cluster size).

def set_params (**params) Used to set the parameters for grid search.

def get_params (deep=True) Used to get the current parameters.

5.4.4. Performance. We try to understand the performance of our Bag of Words implementation by plotting the runtime and memory usage against a fixed number of clustersFig 7, 8 and also against fixed sample sizes. Refer Fig 9, 10.

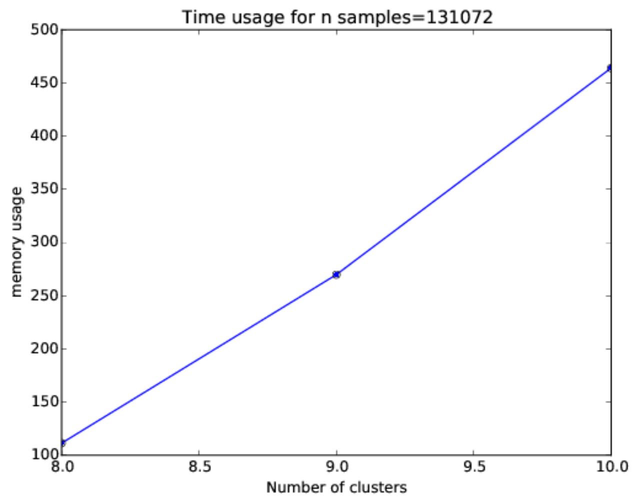


Figure 9. *Bag of Words feature representation: Runtime with fixed sample size. (Sample size in* $\log_2$*).*

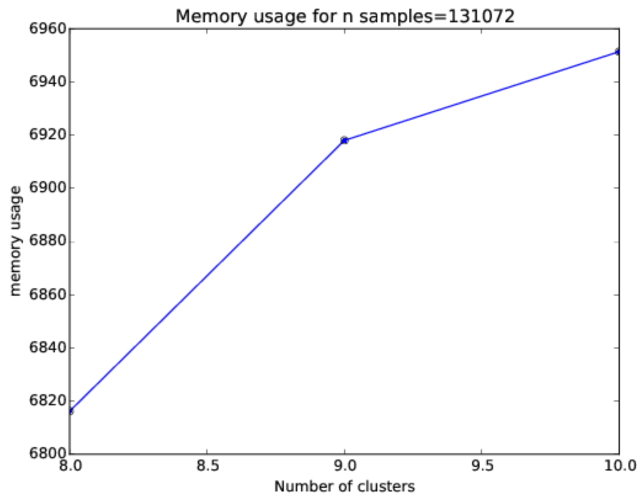


Figure 10. *Bag of Words feature representation: Memory usage with fixed sample size. (Sample size in* $\log_2$*)*

5.4.5. Variance Analysis. We analyze the variance of the Bag of Words representation on a subsample to get an idea of the number of components to use in the downstream pipeline. A plot of the component vs its variance is displayed in Fig 11

5.5. Building Vlad model.

5.5.1. Theorey. Vector of locally aggregated descriptors is a model similar to the Bag of Words model. The main difference between BOW and Vlad is that, Vlad takes into account the spread of the information along with the count of local descriptors belonging to a particular centroid. The vlad representation of a sample with local descriptors of size D and centers K

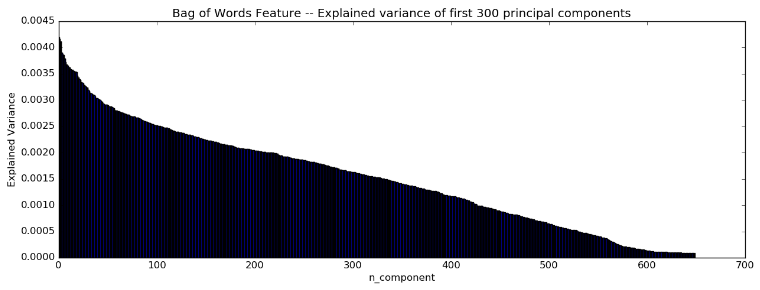


Figure 11. *Explained variance ratio of Bag of Feature representation for a subsample (Subject 1). We get variance measures of 99, 95, 90, 77, 65 and 50 for components 649, 507, 442, 336, 263, 186 respectively.*

will be given by a matrix of size KxD. Thus a Vlad representation would be able to reperesent more information about the domain than what is capable with a Bow model.

5.5.2. Algorithm. The following algorithm enumerates the steps for finding the Vlad representation for a sample with local descriptors.

1. Learn the Codebook $C = \{c_1, c_2 \ldots c_k\}$ of words representing the vocabulary of the domain.
2. Assign the vocabulary for every local descriptor by $c_i = NN(x)$.
3. The Vlad representation is given by difference $x - c_i$ of the vectors assigned to a particular vocabulary c_i.
4. L2 Normalize the above representation and concatenate the rows to form a vector.

Thus we get the Vlad representation by the simple formula

$$V_{i,j} = \sum_{x|NN(x)=c_j} (x_j - c_{i,j})$$

where x_j and $c_{i,j}$ represent the j^{th} component of the descriptor x considered and of its corresponding word c_i

Thus the Vlad representation of a sample with local desccriptors of dimension D and centers K is given by $K \times D$. After concatenating the rows into columns we get a vector of size given by the product of K and D.

5.5.3. Implementation. As discussed in the Bag of Words section, since we are using scikit-learn's pipelines, we want to build an implementation that plays nice with scikit learn. We implemented the fit, transform, set_params and get_params method in our vlad implementation. A brief description of the functions and its operations:

def fit (X, Y=None) For Vlad implementation: learn the vocabulary of the local descriptors.

def transform (X) Transform a sample containing a list of local descriptors into a vector of dimension K x D (product of cluster size and dimension of local descriptors).

def set_params (**params) Used to set the parameters for grid search.

def get_params (deep=True) Used to get the current parameters.

5.5.4. Performance. We try to understand the performance of our Vlad implementation by plotting the runtime and memory usage against a fixed number of clusters. Fig 12, 13 and

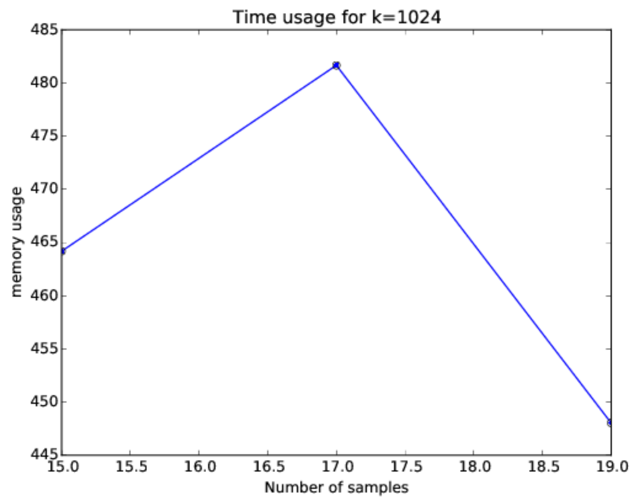


Figure 12. *Vlad feature representation: Runtime with fixed cluster size. (Cluster size in* $\log_2$*).*

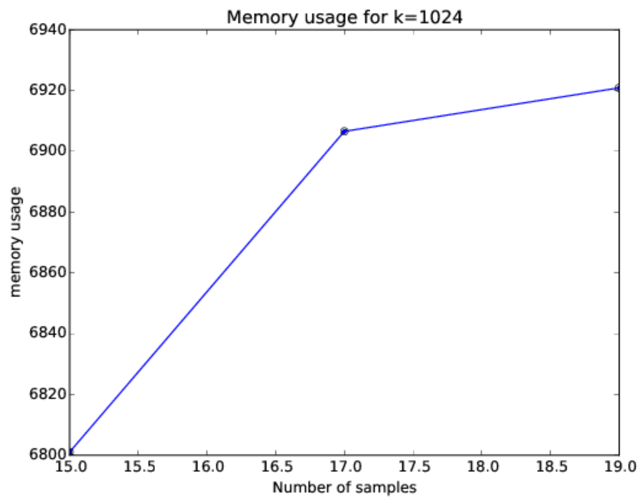


Figure 13. *Vlad feature representation: Memory usage with fixed cluster size. (Cluster size in* $\log_2$*)*

also against fixed sample sizes. Refer Fig 14, 15.

5.5.5. Variance Analysis. We analyze the variance of the Vlad representation on a subsample to get an idea of the number of components to use in the downstream pipeline. A plot of the component vs its variance is displayed in Fig 16

5.6. Building OnlineMiniBatchKMeans model.

5.6.1. Theorey. Online Kmeans is a version of the unsupervised KMeans algorithm for clustering huge amount of samples[13]. It works similar to Stohastic Gradient Descent but

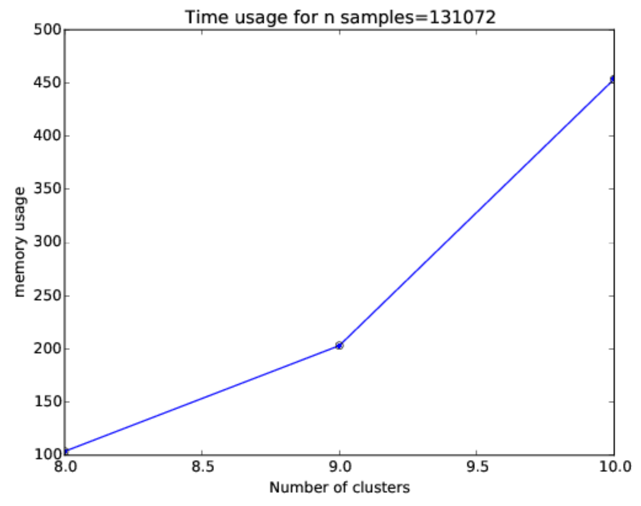


Figure 14. *Vlad feature representation: Runtime with fixed sample size. (Sample size in* $\log_2$*).*

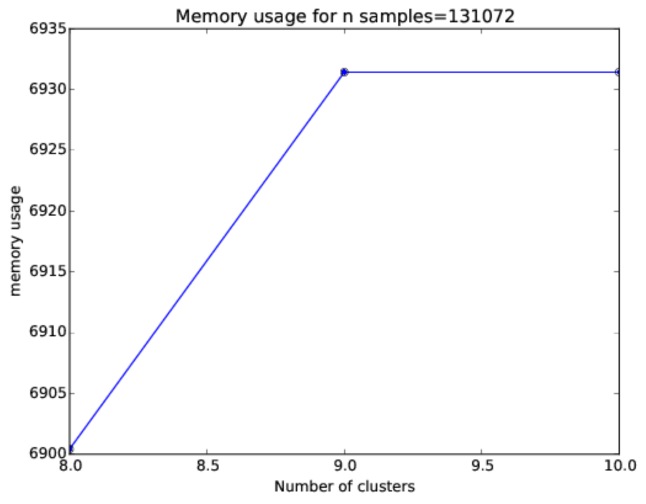


Figure 15. *Vlad feature representation: Memory usage with fixed sample size. (Sample size in* $\log_2$*)*

takes batches of samples drawn randomly from the original sample distribution and running KMeans on the subsample. It is shown that OnlineKMeans has performance comparable to that of KMeans but converges very faster than the originl KMeans algorithm.

5.6.2. Implementation. We implemented online kmeans as a component that plays nice with scikit-learn's pipeline. We also ran a number of experiments with varying cluster sizes to figure out the performance of our algorithm with respect to scikit-learn's KMeans algorithm.

5.6.3. Performance Optimizations. We used convergence factor similar to the one used in Scikit-learn's OnlineKMeans to make sure that our algorithm finishes faster. The performance

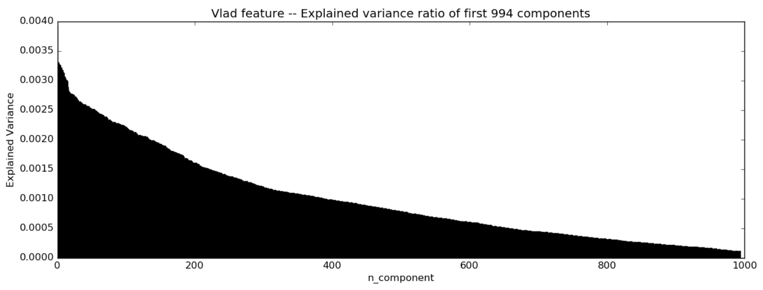


Figure 16. *Explained variance ratio of Vlad representation for a subsample (Subject 1). We get variance measures of 99, 95, 90, 77, 65 and 50 for components 994, 807, 780, 478, 353, 232 respectively.*

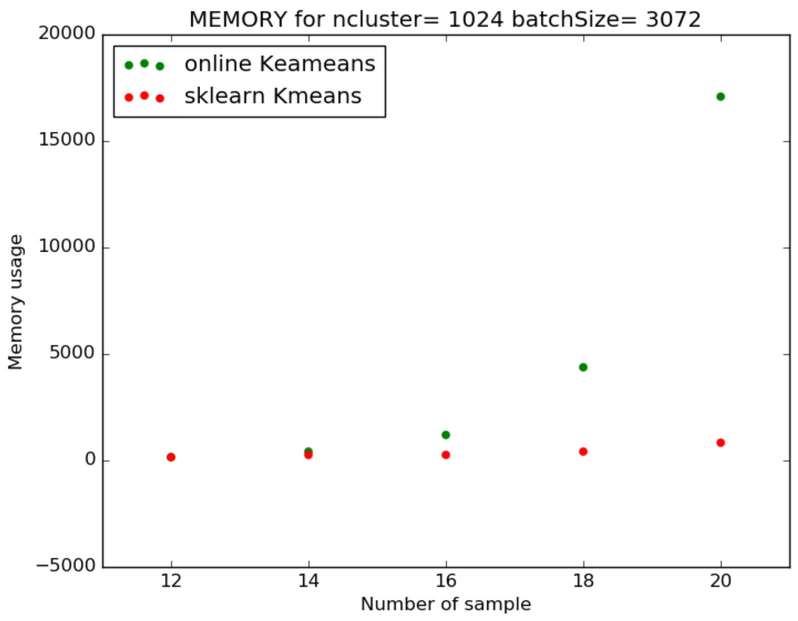


Figure 17. .

of our algorithm is very much similar to that of scikit-learn's implementation till sample size — 2^{17} and number of cluster $K = 2^9$. However, since the performance of our pipeline was the best when the cluster size was 1024 or 2048, we are using the Scikit-Learn's implementation in our final version of our code. Fig 17 and Fig 18 show the memory and time comparision with respect to scikit-learn.

5.7. Putting it all together as a pipeline. We finally built our pipeline using the following components

1. Preprocessing using xDawn
2. VLAD / BOW feature representation using our Vlad and Bowf components.
3. Reduce dimensionality using PCA

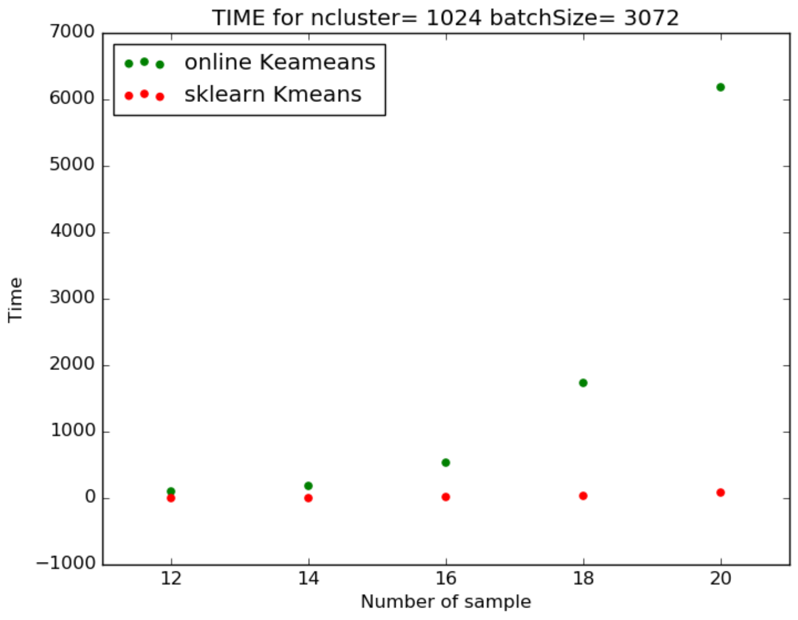


Figure 18.

4. Scaling the features using StandardScaler
5. LinearSVM/GaussianSVM on the scaled feature set
6. Performance estimation using Mean AUC value.

5.8. HyperParameters. There are a number of hyper parameters which we had to learn for each steps of the pipeline.

5.8.1. Preprocessing using xDawn. The value of num_component is a hyperparameter. We performed an exhaustive grid search on the pipeline using values of 2, 3 and 4.

Since the data preprocessing takes considerable amount of time, we decided to store the processed values as numpy arrays ans use them in the pipeline downstream. This saved considerable time for our pipeline.

We were able to observe best performance for num_component = 2.

5.8.2. VLAD / BOWF feature representation. An exhaustive grid search for performed for various values of K ranging from 2^4 to 2^{12}. We were able to observe a plateau between K = 1024 and K = 2048. The best performance was observed for K = 2048, but with K = 4096, the model became computationally intractable.

5.8.3. PCA. We reduced the dimension of the feature set to 90% of the explained variance of the original dataset.

5.8.4. LinearSVM and GaussianSVM. We ran exhaustive grid search for values of C and Gamma ranging from 2^{-4} to 2^4.

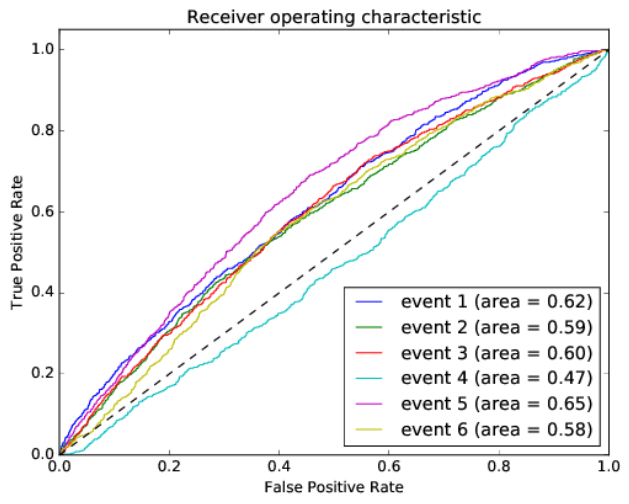


Figure 19. *ROC Partials (n_components=2,C=4,n_culter=2**8).*

6. Results and Analysis. This section contains the results and analysis of our results. We will plot the results for a few cases of grid search, performance, and accuracy score.

6.1. Best Performance. We got the best performance of 0.52 misclassification score with Linear SVM and Bowf representation for parameters

1. xDawn — num_component = 2
2. Bowf — Cluster size (K) = 2048
3. classifier — LinearSVM
4. Classifier parameters — C = 16.000

6.2. ROC curves. The performance criteria we used is the mean ROC curve. Fig 19 and Fig 20 shows the mean ROC curve and actual ROC curve for all the stimulus.

6.3. Grid Search. The plots Fig 21 and Fig 22 shows the accuracy of the cross validation scripts with respect to C values. Note that the accuracy in the plot is with respect to the misclassification error and not the Area Under the Curve.

A sample run of the cross validation with top ten scores is shown in Table 6.3

6.4. Performance. The plots Fig 23 and Fig 24 shows the time usage of the cross validation scripts with respect to C values.

7. Future work and Potential Improvements.

7.1. Feature Selection. On analysing the performance and the choice of features retrospectively, we are able to realize that using just xDawn filters for feature selection might not be a good decision. The baseline uses Linear Discriminant Analysis and a set of Filter Bank along with xDawn for its feature selection. The current Vlad and Bowf model should be trained again by using a better feature set to understand the performance difference.

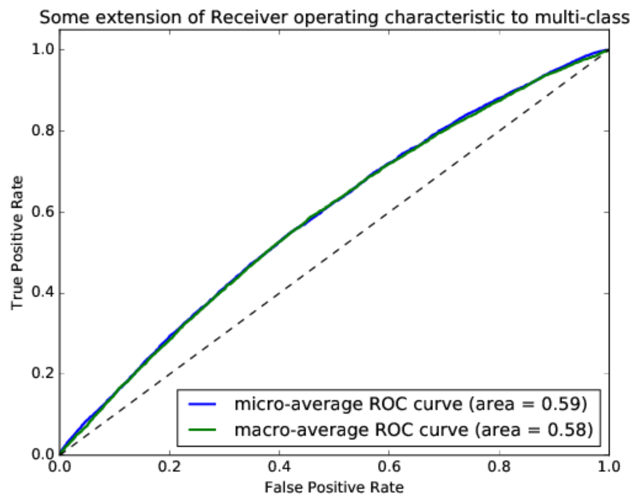


Figure 20. *Mean ROC curve (n_components=2,C=4,n_culter=2**8).*

Table 1
Bag of words score for n component= 2

C	K	score	time
16.0000	2048	0.531901	317.500000
8.0000	2048	0.530237	196.500000
4.0000	2048	0.516204	173.500000
2.0000	2048	0.501374	132.000000
1.0000	2048	0.491392	95.733333
0.5000	2048	0.474392	84.800000
0.2500	2048	0.461661	59.866667
0.1250	2048	0.442564	67.966667
0.0625	2048	0.430555	48.066667
16.0000	1024	0.358941	475.833333

Also, a spatial analysis of the EEG signals with respect to the location of electrodes and analysing the evoked potentials with respect to a particular stimulus whould have given a better understanding of the input data.

The models used in this project should give a much better performance if proper care is taken to select the feature set properly.

7.2. Pipeline. Though we did a grid search on a number of parameters such as C, Gamma, num_components, and K; we didn't do grid search on the dimensionality reduction. This was based on the assumption that for best results the reduced data should still hold the maximum amount of variance. Performing grid search on the number of reduced components such that the variance is at 50, 65, 77, 90, 95 and 99 percentile of the orignal variancce would have given better results.

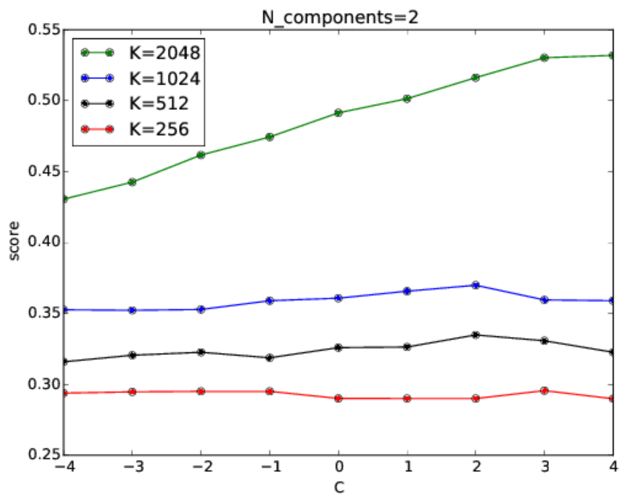


Figure 21. *The C vs score plot for Linear Bow features. When n_components = 2.*

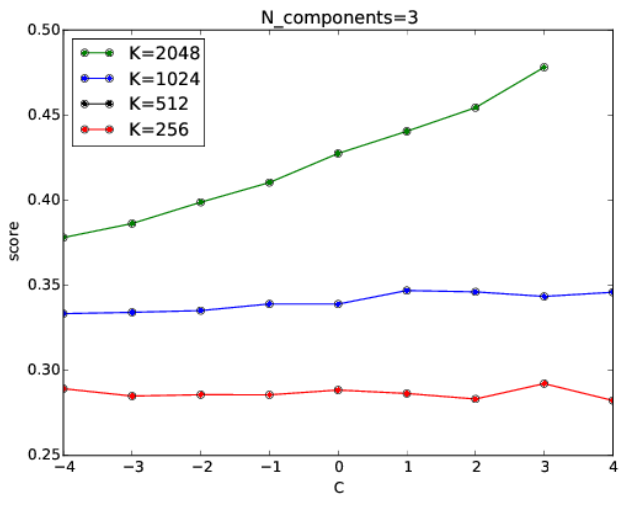


Figure 22. *The C vs score plot for Linear Bow feature When n_components = 3.*

Also, we were unable to run our experiments to get a peak in the plots for all the hyperparameters. The performance was increasing for values of K=2048 and C=16. But we couldn't run it more as the jobs were taking more than 30 hours. A considerable time should be spent to optimize the Vlad/Bowf implementation further before running the implementation.

7.3. Process. We did a lot of analysis such as the separability of the dataset, variance of the features, characteristics of the raw data retrospectively. If these analysis were done before starting up with the pipeline, we might have got better feature representations.

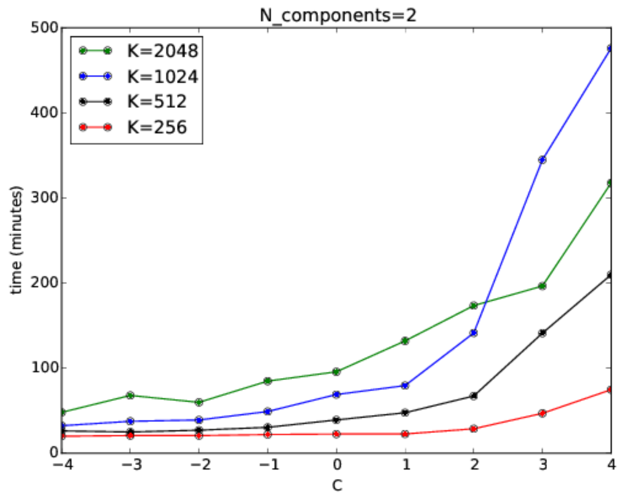


Figure 23. *The C vs time plot for Linear Bow features. When n_components = 2.*

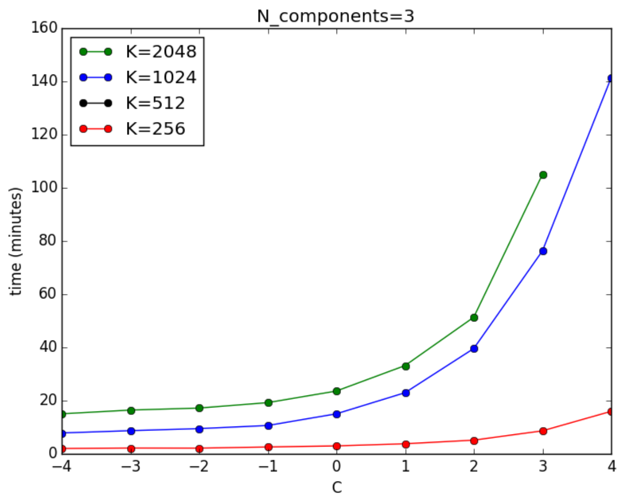


Figure 24. *The C vs time plot for Linear Bow feature When n_components = 3.*

7.4. No future data rule. The current implementation does't abide by the no future data rule. We should find ways to abide by the no future data rule and get good accuracy predictions.

8. Code and Plots. The complete codebase, HPC job outputs, and original plots are all available at our Github repository.[10]

9. Hardware Requirements. The huge size of the dataset requires that we use systems with high capacity. Almost all our jobs in HPC were run with the following specs:
CPU Cores: 12 core.

CPU Nodes: 1 Node.
RAM: 164 GB.
Runtime: Minimum 24 hours. (Jobs with ¿= 2048 clusters got killed even after 30 hours.)
GPU: None

10. Conclusion.

REFERENCES

[1] Relja Arandjeloc and Andrew Zisserman, *All about vlad*, CVPR 2013, (2013). https://courses.cs.washington.edu/courses/cse590v/13au/arandjelovic13.pdf.

[2] Alexandre Barachant and Rafal Cycon, *Kaggle - grasp and lift detection*, August 2015. https://github.com/alexandrebarachant/Grasp-and-lift-EEG-challenge.

[3] Way Consoritum, *Wearable interfaces for hand function recovery*. http://www.wayproject.eu/.

[4] Mne Developers, *Mne — meg & eeg analysis & visualisations*. http://mne-tools.github.io/stable/python_reference.html.

[5] New York University High Performance Computing. http://hpc.nyu.edu.

[6] Kaggle Inc, *The home of data science*, June 2015. https://www.kaggle.com/c/grasp-and-lift-eeg-detection.

[7] Herve Jegou, Matthijs Douze, Cordelia Schmid, and Patrick Perez, *Aggregating local descriptors into a compact image representation*, CVPR 2010, (2010). https://lear.inrialpes.fr/pubs/2010/JDSP10/jegou_compactimagerepresentation.pdf.

[8] Scikit Learn, *Machine learning in python*. http://scikit-learn.org/stable/.

[9] Matthew D Luciw, Ewa Jarocka, and Benoni B Edin, *Multi-channel eeg recordings during 3,936 grasp and lift trials with varying weight and friction*, Scientific Data, (2014). http://dx.doi.org/10.1038/sdata.2014.47.

[10] Anirudhan J Rajagopalan and Michele Ceru, *Grasp and lift eeg detection*. https://github.com/rajegannathan/grasp-lift-eeg-detection.

[11] ———, *Attempt 2 at running the baseline*, September 2015. https://github.com/mc3784/Computational-Machine-Learning-/blob/master/baseline/level1.q.

[12] B. Rivet, A. Souloumiac, V. Attina, and G. Gibert, *xdawn algorithm to enhance evoked potentials: Application to brain computer interface*, Biomedical Engineering, IEEE Transactions on, 56 (2009), pp. 2035–2043. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4760273.

[13] D. Scully, *Web-scale k-means clustering*, In Proceedings of the 19th international conference on World wide web, (2010), p. 11771178.