Fast Random Kernelized Features: Support Vector Machine Classification for High-Dimensional IDC Dataset

Tianshu Huang [*1] **Nimit Kalra** [*2]

Abstract

Random feature maps provide low-dimensional kernel approximations, thereby accelerating the training of support vector machines for large-scale datasets. Through a process of dimensionality reduction via k-means clustering, lifting via a random feature map, and subsequent linear SVM classification in this feature space, we outperform standard Gaussian kernel SVM on the 7500-dimensional Invasive Ductal Carcinoma dataset (Janowczyk & Madabhushi, 2016) in both accuracy and speed. We explore applying two random maps (Random Fourier Features and Random Binning Features) and experiment with different preprocessing methods such as k-means clustering of pixel colors, histogram of gradients (HoG), and applying an HSV transform.

1. Introduction

We aim to improve both classification accuracy and performance of binary classification on the Invasive Ductal Carcinoma (IDC) Identification dataset (Janowczyk & Madabhushi, 2016). This dataset contains mount slide images of Breast Cancer (BCa) specimens from 279 patients scanned at 40x. From these scans, 275, 222 sections were extracted in 50-by-50 pixel patches (see Figure 1) and manually classified as normal or invasive by human experts. Using 10,000 Random Fourier Features (with Gaussian kernel), we achieved an error rate of 13.56%, using only 6 minutes of training. At a slightly higher error margin (13.7%), using fewer features, the training time can be decreased to under 1 minute, presenting a performance improvement of several orders of magnitude over Kernel SVM.

Our code is available on GitHub: `https://github.com/qw3rtman/rf`.

[*]Equal contribution [1]Department of Electrical Engineering, University of Texas, Austin, Texas [2]Department of Computer Science, University of Texas, Austin, Texas. Correspondence to: Tianshu Huang <tianshu.huang@utexas.edu>, Nimit Kalra <nimit@utexas.edu>.

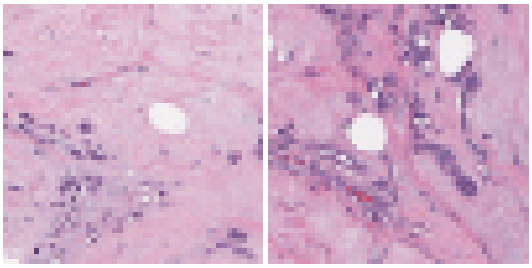


Figure 1. Problem Statement: Normal (Non-Invasive) or Invasive?

2. Random Features

Recall that the kernel trick provides an implicit lifting into a high-dimensional feature space; in other words, the kernel is given by $k(x, y) = \phi(x)^\top \phi(y)$ where ϕ is a high-dimensional feature map. We circumvent the high-dimensionality (which is even more problematic due to the high-dimensional nature of our problem space) by making use of low-dimensional random maps that approximate the kernel. In particular, we explore applying the Random Fourier Feature map and Random Binning Feature map approach from (Rahimi & Recht, 2008) to the IDC classification problem.

2.1. Random Fourier Features

Given a shift-invariant kernel k, Bochner's theorem guarantees it is the characteristic function of some probability distribution $p(\omega)$. Sampling the scale factors ω_1, ω_2 from this probability distribution p and sampling the shift factors b_1, b_2 uniformly on $[0, 2\pi]$ then yields $\mathbb{E}\left[2\cos(\omega_1 x + b_1)\cos(\omega_2 y + b_2)\right] = k(x, y)$.

Moreover, (Rahimi & Recht, 2008) provide an upper bound on the maximum error between the intended kernel k and the Random Fourier Features kernel $z(x) = \sqrt{2}\cos(\omega x + b)$.

We experiment with the shift-invariant Gaussian, Laplacian, and Cauchy kernels given in 2, along with their associated probability distributions.

Kernel	$k(x, y)$	$p(\omega)$
Gaussian	$\exp\left(\frac{-\|x-y\|_2^2}{2}\right)$	$(2\pi)^{-D/2}\exp\left(\frac{-\|\omega\|_2^2}{2}\right)$
Laplacian	$\exp\left(\|x-y\|\right)$	$\prod_i \frac{1}{\pi(1+\omega_i^2)}$
Cauchy	$\prod_i \frac{2}{1+(x_i-y_i)^2}$	$\exp\left(\|\omega\|_1\right)$

Figure 2. Shift-invariant kernels for Random Fourier Features and their associated probability distributions

2.2. Random Binning Features

By partitioning the input space into bins such that $k(x, y) \propto \mathbb{P}$[x and y are in same bin], we can derive an unbiased estimator of our intended kernel. Moreover, (Rahimi & Recht, 2008) prove, via Hoeffding's inequality, that the Random Binning Feature map will uniformly converge to the true kernel exponentially fast in the number of bins while maintaining low variance of the estimator. Additionally, (Wu et al., 2016) provide an upper bound the error of this random feature map.

In our experiments, we sample bin width δ from the probability distribution associated with the Laplacian kernel and uniformly sample the bin shift factor on $[0, \delta]$.

3. Initial Feature Extraction

3.1. k-Means Color Histogram

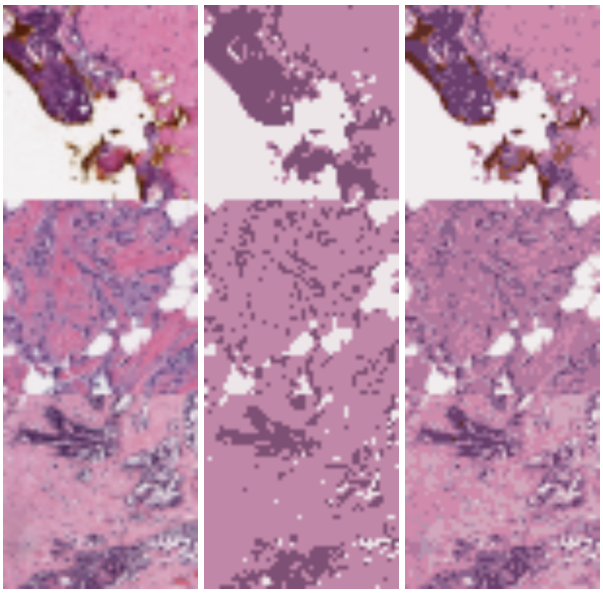


Figure 3. Left: Original Image; Middle: $k = 3$; Right: $k = 10$

When encountered with image data, one common paradigm is to generate a relatively small number of discrete features from the dataset (Wu et al., 2016). With that in mind, we applied k-Means clustering to generate a low-dimension color histogram, and applied the Random Features method to that histogram.

Recall that k-means clustering aims to partition the space into k clusters such that the variance with respect to the mean of each cluster is minimized. We use `skimage.cluster.KMeans` to implement k-means clustering and randomly sample from the input space to pick our k cluster seeds.

3.2. HSV Transform

HSV (Hue, Saturation, Value) is an alternative to the RGB color space. Since HSV encodes color type (hue), vibrancy (saturation), and brightness (value), we hypothesize that it will implicitly bring more key relationships between points in the input space to the feature space, which will boost classification accuracy. We use `skimage.color.RGB2HSV` to implement the transformation.

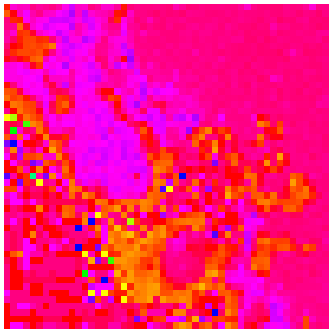


Figure 4. After HSV transform (Value and Saturation set to 100%)

4. Experiment Methodology

4.1. Overview

In order to test our approach, we split the training data (254 patients) into a group of 279 patients for training and a group of 25 patients for testing.

For algorithms that take too long or too much memory to compute, we sample a proportion of the training data, with each class and each patient sampled independently so that the sampled subset of the training data reflects the ratios between classes.

All experiments were run using an Intel i7-6700k processor (4 core, 8 threads; 4GHz) with 16GB of RAM.

4.2. Implementation

Our implementation was written in Python using numpy as a scientific computing framework, and scikit for various basic algorithms used by our implementation.

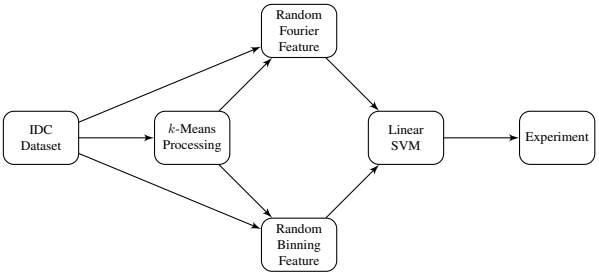


Figure 5. Program Architecture

First, the random feature is generated. Random features are generated using a map-reduce architecture to quickly generate the necessary transformation coefficients. Monte Carlo rejection sampling is used to sample coefficients from the desired densities.

If k-means color histogram features are used, 1% of the training data is first sampled; 10 pixels are then chosen at random from each of these images. This list of colors is used by k-nearest neighbors clustering to determine the k-best approximation of the input colors (SciKit Learn (Pedregosa et al., 2011): `sklearn.cluster.KMeans`). Once loaded, color histograms are computed from the images by counting the number of pixels in each class.

If a hue transform is used, the image is converted to the HSV color space on load (SciKit Image (van der Walt et al., 2014): `skimage.color.RGB2HSV`), and separated into different channels as necessary.

Image loading is conducted using a map-reduce architecture. After loading, images are immediately transformed into the desired feature space. Next, the transformed input space is run through a linear support vector classifier (SciKit Learn (Pedregosa et al., 2011): `sklearn.svm.LinearSVC`). Finally, the test images are loaded and transformed into the feature space using the same procedure, and classified using the previously computed linear model.

5. Results and Performance

Previous implementations of random features have never attempted to use such high dimensions. (Wu et al., 2016) applied random features to the 784-dimension, 28x28 black and white MNIST handwritten numbers dataset; as a collection of 50x50 RGB images, our dataset has almost 10 times the dimensions.

5.1. Baseline

As a preliminary baseline and sanity check, consider the constant classifier (table 5.1), defined by always selecting the class with a higher prior probability. In our data, since there are more benign samples than cancerous samples, the constant classifier always selects benign.

True Negative	20123
True Positive	0
False Negative	5704
False Positive	0
Error	22.1%

Table 1. Constant Baseline

We ran Kernel SVM and Linear SVM on the input data (table 5.3). With 10% of the training data, Kernel SVM was able to reach 13.7% error; however, this comes at significant cost: training took 44 minutes.

As the complexity of the Kernel matrix increases with $O(N^2)$, we can see a quadratic increase in the training time required as the number of training samples used increases. Furthermore, since classification of new points requires translation to the kernel space, classification of new points is $O(N)$, and we can observe a linear increase in the time used per classification.

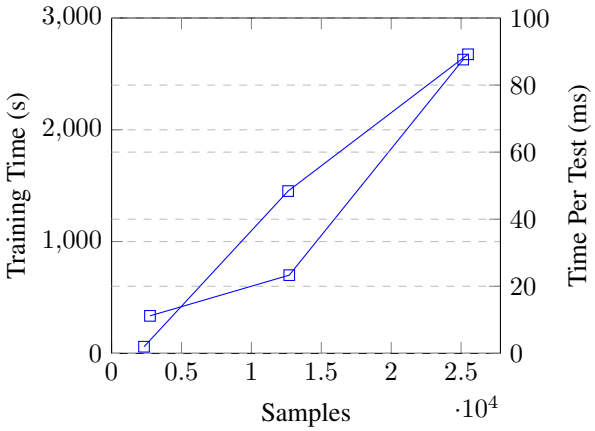


Figure 6. Kernel SVM Baseline Time Complexity

5.2. Fourier Features

One obvious application of random features is directly generate a Random Fourier Feature with the full RGB image as

Training Samples	2745	12708	25827
True Negative	17823	17990	18026
True Positive	3934	4170	4261
False Negative	1770	1534	1443
False Positive	2300	2133	2097
Training (s)	335.27	699.4	2627.97
Classification (ms)	1.97	48.39	89.17
Error	15.8%	14.2%	13.7%

Table 2. Kernel SVM Baseline

Transformation	HSV Transform		No Transform	
Features	2500	7500	2500	7500
True Negative	18176	18216	18159	18236
True Positive	4085	4058	4119	4068
False Negative	1619	1646	1585	1636
False Positive	1947	1907	1964	1887
Training (s)	94.97	269.62	103.54	274.79
Classification (ms)	3.18	9.28	3.28	9.24
Error	13.8%	13.7%	13.7%	13.6%

Table 3. Hue Transform vs No Transform (Training Size: 25827)

Transformation	Hue Only		No Transform	
Features	2500	7500	2500	7500
True Negative	18208	18186	18159	18236
True Positive	4097	4094	4119	4068
False Negative	1607	1610	1585	1636
False Positive	1915	1937	1964	1887
Training (s)	98.2	273.99	103.54	274.79
Classification (ms)	3.33	9.21	3.28	9.24
Error	13.8%	13.7%	13.7%	13.6%

Table 4. Hue Channel vs No Transform (Training Size: 25827)

the input. As the number of features increases, the accuracy converges to the Kernel SVM accuracy, as expected.

Note that when compared to Kernel SVM, Random Fourier Features are extremely robust. With only 2500 features (1/3 of the input dimension), Random Fourier Features with Linear SVC is very close to Kernel SVM classifier's accuracy.

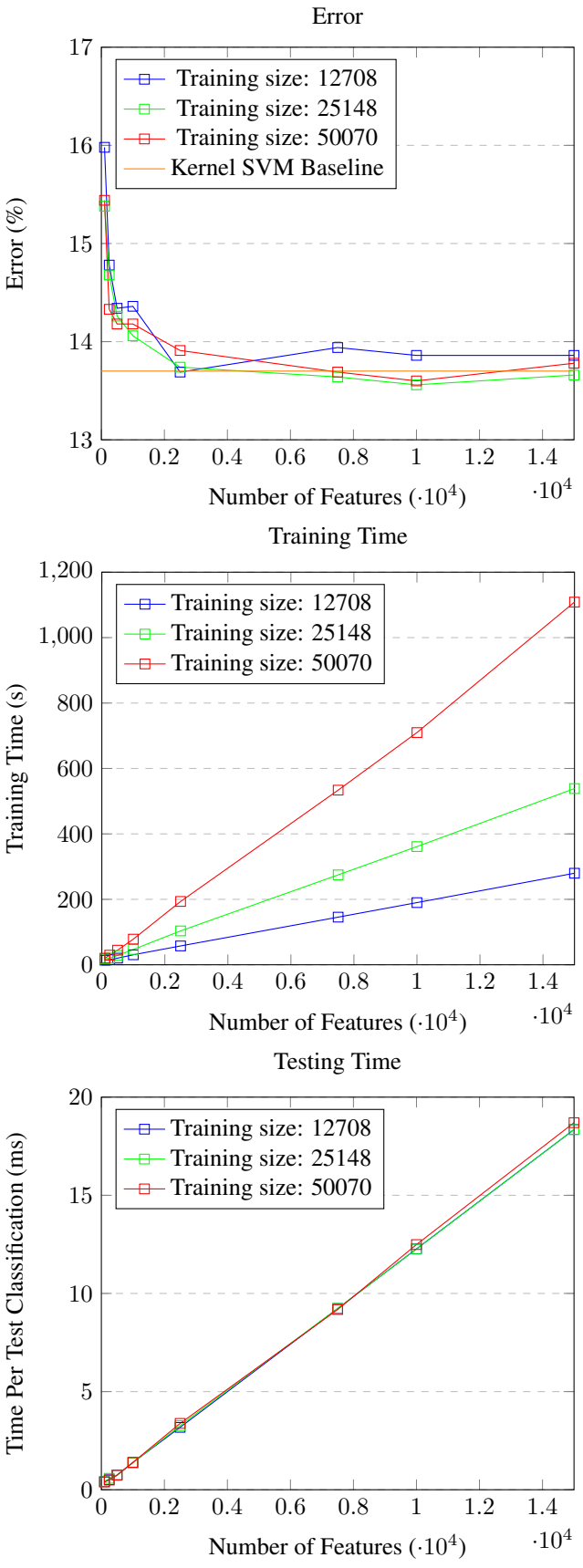
Here, we can observe the advantages of using Random Fourier Features. In order to obtain an accuracy of 13.7%, Random Fourier Feature SVC requires less than 2 minutes of training, compared to Kernel SVM's 44 minutes. Furthermore, Random Fourier Feature SVC classifies in constant time relative to the number of training samples, and therefore does so much faster: each classification at 2500 features takes just over 3ms, almost one-and-half orders of magnitude less than Kernel SVM's 89ms.

5.3. Fourier Features with Hue Transform Pre-Processing

After applying a hue transform (table 5.3), we obtained nearly identical results to applying Random Fourier Features on the original input space. We also applied Random Fourier Features to only the Hue channel of the HSV image (table 5.3), again obtaining a very similar result.

The latter test confirms that the input color space is extremely sparse; despite reducing the color information by a factor of 3, little information appears to be lost.

Figure 7. Performance of Gaussian Fourier Random Features



5.4. Fourier k-Means based Features

Using Fourier Random Features and k-Means color histograms (Figure 8), we obtained a minimum error of 14.2%. As before, we can observe a steady convergence towards the minimum error while increasing the training size; this is accompanied by a linear increase in training time. Amongst all training sizes, we observe a local minimum at $k = 7$.

While this presents a significant reduction in the number of dimensions, compared against Fourier Features applied to the entire image with similar accuracy, this technique does not lead to an increase in speed. This is likely due to the relative computation expense of k-nearest-neighbors clustering: each pixel must be compared to each cluster in order to generate the color histogram.

The main advantage of k-means based features is memory usage: in order to obtain a comparable 14.3% error, full-image-based Fourier Random Features requires at least 500 features compared to the 100 features required for k-means Fourier Random Features to reach 14.2% error.

5.5. Binning k-Means based Features

Using Binning k-Means based features (Figure 9), we were able to obtain a minimum error of 14.1%. Like with Fourier k-Means, there appears to be a local minimum at $k = 7$.

Binning k-means compares favorably to Fourier k-means; however this comes at a cost: binning k-means takes much more time to run. Furthermore, while the minimum of 14.1% is reached using only 10 features, each "feature" takes up much more memory than a Fourier Feature, since instead of 32 or 64-bit floating point numbers, Binning Features must be implemented as binary bit-strings. As there are no commonly-available linear algebra libraries for multiplying packed binary bit vectors, our implementation stores each feature as a 128-byte array, resulting in a much higher memory footprint. Despite similar theoretical efficiency, Random Binning Features are more complicated to implement in practice when compared to Random Fourier Features.

6. Bottlenecks and Potential Improvements

6.1. Processing Power

Processing power will always be a bottleneck. In our implementation, we implement multi-threaded image loading and feature space translation using Python's `multiprocessing` framework. However, the final support vector solver is provided by SciKit-Learn's `LinearSVC` solver, which is not multi-threaded. As such, there is significant room for speed improvement by integrating a multi-threaded SVM solver.

Additionally, due to a issue with `multiprocessing` on

Figure 8. Performance of Fourier k-Means Random Features

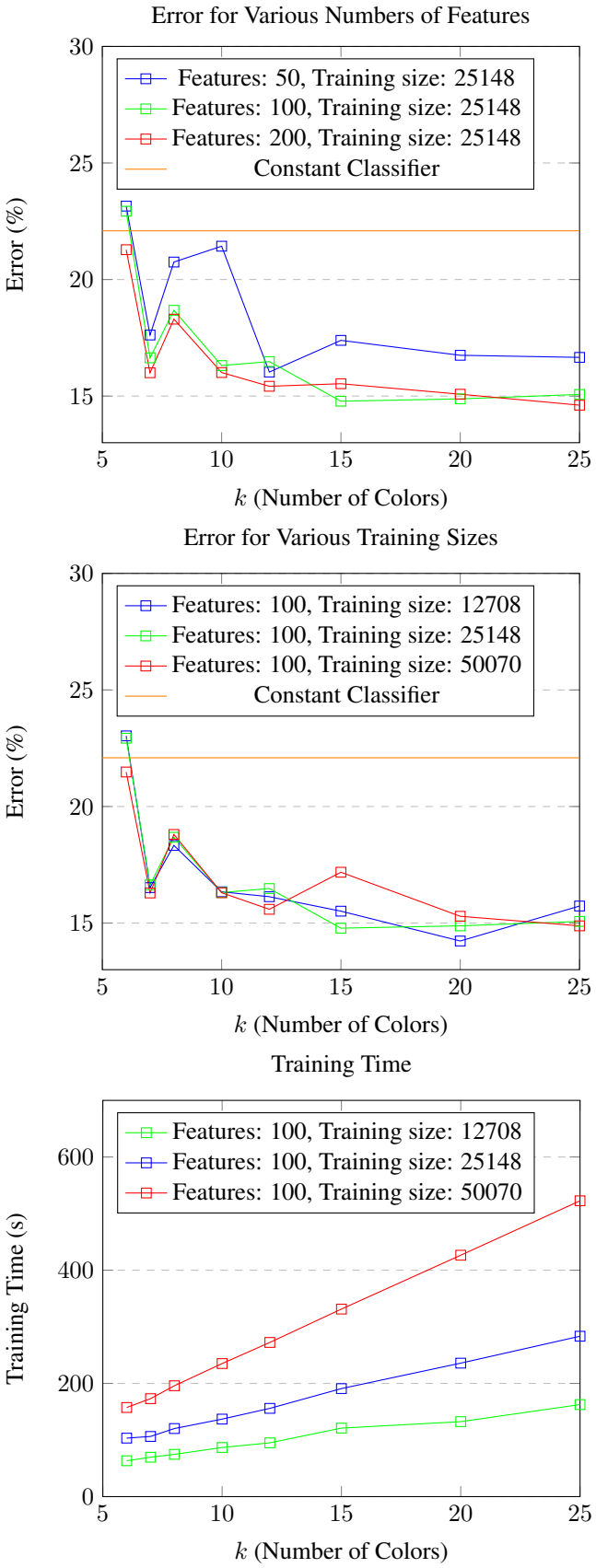
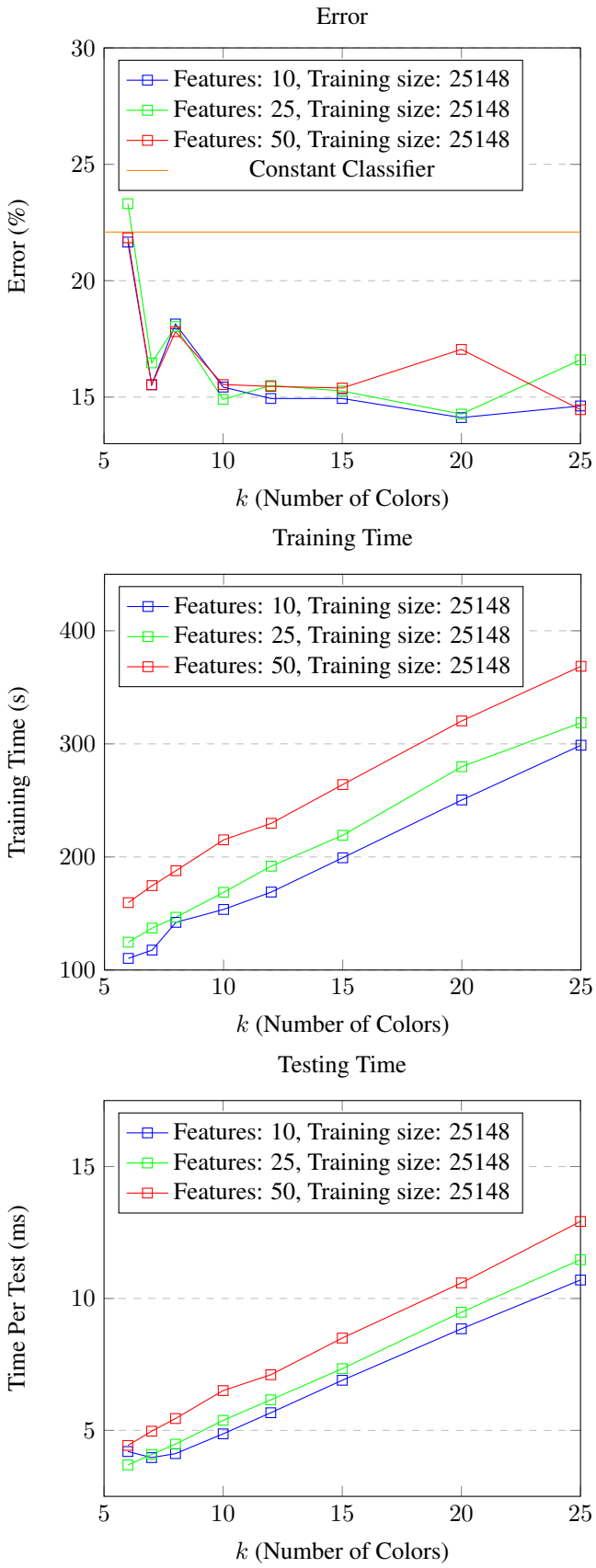


Figure 9. Performance of Binning k-Means Random Features



Linux[1], our implementation cannot be run on Linux with any significant number of threads. Since most supercomputing environments run Linux, this means that we must choose between taking advantage of the large amount of memory available and using the large number of cores. Since training on the entire dataset would be prohibitively time-expensive if not run in a heavily multithreaded environment, we were unable to train Random Features on the entire dataset.

6.2. Memory

As mentioned in the previous section, our current implementation is unable to effectively train on the entire dataset. As our test results appear to have converged, it is unlikely that directly increasing the training size at any given feature size will improve our results; however, further testing with a combination of increased training size and increased number of features may result in greater accuracy.

In addition, more memory could allow for further dimension-lifting before applying Random Fourier Features. Instead of choosing a single pre-processing feature, we could compute several pre-processing transformations, and apply Random Features on the concatenation of these generated features.

6.3. Other Potential Improvements

Many other pre-processing schemes are promising; for example, edge-detection operators such as the Sobel operator could enhance detection of larger shapes. Various convolution kernels such as edge detection kernels could also be used to enhance contrast.

7. Other Applications

When it comes to readily available, easy to use, almost guaranteed to work classifiers, Kernel SVM is one of the most popular classifiers. As such, the relative effectiveness of Random Features opens up many possibilities for applications of Support Vector Machines to applications where speed is critical.

Furthermore, Random Feature SVM allows for constant-time classification independent of the number of training samples, and does not require storage of every single training data point in order to perform future classification, while retaining the accuracy benefits of Kernel SVM. Therefore,

[1]Calling `os.fork()` causes the original processes' memory to become read only; when any objects are accessed, they are copied into a new thread. However, Python manages memory by keeping a reference count for each object. When these objects are read, the reference count is incremented, therefore counting (at least to the operating system) as a memory modification. Therefore, when `os.fork()` is called while there are large objects in the main process (such as a 250MB feature map), the feature map is duplicated to every new thread.

Random Feature SVM is a prime candidate for use in embedded applications where both space and processing power are critical.

For example, with 2500 Random Fourier Features, classification time on a 4GHz x86 platform does not exceed 4ms. Even when scaled down to the 1GHz ARM processor on the low-cost, small-form-factor Raspberry Pi Zero, this should not exceed 25ms; therefore, this puts real-time image classification within reach of low-cost embedded systems.

Acknowledgements

This work was completed by Nimit Kalra and Tianshu Huang while they were undergraduates at UT Austin as part of their final project for a data science course. We thank our professor Chandrajit Bajaj for his suggestions in picking a research direction and providing the IDC classification problem domain, as well as his valuable instruction and guidance throughout the semester.

References

Janowczyk, Andrew and Madabhushi, Anant. Deep learning for digital pathology image analysis: A comprehensive tutorial with selected use cases. *Journal of Pathology Informatics*, 7, 2016.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Rahimi, A. and Recht, B. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, pp. 1177–1184, 2008.

van der Walt, Stéfan, Schönberger, Johannes L., Nunez-Iglesias, Juan, Boulogne, François, Warner, Joshua D., Yager, Neil, Gouillart, Emmanuelle, Yu, Tony, and the scikit-image contributors. scikit-image: image processing in Python. *PeerJ*, 2:e453, 6 2014. ISSN 2167-8359. doi: 10.7717/peerj.453. URL http://dx.doi.org/10.7717/peerj.453.

Wu, Lingfei, Yen, Ian EH, Chen, Jie, and Yan, Rui. Revisiting random binning features: Fast convergence and strong parallelizability. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1265–1274. ACM, 2016.