Classification of SDSS photometric data using machine learning on a cloud

Vishwanath Acharya*, Piyush Singh Bora, Karri Navin, Anisha Nazareth, P. S. Anusha and Shrisha Rao

International Institute of Information Technology-Bangalore, 26/C, Electronics City, Bengaluru 560 100, India

Astronomical datasets are typically very large, and manually classifying the data in them is effectively impossible. We use machine learning algorithms to provide classifications (as stars, quasars and galaxies) for more than one billion objects given photometrically in the Third Data Release of the Sloan Digital Sky Survey (SDSS-III). We have used *k*NN, SVM and random forest algorithms in a distributed environment over the cloud to classify 1,183,850,913 unclassified photometric objects present in the SDSS-III catalog. This catalog contains photometric data for all objects viewed through a telescope and spectroscopic data for a small part of these. Although it is possible to classify all the objects using spectroscopic data, it is impractical to obtain such data for each one of them. To classify such a big dataset on a single machine would be impractically slow, so we have used the Spark cluster computing framework to implement a distributed computing environment over the cloud. We found that writing results (dozens of gigabytes) to the cloud storage is very slow while using *k*NN. Though writing the results with SVM is faster as it is done in parallel, its accuracy is only around 87%, due to lack of a kernel implementation of it in Spark. We then used the random forest algorithm to classify the entire set of 1,183,850,913 objects with an accuracy of 94% in about 17 hours of processing time. The result set is significant as even collecting spectroscopic data for these many objects would take decades, and our classifications can help astronomers and astrophysicists carry out further studies.

Keywords: Astronomical data, classification, cloud computing, distributed algorithms, machine learning.

ASTRONOMICAL datasets are typically very large. The Sloan Digital Sky Survey (SDSS)[1,2] catalog is a large collection of astronomical data which comprises the most detailed three-dimensional maps of the Universe, with deep multi-colour images of one-third of the sky, and spectra for more than three million astronomical objects. One of the problems faced while processing astronomical data is to accurately and efficiently classify observed celestial objects. Classically, when an object was first observed through a telescope, its data was recorded manually and further manual calculations were performed over such data which facilitate in classifying the recorded object as belonging to a particular type of celestial object, say star or galaxy. This approach is infeasible given the sizes of contemporary astronomical datasets.

Processing the entire data from SDSS, even with the most efficient algorithms, takes enormous amounts of time and resources. In the SDSS dataset of our interest, there are two kinds of observational data – spectroscopic and photometric. Different types of objects in the spectroscopic catalog each have a well-described spectrum. Based on the spectral characteristics like redshift, emission peaks, absorption peaks, etc., each object has been classified as a star, quasar or galaxy. SDSS cannot get the spectrum data for all its objects, as it takes about an hour apiece to measure each spectrum; to get data for all the objects viewed would take hundreds of years. The photometric catalog contains colour data about all the objects viewed so far, including objects in the spectroscopic catalog. The colour of the object is measured in five filters: ultraviolet (u), green (g), red (r), and infrared (i) and (z). There are a total of 1,183,850,913 objects in the photometric catalog, out of which only 3,751,358 are spectroscopically classified, while the others are unclassified.

Here we have used machine learning algorithms over the Spark framework[3] to classify the entire unclassified dataset of objects as stars, quasars or galaxies.

Before we delve into further details about the data, we describe the terms 'machine learning' and 'Spark' to indicate some context for the work that is described later.

Machine learning[4] is an approach which automates the process of making predictions for new unclassified data, based on available classified data. (We used spectroscopic data to train the model and make predictions over the photometric data.) Some of the standard algorithms for supervised learning are *k*NN, SVM, random forest, etc. (We were able to get good results using the random forest algorithm.)

Apache Spark[5] is a cluster computing framework to handle big data processing. It works in a manner similar to traditional Hadoop MapReduce, but is much faster. The basic working of MapReduce is to split the input into

*For correspondence. (e-mail: acharya.vishwanath@iiitb.org)

several parts and run a program on these separate parts in parallel at once. Though Spark's functioning is similar to MapReduce, it is much faster than the Hadoop[5] because it uses the concept of resilient distributed datasets (RDDs), which reduces the number of read/write operations to disk. Spark stores the data in the form of an RDD and persists to disk only when necessary, which reduces read/write times and makes Spark a fast engine for large-scale data processing. It is over 100 times faster than Hadoop MapReduce.

We use the astronomical dataset available at SDSS. We utilize only the photometric data of the different classified objects (the objects are classified manually using their spectroscopic data) to train our algorithms, which we then use to identify quasars, stars and galaxies from the unclassified objects.

In order to store such huge amounts of data and make them available across all platforms for accessing, they should be live always. For this purpose, we have used the cloud platform which not only provides the storage but also makes it easy to run programs over the data. Cloud computing is an all-in-one tool which provides computing services such as storage, networking, analytics, etc., over the internet.

Here we have used the Google Cloud[6] platform for our entire environment set-up. Google Cloud provides the infrastructure to run a complete machine learning project. It also frees the users from the overhead of managing and configuring networks. It provides the users with data analytics services to study and analyse the existing data. One such service that we have used is Google Dataproc, which allows the users to create and run Spark frameworks over the cloud. Using the features of Google Dataproc, we have set up our cluster environment. We have then created machine learning models from our training data and used those models to predict the class of each photometric object.

Thus, with the help of machine learning and our cloud set-up, we could automate the entire process by bypassing the need of human intervention and automating the calculation part. Typically, the spectroscopic SDSS data are classified based on their u, g, i, r and z parameters[7]. Using these parameters they measure the bivariate distribution of r^* luminosity with half-light surface brightness, intrinsic $g^*–r^*$ colour and morphology. Based on these measurements, each object is classified as a star, galaxy or quasar. This work would be time-consuming and slow when it has to be done for around 1,183,850,913 objects, were it not for the cloud set-up. Under such circumstances, the combined power of machine learning and distributed computing can become a boon to the astronomical community. Although machine learning algorithms automate the process of making predictions for new data, making predictions for the entire SDSS dataset in a single machine would take an inordinate amount of time. So in order to reduce the time, we have used distributed computing, a computing model in which the operations are distributed among the different systems in a cluster to improve performance[8]. Spark is a distributed framework which is used for large-scale data processing. Initially we put in place a distributed set-up over the cloud using Spark framework. We then applied the random forest algorithm which yielded better results for both binary as well as multi-class classifications, when compared to *k*NN and SVM implementations of Spark. (We however describe the *k*NN and SVM implementations also, as they are standard algorithms and their implementations carry important lessons.)

Classifying even a mere 500,000 objects (a tiny fraction of the photometric catalog) spectroscopically would have taken approximately 57 years using standard astronomical tools. However, with Spark over cloud using machine learning techniques, we were able to classify the entire dataset in less than a day with an accuracy of 94%. Both our source code and result data are available on the cloud (http://tiny.cc/astro-sdss and the result data at https://doi.org/10.6084/m9.figshare.5143255.v1). Therefore this approach would be of use to the astronomy community.

Related work

There have been several efforts at using machine learning with astrophysics data; Zhang and Zhao[9] survey this domain well. Such efforts, however, have mainly focused on the problem of classifying relatively smaller datasets like the Supernova Legacy Survey (SNLS) dataset[10,11]. The main focus of relevant existing work is to implement various machine learning techniques over a small dataset and identify the model which gives the best accuracy among them. In all such related works, attempts have been made to implement machine learning algorithms over a centralized system, without taking full account of scalability. As a consequence, though these efforts are generally successful on a small scale, they cannot be expected to work on complete, large datasets like SDSS-III.

The objective in this study is twofold; first is the use of distributed processing techniques (unlike the centralized ones seen elsewhere[12–15], which are not scalable); and the second is the classification of the entire SDSS-III dataset, rather than small fragments of SDSS or other astronomical data, as done elsewhere[16]. The basic techniques of machine learning are well known and used all over, but to run them at scale in a large distributed setting is what is done in this study.

Problem formulation

Algorithms and dataset

Our aim is to classify the entire SDSS-III photometric data into stars, quasars and galaxies. This in turn requires

us to set up a cloud environment and apply different machine learning classification algorithms to choose the best among them, which can classify the entire data accurately abiding by the evaluation metrics, within a reasonable amount of time. We have used certain standard evaluation metrics to measure the performance of the classifier. The entire result of this classification is shared, which is useful to the astronomical community or any independent researcher.

Distributed environment set-up

The unclassified dataset from the SDSS III catalog is very large. To store such huge data and run machine learning algorithms for classification on local machines would require very high computing power, and a large data storage. To handle this issue we have used the Google Cloud services to set up a cluster for distributed computing, and the Apache Spark framework to distribute our jobs among the machines in the cluster.

Cluster set-up over the cloud

Google Cloud provides the Dataproc[17] service to process large quantities of data easily. It provides Apache Hadoop, Apache Spark, Apache Pig and Apache Hive services to handle large datasets. We have used the Apache Spark framework for cluster management over Google Dataproc. The steps that we have followed to set up a cluster and run Spark jobs over the cluster are: (1) Initially we set up a project over the cloud to use the Google Cloud services; (2) We created four instances, each of which had 7.5 GB RAM: one for master node and the remaining three for worker nodes; (3) These instances were used to set up a cluster using the Dataproc service of Google Cloud; (4) Cloud storage was used as common for storing training data, testing data and programs to be run on the cluster; (5) Using the Dataproc GUI, we could submit the jobs to the cluster using the Spark framework.

Figure 1 shows the basic architecture of our Google Cloud environment. Spark programs run as independent sets of processes on the cluster, coordinated by the SparkContext object in our main program. Spark acquires executor nodes of the cluster. It then sends the application code to these nodes and makes the executor nodes run the tasks in a parallel manner. Figure 2 shows an overview of Spark cluster management.

Data collection from SDSS catalog

The data used were obtained from SDSS[1,2]. The data are of two kinds: spectroscopic and photometric. The spectroscopic data were taken from the DR12 (Data Release 12) Spectroscopic Catalog, and the photometric data from the DR12 Photometric Catalog[18].

The classified data from the Spectroscopic Catalog have the following format:

```
ObjID, class, u, g, r, i, z
where ObjID is the unique object ID.
```

The unclassified data, from the Photometric Catalog, has the following format:

```
ObjID, class = null, u, g, r, i, z
where ObjID is the unique object ID.
```

The SDSS database contains two main tables which are of interest to us: SpecObj and PhotoObj. The SpecObj table contains spectroscopic data (including the class) of spectroscopically classified objects, and the PhotoObj table contains photometric data (the colour data) of all identified objects. The SpecObj table contains a small subset of the objects in the PhotoObj table (those that have been classified). The SQL query used to retrieve our training data is as follows:

```
SELECT p.objID, s.class, p.u, p.g,
p.r,p.i, p.z FROM SpecObj AS s JOIN
PhotoObj AS p ON s. bestObjID = p.
objID WHERE (p.type = 3 OR p.
type = 6).
```

Table 1 shows the result format of the above query.

This query does a full join between the SpecObj and PhotoObj to get the classes and colours of all the objects

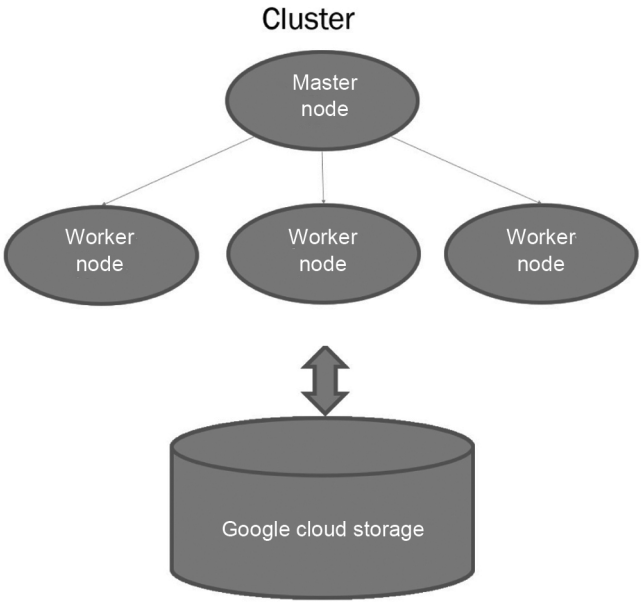

Figure 1. Google Cloud environment.

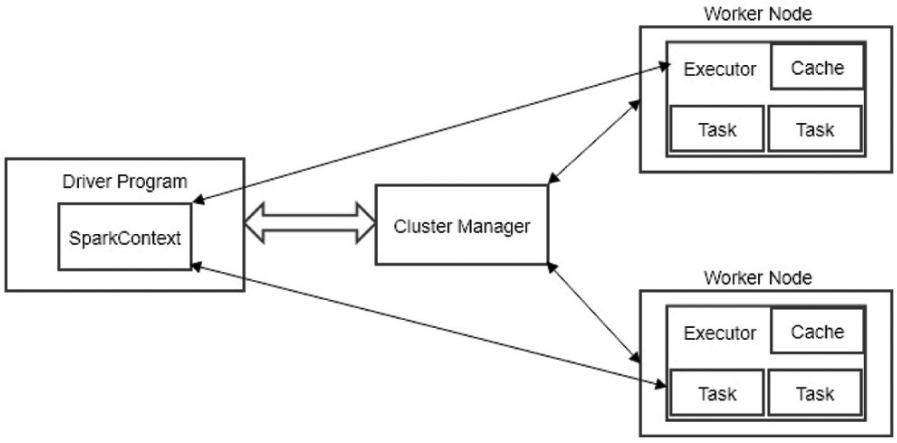

Figure 2. Spark cluster management.

Table 1. Training data

Object ID	Class	*u*	*g*	*r*	*i*	*z*
1237645879551000764	GALAXY	18.55396	25.91849	19.23725	20.98709	18.41879
1237645879551066262	GALAXY	17.20153	19.41061	17.58132	19.41061	16.90159
1237645879562862699	STAR	19.28224	18.95554	19.03111	20.14984	18.78287
1237645879580098737	QSO	20.11118	19.53674	20.00373	20.19055	19.36661

Table 2. Testing data

Object ID	*u*	*g*	*r*	*i*	*z*
1237645943435034783	18.55396	25.91849	19.23725	20.98709	18.41879
1237645943435034775	17.20153	19.41061	17.58132	19.41061	16.90159
1237645943434969150	19.28224	18.95554	19.03111	20.14984	18.78287
1237645943433396364	20.11118	19.53674	20.00373	20.19055	19.36661

in SpecObj. The query used to retrieve our testing data is as follows.

```
SELECT p.objID, p.u, p.g, p.r, p.i,
p.z FROM PhotoObj AS p LEFT OUTER JOIN
SpecObj s ON p. objID = s. bestObjID
WHERE (p. type = 3 OR p. type = 6)
AND s. class is NULL.
```

Table 2 shows the result format of the above query. This query does a left outer join between the SpecObj and the PhotoObj to get the colours of all the objects in PhotoObj where the class is null.

Implementation of kNN and SVM over the data

Our initial approach was to run the *k*NN and SVM algorithms using Spark distributed framework. These algorithms did not give the best results, but we report these negative outcomes as they are of interest.

The *k*NN[4] algorithm classifies a new data point based on the majority of the classes of the *k* nearest training samples to that data point. In our case, the attributes of the object are colours and based on the majority of colours near the data object, the algorithm decides the class of each object.

There is no direct implementation of *k*NN in Spark, so we implemented distributed *k*NN over Spark using the scikit-learn library[19]. The basic steps to implement distributed *k*NN over Spark are as follows.

- Create a Spark context for the program.
- Read the training data from cloud storage and store it as an RDD object.
- For each of these objects compute features as *g*–*r*, *u*–*r*, *r*–*i*, *i*–*z* and store them as an RDD object.
- Convert this RDD object to a stack of numpy tuples[20].
- Create a dataframe to store all the classes corresponding to the respective objects.
- Using scikit-learn, create a *k*NN model using features and classes obtained in steps 4 and 5.

- Create a corresponding broadcast object for the model to distribute the job.
- Read the testing data from cloud and perform steps 2 and 3.
- Using the broadcasted model, predict the class of each object in a parallel manner.
- Finally write the results onto cloud storage.

There were some problems with this *k*NN implementation. First, the accuracy was not satisfactory, as it was around 87%. Due to data conflicts between scikit-learn and Spark, the writing operation was taking linear amount of time, which would have required a month to classify the entire SDSS III data.

We then used SVM[4] for binary classification of SDSS data. The basic concept of SVM is to construct a hyperplane based on the training dataset. It then uses this model to classify the dataset based on which side of the hyper plane the test data belong. According to SVM, there are two groups of data. If the data points are separable by drawing a straight line with all points of one class on one side of the line and all the points of other class on the other side, then such data are linearly separable. For those data which are not linearly separable, they are transformed to higher dimensions using kernel functions to make them separable at higher dimensions. The Spark implementation of SVM is linear[21]. It is based on the loss function formulated by hinge loss[22].

As our data were not linearly separable, the linear SVM of Spark did not give satisfactory results and the accuracy was also around 87%. Though the issue of writing the results to the cloud in parallel was solved using SVM, the accuracy was still not satisfactory. Due to lack of a kernel implementation of SVM in the Spark framework, we were not able to improve the accuracy of the SVM model to more than 90%.

Random forest implementation

Random forest algorithm

Random forests are ensembles of decision trees[23]. Decision trees build classification models in the form of tree structures. A dataset is broken down into smaller and smaller subsets, while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node has two or more branches. Each leaf node represents a classification or decision. It constructs the tree using an information-theoretic entropy function. The entropy function represents the information gain of each of the data points.

The internal unit model used in the random forest algorithm is that of a decision tree. The random forest algorithm builds a set of decision trees separately. The algorithm produces different trees by injecting some randomness in the construction of each decision tree. It then combines the predictions of each tree to predict the final class of the object, which reduces the variance and improves the performance of the predictions made on the test data.

Reasons for choosing the random forest algorithm

The random forest algorithm not only gives good results for binary classification, but it also can be extended for multiclass classification[23]. It does not require feature scaling. The algorithm is capable of capturing nonlinear information inherent in the data. Additionally, it can capture feature interactions.

Training data and parameters used

Here we have used pySpark implementation of random forest over the Google Cloud environment[3]. The parameters used for training decision trees are *g–r*, *u–r*, *r–i*, *i–z*. These parameters along with the 500,000 classified objects are given as input to the algorithm to build a set of decision trees. This training of multiple decision trees is done in parallel with the use of the Spark framework.

Model construction

The master node of the cluster fetches the training dataset from the Google Cloud storage and stores it as an RDD object inside the Spark context. Using this RDD object, it then distributes the training job to all the worker nodes in the cluster to produce the final model. For distribution of jobs within the cluster, the map function of pySpark is used. During this training process, the algorithm attempts techniques like sub-sampling of the original dataset at each iteration and considering different random sets of features to split on at each tree node, to inject randomness in the construction of decision trees. We have used this algorithm for both binary classification and ternary classification of SDSS data. For binary classification, each training object is labelled as 0 for quasar and 1 for non-quasar objects. The same approach is followed to classify into other types such as stars and galaxies. For ternary classification, training objects are labelled as 1 for quasars, 0 for stars and 2 for galaxies.

Final prediction

Prediction of objects classes is done by aggregating the results from all decision trees. The class of an object is decided based on majority voting. Each tree prediction is counted as a vote for one class. The class is predicted to be that which receives the most votes.

Box 1. Pseudocode for ternary classification

```
1   # Creating Spark Context for the cluster environment
2   sc = SparkContext ('local', 'test_script')
3   # Reading Training file from Google Cloud
4   data = sc. textFile (trainingFile)
5
6   # Defining Method to compute features and assign labels to each object
7   def parsePoint (line ):
8         Assign label as 0 for star , 1 for quasar and 2 for galaxy
9         Compute g-r,u-r, r-i, i-z as features for each object
10        return LabeledPoint (label , features )
11
12  # Removing Header from the training file
13  header = data .first ()
14  data = data . filter (lambda line : line != header)
15  # Parsing the training file as per the method defined above
16  parsedData = data .map(parsePoint)
17  # Create a model using Random Forest Algorithm
18  model = RandomForest . trainClassifier (parsedData , numClasses = 3,
          categoricalFeaturesInfo={}, numTrees = 21, featureSubsetStrategy='auto',
          impurity =' entropy', maxDepth = 20, maxBins = 32)
19  testData = sc. textFile (testingFile)
20
21  # Defining Method to parse the testing file
22  def parseTestData (line):
23        Assign object ID as label for each object
24        Compute g-r, u-r, r-i, i-z as features for each object
25        return (label , features)
26
27  # Parsing the testing file as per the method defined above
28  parsedTestData = testData .map(parseTestData)
29
30  # Making predictions using the model created in line 18
31  predictions = model. predict ( parsedTestData .map(lambda x: x[1]))
32  labelsAndPredictions = parsedTestData .map (lambda lp: lp [0]) .zip(predictions)
33
34  # Defining Method to parse the results
35  def getResults (t):
36        Assign object ID as label
37        Assign feature as STAR, GALAXY, QSO for 0, 2, 1 respectively
38        return (label, features)
39
40  # Parsing the results as per the method defined above
41  resultList = labelsAndPredictions .map(getResults)
42  #Save the results back to the cloud
43  resultList . saveAsTextFile (destination)
```

Writing final results

In this study we had 1,183,850,913 unclassified objects. These were stored in the Google Cloud storage. The objects were read from the storage and stored as an RDD object in the Spark context. Using this RDD object, the job of predicting the class for the object was distributed across the worker nodes of the cluster, using model from training step. This result of prediction was stored back into the Google Cloud Platform by writing it into a csv file in a parallel manner.

Explanation of the pseudocode for ternary classification

In line 2 in Box 1, the Spark context is created to handle all Spark jobs in the cluster. In line 4, the training file is read from Google Cloud storage and stored as an RDD object inside the Spark context. From lines 7 to 11, a method parsePoint is defined which splits each data object of the training file into labels and features. For labels it assigns 0 for star, 1 for quasar and 2 for galaxy. It computes *g–r*, *u–r*, *r–i*, *i–z* values for each object and

Table 3. Original data – sample

Object ID	*u*	*g*	*r*	*i*	*z*
1237645879551000764	20.98709	18.55396	19.23725	25.91849	18.41879
1237645879551066262	18.23754	17.20153	17.58132	19.41061	16.90159
1237645879562862699	19.28224	18.95554	19.03111	20.14984	18.78287
1237645879580098737	20.11118	19.53674	20.00373	20.19055	19.36661
1237645943973675227	20.77234	18.22479	19.19735	23.54567	17.63784
1237645943435034783	18.95963	16.81595	17.58961	21.46995	16.40388
1237645943435034775	18.50592	16.20174	17.0387	21.10589	15.65076

Table 4. Results – sample

Object ID	Class
1237645879551000764	GALAXY
1237645879551066262	GALAXY
1237645879562862699	STAR
1237645879580098737	QSO
1237645943973675227	GALAXY
1237645943435034783	STAR
1237645943435034775	STAR

assigns them as features for those objects. From lines 13 to 14, it removes the header from the testing file. In line 16, it calls the parsePoint with the training file RDD object as an argument and collects the parsed training file into another RDD object. In line 18, it creates a model using the random forest algorithm over the parsed RDD object created in line 16. In line 19, it reads the testing file from the Google Cloud storage and stores it as an RDD object. From lines 22 to 25, it defines a method parseTestData, which splits the testing file as labels and features for each object. The object ID is assigned as label and the values of *g–r*, *u–r*, *r–i*, *i–z* as features for each object of the testing file. In line 28, it calls the method parseTestData with the testing file RDD object as an argument to function and collects the parsed testing file into another RDD object. In line 31, using the model created in line 18, it makes predictions for each of the testing file objects and collects the results of the classification into an RDD object. From lines 35 to 41, it defines and calls a method getResults which converts the result RDD object into a readable format by converting labels back to their respective classes. In line 42, it writes results back to the Google Cloud storage.

Results

The results of this study are stored in several large comma-separated values (CSV) files. Each row stores the object ID and the corresponding class to which it belongs. Tables 3 and 4 show the format of our original dataset and the results respectively. The classified results of the entire 1,183,850,913 unclassified objects took around 17 h of processing, and the total size of the classified data is around 40 GB. They have been divided into 7 files, each about 600 MB in size and containing the results of approximately 1.3×10^7 unclassified objects. The results are available online (https://doi.org/10.6084/m9.figshare.5143255.v1).

Results of binary versus multiclass classification

A binary classification is a concept of predicting the classes from a two-class problem, whereas in multiclass classification we are concerned with the class of an object as being one among more than two possible classes. Our initial work was on binary classification of data into quasars and non-quasars. Figure 4 shows the Mollweide projection of a sample of our binary classification results.

The most common measure to assess the performance of a classifier is accuracy, but it ignores many of the factors which should be taken into account when assessing the classifier. Accuracy just gives an idea about the count of correct classifications. This count alone might not be a good classification measure for certain datasets. In particular for SDSS data, 90% of the objects are non-quasars. So any classifier which would classify entire data into non-quasar would get an accuracy of 90% which seems impressive, but that figure ignores all quasar objects in the data. In order to get a better assessment of the classifier we have used a receiver operating characteristics (ROC) graph[24]. This technique assesses the performance of the classifier. This is especially useful for domains like SDSS, where the data have skewed class distribution. ROC graphs are two-dimensional, in which true positive rate is plotted on the *y*-axis and false positive rate on the *x*-axis. An ROC graph depicts the trade-offs between true positive and false positive. Here we have used spark implemented ROC metric to measure accuracy and precision of random forest classifier, and Table 5 presents the results.

The random forest algorithm for binary classification gave an accuracy of 94%. The same approach was used for binary classification of stars and galaxies. The accuracy for all these categories was around 94%.

For multiclass classification, we classified the data into quasars, stars and galaxies. Using the random forest algorithm for multiclass classification, we were able to

Table 5. Confusion matrix

Algorithm	TP	FN	FP	TN	Accuracy	Precision
Binary classification	9290	3594	2315	84960	0.94100	0.80051

Table 6. Results for all algorithms

Algorithm	TP	FN	Fp	TN	Accuracy	Precision
*k*NN	460	12421	781	86800	0.88734	0.37066
SVM	428	12401	813	86820	0.86846	0.34488
Random forest	9290	3594	2315	84960	0.94100	0.80051

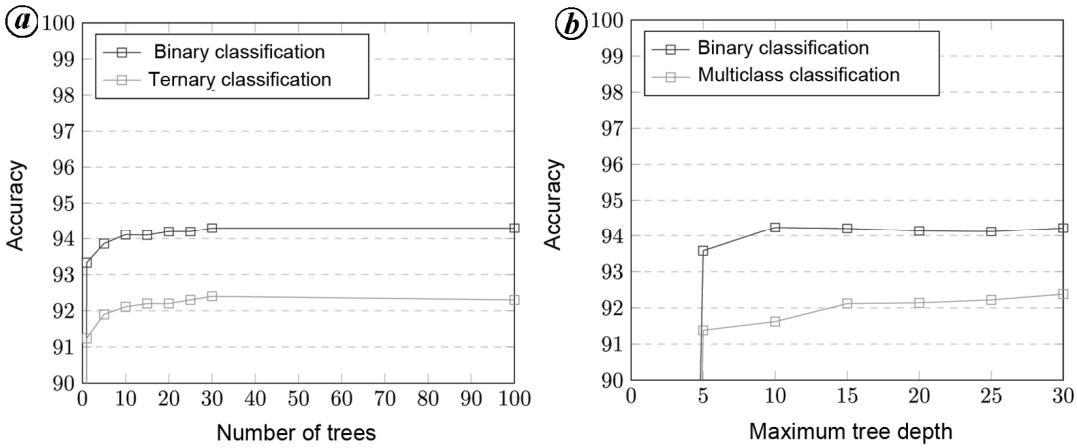

Figure 3. ***a***, Binary versus ternary classification. ***b***, Binary versus multiclass classification.

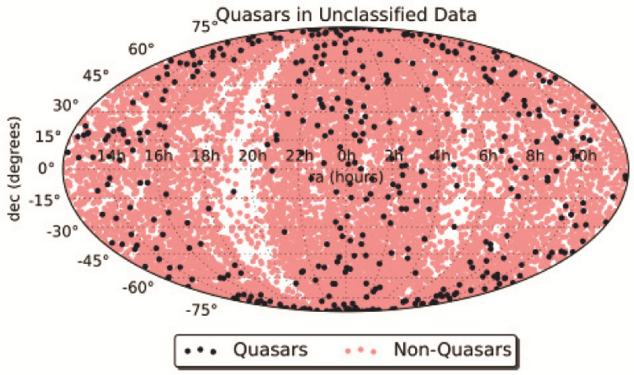

Figure 4. Mollweide projection of sample of binary classification results[28,29].

achieve an accuracy of 92%. Table 5 gives a summary of the results for binary classification.

Parameters relevant to accuracy

The main parameters that are relevant to the accuracy of the algorithm are described below.

Number of trees: A general tendency of decision trees is to overfit to the given data. According to the bias variance lemma, a model which is prone to over-fitting is highly unstable[25]. This suggests that a decision tree is very sensitive to any slight change of data. This property of decision trees leveraged by the random forest algorithm to build diverse models using only a subset of data and features to construct a decision tree. As the trees are diverse from each other, increasing the number of trees decreases the variance in predictions, improving the models test-time accuracy. We found that when the number of trees was 10, the algorithm gave an accuracy of 90% for binary classification and 87% for three-way classification. By increasing the number of trees from 10 to 50 improved our accuracy to 94% for binary classification and 92% for three-way classification (Figure 3 *a*). However, after this point, the accuracy remained the same but increasing the number of trees increased training time linearly. Increasing the number of trees above 100 might yield slightly better results, but it might take much longer to just train the model; so for the entire process of training and testing, it might take more than a day.

Maximum tree depth: This parameter implies the maximum depth each tree can take in the algorithm. Increasing the tree depth makes the model more expressive and powerful. With a maximum depth of 20, we were able to achieve an accuracy of 94% for binary classification and 92% for three-way classification (Figure 3 *b*).

Summary of all algorithms used

Initially, we used the *k*NN and SVM algorithms to train our model. The accuracy for both *k*NN and SVM algorithms over the SDSS data was around 87%. Using the random forest algorithm we were able to achieve better accuracy for both binary classification (94%) and multiclass classification (92%) of the dataset. Table 6 provides a summary of all the results.

Conclusion

The random forest algorithm worked well for SDSS data[1]. We ran the algorithm over the cloud using the Spark framework[3]. We were able to classify the entire dataset of 1,183,850,913 objects into stars, quasars and galaxies with an accuracy of 92%. These objects could be used to identify the distribution of quasars, stars and galaxies in the sky, which in turn can lead to new insights about them.

It seems unlikely that any substantial gain in classification accuracy would result using any other algorithms or approaches on the same dataset, given the classical wisdom that 'invariably, simple models and a lot of data trump more elaborate models based on less data[26]. However, some significant gains could well occur if more spectroscopically classified data (training data) were available. However, given more such training data, better algorithms may also be possible[27].

1. Eisenstein, D. J. *et al.*, SDSS-III: massive spectroscopic surveys of the distant universe, the Milky Way, and extra-solar planetary systems. *Astron. J.*, 2011, **142**, 72.
2. Dawson, D. J. S. K. S. *et al*., The Baryon oscillation spectroscopic survey of SDSS-III. *Astron. J.*, 2013, **145**, 10.
3. https://spark.apache.org/docs/1.2.0/cluster-overview.html
4. Witten, I. H., Frank, E., Hall, M. A. and Pal, C. J., *Data Mining: Practical Machine Learning Tools and Techniques*, Morgan Kaufmann, 2016.
5. http://hadoop.apache.org/
6. https://cloud.google.com/
7. Blanton, M. R. *et al.*, The luminosity function of galaxies in SDSS commissioning data. *The Astron. J.*, 2001, **121**(5), 2358.
8. Peleg, D., *Distributed Computing: A Locality-Sensitive Approach*, Society for Industrial and Applied Mathematics, 2000.
9. Zhang, Y. and Zhao, Y., Astronomy in the big data era. *Data Sci. J.*, 2015, **14**.
10. Supernova legacy survey, 2005; https://tspace.library.utoronto.ca/handle/1807/25390.
11. Möller, A. *et al.*, Photometric classification of type Ia supernovae in the SuperNova Legacy Survey with supervised learning. *J. Cosmol. Astropart. Phys.*, 2016, **12**, 008.
12. Ostrovski, F. *et al.*, VDES j2325-5229 a $z = 2.7$ gravitationally lensed quasar discovered using morphology-independent supervised machine learning. *Mon. Not. R. Astronom. Soc.*, 2017, **465**(4), 4325–4334.
13. Lochner, M., McEwen, J., Peiris, H., Lahav, O. and Winter, M., Photometric SN classification with machine learning. In Kavli Institute for Cosmological Physics Workshop on Photometric Classification of SNIA, Chicago, IL, USA, April 2016.
14. Miller, G. and Berger, E., PS1 classification of SN using ensemble decision tree methods. In Kavli Institute for Cosmological Physics Workshop on Photometric Classification of SNIA, Chicago, IL, USA, April 2016.
15. Moeller, A., SN photometric classification of SNLS data with supervised learning. In Kavli Institute for Cosmological Physics Workshop on Photometric Classification of SNIA, Chicago, IL, USA, April 2016.
16. du Buisson, L., Sivanandam, N., Bassett, B. and Smith, M., ' Machine learning classification of SDSS transient survey images. *Mon. Not. R. Astronom. Soc.*, 2015, **454**(2), 2026–2038.
17. https://cloud.google.com/dataproc/.
18. Alam, S. *et al.*, The eleventh and twelfth data releases of the Sloan Digital Sky Survey: final data from SDSS-III. *Astrophys. J. Suppl.*, 2015, **219**, 12.
19. Pedregosa, F. *et al.*, Scikit-learn: machine learning in python. *J. Mach. Learn. Res.*, 2011, **12**, 2825–2830.
20. Oliphant, T. E., *A Guide to NumPy*, Trelgol Publishing USA, 2006, vol. 1.
21. https://spark.apache.org/docs/1.6.0/api/python/pyspark.mllib.html#pyspark.mllib.classification.SVMModel
22. Rennie, J. D. M. and Srebro, N., Loss functions for preference levels: regression with discrete ordered labels. In *Proceedings of the IJCAI Multidisciplinary Workshop on Advances in Preference Handling*, Kluwer Norwell, MA, 2005, pp. 180–186.
23. Hartshorn, S., *Machine Learning with Random Forests and Decision Trees: A Visual Guide for Beginners*, Amazon Kindle, 2016.
24. Fawcett, T., An introduction to ROC analysis. *Pattern Recognit. Lett.*, 2006, **27**(8), 861–874.
25. Genuer, R., Variance reduction in purely random forests. *J. Nonparametr. Stat.*, 2012, **24**(3), 543–562.
26. Halevy, A., Norvig, P. and Pereira, F., The unreasonable effectiveness of data. *IEEE Intell. Syst.*, 2009, **24**(2), 8–12.
27. Zhu, X., Vondrik, C., Fowlkess, C. C. and Ramanan, D., Do we need more training data? *Int. J. Comput. Vis.*, 2016, **119**(1), 76–92.
28. Lapaine, M., Mollweide map projection, 2011; http://master.grad.hr/hdgg/kogstranica/kog15/2Lapaine-KoG15.pdf
29. Martin, E., The Mollweide projection, 2013; http://balbuceo-sastropy.blogspot.in/2013/09/the-mollweide-projection.html

ACKNOWLEDGEMENTS. We thank Shiva Kumar Malapaka and Ramya Sethuram for their advice on the presentation of this work and acknowledge SDSS-III survey for the data used in this work. Funding for SDSS-III has been provided by the Alfred P. Sloan Foundation, the Participating Institutions, the National Science Foundation, and the US Department of Energy Office of Science.

Received 12 July 2017; revised accepted 5 April 2018

doi: 10.18520/cs/v115/i2/249-257