Analysis of NDVI values dataset

Arman Singh Grewal
School of Computer Science
Lovely Professional University, Phagwara
Reg. no. - 12017712

***Abstract* - This paper is a brief of analysis on Crowdsourced Mapping dataset, freely available on UCI ML repository which was donated by Brian Johnson on 25 May, 2016. First approach was to fit a RandomForestClassifier on the dataset, but accuracy was not on par (64%) as training and testing sets had different probability distributions. Later on we reshuffled the dataset and tried more models. After hyperparameter tuning we achieved accuracy of 97%.**

I. INTRODUCTION

Our dataset consists of a time series NDVI (explained later) values over the Laguna de Bay area, Philippines. It is a typical classification problem with 6 classes - (impervious, farm, forest, grass, orchard, water).

There were no missing values in the dataset, also every value was *np.float64* type. Target label was of object type, and was easily converted to *np.int64* type using *LabelEncoder*.
There were various exploratory techniques like box plots, bar plots, count plots etc. used to understand more about features and their correlation with each other.
Firstly we used the most robust classifier (my personal preference) - RandomForestClassifier, but it gave pretty dull accuracy of 64%. We later find out that training and testing sets were too different from each other, we reshuffle and try again, and SupportVectorClassifier gave the best accuracy ~94%.
Later hyperparameter tuning on regularization parameter 'C' increased the accuracy to ~96%.
Tinkering with some more sklearn features, we got to know that *PowerTransformer()* and *QuantileTransformer()* (with 'uniform'-like distribution set as output) increased the accuracy even further to 97%!

II. APPROACH

A. Why do plants need light? - Photosynthesis

The author assumes that the reader possesses fundamental knowledge of photosynthesis. Therefore, author's explanation of photosynthesis is not a comprehensive biochemical approach to defining the process, but rather serves as a brief refresher.

One must have read this photosynthesis equation somewhere in his life -

$$6\,CO_2 + H_2O \xrightarrow[\text{photosynthesis}]{\text{sunlight}} C_6H_{12}O_6 + 6\,O_2 \quad (1)$$

Plant metabolism is very complex, and certainly not the topic of this term paper. To summarize - plants require four major components to produce energy for sustainment.

- Carbon Dioxide (CO_2): It is absorbed from air using *stomata* and is incorporated into organic molecules, such as sugars.
- Water (H_2O): It provides the hydrogen atoms needed to build energy-rich molecules such as ATP.
- Chlorophyll: It is a pigment that is responsible for capturing light energy during photosynthesis.
- Sunlight: It is the energy source that drives the process of photosynthesis. When light strikes chlorophyll molecules in the leaves of plants, it excites electrons, which are then used to create energy-rich molecules such as ATP and NADPH.

Focusing on chlorophyll - It is found in specialized structures called *chloroplasts*, which are present in the leaves of plants. Chlorophyll absorbs light in the blue and red parts of the electromagnetic spectrum, but reflects green light(500nm - 600nm), giving plants their characteristic green colour.

B. Which light spectrum to use?

I assume, we know the Plank's equation

$$E = h\nu$$

As frequency of violet > red, So, violet light carries higher energy than the red light. Hence, Chlorophyll absorbs blue and violet light more efficiently than red light, but the total amount of blue and violet light available in sunlight is much less than the amount of red light because it is also scattered more easily in the atmosphere. On the other hand, red light is able to penetrate deeper into plant tissues, making it more efficient for photosynthesis.

So, amount of red light absorbed is a better metric in determining the health of vegetation. More red light the plant absorbs, i.e. lesser red light is reflected from leaves, healthier they are

Fig. 1. A healthy plant reflects less red light, so appears less red

Near Infrared Radiation (NIR) is sensitive to the internal structure and water content of leaves. NIR is not absorbed by chlorophyll and is instead reflected out. Therefore, a higher reflectance of NIR radiation indicates a higher chlorophyll content, which indicates a healthier plant.

C. Combining it all in one metric - NDVI

NDVI (Normalized Difference Vegetation Index) is a commonly used remote sensing index that provides information about the health and productivity of vegetation. It is calculated by taking the difference between the reflectance values of near-infrared (NIR) and red light bands of the electromagnetic spectrum and dividing it by their sum.

$$NDVI = \frac{NIR - Red}{NIR + Red}$$

NDVI values range from -1 to 1, with negative values indicating non-vegetated surfaces such as water or bare soil, and higher positive values indicating areas with healthy vegetation cover. The higher the NDVI value, the more healthy and productive the vegetation is considered to be.

NDVI has a wide range of applications in agriculture, forestry, and ecology. It can be used to monitor crop health, estimate crop yields, map land use changes, and assess the impact of climate change on vegetation. NDVI data can be obtained from various remote sensing platforms such as satellites, aerial imagery, and drones.

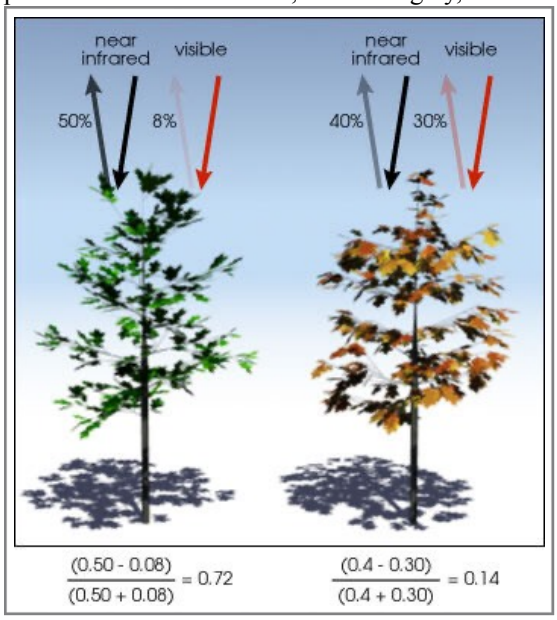

Fig. 2. NDVI values of two plants. Image courtesy of NASA

III. DATA COLLECTION

Whole credit of data collection goes to Brian A Johnson and Kotaru Lizuka. They have also created a research paper [1]

This dataset was derived from geospatial data from two sources:

1) Landsat time-series satellite imagery from the years 2014-2015
2) Crowdsourced geo-referenced polygons with land cover labels obtained from OpenStreetMap. The crowdsourced polygons cover only a small part of the image area, and are used used to extract training data from the image for classifying the rest of the image. The main challenge with the dataset is that both the imagery and the crowdsourced data contain noise (due to cloud cover in the images and inaccurate labelling/digitizing of polygons).

IV. MODELLING THE DATA

A. About the data

We are provided with *training.csv* and *testing.csv* files. Both were imported into a pandas DataFrame of shape - (10545, 29) and (300, 29). One thing to note was that columns were actually in reverse order - target label at front and time series data in descending order. So we reversed the columns for easier understanding.

There were no null values in the dataset, but variance was pretty high.

B. Exploratory Data Analysis (EDA)

First, time-series column names were changed to a date-time object for easier visualization of trends.
We are provided with 6 target classes, namely -
['water' 'forest' 'impervious' 'farm' 'grass' 'orchard']

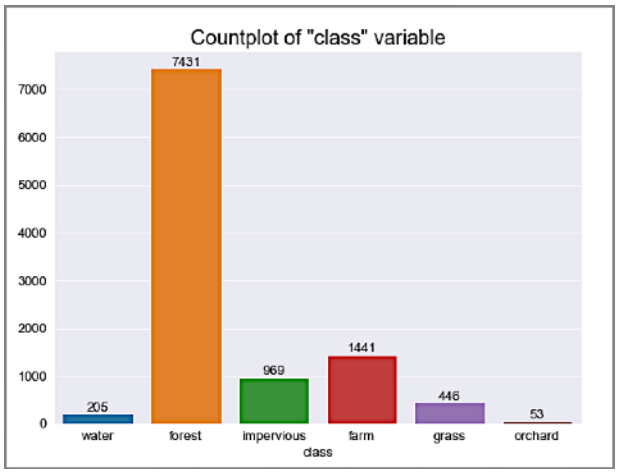

Fig. 3. Count-plot of target classes

As we can see from the Count-plot, forest class is dominant over all other classes.

Let's construct a box plot to check for outliers -

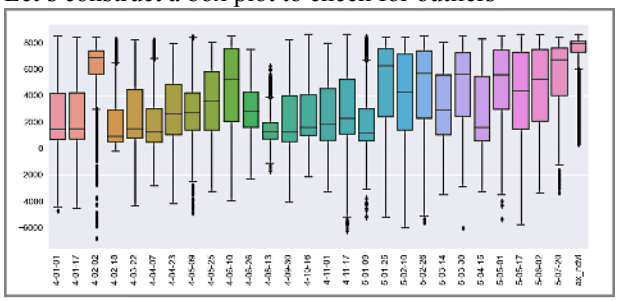

Fig. 4. Box-plot for all features

We can see that there are a lot of outliers in our dataset, this is already specified in UCI ML repository that noise reduction is very challenging in satellite image datasets. Atmospheric conditions like clouds, dust, haze, and fog interfere with sensors and make it difficult to produce accurate results.

Let's take some examples from some of the specific classes and check if they relate in some way with *max_ndvi* value.

Fig. 5. Class: orchard

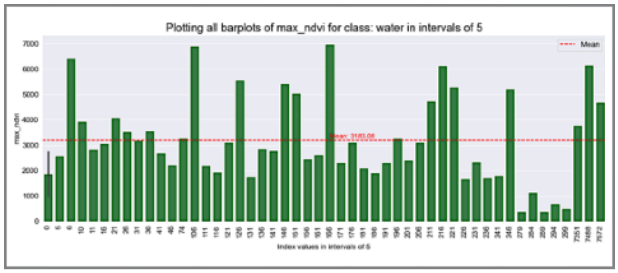

Fig. 6. Class: water

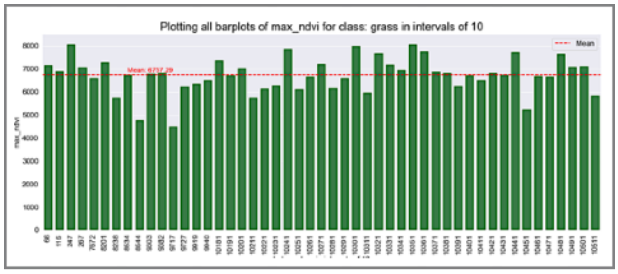

Fig. 7. Class: grass

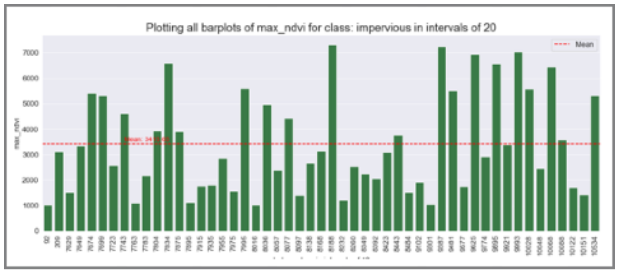

Fig. 8. Class: impervious

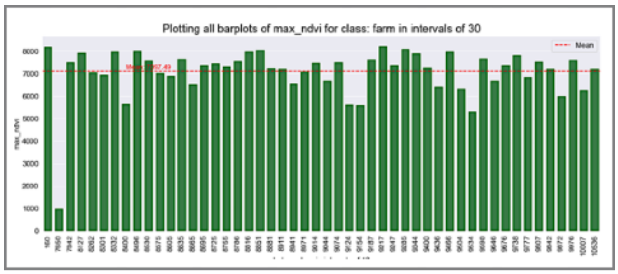

Fig. 9. Class: farm

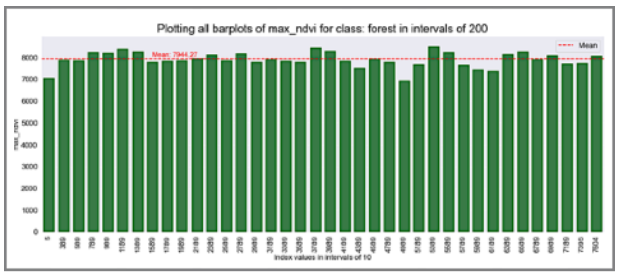

Fig. 10. Class: forest

We can conclude from the above graphs that classes have a lot of standard deviation (also seen form *df.info()*) and linear models will not fit properly.

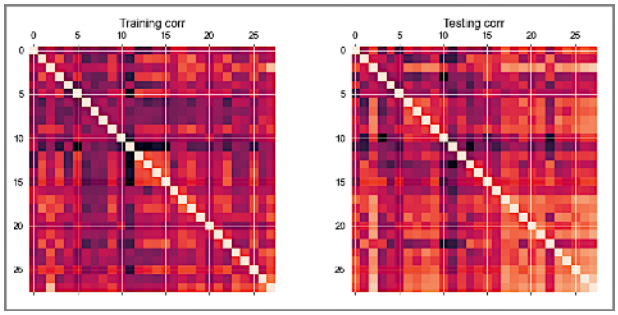

Fig. 11. Training and testing correlation matrices

Note that we can already see the difference b/w distributions of the two, you'll understand why am I saying this as I have surprise for you later on!

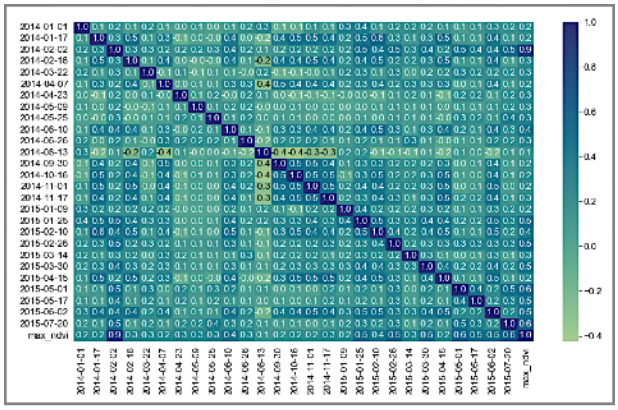

Fig. 12. Detailed and labelled correlation matrix for training dataset

We can infer from this correlation matrix that there isn't that much correlation b/w any attributes

Let's group the data by month and check for any trends in that! -

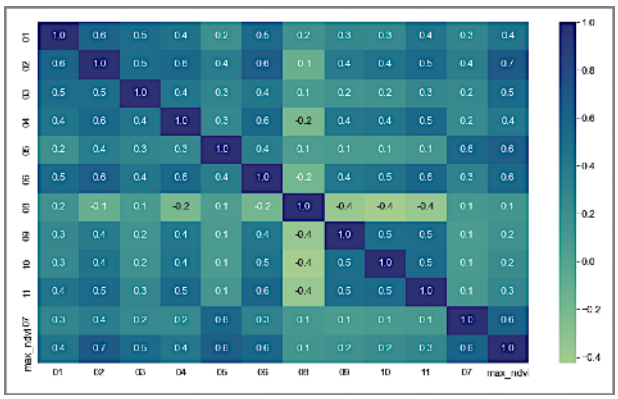

Fig. 12. Detailed and labelled correlation matrix for training dataset aggregated on months

This also did not tell us much about nature of dataset. There is somewhat better correlation among attributes now, but certainly not enough data to make some conclusions.

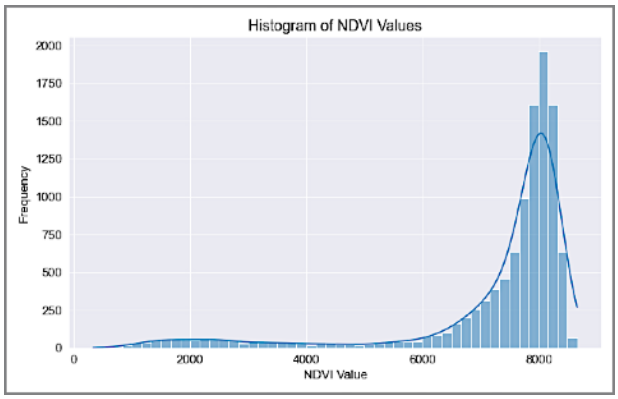

Fig. 13. Histogram + KDE plot to check probability distribution of max_ndvi attribute

We can clearly see a spike over ~8000 NDVI values. We also know that 'Forest' class has maximum frequency (by a relatively large margin) by the count plot earlier, and we also know that mean of forest class ~7940 NDVI value from bar plots. Everything is clicking together perfectly!

C. Feature Engineering

1) LabelEncoder: 'Classes' column was the only column with non-integer-type values. We need to encode it so that a model can be fitted over it. We use sklearn . preprocessing . LabelEncoder for this purpose
Encodings dictionary: {'farm': 0, 'forest': 1, 'grass': 2, 'impervious': 3, 'orchard': 4, 'water': 5}

2*) Splitting dataset:* We already have train and test DataFrames, so splitting the dataset into X_train, X_test, y_train, y_test is easy -

```
X_train = train.drop('class', axis=1)
X_test = test.drop('class', axis=1)
y_train = train['class'].copy()
y_test = test['class'].copy()
```

Each will have a shape of -

- (10545, 28)
- (300, 28)
- (10545,)
- (300,)

D. First phase of model fitting

There are various classification models readily available in sklearn library like - SVM, LogisticRegression, DecisionTree etc. But we choose RandomForestClassifier, as it is an ensemble model and doesn't require much hyper-parameter tuning.
We create a base model with default hyper parameters and fit it on the dataset →

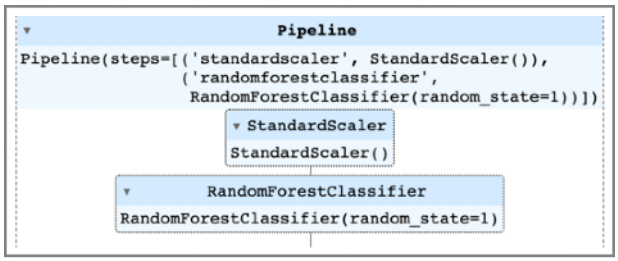

Fig. 14. Base RandomForestClassifier pipeline

	precision	recall	f1-score	support
0	0.67	0.79	0.72	53
1	0.47	0.87	0.61	78
2	0.64	0.19	0.30	36
3	0.88	0.95	0.92	40
4	0.00	0.00	0.00	47
5	1.00	0.83	0.90	46
accuracy			0.64	300
macro avg	0.61	0.61	0.58	300
weighted avg	0.59	0.64	0.58	300

Fig. 15. Classification report of base model

As we can see that we are only getting 64% accuracy, this is not optimal.

Let's see the learning curves to further investigate into this matter.

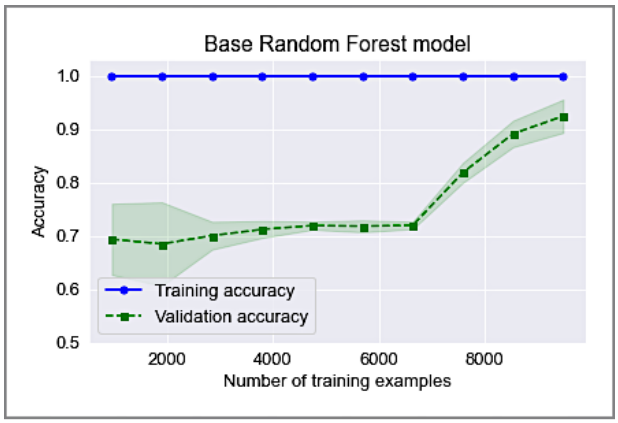

Fig. 16. Training and validation learning curves

We can clearly see in this graph that we have achieved almost 100% training accuracy, pretty early in the dataset, but model doesn't perform that well on test dataset
There can be several reasons for that -

- Overfitting
- Data mismatch
- Lack of regularization

1) Tackling overfitting - Above we have already used *sklearn.model_selction.learning_curve*, with 10 folds to

check how the model performs on various shuffled-validation sets. Let's use the *sklearn.model_selection.cross_validate* to select different folds of data and checking training accuracy for each of them

```
from sklearn.model_selection import cross_validate

cross_validate(pipe, X_test, y_test,cv=60, scoring='accuracy',
                       return_train_score=True)['train_score']
```

✓ 4.6s

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1.])
```

Fig. 17. Using cross-validation

This is something really concerning, we have fitted 60 folds of model on training set, and all have achieved perfect 100% accuracy.
We can very easily conclude that the base model is overfitting our dataset.

2) Checking for data-mismatch - Second we need to check, out test-data should be similar to our training-data.
First red flag that we observe is that our training set had 10545 values whereas our test set has 300 values only (~2.8%). This is bound to cause overfitting!
Secondly let's look at distribution of classes in our test-set →

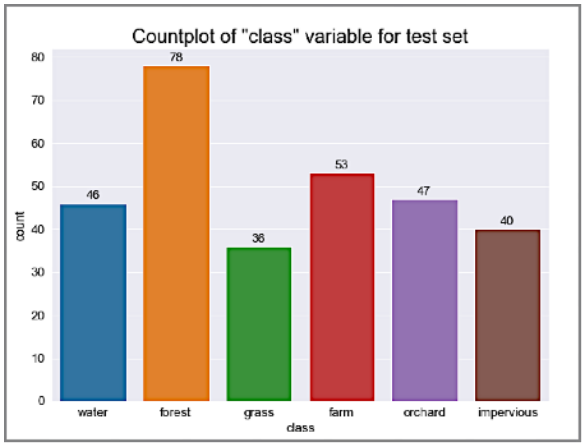

Fig. 18. Count-plot of target classes in test-set

Comparing Fig. 8. with Fig. 1. we will notice that test-set distributions are pretty off.
So, test set is not representative of the distribution of the data that the model is expected to encounter in practice.
We would need to tackle this problem!

E. Handling data mismatch

We have a train and test DataFrames, but the problem is both have different distributions. So we can combine them both using *pd.concat* and then split using *sklearn.model_selection.train_test_split*.

```
from sklearn.model_selection import
train_test_split
X = train.drop('class', axis=1)
y = train['class'].copy()
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=300/len(X))
```

Then we perform the normal steps - LablelEncoder() → StandardScaler() → RandomForestClassifier()

Let's see the classification report ->

	precision	recall	f1-score	support
farm	0.95	0.90	0.92	41
forest	0.94	1.00	0.97	209
grass	0.89	0.62	0.73	13
impervious	0.88	0.85	0.87	27
orchard	1.00	0.25	0.40	4
water	1.00	0.67	0.80	6
accuracy			0.94	300
macro avg	0.94	0.71	0.78	300
weighted avg	0.94	0.94	0.93	300

Fig. 19. Classification report after reshuffling

We can see the drastic improvement in accuracy of the model!

Let's plot the learning curves to be on the safe side

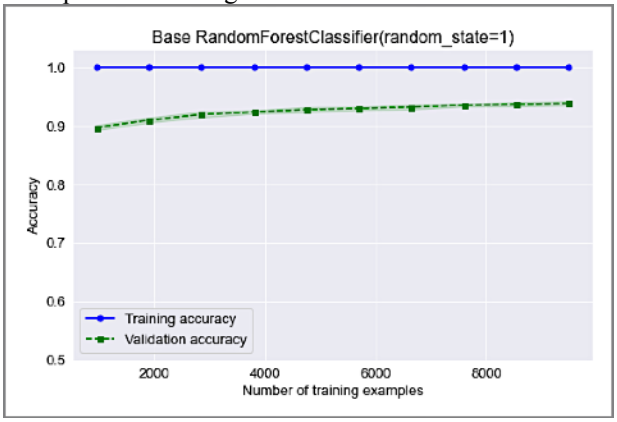

Fig. 20. Learning curves after reshuffling

F. Testing on more ML models

RandomForest performs very well, but we should not be biased towards a model. We should check for more models to actually see how they are performing.
There is a chance we might get surprised!

Let's try SVC first -

```
######## Classification report for SVC ##################
              precision    recall  f1-score   support

        farm       0.90      0.90      0.90        41
      forest       0.98      0.99      0.98       209
       grass       1.00      0.85      0.92        13
  impervious       0.86      0.93      0.89        27
     orchard       1.00      0.75      0.86         4
       water       1.00      0.67      0.80         6

    accuracy                           0.96       300
   macro avg       0.96      0.85      0.89       300
weighted avg       0.96      0.96      0.96       300
```

Fig. 21. Classification report of SVC

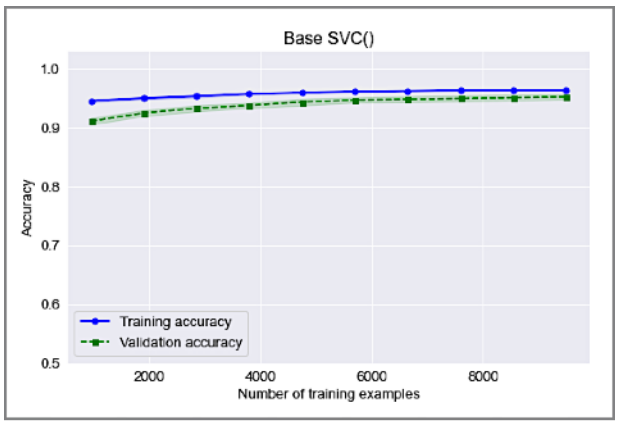

Fig. 22. Learning curves of SVC after reshuffling

Now, Let's try another basic classification model, LogisticRegression-

```
###### Classification report for LogisticRegression ######
              precision    recall  f1-score   support

        farm       0.74      0.68      0.71        41
      forest       0.92      0.95      0.93       209
       grass       0.73      0.62      0.67        13
  impervious       0.89      0.93      0.91        27
    orchard       0.50      0.25      0.33         4
       water       1.00      0.67      0.80         6

    accuracy                           0.88       300
   macro avg       0.80      0.68      0.73       300
weighted avg       0.88      0.88      0.88       300
```

Fig. 23 Classification report of LogisticRegression

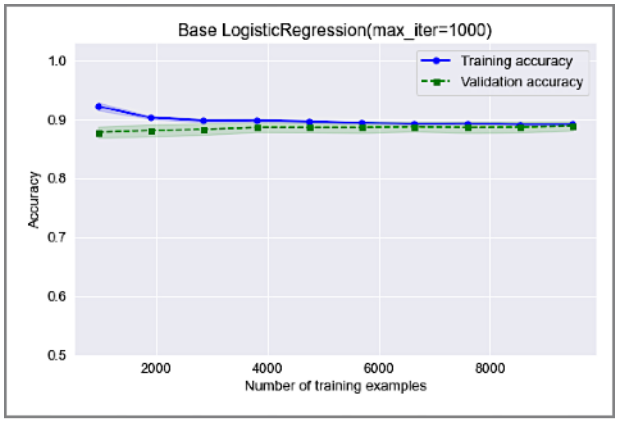

Fig. 24. Learning curves of LR after reshuffling

So, after reshuffling of dataset, SVC performed even better than RandomForestClassifier!!

G. Hyper-parameter tuning on best model

Although SVC is near perfect, let's try to manipulate its hyperparamters to see how better outputs we can get.
One of the most important hyperparameter in SVC is 'C', which is the Regularization parameter.
Let's plot a validation curve for various values of C and judge it on basis of training and validation accuracy. We would be using 10-fold CrossValidation as usual -

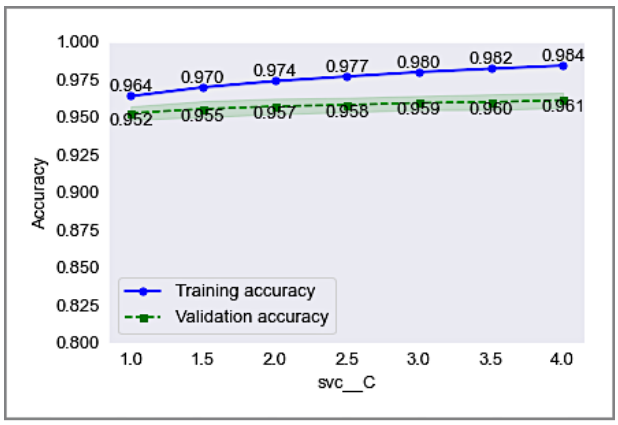

Fig. 25. Validation curve for hyperparameter 'C'

Now, there are two ways of doing hyperparameter tuning -

- GridSearchCV - It is computationally intensive as checks for every combination in parameter dictionary.
- RandomizedSearchCV - Randomly searches for best-parameters, so better approach than the former.

```
random_grid = {
'svc__kernel': ['linear', 'poly', 'rbf'],
'svc__C': list(np.arange(0.5, 5.5, 0.5)),
              }

svc_RSCV = RandomizedSearchCV(
          estimator = pipe_svc,
          param_distributions = random_grid,
          n_iter = 10, cv=10, verbose = 2,
          random_state = 42, n_jobs = -1,
          scoring='accuracy'
          )
svc_RSCV.fit(X_train, y_train)
```

Output was ->

```
Best hyperparameters: {'svc__kernel': 'rbf',
'svc__C': 4.0}
Best training accuracy: 0.9610237686268516
```

But training accuracy doesn't matter that much, we actually need to see how our model performs on test set,
Let's compare both base SVC and hyper tuned SVC -

```
Actual accuracy on test set->
Base model:  0.9366666666666666
RSCV model:  0.9566666666666667
Improvement of 2.14%.
```

So, hyperparameter tuning actually improved our accuracy!!

Also, let's see the model performance using confusion matrices.
We actually have 6-class target, so our confusion matrix would be 6x6

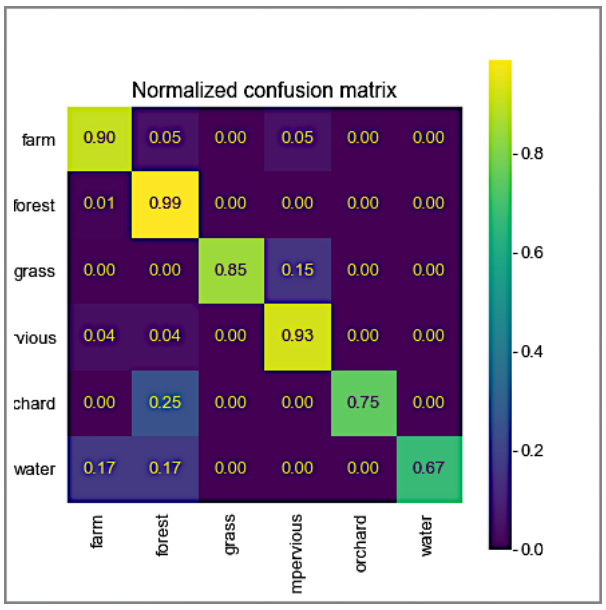

Fig. 26. Normalized Confusion matrix for 6 classes

Let's plot a separate confusion matrix for each class, for better visualization-

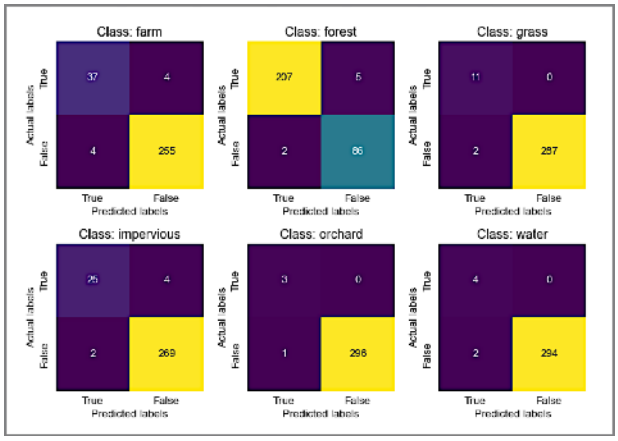

Fig. 27. Six binary confusion matrices for each class

G. Other tried techniques

Here is a list of other techniques I tried to use but accuracy was dropping -

- Using different normalisation techniques→
  - *StandardScaler()* was the best, with test accuracy ***96%***.
  - *MinMaxScaler()* → ***95%***.
  - *MaxAbsScaler()* → ***95%***.
  - *RobustScaler()* → ***95%***.
- Using feature selection to remove unnecessary features →
  - *RFE(n_features_to_select=10)* dropped the accuracy to ***91%***.
  - *SelectKBest()* dropped the accuracy to ***89%***.
  - *SelectPercentile()* dropped the accuracy to ***82%***.
  -
- Using feature engineering to generate more features →
  - *PolynomialFeatures(degree=2)*. In the output accuracy value dropped to ***94%***.
  - *PolynomialFeatures(degree=3),* took an awful lot of time and also dropped the accuracy to ***87%***.
- Using feature transformations to transform data into different probability distributions →
  - *PowerTransformer()* actually increased accuracy to ***97%*** !!
  - *QuantileTransformer()* also increased the accuracy ***97%*** !!
  - *QuantileTransformer(output_distribution='normal')* gave the same accuracy of ***96%***.
  - *SplineTransformer()* gave the same accuracy of ***96%***.
  - Various functions used in FunctionTransformer *FunctionTransformer(<func>)* [Note: Some functions like np.sqrt or np.log have restricted domain(no-negative values), so we need to handle them beforehand otherwise our output will have NaN values, one way to do that is *FunctionTransformer(lambda x: np.log(np.abs(x))*]-

np.exp	***92%***	*np.cos*	***93%***
np.exp2	***94%***	*np.tan*	***70%***
np.expm1	***92%***	*np.sinh*	***95%***
np.log	***92%***	*np.cosh*	***90%***
np.log10	***92%***	*np.tanh*	***96%***
np.log2	***92%***	*np.sinc*	***92%***
np.log1p	***93%***	*np.i0*	***91%***
np.square	***92%***	*np.sign*	***93%***
np.sqrt	***93%***	*np.sign*	***93%***
np.sin	***96%***	*np.abs*	***94%***

V. ACKNOWLEDGEMENT

First and foremost, I would like to thank my supervisor, Jaspreet Kaur ma'm, for his guidance and support throughout this project. Her insights and suggestions were invaluable in shaping the direction of this research.

I would also like to thank the faculty and staff of the Computer Science Department for providing us with the resources and facilities needed to conduct this research.

Finally, I would like to thank my families and friends for their unwavering support and encouragement throughout this project.

Thank you all for your contributions and support.

REFERENCES

1. Brian A. Johnson, B. A. Johnson, & Kotaro Iizuka, K. Iizuka. (0000). Integrating OpenStreetMap crowdsourced data and Landsat time-series imagery for rapid land use/land cover (LULC) mapping: Case study of the Laguna de Bay area of the Philippines. *Applied geography, 67*, 140-149. doi: 10.1016/j.apgeog.2015.12.006