DS Project Report

Team no: 3

Topic: Crop Recommendation Using Ensemble of Machine Learning Model

Abstract:

Agriculture plays a pivotal role in the Indian economy and considered as one of predominant ancient practices. Agriculture contributes major part towards India's GDP. There is a need to increase crop productivity. The production of a particular farm depends upon soil characteristics, environmental characteristics, but the major part goes to crop selection to get a better yield. Farmers sometimes lack the knowledge to choose the best crop for their land using conventional and non-scientific methods. Incorrect crop selection can lead to loss. This work focuses on figuring out the best crop to cultivate to get optimum yield based on the site-specific parameters. Our proposed model takes the data of soil characteristics, environmental characteristics of a farm and the appropriate crop recommendations are given to the farmer based on the parameter values. Crop Recommendation is done through an Ensemble model using KNN, Random Forest, Gaussian Naïve Bayes, Logistic regression, SVM as base learners. To increase overall performance, the ensemble model employed in this work includes decisions from various base learners The Majority Voting mechanism is used for combining these base learners. When compared to other methods, the results achieved with this method are more accurate. The webapp is developed to display the recommended crop when the farmer enters his farm parameters.

Code:

import numpy as np

import pandas as pd

import time

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import GridSearchCV

from sklearn.neighbors import KNeighborsClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.naive\_bayes import GaussianNB

from sklearn.ensemble import VotingClassifier

from mpi4py import MPI

comm = MPI.COMM\_WORLD

rank = comm.Get\_rank()

size = comm.Get\_size()

if rank == 0:

    # Load data

    a= time.time()

    df = pd.read\_csv(r"Crop\_recommendation.csv")

    X = df.drop('label', axis=1)

    y = df['label']

    # Split data

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=0)

    b= time.time()

    tl = b-a

    print("Time taken to load data: ", tl)

    st = time.time()

    #create new a knn model

    knn = KNeighborsClassifier()

    #create a dictionary of all values we want to test for n\_neighbors

    params\_knn = {'n\_neighbors': np.arange(1,10), 'p': [1, 2]}

    #use gridsearch to test all values for n\_neighbors

    knn\_gs = GridSearchCV(knn, params\_knn, cv=5)

    #fit model to training data

    knn\_gs.fit(X\_train, y\_train)

    #save best model

    knn\_best = knn\_gs.best\_estimator\_

    #check best n\_neigbors value

    print(knn\_gs.best\_params\_)

    #create a new random forest classifier

    rf = RandomForestClassifier()

    #create a dictionary of all values we want to test for n\_estimators

    params\_rf = {'n\_estimators': [50, 100, 150], 'criterion': ['gini', 'entropy']}

    #use gridsearch to test all values for n\_estimators

    rf\_gs = GridSearchCV(rf, params\_rf, cv=5)

    #fit model to training data

    rf\_gs.fit(X\_train, y\_train)

    #save best model

    rf\_best = rf\_gs.best\_estimator\_

    #check best n\_estimators value

    print(rf\_gs.best\_params\_)

    #create a new logistic regression model

    log\_reg = LogisticRegression(max\_iter=1000, solver='liblinear')

    #fit the model to the training data

    log\_reg.fit(X\_train, y\_train)

    svc = SVC()

    #create a dictionary of all values we want to test for n\_estimators

    params\_svc = {'C': [0.1, 1, 10], 'gamma': [0.1, 0.01, 0.001]}

    #use gridsearch to test all values for n\_estimators

    svc\_gs = GridSearchCV(svc, params\_svc, cv=5)

    #fit model to training data

    svc\_gs.fit(X\_train, y\_train)

    #save best model

    svc\_best = svc\_gs.best\_estimator\_

    print(svc\_gs.best\_params\_)

    nb = GaussianNB()

    nb.fit(X\_train, y\_train)

    #test the three models with the test data and print their accuracy scores

    print('knn: {}'.format(knn\_best.score(X\_test, y\_test)))

    print('rf: {}'.format(rf\_best.score(X\_test, y\_test)))

    print('log\_reg: {}'.format(log\_reg.score(X\_test, y\_test)))

    print('svc: {}'.format(svc\_best.score(X\_test, y\_test)))

    print('nb: {}'.format(nb.score(X\_test, y\_test)))

    #create a dictionary of our models

    estimators=[('knn', knn\_best), ('rf', rf\_best), ('log\_reg', log\_reg) , ('svc', svc\_best), ('nb', nb)]

    #create our voting classifier, inputting our models

    ensemble = VotingClassifier(estimators, voting='hard')

    #fit model to training data

    ensemble.fit(X\_train, y\_train)

    #test our model on the test data

    print("Accuracy: ", ensemble.score(X\_test, y\_test))

    et = time.time()

    ts = (et-st) + tl

    sp = time.time()

    comm.bcast(X\_train, root=0)

    comm.bcast(y\_train, root=0)

    comm.bcast(X\_test, root=0)

    comm.bcast(y\_test, root=0)

elif rank == 1:

    X\_train = comm.bcast(None, root=0)

    y\_train = comm.bcast(None, root=0)

    X\_test = comm.bcast(None, root=0)

    y\_test = comm.bcast(None, root=0)

    #create new a knn model

    knn = KNeighborsClassifier()

    #create a dictionary of all values we want to test for n\_neighbors

    params\_knn = {'n\_neighbors': np.arange(1, 10), 'p': [1, 2]}

    #use gridsearch to test all values for n\_neighbors

    knn\_gs = GridSearchCV(knn, params\_knn, cv=5)

    #fit model to training data

    knn\_gs.fit(X\_train, y\_train)

    #save best model

    knn\_best = knn\_gs.best\_estimator\_

    #check best n\_neigbors value

    print(knn\_gs.best\_params\_)

    comm.send(knn\_best, dest=0, tag=11)

elif rank == 2:

    X\_train = comm.bcast(None, root=0)

    y\_train = comm.bcast(None, root=0)

    X\_test = comm.bcast(None, root=0)

    y\_test = comm.bcast(None, root=0)

    #create a new random forest classifier

    rf = RandomForestClassifier()

    #create a dictionary of all values we want to test for n\_estimators

    params\_rf = {'n\_estimators': [50, 100, 150], 'criterion': ['gini', 'entropy']}

    #use gridsearch to test all values for n\_estimators

    rf\_gs = GridSearchCV(rf, params\_rf, cv=5)

    #fit model to training data

    rf\_gs.fit(X\_train, y\_train)

    #save best model

    rf\_best = rf\_gs.best\_estimator\_

    comm.send(rf\_best, dest=0, tag=22)

    #check best n\_estimators value

    print(rf\_gs.best\_params\_)

elif rank == 3:

    X\_train = comm.bcast(None, root=0)

    y\_train = comm.bcast(None, root=0)

    X\_test = comm.bcast(None, root=0)

    y\_test = comm.bcast(None, root=0)

    #create a new logistic regression model

    log\_reg = LogisticRegression(max\_iter=1000, solver='liblinear')

    #fit the model to the training data

    log\_reg.fit(X\_train, y\_train)

    #save best model

    comm.send(log\_reg, dest=0, tag=33)

elif rank == 4:

    X\_train = comm.bcast(None, root=0)

    y\_train = comm.bcast(None, root=0)

    X\_test = comm.bcast(None, root=0)

    y\_test = comm.bcast(None, root=0)

    #create a new logistic regression model

    svc = SVC()

    #create a dictionary of all values we want to test for n\_estimators

    params\_svc = {'C': [0.1, 1, 10], 'gamma': [0.1, 0.01, 0.001]}

    #use gridsearch to test all values for n\_estimators

    svc\_gs = GridSearchCV(svc, params\_svc, cv=5)

    #fit model to training data

    svc\_gs.fit(X\_train, y\_train)

    #save best model

    svc\_best = svc\_gs.best\_estimator\_

    comm.send(svc\_best, dest=0, tag=44)

elif rank == 5:

    X\_train = comm.bcast(None, root=0)

    y\_train = comm.bcast(None, root=0)

    X\_test = comm.bcast(None, root=0)

    y\_test = comm.bcast(None, root=0)

    #create a new logistic regression model

    nb = GaussianNB()

    #fit the model to the training data

    nb.fit(X\_train, y\_train)

    #save best model

    comm.send(nb, dest=0, tag=55)

if rank == 0:

    knn\_best = comm.recv(source=1, tag=11)

    rf\_best = comm.recv(source=2, tag=22)

    log\_reg = comm.recv(source=3, tag=33)

    svc\_best = comm.recv(source=4, tag=44)

    nb = comm.recv(source=5, tag=55)

    #test the three models with the test data and print their accuracy scores

    print('knn: {}'.format(knn\_best.score(X\_test, y\_test)))

    print('rf: {}'.format(rf\_best.score(X\_test, y\_test)))

    print('log\_reg: {}'.format(log\_reg.score(X\_test, y\_test)))

    print('svc: {}'.format(svc\_best.score(X\_test, y\_test)))

    print('nb: {}'.format(nb.score(X\_test, y\_test)))

    #create a dictionary of our models

    estimators=[('knn', knn\_best), ('rf', rf\_best), ('log\_reg', log\_reg) , ('svc', svc\_best), ('nb', nb)]

    #create our voting classifier, inputting our models

    ensemble = VotingClassifier(estimators, voting='hard')

    #fit model to training data

    ensemble.fit(X\_train, y\_train)

    #test our model on the test data

    print("Accuracy: ", ensemble.score(X\_test, y\_test))

    ep = time.time()

    tp = (ep-sp) + tl

    print("Sequntial time: ", ts)

    print("Parallel time: ", tp)

    print("Speedup: ", ts/tp)

    print("Number of processors: ", size)

    print("Efficiency: ", (ts/tp)/6)

Result Screenshot:

