Cereal_Data_Challenge

Jonas Grove

10/5/2020

Data Preparation

Data process description

The provided weather and site data was integrated into the crop and cereal data, in order to expand the number of features available to be used to predict Assessment Score, using R. The weather data set was extensive, having measurements of 5 variables from every day throughout the plants life cycle, which allows for a lot of possible features. To reduce the dimensionality of the data set, the growing cycles were broken into four growing quarters (q1, q2, q3 and q4) and the average of the daily measurements within these quarters was averaged for each of the weather variables (A-F). These data were merged by matching weather samples and crop and grain samples by Site ID. The site data and crop data was merged such that all of the soil variables were matched to crop samples by Site ID. The dates were removed from the data, because the sample data already included a parameter Growth stage. The prepped data was written to a file and further analysis was performed using python and Tensorflow.

Code Block 1. Data Preparation.

```
setwd("/Users/jonasgrove/Projects/pepsi_cereal")
#install.packages("readxl")
library("readxl")

#load all data into frames
crop_data <- read_excel("nyas-challenge-2020-data.xlsx", sheet = "Crop and Grain Data")
weather_data <- read_excel("nyas-challenge-2020-data.xlsx", sheet = "Weather Data")
site_data <- read_excel("nyas-challenge-2020-data.xlsx", sheet = "Site Data")

#remove year and whitespace from site ID to make uniform
crop_data$`Site ID` <- gsub("Year", "", crop_data$`Site ID`)
crop_data$`Site ID` <- gsub("[[:space:]]", "", crop_data$`Site ID`)
weather_data$`Site ID` <- gsub("[[:space:]]", "", weather_data$`Site ID`)
site_data$`Site ID` <- gsub("[[:space:]]", "", site_data$`Site ID`)

##break weather data into 4 growing quarters
factors_sites <- as.factor(site_data$`Site ID`)
site_levels <- levels(factors_sites)
data_div_num <- 4
weather_all <- data.frame()
```

```
for (site_level in site_levels){
  site_subs <- subset(weather_data, subset = (weather_data$'Site ID' == site_level))
  div_num <- nrow(site_subs)%/%data_div_num
  q1 <- site_subs[1:div_num,]
  q2 <- site_subs[div_num+1:div_num,]
  q3 <- site_subs[2*div_num+1:div_num,]
  q4 <- site_subs[3*div_num+1:div_num,]

#average all daa from four different different growing quarter for vars A-F
  weather_new <- data.frame('Site ID' = site_level)
  all_qs = list(q1,q2,q3,q4)
  i = 2
  for (q in all_qs){
    j=i+5
    weather_vars <- data.frame("Weather Variable A"=c(mean(q$`Weather Variable A`)),
                               "Weather Variable B"=c(mean(q$`Weather Variable B`)),
                               "Weather Variable C"=c(mean(q$`Weather Variable C`)),
                               "Weather Variable D"=c(mean(q$`Weather Variable D`)),
                               "Weather Variable E"=c(mean(q$`Weather Variable E`)),
                               "Weather Variable F"=c(mean(q$`Weather Variable F`)))
    weather_new[,i:j] <- weather_vars
    i=i+6}
  weather_all[nrow(weather_all)+1,1:25] <- weather_new
  }

#merge all frames
all_data <- merge(x=crop_data, y=site_data, by.x = 'Site ID', by.y = 'Site ID')
all_data <- merge(x=all_data, y=weather_all, by.x = 'Site ID', by.y = 'Site.ID')

#obtain targets (assessment and assessment type)
targets <- all_data[,6:7]

#remove dates, as this info is captured in
all_data$`Assessment Date (mm/dd/year)` <- all_data$`Sowing Date (mm/dd/year)` <- all_data$`Harvest Dat



#write features and targets to file
write.table(all_data, file = "/Users/jonasgrove/Projects/pepsi_cereal/prep_cereal_data", sep = "\t", ro
```

The data was further prepared using matplotlib and sklearn. The Assessment type variable was converted into 28 dummy variables and the Variety was turned into two dummy variables. All missing Assessment samples were removed and the data was scaled using sklearn RobustScaler.

Dummy Variables and Scaling

```
import pandas as pd
import tensorflow as tf
from sklearn import preprocessing
import numpy as np
```

```
all_data = pd.read_csv("prep_cereal_data", sep='\t', header=0)


#turn assessment type and variety into dummy vars
at_dums = pd.get_dummies(all_data["Assessment Type"])
all_data = all_data.drop(columns="Assessment Type")
all_data = pd.concat([all_data, at_dums], axis=1)

var_dums = pd.get_dummies(all_data["Variety"])
all_data = all_data.drop(columns="Variety")
all_data = pd.concat([all_data, var_dums], axis=1)

all_data = all_data.drop(columns="Site ID")
all_data = all_data.dropna()
all_data = all_data[all_data["Assessment Score"] != '*']

#split features and target
Y = all_data["Assessment Score"]
X = all_data.drop(columns="Assessment Score")

#scale features
from sklearn.preprocessing import RobustScaler
transformer = RobustScaler().fit(X)
X = transformer.transform(X)

Y = np.array(Y)
Y[Y == ''] = 0.0
Y = Y.astype(np.float)
```

Model Development

```
#make dense network model
import neural_net
NeuralNet = neural_net.NeuralNet

#check accuracy
from sklearn.metrics import mean_squared_error

test_layers_up = 15
test_nodes_up = 500
out_file = open("nyas_score_model.txt", "w")
all_scores = []

#loop through possible node layer cobos
for i in range(5,test_layers_up):

    for j in range(100,test_nodes_up):
        crop_score_model = NeuralNet(X, Y, i, j, "r", 20)
        y_pred = crop_score_model.test_result
        loss = mean_squared_error(crop_score_model.y_test ,y_pred)
        result = "layers: " + str(i) + "nodes: " + str(j) + "loss: "+ str(loss)
        out_file.write(result)
        print(result)
        all_scores.append(result)

print(all_scores)
out_file.close()
```

A fully connected neural network was made using the keras API built on tensor flow, and trained on the previously prepped crop data. The model was developed by training on different node and layer number combinations, staring by iterating through layers 1-20 and nodes 100-500. Good results were seen starting at 5 layers, based on the mean squared error (MSE), so iterations were changed to train models with nodes from 100 to 500 and layers 5 to 15. A combination was found with a MSE of 94, which had 5 layers with 259 nodes each. These hyperparameters were used to develop the final model.

Results

The final model was made using 5 layers with 249 layers and was trained over 500 and 1000 epochs which seem to result in an MSE between 200 and 300. Given more data, more testing and training could be performed and the model could be more refined. More data would also allow for a larger model to be developed and would likely result in higher accuracy. Another way that this model could be improved is by standardizing the assessment type. The assessment type seems to affect the assessment score, so it was included as dummy variables, but building the model on a larger data set that all has the same assessment type would likely be more accurate. Another method to deal with the assessment type could be to normalize all the assessment scores by assessment type, such that they all have the same mean and standard deviation. Overall a model was developed that is capable of acurately predicting the assessment with a MSE of 94, which could greatly be improved given more data and more refinement to the model.

Neural Network Class

```
#!/usr/bin/env python

import pandas as pd
import tensorflow as tf
from sklearn import preprocessing
import numpy as np

class NeuralNet:

    def __init__(self, X, Y, number_of_layers, nodes_per_layer, pred, epoch_num):
        from sklearn.model_selection import train_test_split
        self.input_nodes_num = X.shape[1]
        self.input_layer = tf.keras.layers.Input(shape=(self.input_nodes_num,))
        self.number_of_layers = number_of_layers
        self.nodes_per_layer = nodes_per_layer
        self.epochs = epoch_num

        self.pred = pred

        self.x_train, self.x_develop, self.y_train, self.y_develop = train_test_split(X, Y, test_size=0

        self.x_train, self.x_test, self.y_train, self.y_test = train_test_split(self.x_train, self.y_tra

        self.model = self.train()

        self.test_result = self.predict_nn(self.x_test)

    def train(self):

        hidden_layer = tf.keras.layers.Dense(self.nodes_per_layer)(self.input_layer)
        hidden_layer = tf.keras.layers.LeakyReLU()(hidden_layer)

        for i in range(self.number_of_layers - 1):
            hidden_layer = tf.keras.layers.Dense(self.nodes_per_layer)(hidden_layer)
            hidden_layer = tf.keras.layers.LeakyReLU()(hidden_layer)
```

```
        if self.pred == "r":
            output_layer = tf.keras.layers.Dense(1)(hidden_layer)
            nn_model=tf.keras.models.Model(self.input_layer,output_layer)
            nn_model.compile(loss='mse',optimizer='adam')
            nn_model.fit(self.x_train, self.y_train ,epochs=self.epochs,validation_split=0.5)

        elif self.pred == "m":
            output_layer = tf.keras.layers.Dense(3,activation='softmax')(hidden_layer)
            nn_model = tf.keras.models.Model(input_layer,output_layer)
            nn_model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
            nn_model.fit(self.x_train, self.y_train ,validation_data=(self.x_develop,self.y_develop_one

        elif self.pred == "b":

            output_layer = tf.keras.layers.Dense(1,activation='sigmoid')(hidden_layer)
            nn_model = tf.keras.models.Model(self.input_layer, output_layer)
            nn_model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
            nn_model.fit(self.x_train, self.y_train, epochs=self.epochs, validation_data=(self.x_develo

        return nn_model

    def predict_nn(self,x):

        prediction = self.model.predict(x)

        return prediction
```