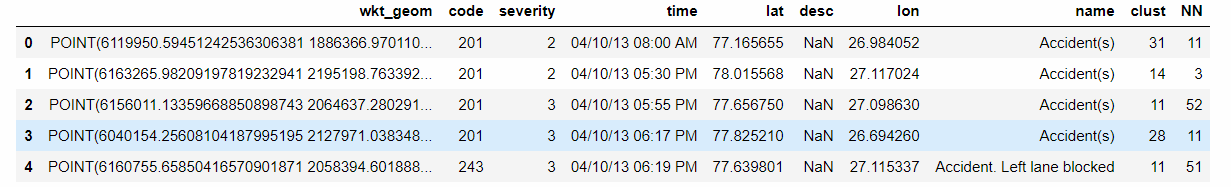
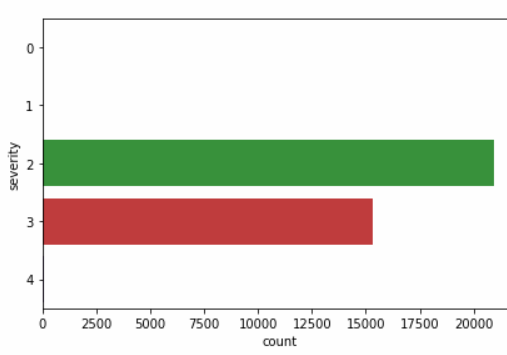
Import libraries and the data file

1. %matplotlib inline
2. **import** warnings
3. warnings.filterwarnings('ignore')
4. **import** pandas as pd
5. **import** numpy as np
6. **import** matplotlib.pyplot as plt
7. **import** seaborn as sns
9. path = "C:/Users/Rahul/Desktop/Desktop Items/Study/notes vi & vii sem/DM project/Project DM/"
10. df = pd.read\_csv( path + "accidents.csv" )
11. **print** ("The dataset has %d rows and %d columns" % (df.shape[0] , df.shape[1]))
12. df.head()

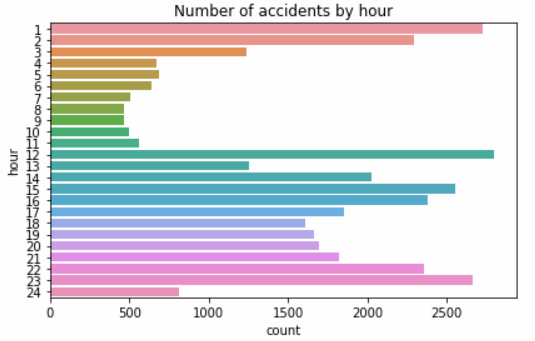


Characteristics and summary statistic of data with graphs

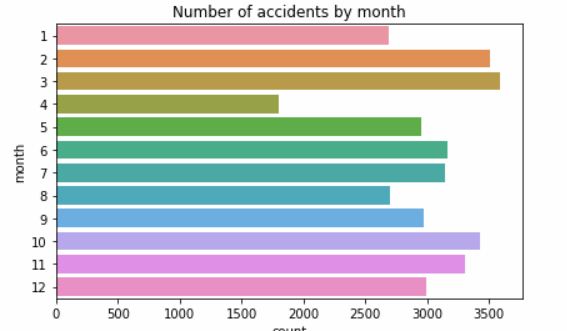
1. #character stick graph
2. #severity vs count
4. sns.countplot(y = "severity" , data = df )
5. plt.tight\_layout()



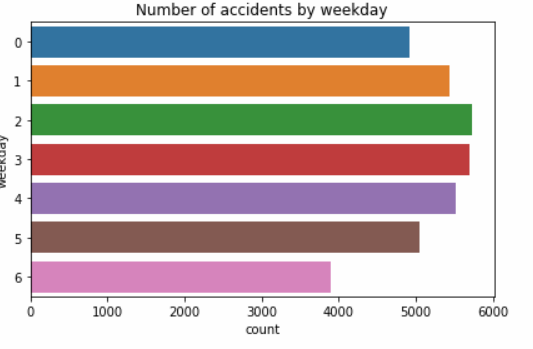
1. #classifiction , hour vs counts
3. sns.countplot(y = "hour" , data = df , order = range(1,25))
4. plt.title("Number of accidents by hour")
5. plt.tight\_layout()



1. #classification , month vs count
3. sns.countplot(y = "month" , data = df)
4. plt.title("Number of accidents by month")
5. plt.tight\_layout()



1. #classification , weekdays vs count
3. sns.countplot(y = "month" , data = df)
4. plt.title("Number of accidents by month")
5. plt.tight\_layout()



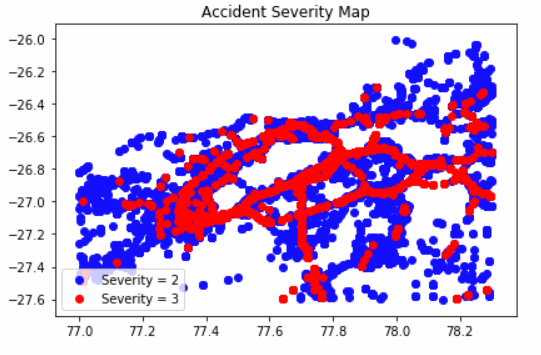
Error’s and Outlier removing

1. #here we check for outlier and boundaries
2. pd.DataFrame( {"count": df["severity"].value\_counts().values } , index = df["severity"].value\_counts().index )
3. df = df.loc[df["severity"] >  1].loc[df["severity"] < 4]



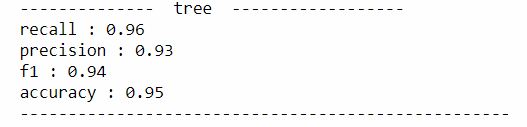
Distributions of features and labels

1. #plotting the dataset with a different color depending on the severity (2,3)
2. #longitude vs lattitude
4. df2 = df.loc[df["severity"] == 2]
5. df3 = df.loc[df["severity"] == 3]
7. xx2 , yy2 = df2["lat"] , -df2["lon"]
8. xx3 , yy3 = df3["lat"] , -df3["lon"]
10. pts2 = plt.scatter(xx2,yy2,color = 'b' )
11. pts3 = plt.scatter(xx3,yy3,color = 'r' )
12. plt.legend((pts2, pts3), ('Severity = 2', 'Severity = 3'),loc='lower left')
13. plt.title("Accident Severity Map")
14. plt.tight\_layout()



Preprocessing, covert the time format to months, weekdays, years, hours

1. df["month"] = df["time"].apply(**lambda** x:int(x[:2]))
2. df["day"] = df["time"].apply(**lambda** x:int(x[3:5]))
3. df["year"] = df["time"].apply(**lambda** x:int(x[6:8]))
4. df["hour"] =  df["time"].apply(**lambda** x: int(x[9:11]) **if** str(x)[15] == 'A' **else** 12 + int(x[9:11])  )
5. df["lon"] = df["lon"].apply(**lambda** x:abs(x)) #so that multinomialNB works (only with positive features)
6. #creating the date at the datetime format (easier to deal with)
7. df[ "date" ]= df[["month" , "day" ,"year"]].apply(**lambda** x:pd.datetime(month = x['month'] , day = x['day']  , year = 2000+x["year"]), axis = 1)
8. df["weekday"] =  df["date"].apply(**lambda** x:x.weekday())

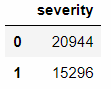


Applying Cross Validation and Splitting the data into training set and test sets , ratio of 2:8

1. X = df[["month" , "hour" , "year", "weekday" ,"lon" , "lat"]]
2. y = df["severity"].apply(**lambda** x:x-2) # shifting to 0-1 values instead of 2-3
4. #here we assign test size as 20% of actual data set
5. # random state is set as 42 ( or 1 ) also from the reference of " The Hitchhiker's Guide to the Galaxy"
6. # we defined random state to get consistent and same results , regardless of the training iterations
7. # so that the values in the train and test sets are homogenous
9. **from** sklearn.cross\_validation **import** train\_test\_split
10. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=42)

Basic Algorithm Model

1. #basic model to check accurary (worst)
3. sev = y.value\_counts()
4. pd.DataFrame(sev)
5. #here we will calculate the worst accuracy
7. **print** ("worst accuracy: " , max(sev)/float(sum(sev)))



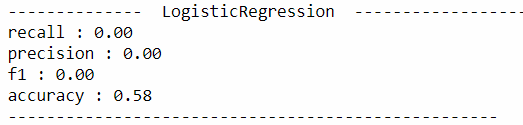


**Applied algorithm’s**

Logistic Regression

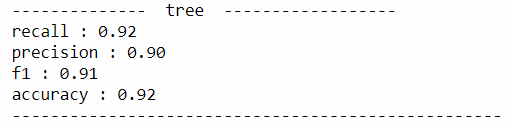
1. #logreg (it predicts everything to 0, the most common class)
2. # a predictive analysis , where we comapre a relation between binary varaible
3. # and other ordinal and nominal and indepdendent variables
4. #also here we are not using any paramneter's
5. #use from training set and test sets
6. #for cross valiadtion

9. **from** sklearn.linear\_model **import** LogisticRegression
10. clf = LogisticRegression()
11. clf.fit(X\_train,y\_train)
12. y\_pred = pd.Series(clf.predict(X\_test))
13. printScores(y\_test, y\_pred, "LogisticRegression")



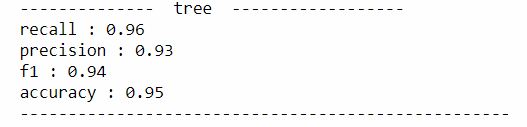
Tree Algorithm

1. #tree
3. #structured tree where it containts root nodes , leaf node
4. #each brach of the tre depicts outcome of the tree
6. # use from training set and test sets
7. #for cross valiadtion
9. **from** sklearn **import** tree
10. clf = tree.DecisionTreeClassifier()
11. clf.fit(X\_train,y\_train)
12. y\_pred = clf.predict(X\_test)
13. printScores(y\_test, y\_pred, "tree")



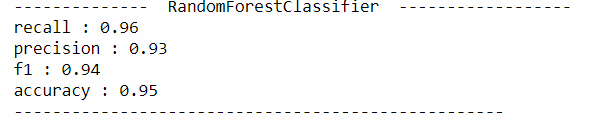
Tree algorithm (modified training set)

1. #tree
2. #MODIFIED TRAINING SET of lat and longitude
3. #this one is parametrized
4. # use from training set and test sets , only 2 variables , lat and lon
5. #for cross validation
7. X\_train2 , X\_test2 = X\_train[["lat" , "lon" ]]  , X\_test[["lat" , "lon" ]]
8. **from** sklearn **import** tree
9. clf = tree.DecisionTreeClassifier()
10. clf.fit(X\_train2,y\_train)
11. y\_pred = clf.predict(X\_test2)
12. printScores(y\_test, y\_pred, "tree")



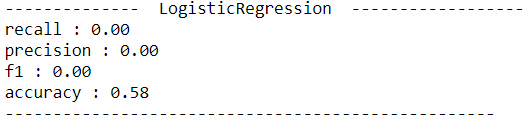
Random Forest Algorithm (modified training set)

1. #random forest
3. # use from training set and test sets
4. # for cross valiadtion
5. #n\_estiamotor is 100 , as default from update of 2.10
7. **from** sklearn.ensemble **import** RandomForestClassifier
8. clf = RandomForestClassifier(n\_estimators = 100)
9. clf.fit(X\_train2,y\_train)
10. y\_pred = clf.predict(X\_test2)
11. printScores(y\_test, y\_pred, "RandomForestClassifier")



Logistic Regression (Modified training Set)

1. #logreg (it predicts everything to 0, the most common class)
3. # use from training set and test sets
4. #for cross valiadtion
6. **from** sklearn.linear\_model **import** LogisticRegression
7. clf = LogisticRegression()
8. clf.fit(X\_train2,y\_train)
9. y\_pred = pd.Series(clf.predict(X\_test2))
10. printScores(y\_test, y\_pred, "LogisticRegression")



Training set graph prediction

1. #drawing the prediction graph
3. training\_set\_size = [0.05\*i **for** i **in** range(1,21)]
4. accuracy = []
5. **from** sklearn **import** tree
6. **for** size **in** training\_set\_size:
7. # won't be using the test in that case...this is just a way of splitting the data
8. X\_train2, X\_test2, y\_train2, y\_test2 = train\_test\_split(X\_train, y\_train, test\_size=1-size, random\_state=42)
9. clf = tree.DecisionTreeClassifier()
10. clf.fit(X\_train2,y\_train2)
11. y\_pred = clf.predict(X\_test)
12. #printScores(y\_test, y\_pred, "tree")
13. accuracy.append(  (X\_train2.shape[0] ,accuracy\_score(y\_test, y\_pred) )  )
15. xx = [w[0] **for** w **in** accuracy]
16. yy = [w[1] **for** w **in** accuracy]
17. plt.scatter(xx,yy)
18. plt.xlabel('Training Set Size')
19. plt.ylabel('Accuracy')
20. plt.title("Learning Curve: Accuracy vs training set size")
21. plt.tight\_layout()

