**Statoil Iceberg challenge:**

* First we have downloaded the .7Z files and extracted the json files and loaded it into pandas dataframe and found that there are missing values in “inc\_angle” column
* So, we replaced the missing(“na”) values with its mean.
* **First method:**

Used Keras module for building Convolutional Neural Network model for this image classification problem.

Transformed each array of image(5625 array values) into a feature attribute and added “band\_2” column.

The output variable will be "is\_iceberg".

Splitted the data in the ratio of 60(train), test (40).

Built a sequential model of neural net.

Used different layers of neural nets like 3, 4 and 5 layers.

Added different neuron counts in each layer with different activation functions.

As it is a binary classifier we used “sigmoid” activation function in the last level.

Compiled the model using “adam” optimizer.

Evaluated the model on the train data. The accuracy level of prediction is coming around 51%.

* **Second method:**

Observed that only the image of the “iceberg” or “ship” is captured in the 30\*30 [20:50,20:50] area of 75\*75.

Extracted that area pixels and used both” band\_1” and ” band\_2” images and “inc\_angle”(1801 features).

Followed the first method with different number of activation layers, different activation functions.

This time got an accuracy level of 53%.

* **Third method:**

Used the same features from the second method and built a KNeighborsClassifier (n=3) model.

Splitted the data in the ratio of 90(train), test (10).

Fitted the model on training data and predicted the values of test data.

Found out the confusion matrix.

Got an accuracy level of 59%

Used the 10-fold cross validation technique on the whole data and got an average accuracy of 79%.

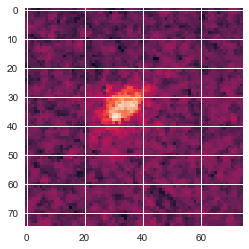
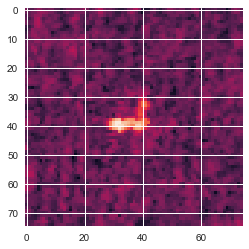
Right now, we are using Trial and error method for finding the optimal solution by varying the model and parameters.

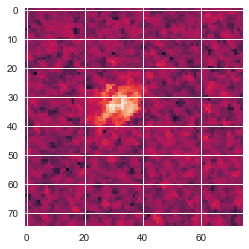
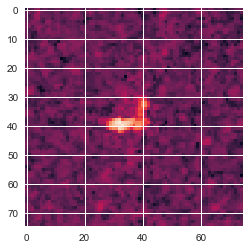
**Preprocessing the data:**

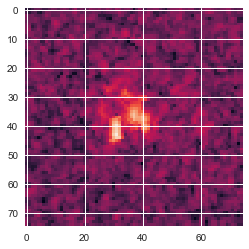
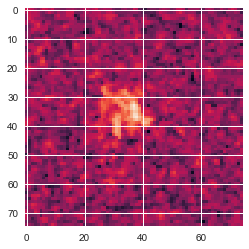
There are some missing values in the “incidence angle” column from the data.

Replaced the missing value with the mean of the “incidence angle” . So that we do not drop many rows in the dataset.

Some sample images from the data

One observation is that most of the data is concentrated on the [20:50,20:50] matrix grid and the rest of the image has ocean as a background.

Second observation is that there is not much difference if we take **band\_1 data or band\_2 data.**

**Reshape the data in each column of band\_1 and band\_2 and select the [**20:50,20:50] matrix grid, and reshape it to an array and convert each value of the array into a attribute for building the model

**We have 900\*2 + 1(inc\_angle)= 1801 attributes and one value to predict i.e., is\_iceberg(Binary Variable).**

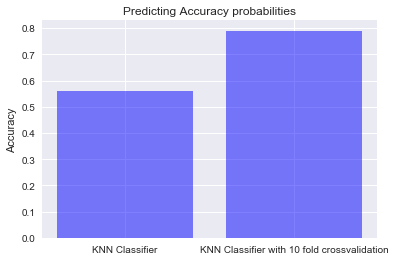
**Implementing KNearest Neighbours classifier:**

**Splitted the data into train and test in the ratio of 80/20**

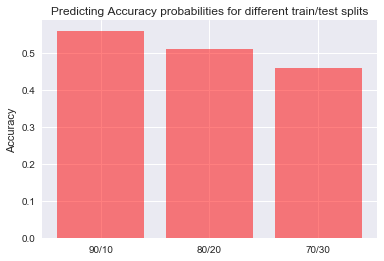
**Fitted the Scikit learn’s KNearest Neighbours classifier model to fit the data on the training data.**

**Found out the confusion matrix and got an accuracy of 56% for prediction on test data**

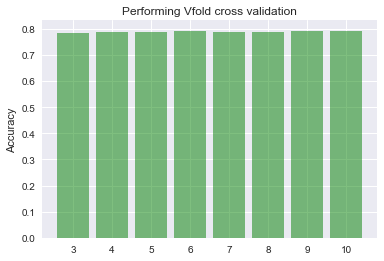
**Used Scikit learn’s 10 fold cross validation score to find the mean accuracy and got an accuracy of 79% on the total data.**



**If we do Vfold cross validation the model predicting accuracy will improve**



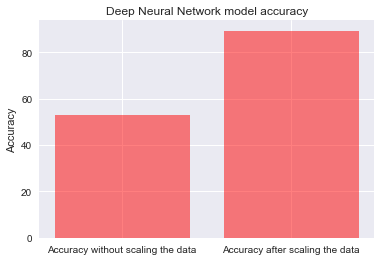
**For this data the here maximum accuracy is achieved when we keep the train/test ratio to be 90/10.**



**We can conclude that for this data set Vfold cross validation is independent of number of cross validations we have done.**

**Implementing TensorFlow (Keras Deep Neural network):**

**Splitted the data into train and test in the ratio of 60/40**



**Accuracy Improves way better than expected for deep neural nets if we scale the attributes.**

from sklearn import preprocessing

import pandas as pd

import json

import numpy as np

import matplotlib.pyplot as plt

from PIL import Image

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

from keras.models import Sequential

from keras.layers import Dense

import numpy

# fix random seed for reproducibility

numpy.random.seed(7)

with open("train.json") as f:

d=json.loads(f.read())

df=pd.DataFrame(d)

"""df=df.drop(["id","band\_2"],axis=1)"""

df.info()

df=df.replace("na",40)

dataa=[]

for i in range(1604):

data1=df.ix[i,0]

dataf=np.reshape(data1,(75,75))

dataf2=dataf[20:50,20:50]

dataf3=np.reshape(dataf2,(900,))

dataa.append(dataf3)

dataa=np.array(dataa)

"""

names=[i for i in range(1604)]

X=pd.DataFrame()

for i in range(1604):

data1=df.ix[i,0]

dataf=np.reshape(data1,(75,75))

FD=dataf[20:50,20:50].reshape(900,1)

for j in range(900):

X.ix[i,j]=FD[j]

data=[]

for i in range(1604):

data.append(df.ix[i,0])

"""

datab=[]

for i in range(1604):

data1=df.ix[i,1]

dataf=np.reshape(data1,(75,75))

dataf2=dataf[20:50,20:50]

dataf3=np.reshape(dataf2,(900,))

datab.append(dataf3)

datab=np.array(datab)

df1=pd.DataFrame(dataa)

df2=pd.DataFrame(datab)

df2.columns=np.arange(900,1800)

X1=pd.concat([df1, df2], axis=1)

X1[1800]=df["inc\_angle"]

#X=pd.concat([X1, df["inc\_angle"]], axis=1)

X=np.array(X1)

X=preprocessing.scale(X)

y=df.ix[:,"is\_iceberg"]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.4, random\_state=42)

# create model

model = Sequential()

model.add(Dense(1801, input\_dim=1801, activation="linear"))

model.add(Dense(700, activation='softplus'))

model.add(Dense(100, activation='tanh'))

model.add(Dense(10, activation='relu'))

model.add(Dense(1, activation='sigmoid'))

# Compile model

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Fit the model

model.fit(X\_train, y\_train, epochs=30, batch\_size=100)

# evaluate the model

scores = model.evaluate(X\_train, y\_train)

print("\n%s: %.2f%%" % (model.metrics\_names[1], scores[1]\*100))

KNN

import pandas as pd

import json

import numpy as np

import matplotlib.pyplot as plt

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

from sklearn import svm

# fix random seed for reproducibility

numpy.random.seed(7)

with open("train.json") as f:

d=json.loads(f.read())

df=pd.DataFrame(d)

"""df=df.drop(["id","band\_2"],axis=1)"""

df.info()

df=df.replace("na",40)

dataa=[]

for i in range(1604):

data1=df.ix[i,0]

dataf=np.reshape(data1,(75,75))

dataf2=dataf[20:50,20:50]

dataf3=np.reshape(dataf2,(900,))

dataa.append(dataf3)

dataa=np.array(dataa)

"""

names=[i for i in range(1604)]

X=pd.DataFrame()

for i in range(1604):

data1=df.ix[i,0]

dataf=np.reshape(data1,(75,75))

FD=dataf[20:50,20:50].reshape(900,1)

for j in range(900):

X.ix[i,j]=FD[j]

data=[]

for i in range(1604):

data.append(df.ix[i,0])

"""

datab=[]

for i in range(1604):

data1=df.ix[i,1]

dataf=np.reshape(data1,(75,75))

dataf2=dataf[20:50,20:50]

dataf3=np.reshape(dataf2,(900,))

datab.append(dataf3)

datab=np.array(datab)

df1=pd.DataFrame(dataa)

df2=pd.DataFrame(datab)

df2.columns=np.arange(900,1800)

X1=pd.concat([df1, df2], axis=1)

X1[1800]=df["inc\_angle"]

#X=pd.concat([X1, df["inc\_angle"]], axis=1)

X=X1

#X=np.array(X1)

y=df.ix[:,"is\_iceberg"]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

"""svm model

clf = svm.SVC()

clf.fit(X\_train, y\_train)

y\_predict=clf.predict(X\_test)

"""

import seaborn as sns

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import confusion\_matrix

neigh = KNeighborsClassifier(n\_neighbors=3)

from sklearn.model\_selection import cross\_val\_score

neigh.fit(X\_train, y\_train)

y\_predict=neigh.predict(X\_test)

tn, fp, fn, tp = confusion\_matrix(y\_test, y\_predict, labels=[0,1]).ravel()

print((tn, fp, fn, tp))

accuracy=100\*(tp+fn)/len(y\_predict)

print(accuracy)

sns.set()

scores = cross\_val\_score(neigh, X, y, cv=10)

print(100\*np.mean(scores))

**Spooky Author Identification:**

* Downloaded the training and test data from the Kaggle and observed that the data does not have any missing values.
* Splitted the sentence of each author into words and stored in three different arrays (one for each author).
* Filtered out the special characters.
* Converted all words into lower letters of alphabets.
* Built a Pandas dataframe with columns as words and rows as authors. [word counts are stored in here]
* **First Method:**

Used Naïve-Bayes theorem for finding the probability that the given line is written by a particular author.

Stored the results in a csv file and uploaded it in the Kaggle and found out that our position is in the 30th percentile.

* **Second Method:**

Deleted all the stopping words from the words data frame.

Followed the same procedure as the First method this time our position has improved to 37th percentile.

* **Third Method:**

This time we have splitted the data into test and train from the training data itself.

Log loss error is coming around 0.8.

The major drawback of this method is that when many new words comes from any author’s sentence this method will work poorly. We are trying to find a new way to tackle this problem.

**Preprocessing the data:**

Splitted all the sentences into words.

Converted all the words into lower alphabetical order.

Removed all the special characters.

Created a data frame with words as columns and authors as rows.

Stored the count of each words written by each author in this dataframe.

Observed that many of the data values are zeros. So, filled it with zero as it means that word is not used by the particular author.

**Modelling the data**

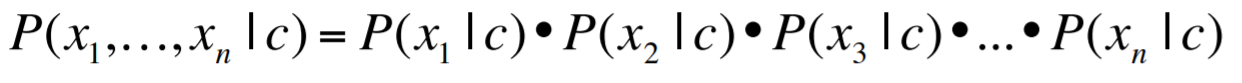
Splitting the data into training and test in the ratio of 60:40

Doing the same preprocessing with the test data.

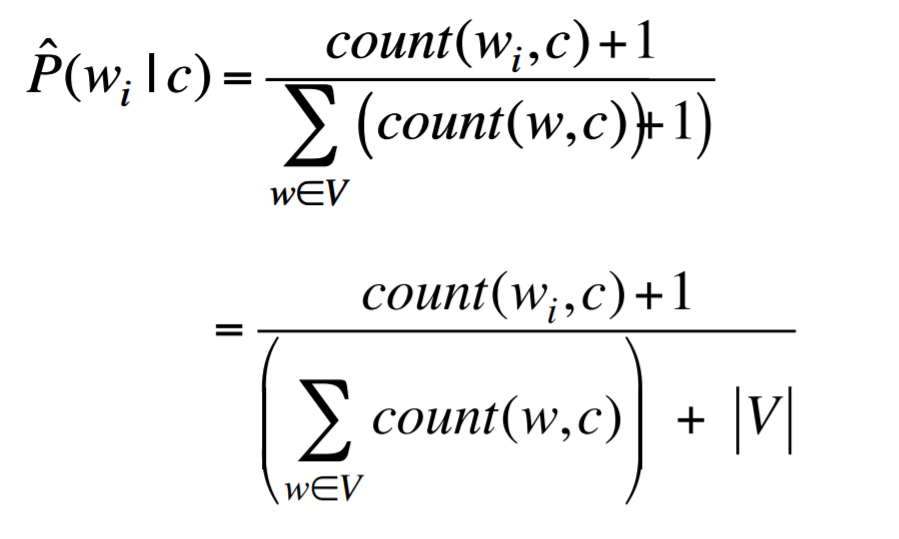
<https://web.stanford.edu/class/cs124/lec/naivebayes.pdf>

<https://www.youtube.com/watch?v=EGKeC2S44Rs&t=569s>

By following Naïve bayes we need to find out the probability of each author i.e., the number of sentences written by the particular author. After that treated each word independently for finding the conditional probabilities for finding the posterior probabilities



As there is a multiplication over here when a word is not present in our training set this causes zero value. So, we use modified Naïve bayes for this



First calculate the sum of each column and storing that in a variable

Observing the above formula, we can transform our data frame by adding 1 to each word

In above formula |V| means total number of count of frequencies of all the words in the dataframe

Initilize each list for each author go through each line of a test set (bag of words). Using modified Naïve bayes we calculate the probability that each line is written by one of the three authors.

After that we get three lists with probabilities

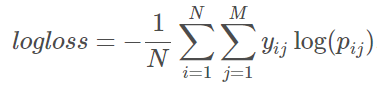
Convert those lists into a dataframe with rows as id’s of a particular sentence and columns as a probability that the line is written by that author.

Divide each element with its row sum, doing so we get the row sum equal to one and get the calculated probabilities of each author.

Actual probabilities are stored in our test set we can use pandas **get\_dummies** function i.e., if the line is written by author the actual probability for that author is 1 and for other two authors it is zero.

Log loss calculation for the test set (from Kaggle):

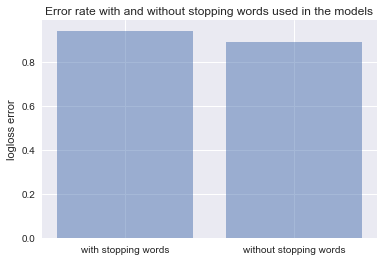
Log loss evaluated using multi-class logarithmic loss. Each id has one true class. For each id, you must submit a predicted probability for each author. The formula is then:



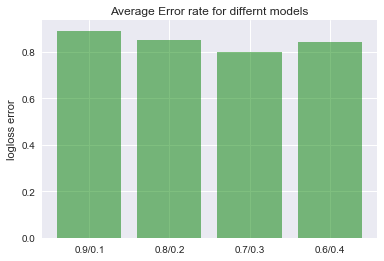
where N is the number of observations in the test set, M is the number of class labels (3 classes), loglog is the natural logarithm, yijyij is 1 if observation ii belongs to class jj and 0 otherwise, and pijpij is the predicted probability that observation ii belongs to class jj

If we get a minimum value from this then our error is very less. This is how a goodness of our model is defined.

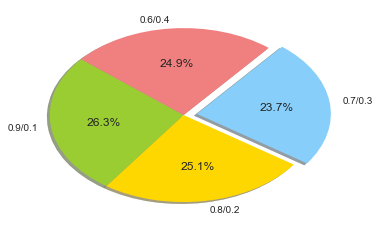
Suppose for example if we get probability of an author belonging to particular class as 0.6 then 1\*log(1/0.6) +0+0 will be our error for that particular sentence.

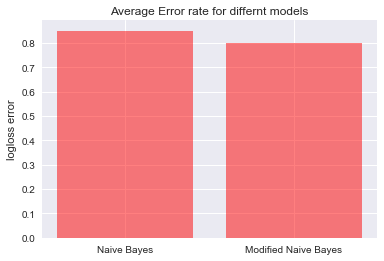


If we remove stopping words the error rate is decreased



Error rate is minimum for a train/test split of 70/30





Logloss error is minimum if we use Modified Naïve bayes

import pandas as pd

import math

import numpy as np

import seaborn as sns

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

df=pd.read\_csv("train.csv")

X=df.ix[:,['id', 'text',]]

Y=df.ix[:,["author"]]

#train test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.4, random\_state=0)

X\_train["author"]=y\_train

df=X\_train

E=['','','']

r=0

A = df["author"].unique()

dim=[]

for i in A:

dim.append(len(df[df["author"] == i]))

z5=sum(dim)

pa=[i/z5 for i in dim]#probability of that author

for j in range(len(A)):

p=df["author"]==A[j]

k= df.ix[p,"text"]

for i in k:

E[r]=E[r]+i

r=r+1

for l in range(len(E)):

E[l]=E[l].split(" ")

for l in range(len(E)):

E[l]=[w.lower() for w in E[l]]

E[l]=[w for w in E[l] if w.isalpha()]

d=[{},{},{}]

for l in range(len(E)):

for w in E[l]:

if w in d[l]:

d[l][w]= d[l][w]+1

else:

d[l][w]=1

#convering into data frame and filling Nans with zeros

W= pd.DataFrame(d)

W=W.fillna(0)

V=W.shape[1]

sum=W.sum(axis=1)

den=(sum+V)

X=W+1

den=den

for i in X.columns:

for j in range(3):

X.ix[j,i]=X.ix[j,i]/den[j]

#test data

X\_test["author"]=y\_test

dt=X\_test

j=list(dt.ix[:,1])

j1=[j[i].lower() for i in range(len(j))]

j2=[j1[i].split(" ") for i in range(len(j1))]

id1=list(dt.ix[:,0])

E1=[]

H1=[]

M1=[]

for i in range(len(j2)):

E=pa[0]

H=pa[1]

M=pa[2]

for j in range(len(j2[i])):

if j2[i][j] in X.columns:

E\*=X.ix[0,j2[i][j]]

H\*=X.ix[1,j2[i][j]]

M\*=X.ix[2,j2[i][j]]

else:

E\*=1/3

H\*=1/3

M\*=1/3

E1.append(E)

M1.append(M)

H1.append(H)

op=pd.DataFrame(np.column\_stack([E1,H1,M1]),columns=["EAP","HPL","MWS"])

op1= op.apply(np.sum,axis=1)# finding each row sum

for i in op.columns:

for j in range(len(op)):

op.ix[j,i]=op.ix[j,i]/op1[j]

Y\_actual=pd.get\_dummies(y\_test)

n\_test=len(Y\_actual)

headernames=op.columns

Y\_actual.columns = headernames

Y\_a=Y\_actual.set\_index(op.index)

sum1=0

op=op.fillna(op.mean())

for i in range(n\_test):

for j in range(3):

sum1+=Y\_a.ix[i,j]\*(math.log(op.ix[i,j]))

error=-sum1/n\_test