In this notebook I am going to finalize, in which way should I use my models so that I get maximum performance.
Since we have three models one machine learning, one CNN model and one NLP model all trained on different data. So, we have to find a method to combine all models so that we get maximum performance.

```
!pip install tensorflow-text
```

```
Collecting tensorflow-text
  Downloading https://files.pythonhosted.org/packages/c0/ed/bbb51e9eccca0c2bt
     |████████████████████████████████| 4.3MB 8.4MB/s
Requirement already satisfied: tensorflow<2.6,>=2.5.0 in /usr/local/lib/pytho
Requirement already satisfied: tensorflow-hub>=0.8.0 in /usr/local/lib/pythor
Requirement already satisfied: h5py~=3.1.0 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: opt-einsum~=3.3.0 in /usr/local/lib/python3.7/
Requirement already satisfied: flatbuffers~=1.12.0 in /usr/local/lib/python3.
Requirement already satisfied: tensorflow-estimator<2.6.0,>=2.5.0rc0 in /usr/
Requirement already satisfied: absl-py~=0.10 in /usr/local/lib/python3.7/dist
Requirement already satisfied: astunparse~=1.6.3 in /usr/local/lib/python3.7/
Requirement already satisfied: typing-extensions~=3.7.4 in /usr/local/lib/pyt
Requirement already satisfied: protobuf>=3.9.2 in /usr/local/lib/python3.7/di
Requirement already satisfied: wheel~=0.35 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: tensorboard~=2.5 in /usr/local/lib/python3.7/c
Requirement already satisfied: termcolor~=1.1.0 in /usr/local/lib/python3.7/c
Requirement already satisfied: grpcio~=1.34.0 in /usr/local/lib/python3.7/dis
Requirement already satisfied: gast==0.4.0 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: wrapt~=1.12.1 in /usr/local/lib/python3.7/dist
Requirement already satisfied: google-pasta~=0.2 in /usr/local/lib/python3.7/
Requirement already satisfied: numpy~=1.19.2 in /usr/local/lib/python3.7/dist
Requirement already satisfied: keras-preprocessing~=1.1.2 in /usr/local/lib/p
Requirement already satisfied: keras-nightly~=2.5.0.dev in /usr/local/lib/pyt
Requirement already satisfied: six~=1.15.0 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: cached-property; python_version < "3.8" in /us
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.7/
Requirement already satisfied: google-auth<2,>=1.6.3 in /usr/local/lib/pythor
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.7/di
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/li
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in /usr/
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/pythor
Requirement already satisfied: cachetools<5.0,>=2.0.0 in /usr/local/lib/pytho
Requirement already satisfied: rsa<5,>=3.1.4; python_version >= "3.6" in /usr
Requirement already satisfied: importlib-metadata; python_version < "3.8" in
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/pyt
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /us
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.7/di
Installing collected packages: tensorflow-text
Successfully installed tensorflow-text-2.5.0
```

```
#importing libraries
import pandas as pd
import numpy as np
from numpy import asarray
from tqdm import tqdm
import matplotlib.pyplot as plt
import seaborn as sns

import pickle
import os
from PIL import Image

from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

import tensorflow as tf
from keras.models import load_model
import tensorflow_text as text
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import warnings
warnings.filterwarnings("ignore")
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
#loading the dataset
df = pd.read_csv('/content/drive/MyDrive/Applied_ai/test dataset/test_dataset.csv'
```

```
df.shape
```

```
(4984, 68)
```

```
'''
during object detection in image I have exteacted 3 objects with their respective
are stored in a list and list is in the form of string. SO I am going to convert t
different features for these probabilities.
'''
```

```
#converting lists of string type to list
from ast import literal_eval
df['img_feature_pred'] = df['img_feature_pred'].apply(literal_eval)
```

```
#getting probabitity values in three different lists to make them as seperate feat
img_feature_pred_1 = []
img_feature_pred_2 = []
img_feature_pred_3 = []
for i in df['img_feature_pred']:
  img_feature_pred_1.append(i[0])
  img_feature_pred_2.append(i[1])
```

```
    img_feature_pred_3.append(i[2])

#creating features for probability values
df['img_feature_pred_1'] = img_feature_pred_1
df['img_feature_pred_2'] = img_feature_pred_2
df['img_feature_pred_3'] = img_feature_pred_3

y = df['dank_or_not']
X = df.drop(['dank_or_not'], axis=1)
```

Since our project is to predict the dankness of the meme and it will be done before posting the meme on social networking sites. So, before posting data such as num_comments, upvote_ratio, score etc will not be available for the post and also it will be good if we predict the dankness of memes irrespective of subscribers count on the page. So, I am going to drop those features.

```
#dropping features as discussed above
df.drop(['img_feature_pred','is_original_content','num_comments','upvote_ratio','s
```

Seperating datasets to predict from different models

```
#Dataset for machine learning model
X_ml = X[['img_feature_pred_1','img_feature_pred_2','img_feature_pred_3','avg_h','
    'num_words','thumbnail_height','thumbnail_width','gray', 'white', 'faded color
    'light blue', 'brown', 'yellow', 'dark cyan', 'light orange', 'dark green', 'c
    'dark orange', 'light red','web_site','book_jacket','packet','mud_turtle']]

#dataset for CNN model
X_cnn = df[['url']]
X_cnn.url = X_cnn.url.str.split('/').str[-1] + ('.png')
X_cnn = pd.DataFrame(X_cnn)

#dataset for NLP model
X_bert = df['text']
```

Loading models

```
#ML model
ml_model = pickle.load(open('/content/drive/MyDrive/Applied_ai/models/dankornot_ml

#CNN model
```

```
cnn_model = load_model('/content/drive/MyDrive/Applied_ai/models/resnet_model/resn


#bert model
bert_model = tf.saved_model.load('/content/drive/MyDrive/Applied_ai/models/bert_mo
```

Predictions

Predicting using ML model

```
ml_pred_prob = ml_model.predict_proba(X_ml)[:,-1]
ml_pred = ml_model.predict(X_ml)
```

Predicting using CNN model

```
#Reference : https://machinelearningmastery.com/how-to-manually-scale-image-pixel-
cnn_pred_prob = []
for image in tqdm(X_cnn.url, position=0):
  path = '/content/drive/MyDrive/Applied_ai/meme_images/'+image
  img = Image.open(path)
  pixels = asarray(img)
  pixels = pixels.astype('float32')
  pixels /= 255.0
  pixels.resize(224,224,3)
  pixels = np.expand_dims(pixels, axis=0)
  cnn_prediction = cnn_model.predict(pixels)
  cnn_pred_prob.append(cnn_prediction[0][0])
```

```
100%|██████████| 4984/4984 [1:01:41<00:00,  2.13it/s]
100%|██████████| 4984/4984 [1:01:41<00:00,  1.35it/s]
```

```
cnn_pred = np.array(cnn_pred_prob).round().astype('int')
```

Predicting using NLP model

```
nlp_pred_prob = []
for text in tqdm(X_bert, position=0):
  try:
    bert_predict = tf.sigmoid(bert_model(tf.constant([text])))
    nlp_pred_prob.append(np.array(bert_predict)[0][0])
  except:
    nlp_pred_prob.append(0)
```

```
100%|██████████| 4984/4984 [00:35<00:00, 141.98it/s]
```

```
nlp_pred = np.array(nlp_pred_prob).round().astype('int')
```

Creating a dataset of predicted probabilities values

```
prob_pred_df = pd.DataFrame(columns = ['ml_pred','cnn_pred','nlp_pred'])
```

```
prob_pred_df['ml_pred'] = ml_pred_prob
prob_pred_df['cnn_pred'] = cnn_pred_prob
prob_pred_df['nlp_pred'] = nlp_pred_prob
```

Creating a dataset of predicted labels

```
pred_df = pd.DataFrame(columns = ['ml_pred','cnn_pred','nlp_pred'])
```

```
pred_df['ml_pred'] = ml_pred
pred_df['cnn_pred'] = cnn_pred
pred_df['nlp_pred'] = nlp_pred
```

Performance of models by taking mean of probability values.

Here I am going to combine all the three models by taking the mean of probability values predicted by them for each data. According to that mean value I will get predicted labels and compare these labes with original labels.

```
prob_pred_df = pd.read_csv('/content/drive/MyDrive/Applied_ai/datasets/prob_pred_d
pred_df = pd.read_csv('/content/drive/MyDrive/Applied_ai/datasets/pred_df')
```

```
mean_all = prob_pred_df.mean(axis=1)
mean_ml_cnn = prob_pred_df[['ml_pred','cnn_pred']].mean(axis=1)
mean_cnn_nlp = prob_pred_df[['cnn_pred','nlp_pred']].mean(axis=1)
mean_nlp_ml = prob_pred_df[['nlp_pred','ml_pred']].mean(axis=1)
```

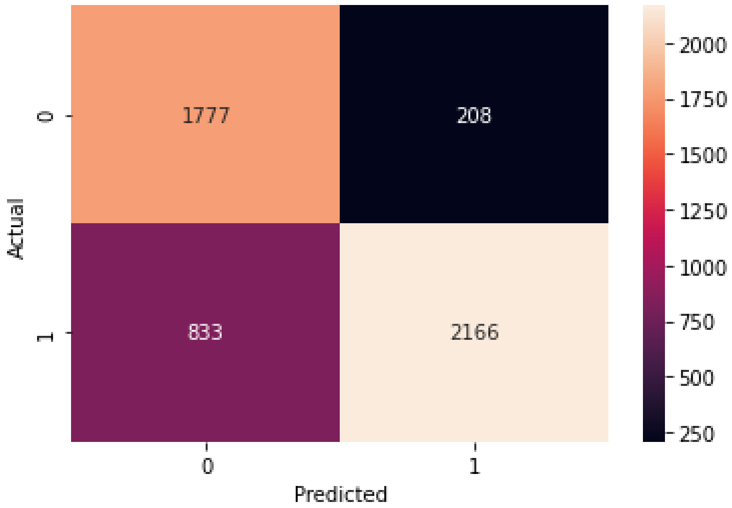
```
label_all = mean_all.round().astype('int')
label_ml_cnn = mean_ml_cnn.round().astype('int')
label_cnn_nlp = mean_cnn_nlp.round().astype('int')
label_nlp_ml = mean_nlp_ml.round().astype('int')
```

```
print('Accuracy all : ', accuracy_score(label_all, y))
print('Accuracy ml_cnn : ', accuracy_score(label_ml_cnn, y))
print('Accuracy cnn_nlp : ', accuracy_score(label_cnn_nlp, y))
print('Accuracy nlp_ml : ', accuracy_score(label_nlp_ml, y))
```

```
    Accuracy all :  0.791131621187801
```

```
Accuracy ml_cnn :  0.8802166934189406
Accuracy cnn_nlp :  0.4871589085072231
Accuracy nlp_ml :  0.7375601926163724
```

```
#plotting confusion matrix
cm = confusion_matrix(prob_pred_df['label'], y)
sns.heatmap(cm, annot=True, fmt='g')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



Performance of models by taking mode of labels.

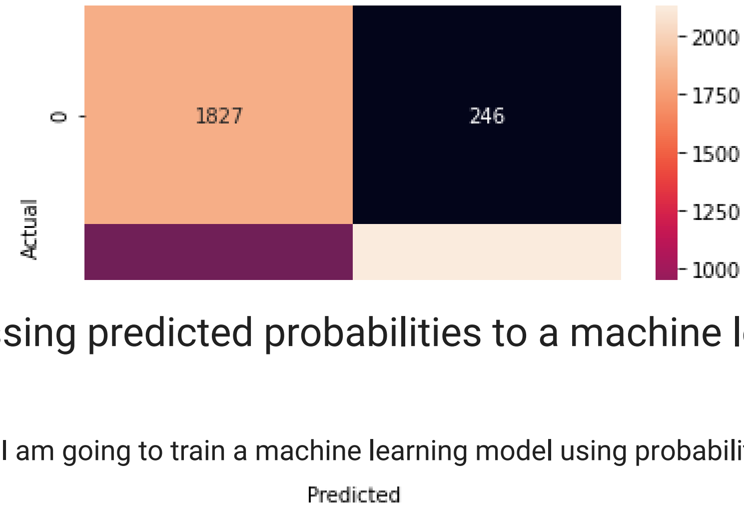
Here I am going to take the mode of labels predicted by each model and that mode will be our final prediced label by all models. Then I will compare this mode with original target value.

```
mode_all = pred_df.mode(axis=1)
```

```
print('Accuracy : ', accuracy_score(mode_all, y))
```

```
Accuracy :  0.7935393258426966
```

```
#plotting confusion matrix
cm_p = confusion_matrix(mode_all, y)
sns.heatmap(cm_p, annot=True, fmt='g')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



Passing predicted probabilities to a machine learning algorithm.

Here I am going to train a machine learning model using probabilities values.

```
#using logistic regression
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV


params = {'max_iter' : [100,500,1000,2000]}


lr = LogisticRegression()
clf = GridSearchCV(lr, param_grid=params, scoring='accuracy', cv=5, return_train_s


X = prob_pred_df[['ml_pred', 'cnn_pred', 'nlp_pred']]
y = y


#training the model
clf.fit(X, y)
```

```
GridSearchCV(cv=5, error_score=nan,
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=Fals
                                          fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=None,
                                          max_iter=100, multi_class='auto',
                                          n_jobs=None, penalty='l2',
                                          random_state=None, solver='lbfgs',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'max_iter': [100, 500, 1000, 2000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='accuracy', verbose=0)
```

```
print('logistic regression cv train score : ',clf.cv_results_['mean_train_score'].
```

```
logistic regression cv train score :  0.8897473332966042
```

```
print('logistic regression cv test score : ',clf.cv_results_['mean_test_score'].me
```

```
logistic regression cv test score :  0.8902508731012315
```

From all the three methods we can see that logistic regression is giving the highest accuracy than the mean and mode method. But we have very less data to train the machine learning model so, in this case mean and mode method looks more promising.
Mean and mode methods are giving same accuracy, So I am going to use mean of probabilities predicted by all the three models.

✓ 0s completed at 9:45 PM