# Data Preprocessing

Even in this project, we are bound to do certain data preprocessing steps which includes data cleaning, preparation, transformation, and dimensionality reduction, which convert the raw data into a form that is suitable for further processing.

First we will look into usual dataset level cleaning and then later into text pre processing.

1. Missing Value Handling

# Number of missing values in each column

missing = pd.DataFrame(df.isnull().sum()).rename(columns = {0: 'missing'})

# Create a percentage of missing values

missing['percent'] = missing['missing'] / len(df)

# sorting the values in desending order to see highest count on the top

missing.sort\_values('percent', ascending = False)

****

Description feature has one missing value. We are dropping that missing value observation from the dataset.

# removing missing values in description

df=df[pd.notnull(df['description'])]

Text pre-processing

We already observed that the value text pro processing adds to any text related tasks. Text pre-processing tasks includes

* Removing punctuation
* Removing numbers
* Converting text to lower case (no capital letters)
* Removing extra whitespace
* Removing stop-words (extremely common words which do not provide any analytic information and tend to be of little value i.e. a, and, are etc.)
* Stammering & Lemmatization (not in the scope of current work due to system memory limitation)

Data before text pre processing

df['description'][4]

'Key Features of dilli bazaaar Bellies, Corporate Casuals, Casuals Material: Fabric Occasion: Ethnic, Casual, Party, Formal Color: Pink Heel Height: 0,Specifications of dilli bazaaar Bellies, Corporate Casuals, Casuals General Occasion Ethnic, Casual, Party, Formal Ideal For Women Shoe Details Weight 200 g (per single Shoe) - Weight of the product may vary depending on size. Heel Height 0 inch Outer Material Fabric Color Pink'

# Remove punctuation

df['description'] = df['description'].str.replace(r'[^\w\d\s]', ' ')

# Replace whitespace between terms with a single space

df['description'] = df['description'].str.replace(r'\s+', ' ')

# Remove leading and trailing whitespace

df['description'] = df['description'].str.replace(r'^\s+|\s+?$', '')

# converting to lower case

df['description'] = df['description'].str.lower()

# Replace numberslike price values with 'numbr'

df['description'] = df['description'].str.replace(r'\d+(\.\d+)?', 'numbr')

df['description'][4]

'key features of dilli bazaaar bellies corporate casuals casuals material fabric occasion ethnic casual party formal color pink heel height numbr specifications of dilli bazaaar bellies corporate casuals casuals general occasion ethnic casual party formal ideal for women shoe details weight numbr g per single shoe weight of the product may vary depending on size heel height numbr inch outer material fabric color pink'

The stopwords are imported from the nltk library and are removed from description. There are two kinds of stopwords.

1. General stopwords like ‘an’, ‘in’ and so on which appears everywhere irrespective of domains. It doent matter where the text in coming from.
2. There are few domain specific stopwords. For example, ‘buy’, ‘com’, ‘cash’ and so on which can appear only in certain domains like E commerce and retail. We need to remove them as well.

# Removing Stop words

stop = stopwords.words('english')

pattern = r'\b(?:{})\b'.format('|'.join(stop))

df['description'] = df['description'].str.replace(pattern, '')

df['description'] = df['description'].str.replace(r'\s+', ' ')# Removing single characters

df['description'] = df['description'].apply(lambda x: " ".join(x for x in x.split() if len(x)>1))

df['description'][4]

'key features dilli bazaaar bellies corporate casuals casuals material fabric occasion ethnic casual party formal color pink heel height numbr specifications dilli bazaaar bellies corporate casuals casuals general occasion ethnic casual party formal ideal women shoe details weight numbr per single shoe weight product may vary depending size heel height numbr inch outer material fabric color pink'

# Removing domain realted stop words from description

specific\_stop\_words = ["numbr", "rs","flipkart","buy","com","free","day","cash","replacement","guarantee","genuine","key","feature","delivery","products","product","shipping", "online","india","shop"]

df['description'] = df['description'].apply(lambda x: " ".join(x for x in x.split() if x not in specific\_stop\_words))

That is the end of text pre processing. We did most of the pre processing steps to clean the text and make it ready for feature engineering.

Feature engineering

In order to apply text classification, the unstructured format of text has to be converted into a structured format for the simple reason that it is much easier for machines to deal with numbers than text. We are considering only one feature "description" in this project. Extracted the words from description column after text pre-processing and using as our own vocabulary.

We will be using deep learning algorithms to build the classifier and feature extraction should be carried out accordingly. We will use keras tokenizer to function to generate features. We are setting max\_length to 200 which means, we are only considering 200 features for the classifier. This number will also decide the accuracy and the ideal number can be obtained from hyper parameter tuning. This process is called padding.

max\_length= 200

tokenizer = Tokenizer()

tokenizer.fit\_on\_texts(df['description'])

clean\_description = tokenizer.texts\_to\_sequences(df['description'])

#padding

X = pad\_sequences( clean\_description, maxlen= max\_length)

Just like features, we need to convert target variable which is category to number as well. We will use one hot encoding to do so. The function used for this is *LabelEncoder* from sklearn.

# one hot encoding for Target

from sklearn.preprocessing import LabelEncoder

num\_class = len(np.unique(df.product\_category\_tree.values))

y = df['product\_category\_tree'].values

encoder = LabelEncoder()

encoder.fit(y)

y = encoder.transform(y)

# 

# Train-Test Split

The data is split into two parts one for training the model and one for evaluating the model.

The **train\_test\_split** library from **sklearn.model\_selection** is imported to split the dataframe into two parts.

#train test split into X and y

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=1) #train 80, test 20

print(X\_train.shape)

print(X\_test.shape)

print(y\_train.shape)

print(y\_test.shape)

(14025, 200)

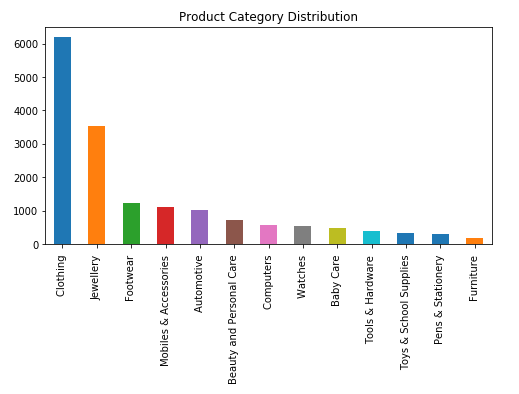
(3507, 200)

(14025,)

(3507,)

Resampling

When dealing with data in the context of classification, two different specific cases can be encountered. The datasets can either be balanced or imbalanced. Imbalanced datasets refer to a situation in which classes are not equally represented. We have observed that, this e-commerce dataset also with imbalance data, where clothing and jewellary categories has more observations than other categories as shown in the below figures. So we need to apply resampling methods to get balanced data set for training the model.



The below mentioned techniques handles the unbalanced dataset.

1. Undersampling

Undersampling aims at balancing the dataset distribution by randomly removing some observations from the majority class until the dataset is balanced out. The drawback of this method is that it removes observations from the majority class and therefore may lead to important information loss in the training dataset.

RandomUnderSampler function from the imblearn.under\_sampling package is used for undersampling.

from imblearn.under\_sampling import RandomUnderSampler

sample = RandomUnderSampler(return\_indices=True)

X\_undersample, y\_undersample, id\_rus = sample.fit\_sample(X, y)

After applying under sampling, the data distribution

for i in range(y\_undersample.max()):

print(i,len(y\_undersample[y\_undersample==i]))

0 265

1 265

2 265

3 265

4 265

5 265

6 265

7 265

8 265

9 265

10 265

11 265

12 265

1. Oversampling

Oversampling aims at randomly replicating instances from the minority class (and thus increasing the minority population) until all the classes of the dataset get balanced. Even if this method has the advantage not to lose information, replicating instances in the training dataset may lead to over fitting.

SMOTE is used for oversampling the data. SMOTE proposes several variants by identifying specific samples. The SMOTE () function from **imblearn.over\_sampling** is used to implement this.

from imblearn.over\_sampling import SMOTE

sm = SMOTE(random\_state=2)

X\_train\_res, y\_train\_res = sm.fit\_sample(X\_train, y\_train.ravel())

After applying over- sampling, the data distribution

for i in range(y\_train\_res.max()):

print(i,len(y\_train\_res[y\_train\_res==i]))

0 4947

1 4947

2 4947

3 4947

4 4947

5 4947

6 4947

7 4947

8 4947

9 4947

10 4947

11 4947

12 4947

# 

# Model Building & Prediction

A list of below classifiers was used for creating various classification models, which can be further used for prediction.

* Simple Baseline Artificial Neural Networks (ANNs)
* [Recurrent neural networks (](http://adventuresinmachinelearning.com/recurrent-neural-networks-lstm-tutorial-tensorflow/) RNN-LSTM)
* [Convolutional](https://developers.google.com/machine-learning/glossary/#convolutional_layer) Neural Networks

## ANN with Imbalance data

Let s start with basic neural network using the imbalanced data.

inputs = Input(shape=(max\_length, ))

embedding\_layer = Embedding(vocab\_size,

128,

input\_length= max\_length)(inputs)

x = Flatten()(embedding\_layer)

x = Dense(32, activation='relu')(x)

predictions = Dense(num\_class, activation='softmax')(x)

model = Model(inputs=[inputs], outputs=predictions)

model.compile(optimizer='adam',

loss='categorical\_crossentropy',

metrics=['acc'])

model.summary()

filepath="weights-simple.hdf5"

checkpointer = ModelCheckpoint(filepath, monitor='val\_acc', verbose=1, save\_best\_only=True, mode='max')

history = model.fit([X\_train], batch\_size=64, y=to\_categorical(y\_train), verbose=1, validation\_split=0.25,

shuffle=True, epochs=5, callbacks=[checkpointer])

predicted = model.predict(X\_test)

predicted = np.argmax(predicted, axis=1)

accuracy\_score(y\_test, predicted)

0.974337040205303

1. ANN with balance data using Oversampling

inputs = Input(shape=(MAX\_LENGTH, ))

embedding\_layer = Embedding(vocab\_size,

128,

input\_length=MAX\_LENGTH)(inputs)

x = Flatten()(embedding\_layer)

x = Dense(32, activation='relu')(x)

predictions = Dense(num\_class, activation='softmax')(x)

model = Model(inputs=[inputs], outputs=predictions)

model.compile(optimizer='adam',

loss='categorical\_crossentropy',

metrics=['acc'])

model.summary()

filepath="weights-simple.hdf5"

checkpointer = ModelCheckpoint(filepath, monitor='val\_acc', verbose=1, save\_best\_only=True, mode='max')

history = model.fit([X\_train\_res], batch\_size=64, y=to\_categorical(y\_train\_res), verbose=1, validation\_split=0.25,

shuffle=True, epochs=5, callbacks=[checkpointer])

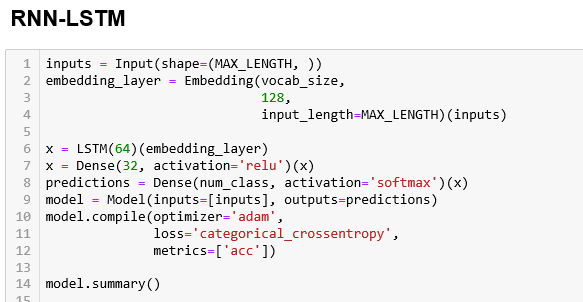
predicted = model.predict(X\_test)

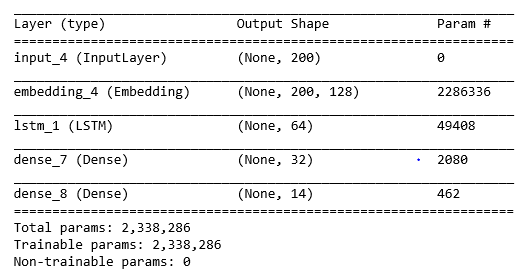
predicted = np.argmax(predicted, axis=1)

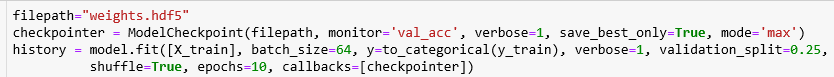
accuracy\_score(y\_test, predicted)

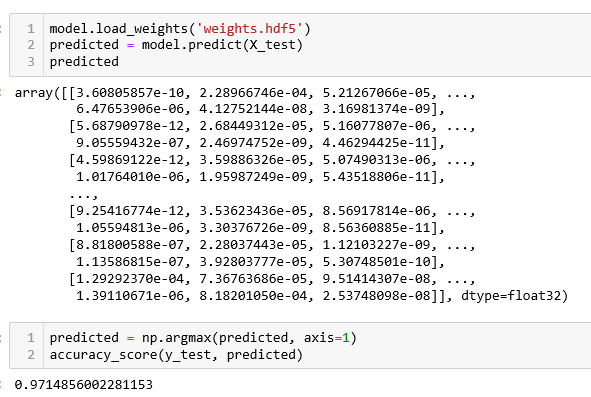
0.9421157684630739

1. ANN with balance data using UnderSampling









# Evaluation of the model

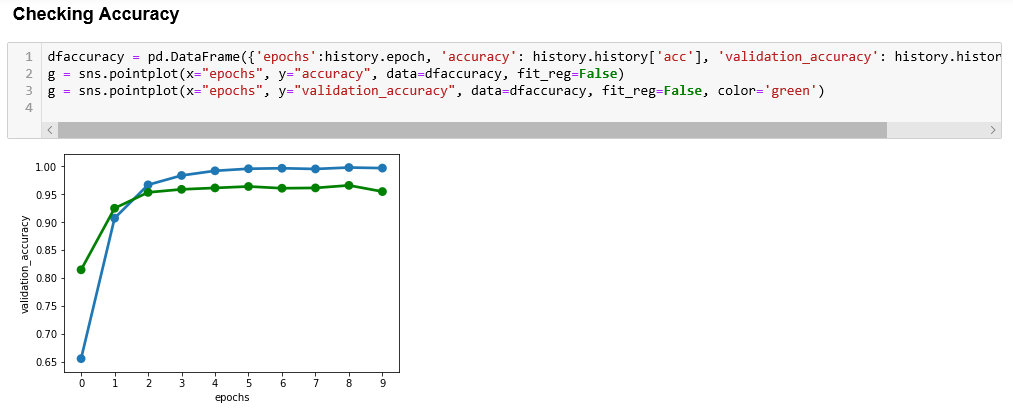
K-fold cross validation is the most commonly used method in order to evaluate the performance of a classifier.

With this method, the dataset is partitioned into K equally-sized disjoint folds. For each of the K iterations, the training set is composed of the examples from (K-1) folds aiming at training the classifier whereas the examples of the remaining fold represent the testing dataset, used to evaluate the classifier. Each iteration gives a value of error, which is used for the computation of the overall true error estimate.

The main advantage of this method is that each record appears at least once in the training dataset, and at least once in the testing dataset. Therefore, we ensure to train and test the whole dataset.

In practice, the value of K is set to either 5 or 10 depending upon the size of the dataset.

The performance of a model is evaluated by evaluation measure accuracy.



## 

## Hyper parameters tuning

Parameters which define the model architecture are referred to as **hyper parameters** and thus this process of searching for the ideal model architecture is referred to as hyper parameter tuning.

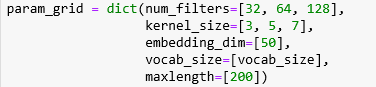
**Randomized Search Parameter Tuning**

Randomized Search is an approach to parameter tuning that will methodically build and evaluate a model for each combination of algorithm parameters specified in a grid.

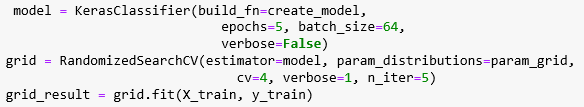
We have applied the hyper parameter tuning on CNN algorithm using keras classifier and Randomized search CV method.

**The Steps for Hyperparameters tuning includes:**

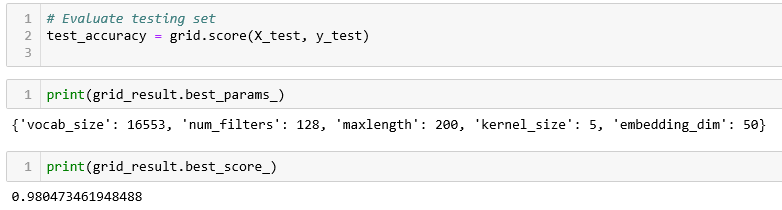
**Parameters Grid:**



**Running the Randomized Search:**

****

**Find the best parameters of the model:**

****

# 

# Results

The performance of a model is evaluated by evaluation measure accuracy.

|  |  |
| --- | --- |
| **Classifier** | **Accuracy** |
| Simple Baseline ANN | 0.97434 |
| RNN-LSTM | 0.97149 |
| CNN with 1Dimensional | 0.96635 |
| CNN with 1Dimensional  after Hyper tuning | **0.98047** |
| Simple Baseline ANN with over sampling | 0.94212 |
| Simple Baseline ANN with under sampling | 0.89165 |

The CNN model with 1D gave better performance compare to other models.

# Conclusions & Future Scope

The 1D convolutional Neural Networks performed better than simple baseline ANN model.We created our own word embeddings and did not use pretrained embeddings as the vocabulary. We can try with pretrained embeddings like Word2Net etc. We can apply stemming & Lemmatization in text pre processing. We have used “description” feature only. We can use