PORTO SEGURO SELF DRIVER PREDICTION

ABSTRACT

Nothing ruins the thrill of buying a brand new car more quickly than seeing your new insurance bill. The sting's even more painful when you know you're a good driver. It doesn't seem fair that you have to pay so much if you've been cautious on the road for years. Inaccuracies in car insurance company's claim predictions raise the cost of insurance for good drivers and reduce the price for bad ones.

In this project, we are going to build a model that predicts the probability that a driver will initiate an auto insurance claim in the next year using the data, which has information of auto insurance policy holder. We are going to build a model using neural networks in keras[1] having tensor flow as a backend. As, we are using textual data, we are going to use word embedding concept.

A more accurate prediction will allow the insurance company to further tailor their prices, and hopefully make auto insurance coverage more accessible to more drivers. So, the probability of increasing the profits for insurance company is much higher by using the predictions from this model. Now, we are now are going to build a prediction model, which predicts the probability that an auto insurance policy holder files a claim

[1] Keras – Deep Learning Library

INTRODUCTION

The heart of any model is neural network, so it is very important to decide which concept you are using while building the layers of neural network. As we are dealing with textual data, we are going to use embedding concept for our neural network.

Word embeddings provide a dense representation of words and their relative meanings. They are an improvement over sparse representations used in simpler bag of word model representations. Word embeddings can be learned from text data and reused among projects. They can also be learned as part of fitting a neural network on text data.

Training data plays a vital role in the performance and efficiency of the model. So, we should use correct data to train our model. In our model we are going use porto seguro's [i]auto insurance customers data to train our model.

In the training data[2], features that belong to similar groupings are tagged as such in the feature names (e.g., ind, reg, car, calc). In addition, feature names include the postfix bin to indicate binary features and cat to indicate categorical features. Features without these designations are either continuous or ordinal. Values of -1 indicate that the feature was missing from the observation. The target column's signifies whether or not a claim was filed for that policy holder.

[2] Data - https://www.kaggle.com/c/porto-seguro-safe-driver-prediction/data

PROBLEM STATEMENT

In this project, we are going to build a model for kaggle competition[3]. We are going to build a prediction model for Porto seguro, one of Brazil's largest auto and homeowner insurance companies. Inaccuracies in car insurance company's claim predictions raise the cost of insurance for good drivers and reduce the price for bad ones.

In this project, we are going to build a model that predicts the probability that a driver will initiate an auto insurance claim in the next year. While Porto Seguro has used machine learning for the past 20 years, they're looking to Kaggle's machine learning community to explore new, more powerful methods. A more accurate prediction will allow them to further tailor their prices, and hopefully make auto insurance coverage more accessible to more drivers.

Building from bottom up, there are a number of things to consider and get right.

- In the beginning, it is very important to look at the data to make it suitable for training the neural network, which is called data cleaning
- we need to choose the neural network which works best for our problem
- we need to create the model

[3] Kaggle - https://www.kaggle.com/c/porto-seguro-safe-driver-prediction

- Fit the model
- Evaluate the model
- Make predictions

LITERATURE REVIEW

The Literature Review is split into the three components: They are

(1) Deep Learning

(2) Python

(3) keras

1. Deep Learning

Deep learning is a subset of a subset of the way that Machine's think. If we want to know about Deep learning we need to have a basic idea on Artificial Intelligence and Machine Learning. So, what is AI and Machine Learning.

Artificial intelligence

Artificial intelligence is the broadest term we use for technology that allows machines to mimic human behavior. AI can include logic, if-then rules, and more.

Machine Learning

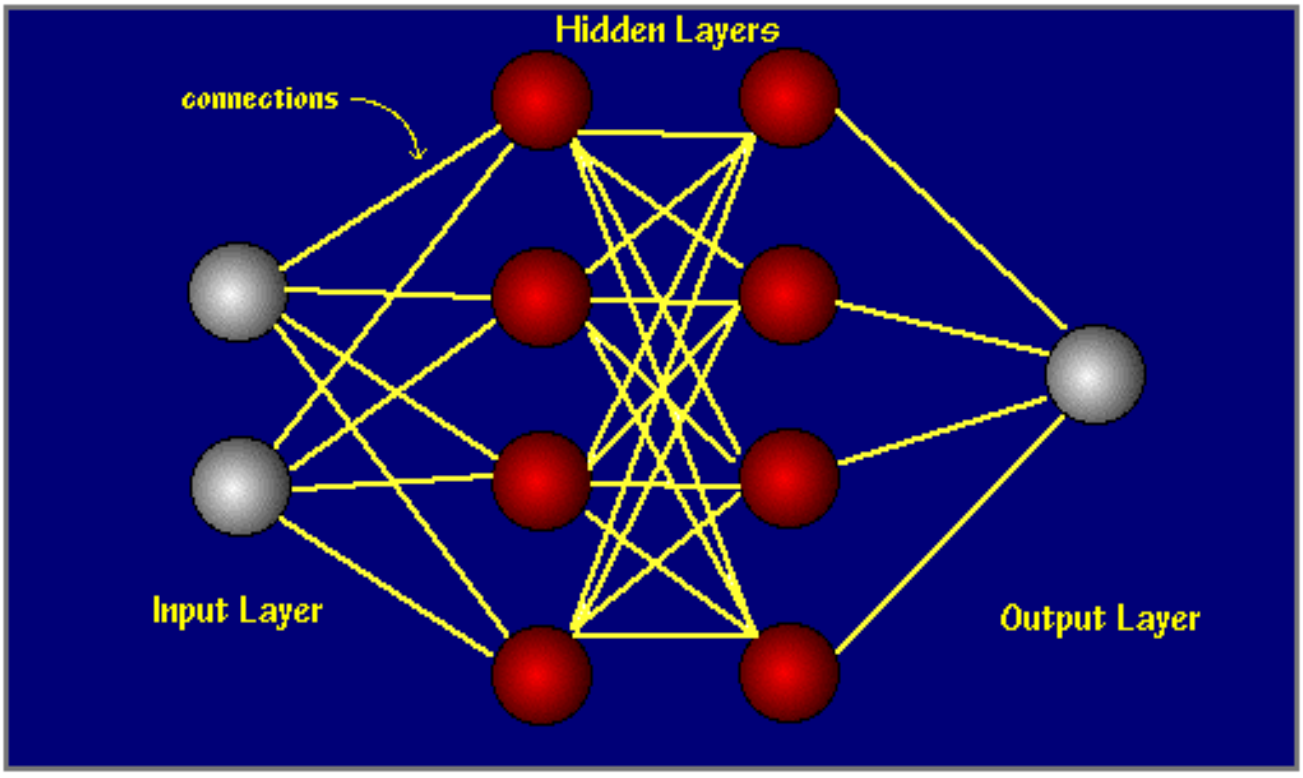
One subset of AI technology is machine learning. Machine learning allows computers to use statistics and use the experience to improve at tasks.

Deep Learning

A subset of machine learning is deep learning, which uses algorithms to train computers to perform tasks by exposing neural nets to huge amounts of data, allowing them to predict responses without needing to actually complete every task.

Neural Networks

Neural networks are typically organized in layers. Layers are made up of a number of interconnected 'nodes' which contain an 'activation function'. Patterns are presented to the network via the 'input layer', which communicates to one or more 'hidden layers' where the actual processing is done via a system of weighted 'connections'. The hidden layers then link to an 'output layer' where the answer is output as shown in the graphic below.



Most ANNs contain some form of 'learning rule' which Modifies the weights of the connections according to the input patterns that it is presented with. In a sense, ANNs[1] learn by example as do their biological counterparts; a child learns to recognize dogs from examples of dogs. Although there are many different kinds of learning rules used by neural networks, this demonstration is concerned only with

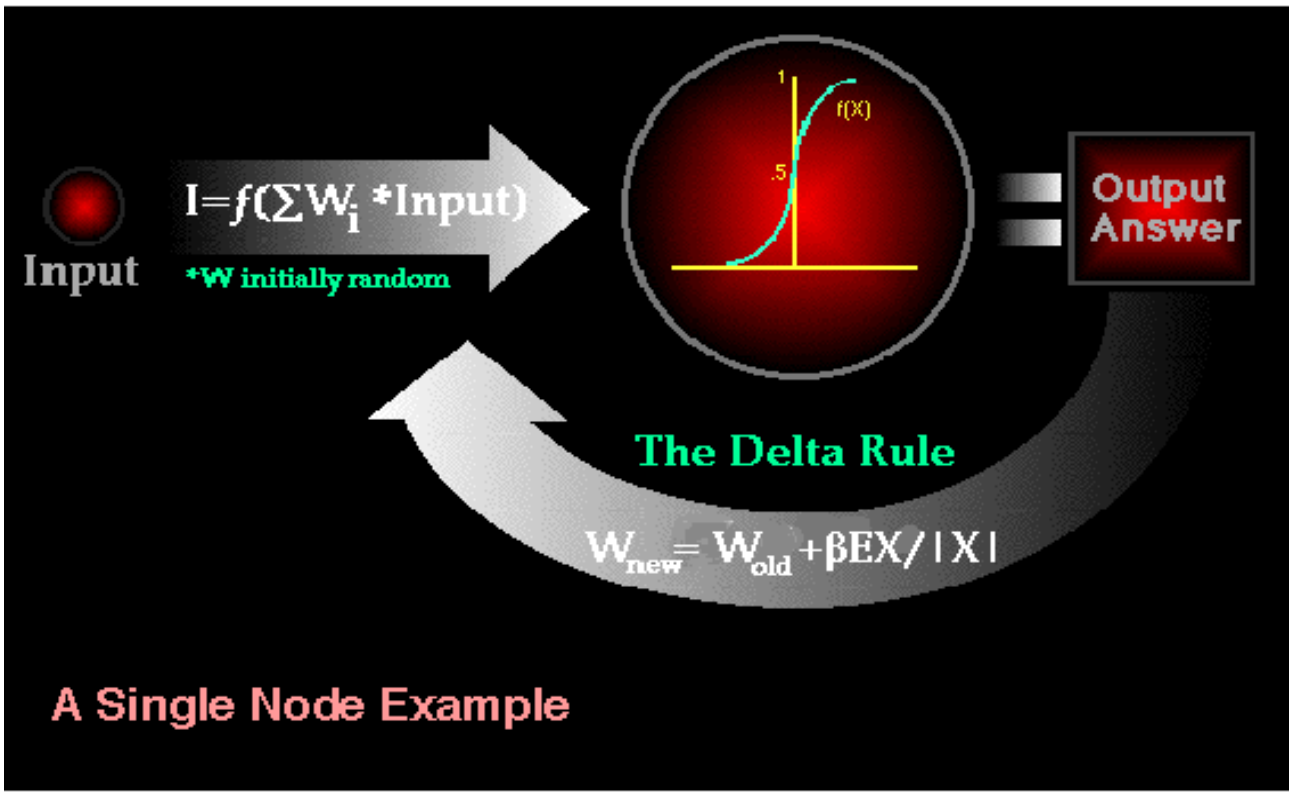
one; the delta rule. The delta rule is often utilized by the most common class of ANNs[4] called BPNNs[5]. Back propagation is an abbreviation for the backwards propagation of error. With the delta rule, as with other types of back propagation, 'learning' is a supervised process that occurs with each cycle or 'epoch' (i.e. each time the network is presented with a new input pattern) through a forward activation flow of outputs, and the backwards error propagation of weight adjustments. More simply, when a neural network is initially presented with a pattern it makes a random 'guess' as to what it might be.

It then sees how far its answer was from the actual one and makes an appropriate adjustment to its connection weights.

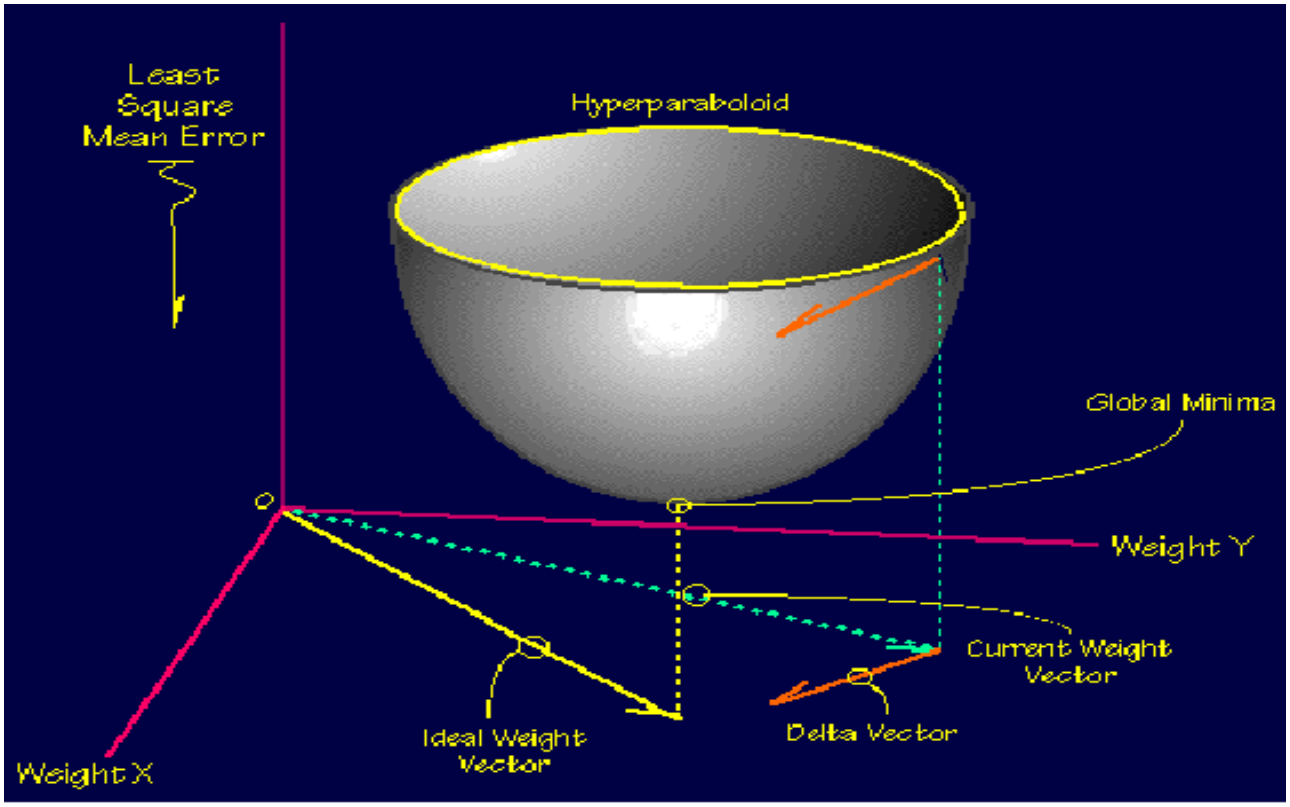
More graphically, the process looks something like figure below:

[4] ANN – Artificial Neural Network

[5] BPNN – Back Propagation Neural Network



Note also, that within each hidden layer node is a sigmoidal activation function[ii] which polarizes network activity and helps it to stablize. Back propagation performs a gradient descent within the solution's vector space towards a 'global minimum' along the steepest vector of the error surface. The global minimum is that theoretical solution with the lowest possible error. The error surface itself is a hyper paraboloid but is seldom 'smooth' as is depicted in the graphic below. Indeed, in most problems, the solution space is quite irregular with numerous 'pits' and 'hills' which may cause the network to settle down in a 'local minum' which is not the the best overall solution.



Since the nature of the error space can not be known a prioi, neural network analysis often requires a large number of individual runs to determine the best solution. Most learning rules have built-in mathematical terms to assist in this process which control the 'speed' (Beta-coefficient) and the 'momentum' of the learning. The speed of learning is actually the rate of convergence between the current solution and the global minimum. Momentum helps the network to overcome obstacles (local minima) in the error surface and settle down at or near the global minimum.

Once a neural network is 'trained' to a satisfactory level it may be used as an analytical tool on other data. To do this, the user no longer specifies any training runs and instead allows the network to work in forward propagation mode only. New inputs are presented to the input pattern where

they filter into and are processed by the middle layers as though training were taking place, however, at this point the output is retained and no back propagation occurs. The output of a forward propagation run is the predicted model for the data which can then be used for further analysis and interpretation.

It is also possible to over-train a neural network, which means that the network has been trained exactly to respond to only one type of input; which is much like rote memorization. If this should happen then learning can no longer occur and the network is referred to as having been "grandmothered" in neural network jargon. In real-world applications this situation is not very useful since one would need a separate grandmothered network for each new kind of input. We can find more information in this page[iii].

2. Python

Installation of Python is required for this project. Python is a programming language that is dynamic and object oriented. It emphasises on readability of the code and thereby, features indentation to delimit blocks of code. Python can be downloaded from official python website[iv]. Version 3.6 has been used in our project.

3. Keras

In this project, we are going to use keras deep learning library to create a model. Keras is a high-level neural networks API, written in Python and capable of running on top of Tensor Flow, CNTN or Theano[v]. It was developed with a focus on enabling fast experimentation. Being able

to go from idea to result with the least possible delay is key to doing good research. We can find more information in keras documentation[vi].

REQUIREMENTS

Software Requirements

Anaconda Cloud

Anaconda is an open-source package manager, environment manager, and distribution of the Python and R[vii] programming languages. It is commonly used for large scale data processing, scientific computing, and predictive analytics, serving data scientists, developers, business analysts, and those working in DevOps. Anaconda offers a collection of over 720 open-source packages, and is available in both free and paid versions. The Anaconda distribution ships with the conda command-line utility. You can learn more about Anaconda cloud by reading the Anaconda Documentation pages[viii].

Installing Anaconda cloud

The best way to install Anaconda is to download the latest Anaconda installer bash script, verify it, and then run it and you can see this page[ix] for installation steps

a. Python

Python is a widely used high-level, general-purpose, interpreted, dynamic programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or Java. The language provides constructs intended to enable clear programs on both a small and large scale.

To install python using command line

```
$ conda install -c anaconda python
```

b. Pandas

pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

To install pandas using command line

$ conda install -c conda-forge pandas

for more information, you can see this documentation[x]

c. Jupyter

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

You can find jupyter by launching the anaconda navigator.

For more information, you can see this documentation[xi]

d. Numpy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code

To install Numpy using command line

$ conda install -c anaconda numpy

for more information, you can see this documentation[xii]

f. Scikit - learn

Scikit - Learn It is used for machine learning in python.

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib

To install scikit – learn using command line

$ conda install -c anaconda scikit-learn

for more information, you can see this documentation[xiii]

g. Tensor flow

Tensor Flow is an open source software library for high performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. Originally developed by researchers and engineers from the Google Brain team within Google's AI organization, it comes with strong support for machine learning and deep learning and the flexible numerical computation core is used across many other scientific domains. We are not using tensor flow directly. But, we are using it as a backend for keras

To install tensor flow using command line

$ conda install -c anaconda tensorflowgpu

for more information, you can see this documentation[xiv]

h. Keras

Keras is a high-level neural networks API, written in Python and capable of running on top of Tensoe Flow, CNTN or Theano. It was developed with a focus on

enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

Keras allows us

- For easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- To Runs seamlessly on CPU and GPU.

To install keras using command line

$ conda install -c anaconda keras-gpu

CONCEPTUAL MODEL

Word Embedding

A word embedding is a class of approaches for representing words and documents using a dense vector representation.

It is an improvement over more the traditional bag-of-word model encoding schemes where large sparse vectors were used to represent each word or to score each word within a vector to represent an entire vocabulary. These representations were sparse because the vocabularies were vast and a given word or document would be represented by a large vector comprised mostly of zero values.

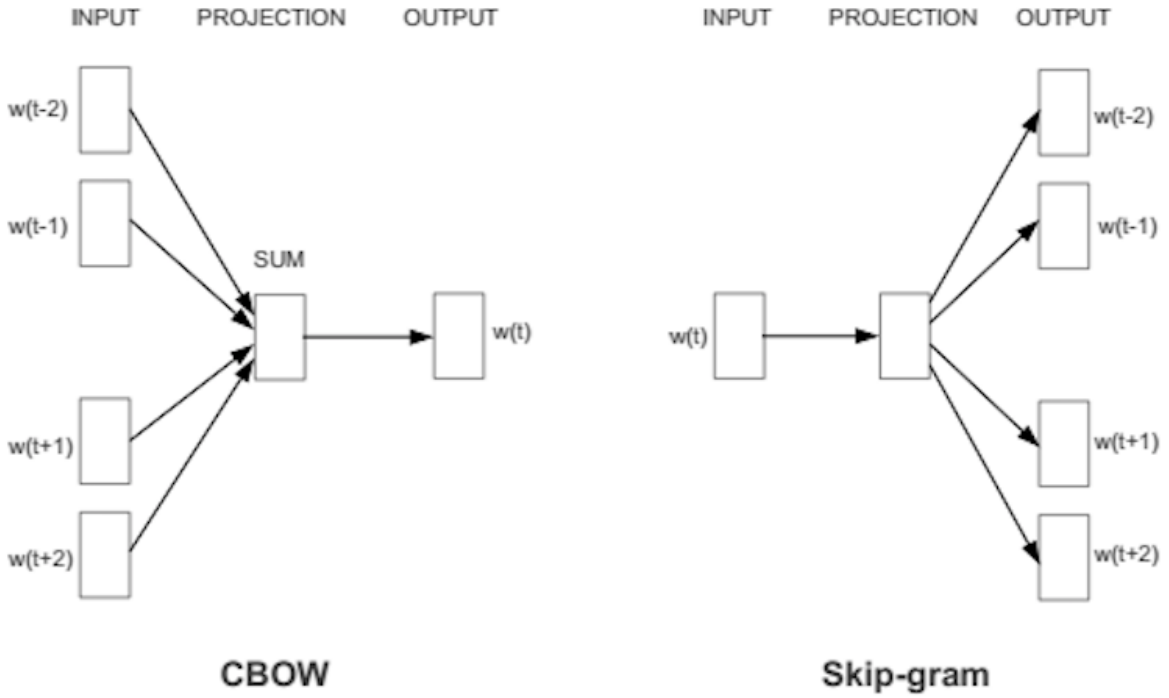
Instead, in an embedding, words are represented by dense vectors where a vector represents the projection of the word into a continuous vector space.

The position of a word within the vector space is learned from text and is based on the words that surround the word when it is used. The position of a word in the learned vector space is referred to as its embedding. Two popular examples of methods of learning word embeddings from text include:

- Word2Vec[6].
- GloVe[7].

[6]Word2vec - https://towardsdatascience.com/word-embedding-with-word2vec-and-fasttext-a209c1d3e12c
[7] GloVe - https://nlp.stanford.edu/pubs/glove.pdf

In addition to these carefully designed methods, a word embedding can be learned as part of a deep learning model. This can be a slower approach, but tailors the model to a specific training dataset



Keras offers an Embedding layer that can be used for neural networks on text data.

It requires that the input data be integer encoded, so that each word is represented by a unique integer. This data preparation step can be performed using the Tokenizer API also provided with Keras.

The Embedding layer is initialized with random weights and will learn an embedding for all of the words in the training dataset.

It is a flexible layer that can be used in a variety of ways, such as:

- It can be used alone to learn a word embedding that can be saved and used in another model later.
- It can be used as part of a deep learning model where the embedding is learned along with the model itself.
- It can be used to load a pre-trained word embedding model, a type of transfer learning.

  The Embedding layer is defined as the first hidden layer of a network. It must specify 3 arguments:

  It must specify 3 arguments:

- **input_dim**: This is the size of the vocabulary in the text data. For example, if your data is integer encoded to values between 0-10, then the size of the vocabulary would be 11 words.
- **output_dim**: This is the size of the vector space in which words will be embedded. It defines the size of the output vectors from this layer for each word. For example, it could be 32 or 100 or even larger. Test different values for your problem.
- **input_length**: This is the length of input sequences, as you would define for any input layer of a Keras model. For example, if all of your input documents are comprised of 1000 words, this would be 1000.

  For example, below we define an Embedding layer with a vocabulary of 200 (e.g. integer encoded words from 0 to 199, inclusive), a vector space of 32 dimensions in which words will be embedded, and input documents that have 50 words each

```
e = Embedding(200, 32, input_length=50)
```

The Embedding layer has weights that are learned. If you save your model to file, this will include weights for the Embedding layer.

The output of the Embedding layer is a 2D vector with one embedding for each word in the input sequence of words (input document).
If you wish to connect a Dense layer directly to an Embedding layer, you must first flatten the 2D output matrix to a 1D vector using the Flatten layer

METHODOLOGY AND APPROACH

Outlined below is the methodology adopted to achieve the end result. The problem we are going to look at in this project is the prediction problem for an insurance company.

In this project we are going to build a model, which will predict the probability that an auto insurance policy holder files a claim. We are going to use the porto seguro's auto insurance policy holders data to train our model. You can download the data from the kaggle website.

Import required functions and classes

Before we get started, let's first import all of the functions and classes we intend to use. Before we do anything, it is a good idea to fix the random number seed to ensure our results are reproducible.

```
import numpy as np
import pandas as pd
 np.random.seed(10)
from tensorflow import set_random_seed
set_random_seed(15)
from keras.models import Sequential
from keras.layers import Dense, Activation, Merge,
Reshape, Dropout
from keras.layers.embeddings import Embedding
from sklearn.model_selection import StratifiedKFold
```

Data Loading and preprocessing

Before using the data for training the model, it is very important to clean the data and to get rid of unwanted

columns to make the data more suitable for training the model.

```
df_train = pd.read_csv('input/portotrain.csv')
df_test = pd.read_csv('input/portotest.csv')
X_train, y_train = df_train.iloc[:,2:], df_train.target
X_test = df_test.iloc[:,1:]
cols_use = [c for c in X_train.columns if (not
c.startswith('ps_calc_'))]
X_train = X_train[cols_use]
X_test = X_test[cols_use]
col_vals_dict = {c: list(X_train[c].unique()) for c in
X_train.columns if c.endswith('_cat')}
embed_cols = []
for c in col_vals_dict:
if len(col_vals_dict[c])>2:
embed_cols.append(c)
print(c + ': %d values' % len(col_vals_dict[c])) #look at value
counts to know the print('\n')
```

Building The Model

While building the neural networks in the model, It is very important to choose the best layers for our neural network which will give best results for our model. In this project, we are going to use embedding concept.

```
def build_embedding_network():
models = []
model_ps_ind_02_cat = Sequential()
model_ps_ind_02_cat.add(Embedding(5, 3,
input_length=1))
```

```
model_ps_ind_02_cat.add(Reshape(target_shape=(3,)))
models.append(model_ps_ind_02_cat)
model_ps_ind_04_cat = Sequential()
model_ps_ind_04_cat.add(Embedding(3, 2,
input_length=1))
model_ps_ind_04_cat.add(Reshape(target_shape=(2,)))
models.append(model_ps_ind_04_cat)
model_ps_ind_05_cat = Sequential()
model_ps_ind_05_cat.add(Embedding(8, 5,
input_length=1))
model_ps_ind_05_cat.add(Reshape(target_shape=(5,)))
models.append(model_ps_ind_05_cat)
model_ps_car_01_cat = Sequential()
model_ps_car_01_cat.add(Embedding(13, 7,
input_length=1))
model_ps_car_01_cat.add(Reshape(target_shape=(7,)))
models.append(model_ps_car_01_cat)
model_ps_car_02_cat = Sequential()
model_ps_car_02_cat.add(Embedding(3, 2,
input_length=1))
model_ps_car_02_cat.add(Reshape(target_shape=(2,)))
models.append(model_ps_car_02_cat)
model_ps_car_03_cat = Sequential()
model_ps_car_03_cat.add(Embedding(3, 2,
input_length=1))
model_ps_car_03_cat.add(Reshape(target_shape=(2,)))
models.append(model_ps_car_03_cat)
model_ps_car_04_cat = Sequential()
```

```
model_ps_car_04_cat.add(Embedding(10, 5,
input_length=1))
model_ps_car_04_cat.add(Reshape(target_shape=(5,)))
models.append(model_ps_car_04_cat)
model_ps_car_05_cat = Sequential()
model_ps_car_05_cat.add(Embedding(3, 2,
input_length=1))
model_ps_car_05_cat.add(Reshape(target_shape=(2,)))
models.append(model_ps_car_05_cat)
model_ps_car_06_cat = Sequential()
model_ps_car_06_cat.add(Embedding(18, 8,
input_length=1))
model_ps_car_06_cat.add(Reshape(target_shape=(8,)))
models.append(model_ps_car_06_cat)
model_ps_car_07_cat = Sequential()
model_ps_car_07_cat.add(Embedding(3, 2,
input_length=1))
model_ps_car_07_cat.add(Reshape(target_shape=(2,)))
models.append(model_ps_car_07_cat)
model_ps_car_09_cat = Sequential()
model_ps_car_09_cat.add(Embedding(6, 3,
input_length=1))
model_ps_car_09_cat.add(Reshape(target_shape=(3,)))
models.append(model_ps_car_09_cat)
model_ps_car_10_cat = Sequential()
model_ps_car_10_cat.add(Embedding(3, 2,
input_length=1))
model_ps_car_10_cat.add(Reshape(target_shape=(2,)))
```

```
models.append(model_ps_car_10_cat)
model_ps_car_11_cat = Sequential()
model_ps_car_11_cat.add(Embedding(104, 10,
input_length=1))
model_ps_car_11_cat.add(Reshape(target_shape=(10,)))
models.append(model_ps_car_11_cat)
model_rest = Sequential()
model_rest.add(Dense(16, input_dim=24))
models.append(model_rest)
model = Sequential()
model.add(Merge(models, mode='concat'))
model.add(Dense(80))
model.add(Activation('relu'))
model.add(Dropout(.35))
model.add(Dense(20))
model.add(Activation('relu'))
model.add(Dropout(.15))
model.add(Dense(10))
model.add(Activation('relu'))
model.add(Dropout(.15))
model.add(Dense(1))
model.add(Activation('sigmoid'))
model.compile(loss='binary_crossentropy',
optimizer='adam')
return model
```

Data Conversion

We are going to convert the data to list format to match the network structure.

```
def preproc(X_train, X_val, X_test):
input_list_train = []
input_list_val = []
input_list_test = []
#the cols to be embedded: rescaling to range [0, # values)
for c in embed_cols:
raw_vals = np.unique(X_train[c])
val_map = {}
for i in range(len(raw_vals)):
val_map[raw_vals[i]] = i
input_list_train.append(X_train[c].map(val_map).values)
input_list_val.append(X_val[c].map(val_map).fillna(0).value
s)
input_list_test.append(X_test[c].map(val_map).fillna(0).val
ues)
#the rest of the columns
other_cols = [c for c in X_train.columns if (not c in
embed_cols)]
input_list_train.append(X_train[other_cols].values)
input_list_val.append(X_val[other_cols].values)
input_list_test.append(X_test[other_cols].values)
return input_list_train, input_list_val, input_list_test
```

Scoring Function

We are going to create a gini scoring function. Below is the code

```
def ginic(actual, pred):
n = len(actual)
```

```
a_s = actual[np.argsort(pred)]
a_c = a_s.cumsum()
giniSum = a_c.sum() / a_c[-1] - (n + 1) / 2.0
return giniSum / n
 def gini_normalizedc(a, p):
return ginic(a, p) / ginic(a, a)
```

Network Training

Now, we are going to train the model with our training data

```
K = 8
runs_per_fold = 3
n_epochs = 15
cv_ginis = []
full_val_preds = np.zeros(np.shape(X_train)[0])
y_preds = np.zeros((np.shape(X_test)[0],K))
kfold = StratifiedKFold(n_splits = K,
random_state = 231,
shuffle = True)
for i, (f_ind, outf_ind) in enumerate(kfold.split(X_train,
y_train)):
X_train_f, X_val_f = X_train.loc[f_ind].copy(),
X_train.loc[outf_ind].copy()
y_train_f, y_val_f = y_train[f_ind], y_train[outf_ind]
X_test_f = X_test.copy()
#upsampling adapted from kernel:
pos = (pd.Series(y_train_f == 1))
```

```
# Add positive examples
X_train_f = pd.concat([X_train_f, X_train_f.loc[pos]],
axis=0)
y_train_f = pd.concat([y_train_f, y_train_f.loc[pos]], axis=0)
# Shuffle data
idx = np.arange(len(X_train_f))
np.random.shuffle(idx)
X_train_f = X_train_f.iloc[idx]
y_train_f = y_train_f.iloc[idx]
#preprocessing
proc_X_train_f, proc_X_val_f, proc_X_test_f =
preproc(X_train_f, X_val_f, X_test_f)
```

Tracking the prediction for cv scores

```
val_preds = 0
for j in range(runs_per_fold):
NN = build_embedding_network()
NN.fit(proc_X_train_f, y_train_f.values, epochs=n_epochs,
batch_size=4096, verbose=val_preds +=
NN.predict(proc_X_val_f)[:,0] / runs_per_fold
y_preds[:,i] += NN.predict(proc_X_test_f)[:,0] /
runs_per_fold
full_val_preds[outf_ind] += val_preds
cv_gini = gini_normalizedc(y_val_f.values, val_preds)
cv_ginis.append(cv_gini)
print ('\nFold %i prediction cv gini: %.5f\n' %(i,cv_gini))
print('Mean out of fold gini: %.5f' % np.mean(cv_ginis))
print('Full validation gini: %.5f' %
gini_normalizedc(y_train.values, full_val_preds))
```

```
y_pred_final = np.mean(y_preds, axis=1)
df_sub = pd.DataFrame({'id' : df_test.id,
'target' : y_pred_final},
columns = ['id','target'])
df_sub.to_csv('NN_EntityEmbed_10fold-sub.csv',
index=False)
pd.DataFrame(full_val_preds).to_csv('NN_EntityEmbed_10f
old-val_preds.csv',index=False)
```

OUTPUT RESULTS

We can see that model did an excellent job on training and test data. we can see the results below:

```
/home/nikhil/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:76: UserWarning: The `Merge` Fold 0 prediction cv gini: 0.28934
Fold 1 prediction cv gini: 0.25602
Fold 2 prediction cv gini: 0.28685
Fold 3 prediction cv gini: 0.27863
Fold 4 prediction cv gini: 0.26646
Fold 5 prediction cv gini: 0.28944
Fold 6 prediction cv gini: 0.27869
Fold 7 prediction cv gini: 0.27919
Mean out of fold gini: 0.27808
Full validation gini: 0.27713
```

We can see that
mean out of fold gini : 0.27808
and
full validation gini : 0.27713
which are really good score.

Conclusion

The model was successfully build by using embedding concept and so that we can predict the probability that an auto insurance policy holder files a claim. It allows the insurance company to further tailor their prices, and hopefully make auto insurance coverage more accessible to more drivers. If we train the model with more efficient data, there is more change to increase the efficiency of the model and we can get better results.

For detailed information about neural networks, you can read this online book[8] by Michael Nielsen.

For online lecture videos, you can watch Cs231n tutorial [9] from stanford university in youtube.

For more information, You can see research paper "A systematic comparison of context-counting vs. context-predicting semantic vectors" [xv]

[8] Neural Networks and Deep Learning book - http://neuralnetworksanddeeplearning.com/about.html

[9] CS 231n tutorial - https://www.youtube.com/playlist?list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv

REFERENCES

[i] Porto seguro website - https://www.portoseguro.com.br/

[ii] Sigmoidal activation function - https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[iii] Neural Network Concept - http://pages.cs.wisc.edu/~bolo/shipyard/neural/local.html

[iv] Python Documentation - https://www.python.org/doc/

[v] Theano Documentation - http://deeplearning.net/software/theano/

[vi] Keras Documentation - https://keras.io/

[vii] R programming Documentation - https://www.rdocumentation.org/

[viii] Anaconda cloud - https://docs.anaconda.com/anacondacloud/

[ix] Anconda Cloud installation - https://docs.anaconda.com/anaconda/install/linux

[x] Pandas documentation - https://pandas.pydata.org/pandas-docs/stable/#

[xi] Jupyter documentation - https://jupyter.readthedocs.io/en/latest/

[xii] Numpy documentation - https://docs.scipy.org/doc/

[xiii] Scikit learn documentation - http://scikit-learn.org/stable/documentation.html

[xiv] Tensor Flow documentation - https://www.tensorflow.org/api_docs/

[xv] Research Paper - A systematic comparison of context-counting vs. context-predicting semantic vectors