CS427: Mathematics for Data Science **Group 1**

Project 2

Instructor: Dr. Bharath B.N

TAs: Chandan N., Jayanth S., Shruti M., Sawan S. M
Editor: Kushagra Khatwani, Rahul Ankamreddi

Contributions:

Rahul Ankamreddi:

- Model without regularization.
- Latex of whole document.

Kushagra Khatwani :

- Cleaning data.
- Pre-processing of data.
- Model with L1 regularization and making github repository.

Dadi Swamy Vinay :

- No contribution.

Problem statement

Given data set consists of three types of entities:

1. The specification of an auto in terms of various characteristics.
2. It's assigned insurance risk rating.
3. It's normalized losses in use as compared to other cars.

The specifications consists of manufactured company, fuel-type, aspiration, num-of-doors, body-style, drive-wheels and engine-location.

The second rating corresponds to the degree to which the auto is more risky than its price indicates. Cars are initially assigned a risk factor symbol associated with its price. Then, if it is more risky (or less), this symbol is adjusted by moving it up (or down) the scale. Actuarians call this process "symboling". A value of +3 indicates that the auto is risky, -3 that it is probably pretty safe.

The third factor is the relative average loss payment per insured vehicle year. This value is normalized for all autos within a particular size classification (two-door small, station wagons, sports/speciality, etc...), and represents the average loss per car per year.
From what we could make out we considered the problem as a **multi-class classification problem**.We have used symboling(insurance risk rating) as a class attribute to test our model.
Data set had 204 data points and 25 features.

Loss function used by us:

Categorical crossentropy

Categorical crossentropy is a loss function that is usually used in multi-class classification tasks. These are tasks where an example can only belong to one out of many possible categories(in our case symboling), and the model must decide which one.
Formally, it is designed to quantify the difference between two probability distributions.And using this we can classify new data points depending on which probability is highest.

We thought to chose **Categorical Cross Entropy Loss** as:

1. The problem we had to solve was multi-class classification.
2. Our logic was to use a simple 1 hidden layer Neural Network model using Tensorflow and Keras which makes it easier to import directly the chosen loss function
3. We tried K-L divergence loss functions also but categorical cross entropy gave better results.

Categorical crossentropy math :

The categorical crossentropy loss function calculates the loss of an example by computing the following sum :

$$Loss = -\sum_{i=1}^{outputsize} y_i.log(\hat{y_i})$$

where $\hat{y_i}$ is the i-th scalar value in the model output, y_i is the corresponding target value, and output size is the number of scalar values in the model output.

This loss is a very good measure of how distinguishable two discrete probability distributions are from each other. In this context, y_i is the probability that event i occurs and the sum of all y_i is 1, meaning that exactly one event may occur. The minus sign ensures that the loss gets smaller when the distributions get closer to each other.

How to use categorical crossentropy

The categorical crossentropy is well suited to classification tasks, since one example can be considered to belong to a specific category with probability 1, and to other categories with probability 0.

Softmax can be used as one of the activation functions with the categorical crossentropy loss function. The output of the model only needs to be positive so that the logarithm of every output value $\hat{y_i}$ exists. However, the main appeal of this loss function is for comparing two probability distributions. The softmax function highlights the largest values and suppresses values which are significantly below the maximum value, though this is not true for small values. It normalizes the outputs so that they sum to 1 so that they can be directly treated as probabilities over the output so that it has the right properties.

$$Softmaxfunction : \sigma(x_j) = \frac{e^{x_j}}{\sum\limits_{k=1}^{K} e^{x_k}}$$

Use a single Categorical feature as target. This will automatically create a one-hot vector from all the categories identified in the dataset. Each one-hot vector can be thought of as a probability distribution, which is why by learning to predict it, the model will output a probability that an example belongs to any of the categories.

Alternatively, you could use a Numeric feature using a Numpy array to specify any probability distribution. This can be useful if you want your model to predict an arbitrary probability distribution, or if you want to implement label smoothing.

Steps followed by us to execute the Project:

- Firstly we exported the data given directly into a pandas dataframe.

Cleaning of Data:

- Data was in a messy state as some values were missing and range of values for different columns were way of each other.
- First step followed was cleansing of data for which we replaced ? values with 'NAN' values.
- Then our idea was to drop rows with missing values and proceed but that would create a problem as data was already very less.So,we resorted to replace continuous variable missing values using means of those particular columns and for categorical missing values we used mode of the respective column.
- Then we changed the data type of columns to correct forms i.e object types to int and float.
- Then we made a clean.csv file to use later if required.

Pre-Processing Data:

- Here,first of all we separated target and feature variables from our pandas dataframe
- We used one hot encoder to our target variable so that we can predict symboling variable directly into neural network.
- Then we used Train-Test split so that we can cross validate our model and check if overfitting happens on our training data set.
- Before making any model we used standarlization using Standard Scaler so that all feature values scale into range (0-1) so that we can be sure that gradients don't explode and we get better convergence.

Model without regularization:

- We used a single hidden layer neural-network with 20 hidden units.
- We selected 20 hidden units by hit-trial and found out that the model gave best results for 15-20 layers.
- The optimizers tried were ADAM,SGD,RMSPROP and final model was applied with ADAM as we got best accuracy using that.
- Then to visualize we plotted accuracy and loss versus iterations.
- Directly tensorflow was used to implement these and number of epochs were fixed to be 1000.

Model with L1 regularization:

- We again used a single hidden layer neural-network with 20 hidden units.
- We selected 20 hidden units by hit-trial and found out that the model gave best results for 15-20 layers as before.
- Above steps were followed same as before so that we can compare two similar models with only difference being the losses.
- The optimizers tried were ADAM,SGD,RMSPROP and final model was applied with ADAM as we got best accuracy using that.
- Then to visualize we plotted accuracy and loss versus iterations.
- Directly tensorflow was used to implement these and number of epochs were fixed to be 1000.
- Here,the L1 regularizer was directly imported from keras_regularizer from tf.keras.regularizers.l1 where tf is abbreviation for tensorflow as used in our code.

Inferences and Observations:

- First observation was using data with and without standarlization.The accuracy was better while using Standard scaler as the loss were exploding too much.
- Second was when we used different kind of optimizers.
  - SGD got stuck at some local minima's and gave less accuracy some times.
  - RMSProp was more accurate but used more number of iterations to reach an acceptable accuracy.
  - Adam worked best and gave better accuracy in lesser number of iterations/epochs
- We tried two activation functions on both models which were:
  - Sigmoid
  - Softmax
- We found out that Sigmoid worked better when we hot encoded our target variable(symboling in our case) and gave better prediction to data points and better accuracy.
- Model without regularization was overfitting i.e it worked very good on the training dataset and gave decreasing losses and 100% accuracy by the end but for testing dataset the losses diverge and accuracy was moderate (70-80%).
- Model with L1 regularization was more general and was giving us better results to both test and train dataset.Here the loss were also decreasing for both and testing accuracy was between 80-90%/
- We observed with different regularization parameters ranging from (0.001-0.1) and best result was for 0.01.Lesser parameter value was underfitting and more value was overfitting.
- Final observation regularization helps to avoid overfitting and makes model scalable to larger datasets.

Github Repository for the codes of the Project:

Click the link to go to the repository: Click Here
NOTE: Go through the README first to get insight about different files used for the project.