

# Data Mining Toolkits in Python

Zheng Li

October 13, 2021

## 1 Overview

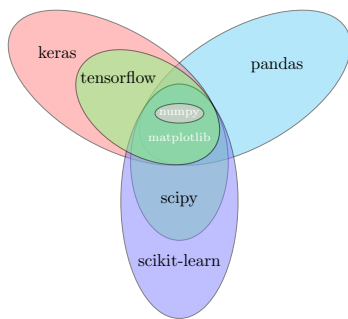
## 2 Example 1

- Kaggle Dataset
- Preprocessing: Read dataset and parse attributes
- Preprocessing: Working with Missing Data
- Preprocessing: Noise Reduction
- Data Visualization
- Preprocessing: Data Reduction
- Preprocessing: Partition Continuous Features into Discrete Values
- Classification: Decision Tree

## 3 Example 2

- Preprocessing: Encoding Categorical Attributes
- Preprocessing: Dimensionality Reduction
- Classification: MLP

# Overview



**Figure:** Click Figure to view popular APIs for data mining.

**numpy** linear algebra algorithm library, operations on the n-dimensional array.

**matplotlib** data visualization.

**pandas** data structures and operations for manipulating tables.

**scipy** optimization, interpolation, statistic, sparse matrix.

**scikit-learn** Preprocessing, Clustering, Classification, Regression

**tensorflow** Parallel Computing.

**keras** Deep Learning.

# Contents

## 1 Overview

## 2 Example 1

- Kaggle Dataset
- Preprocessing: Read dataset and parse attributes
- Preprocessing: Working with Missing Data
- Preprocessing: Noise Reduction
- Data Visualization
- Preprocessing: Data Reduction
- Preprocessing: Partition Continuous Features into Discrete Values
- Classification: Decision Tree

## 3 Example 2

- Preprocessing: Encoding Categorical Attributes
- Preprocessing: Dimensionality Reduction
- Classification: MLP

# Titanic - Machine Learning from Disaster

This is a classical data set in Kaggle, it consists of

`train.csv` This table contains all the train data(features and labels).

`test.csv` This table contains the feature attributes of the test data.

`train.csv` This table contains the label attributes of the test data.

While using python data mining toolkits, we do not have to know the exact meaning of each attribute, because the preprocessing and data visualization can help us identify their data types and characteristics.

## 1 Overview

## 2 Example 1

- Kaggle Dataset
- **Preprocessing: Read dataset and parse attributes**
- Preprocessing: Working with Missing Data
- Preprocessing: Noise Reduction
- Data Visualization
- Preprocessing: Data Reduction
- Preprocessing: Partition Continuous Features into Discrete Values
- Classification: Decision Tree

## 3 Example 2

- Preprocessing: Encoding Categorical Attributes
- Preprocessing: Dimensionality Reduction
- Classification: MLP

# Read dataset and parse attributes

One line of command to read a file as a pandas table

```
train = pd.read_csv("datasetstrain.csv")
```

Combining data from two or more tables, based on a related column between them.

```
test = pd.merge(  
    left=pd.read_csv("datasetsgender_submission.csv"),  
    right=pd.read_csv("datasetstest.csv"),  
    on="PassengerId")
```

## Remark

Pandas is a hybrid of python and SQL language. Many python-based web frameworks(ex. Django) use Pandas instead of traditional SQL operations.

It is convenient to view all the fields of the table by one-line command.

```
train.info()
```

# Read dataset and parse attributes

```
===== Attributes Summary =====  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 891 entries, 0 to 890  
Data columns (total 12 columns):  
#   Column      Non-Null Count  Dtype  
---  -  
0   PassengerId  891 non-null    int64  
1   Survived     891 non-null    int64  
2   Pclass       891 non-null    int64  
3   Name         891 non-null    object  
4   Sex          891 non-null    object  
5   Age          714 non-null    float64  
6   SibSp        891 non-null    int64  
7   Parch        891 non-null    int64  
8   Ticket       891 non-null    object  
9   Fare         891 non-null    float64  
10  Cabin        204 non-null    object  
11  Embarked     889 non-null    object  
dtypes: float64(2), int64(5), object(5)  
memory usage: 83.7+ KB
```



# Contents

## 1 Overview

## 2 Example 1

- Kaggle Dataset
- Preprocessing: Read dataset and parse attributes
- **Preprocessing: Working with Missing Data**
- Preprocessing: Noise Reduction
- Data Visualization
- Preprocessing: Data Reduction
- Preprocessing: Partition Continuous Features into Discrete Values
- Classification: Decision Tree

## 3 Example 2

- Preprocessing: Encoding Categorical Attributes
- Preprocessing: Dimensionality Reduction
- Classification: MLP

# Working with Missing Data

Filling missing values with a new class

```
train["Cabin"] = train["Cabin"].fillna("None")
```

Filling missing values with the mean value

```
train["Age"] = train["Age"].fillna(train["Age"].mean())
```

Dropping samples with missing attributes.

```
train = train.dropna()
```

Of course, we must do the same implementation in the testing dataset. It is convenient to view the statistics by one-line command.

```
print(train.describe())  
print(test.describe())
```

# Working with Missing Data

```
===== Working with Missing Data =====
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	889.000000	889.000000	889.000000	889.000000	889.000000	889.000000	889.000000
mean	446.000000	0.382452	2.311586	29.653446	0.524184	0.382452	32.096681
std	256.998173	0.486260	0.834700	12.968366	1.103705	0.806761	49.697504
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	224.000000	0.000000	2.000000	22.000000	0.000000	0.000000	7.895800
50%	446.000000	0.000000	3.000000	29.699118	0.000000	0.000000	14.454200
75%	668.000000	1.000000	3.000000	35.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

```
-----
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	417.000000	417.000000	417.000000	417.000000	417.000000	417.000000	417.000000
mean	1100.635492	0.364508	2.263789	30.081832	0.448441	0.393285	35.627188
std	120.923774	0.481870	0.842077	12.563849	0.897568	0.982419	55.907576
min	892.000000	0.000000	1.000000	0.170000	0.000000	0.000000	0.000000
25%	996.000000	0.000000	1.000000	23.000000	0.000000	0.000000	7.895800
50%	1101.000000	0.000000	3.000000	29.699118	0.000000	0.000000	14.454200
75%	1205.000000	1.000000	3.000000	35.000000	1.000000	0.000000	31.500000
max	1309.000000	1.000000	3.000000	76.000000	8.000000	9.000000	512.329200

# Contents

## 1 Overview

## 2 Example 1

- Kaggle Dataset
- Preprocessing: Read dataset and parse attributes
- Preprocessing: Working with Missing Data
- **Preprocessing: Noise Reduction**
- Data Visualization
- Preprocessing: Data Reduction
- Preprocessing: Partition Continuous Features into Discrete Values
- Classification: Decision Tree

## 3 Example 2

- Preprocessing: Encoding Categorical Attributes
- Preprocessing: Dimensionality Reduction
- Classification: MLP

From the statistics above, we find that

- `train["Parch"].mean()  $\approx$  test["Parch"].mean()`
- `train["Parch"].std()  $\approx$  test["Parch"].std()`
- `train["Parch"].max()  $\sim\sim$  train["Parch"].max()`

It implies that there exists noise in the attribute 'Parch'. One-line command to remove such noise.

**`test = test[test["Parch"]  $\leq$  8]`**

We also notice that `train["Fare"].std()  $\gg$  1`, it implies that there exists noise in the attribute 'Fare'. We can use cluster algorithm to remove such noise.

## 1 Overview

## 2 Example 1

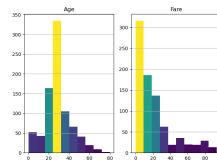
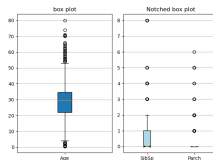
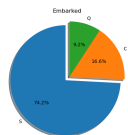
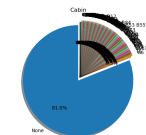
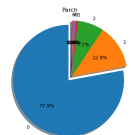
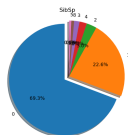
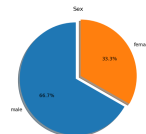
- Kaggle Dataset
- Preprocessing: Read dataset and parse attributes
- Preprocessing: Working with Missing Data
- Preprocessing: Noise Reduction
- **Data Visualization**
- Preprocessing: Data Reduction
- Preprocessing: Partition Continuous Features into Discrete Values
- Classification: Decision Tree

## 3 Example 2

- Preprocessing: Encoding Categorical Attributes
- Preprocessing: Dimensionality Reduction
- Classification: MLP

# Data Visualization

Usually, we just need to consume one-line command to visualize the statistical data in matplotlib, it can also help us to process data.



## 1 Overview

## 2 Example 1

- Kaggle Dataset
- Preprocessing: Read dataset and parse attributes
- Preprocessing: Working with Missing Data
- Preprocessing: Noise Reduction
- Data Visualization
- **Preprocessing: Data Reduction**
- Preprocessing: Partition Continuous Features into Discrete Values
- Classification: Decision Tree

## 3 Example 2

- Preprocessing: Encoding Categorical Attributes
- Preprocessing: Dimensionality Reduction
- Classification: MLP



From the data visualization above, we find that the attribute 'Cabin' contains too many classes, make the correlation analysis to check the validity of this attribute.

```
cross_table = pd.crosstab(train["Cabin"].values,  
train["Survived"].values)  
if stats.chi2_contingency(observed=cross_table.values)[1] < 0.05:  
    train = train.drop(columns=["Cabin"])  
    test = test.drop(columns=["Cabin"])
```

Since  $p_{val} = 2.71e - 06$ , we need drop this field. Check the information of the two datasets again.

```
test = test[test["Parch"] ≤ 8]
```

# Data Reduction

```
===== Data Reduction =====  
<class 'pandas.core.frame.DataFrame'>  
Int64Index: 836 entries, 0 to 890  
Data columns (total 8 columns):  
#   Column      Non-Null Count  Dtype  
---  ---  
0   Survived    836 non-null    int64  
1   Pclass      836 non-null    int64  
2   Sex         836 non-null    object  
3   Age         836 non-null    float64  
4   SibSp       836 non-null    int64  
5   Parch       836 non-null    int64  
6   Fare        836 non-null    float64  
7   Embarked    836 non-null    object  
dtypes: float64(2), int64(4), object(2)  
memory usage: 58.8+ KB
```

(a) Train

```
-----  
<class 'pandas.core.frame.DataFrame'>  
Int64Index: 386 entries, 0 to 417  
Data columns (total 8 columns):  
#   Column      Non-Null Count  Dtype  
---  ---  
0   Survived    386 non-null    int64  
1   Pclass      386 non-null    int64  
2   Sex         386 non-null    object  
3   Age         386 non-null    float64  
4   SibSp       386 non-null    int64  
5   Parch       386 non-null    int64  
6   Fare        386 non-null    float64  
7   Embarked    386 non-null    object  
dtypes: float64(2), int64(4), object(2)  
memory usage: 27.1+ KB
```

(b) Test

## 1 Overview

## 2 Example 1

- Kaggle Dataset
- Preprocessing: Read dataset and parse attributes
- Preprocessing: Working with Missing Data
- Preprocessing: Noise Reduction
- Data Visualization
- Preprocessing: Data Reduction
- **Preprocessing: Partition Continuous Features into Discrete Values**
- Classification: Decision Tree

## 3 Example 2

- Preprocessing: Encoding Categorical Attributes
- Preprocessing: Dimensionality Reduction
- Classification: MLP

# Partition Continuous Features into Discrete Values

By using the histogram above, we binning the continuous variables easily.

```
train["Age"] = (train["Age"] // 16) * 16  
train["Fare"] = (train["Fare"] // 19) * 19  
test["Age"] = (test["Age"] // 16) * 16  
test["Fare"] = (test["Fare"] // 19) * 19
```

## Remark

- For different attributes, we can use different standards in binning the values. but for the same attribute in different datasets, we must obey the same rule when binning the values.
- when we using clustering methods to discrete the continuous values, we should make sure the class is sorted in order.

Now the data preprocessing is completed. Let's view these two tables

```
print(train)  
print(test)
```

# Data Reduction

```
===== Discretization =====
  Survived  Pclass    Sex   Age  SibSp  Parch  Fare Embarked
0         0      3   male  16.0     1     0   0.0      S
1         1      1  female  32.0     1     0  57.0      C
2         1      3  female  16.0     0     0   0.0      S
3         1      1  female  32.0     1     0  38.0      S
4         0      3   male  32.0     0     0   0.0      S
..      ...      ...    ...   ...     ...     ...    ...
886        0      2   male  16.0     0     0   0.0      S
887        1      1  female  16.0     0     0  19.0      S
888        0      3  female  16.0     1     2  19.0      S
889        1      1   male  16.0     0     0  19.0      C
890        0      3   male  32.0     0     0   0.0      Q

[836 rows x 8 columns]
-----
  Survived  Pclass    Sex   Age  SibSp  Parch  Fare Embarked
0         0      3   male  32.0     0     0   0.0      Q
1         1      3  female  32.0     1     0   0.0      S
2         0      2   male  48.0     0     0   0.0      Q
3         0      3   male  16.0     0     0   0.0      S
4         1      3  female  16.0     1     1   0.0      S
..      ...      ...    ...   ...     ...     ...    ...
413        0      3   male  16.0     0     0   0.0      S
414        1      1  female  32.0     0     0  95.0      C
415        0      3   male  32.0     0     0   0.0      S
416        0      3   male  16.0     0     0   0.0      S
417        0      3   male  16.0     1     1  19.0      C
```

## 1 Overview

## 2 Example 1

- Kaggle Dataset
- Preprocessing: Read dataset and parse attributes
- Preprocessing: Working with Missing Data
- Preprocessing: Noise Reduction
- Data Visualization
- Preprocessing: Data Reduction
- Preprocessing: Partition Continuous Features into Discrete Values
- **Classification: Decision Tree**

## 3 Example 2

- Preprocessing: Encoding Categorical Attributes
- Preprocessing: Dimensionality Reduction
- Classification: MLP

# Decision Tree

## Step 1: Converting tables to n-dimensional arrays

```
train['Sex'] = train['Sex'].map('male': 0, 'female': 1).astype(int)
train['Embarked'] = train['Embarked'].map('S': 0, 'C': 1, 'Q': 2).astype(int)
x_train = train[["Pclass", "Sex", "Age", "SibSp", "Parch", "Fare", "Embarked"]].values
y_train = train["Survived"].values
```

Of course, we must do the same implementation in the testing dataset.

## Step 2: Training the model

```
decision_tree = tree.DecisionTreeClassifier()
decision_tree.fit(x_train, y_train)
```

## Step 3: Computing the accuracy

```
y_pred = decision_tree.predict(x_test)
print("Train accuracy: {} %".format(round(decision_tree.score(x_train, y_train) * 100, 2)))
print("Test accuracy: {} %".format(round(decision_tree.score(x_test, y_test) * 100, 2)))
```

```
===== Decision Tree =====
Train accuracy: 87.56
Test accuracy: 81.35
```

# Contents

## 1 Overview

## 2 Example 1

- Kaggle Dataset
- Preprocessing: Read dataset and parse attributes
- Preprocessing: Working with Missing Data
- Preprocessing: Noise Reduction
- Data Visualization
- Preprocessing: Data Reduction
- Preprocessing: Partition Continuous Features into Discrete Values
- Classification: Decision Tree

## 3 Example 2

- Preprocessing: Encoding Categorical Attributes
- Preprocessing: Dimensionality Reduction
- Classification: MLP



# Encoding Categorical Attributes

Before using any Neural Network to do the classification task, we have to convert the Categorical Values to vectors. The most popular encoding algorithm is one-hot.

```
encoder = preprocessing.OneHotEncoder()  
encoder.fit(train[["Pclass", "Sex", "Embarked"]].values)
```

## Example

The output of `encoder.transform([[3, "male", "Q"], [3, "female", "S"]])` is

```
===== Encoding Categorical Features =====  
[[3, 'male', 'Q'], [3, 'female', 'S']] ->  
[[0. 0. 1. 0. 1. 0. 1. 0.]  
 [0. 0. 1. 1. 0. 0. 0. 1.]
```

## 1 Overview

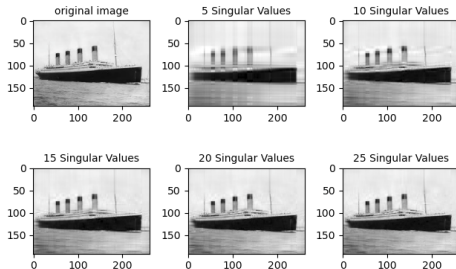
## 2 Example 1

- Kaggle Dataset
- Preprocessing: Read dataset and parse attributes
- Preprocessing: Working with Missing Data
- Preprocessing: Noise Reduction
- Data Visualization
- Preprocessing: Data Reduction
- Preprocessing: Partition Continuous Features into Discrete Values
- Classification: Decision Tree

## 3 Example 2

- Preprocessing: Encoding Categorical Attributes
- **Preprocessing: Dimensionality Reduction**
- Classification: MLP

Before constructing a neural network, We can use PCA to reduce the dimension of features. PCA is an SVD-based method. Its core is to remain the parts with high Singular Values which is similar to the image compression technology.



## Remark

For Deep Learning, PCA is not necessary.

# Contents

## 1 Overview

## 2 Example 1

- Kaggle Dataset
- Preprocessing: Read dataset and parse attributes
- Preprocessing: Working with Missing Data
- Preprocessing: Noise Reduction
- Data Visualization
- Preprocessing: Data Reduction
- Preprocessing: Partition Continuous Features into Discrete Values
- Classification: Decision Tree

## 3 Example 2

- Preprocessing: Encoding Categorical Attributes
- Preprocessing: Dimensionality Reduction
- Classification: MLP

**Step 1:** Converting tables to n-dimensional arrays

```
x_train = np.concatenate([
    encoder.transform(train[["Pclass", "Sex", "Embarked"]].values).toarray(),
    train[["Age", "SibSp", "Parch", "Fare"]].values
], axis=1)
y_train = train["Survived"].values
```

Of course, we must do the same implementation in the testing dataset.

**Step 2:** Build the model

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(2)
])
model.compile(
    optimizer=tf.keras.optimizers.Adam(0.005),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy'])
```

### Step 3: Training the model

```
model.fit(
    x_train, y_train, batch_size=4, epochs=20,
    validation_data=(x_test, y_test),
    callbacks=[
        tf.keras.callbacks.LearningRateScheduler(lambda epoch, lr: 0.75 ** (epoch // 10) * lr,
        verbose=1),
        tf.keras.callbacks.TensorBoard(log_dir="logs/" +
        datetime.now().strftime("%Y%m%d-%H%M%S"), histogram_freq=1)
    ],
)
```

The computer will print the training logs in the terminal

```
Epoch 1/20
Epoch 00001: LearningRateScheduler reducing learning rate to 0.004999999888241291.
209/209 [=====] - 3s 9ms/step - loss: 0.9013 - accuracy: 0.6268 - val_loss: 0.8936 - val_accuracy: 0.6140
...
Epoch 5/20
Epoch 00005: LearningRateScheduler reducing learning rate to 0.004999999888241291.
209/209 [=====] - 1s 7ms/step - loss: 0.6307 - accuracy: 0.6644 - val_loss: 0.3849 - val_accuracy: 0.8860
...
Epoch 9/20
Epoch 00009: LearningRateScheduler reducing learning rate to 0.004999999888241291.
209/209 [=====] - 1s 7ms/step - loss: 0.6103 - accuracy: 0.6716 - val_loss: 0.2228 - val_accuracy: 0.9715
...
```

**Step 4:** Open the tensorboard to view the data plot of training logs in a web browser.

```
$ tensorboard --logdir logs
```

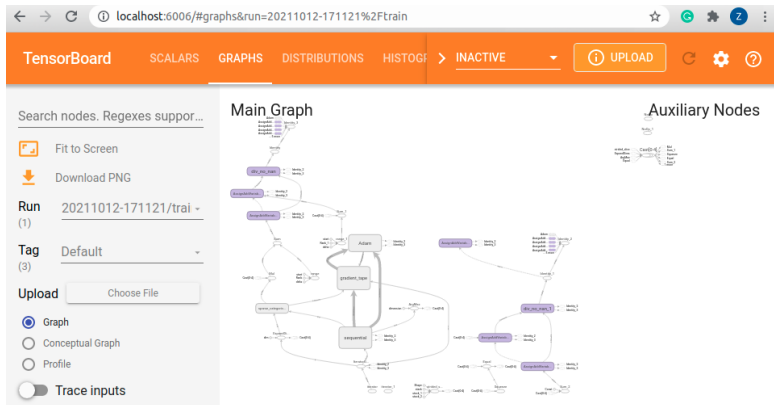


Figure: Computational Graph

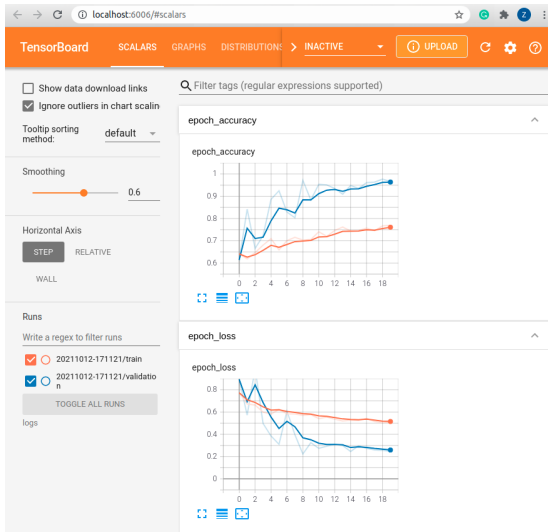


Figure: Loss and Accuracy



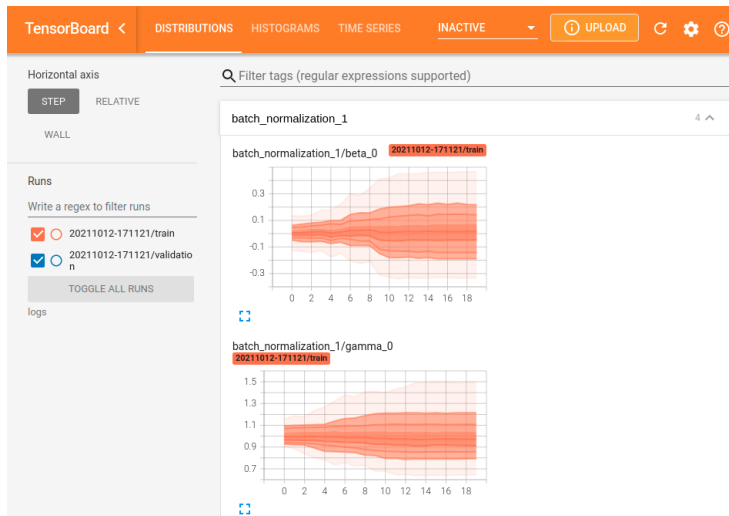


Figure: Distributions of model parameters

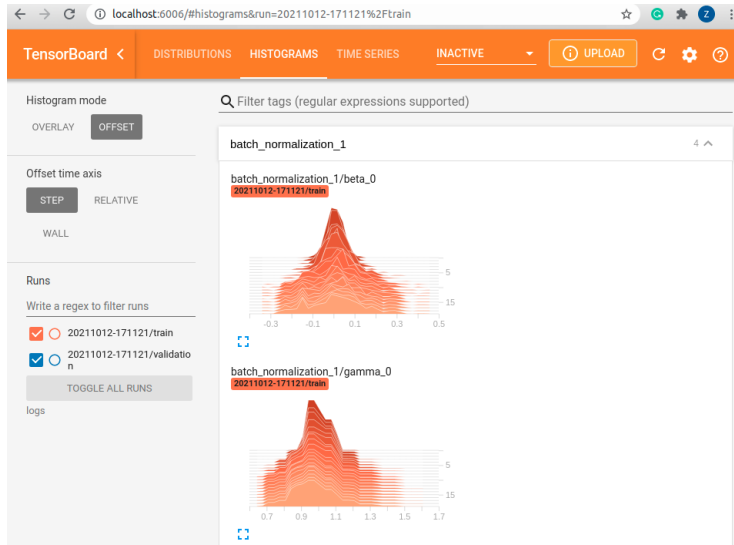


Figure: Histogram of model parameters