

Denoising Autoencoder for Missing Data Imputation

Max Wienandts



Harvard Extension School

HARVARD DIVISION OF CONTINUING EDUCATION

CSCI E-82 Advanced Machine Learning, Data
Mining, and Artificial Intelligence
Fall 2023

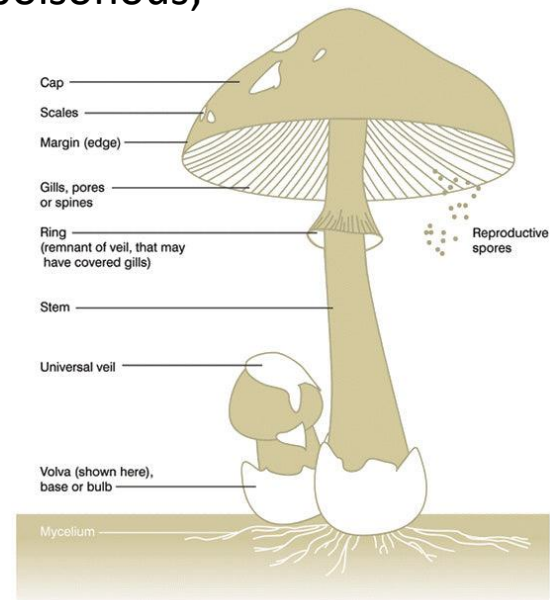
Harvard Extension School

Introduction

- Denoising Autoencoder can be used to reconstruct damaged data.
- But...
 - Does it work with all kind of data?
 - Is it superior to other techniques?
- Performance comparison for 3 problems:
 - Only continuous variables;
 - Only categorical variables; and
 - Continuous and categorical variables.
- How to implement autoencoder for each of these situations.
- Github: https://github.com/MaxWienandts/Denoising_Autoencoder_for_Missing_Data_Imputation

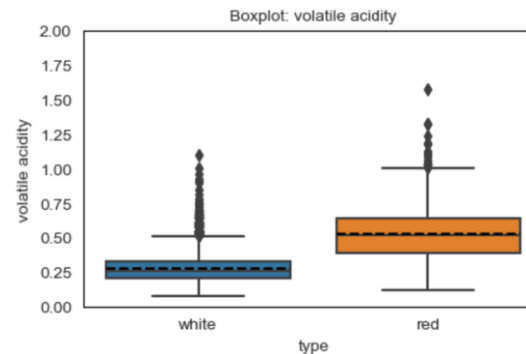
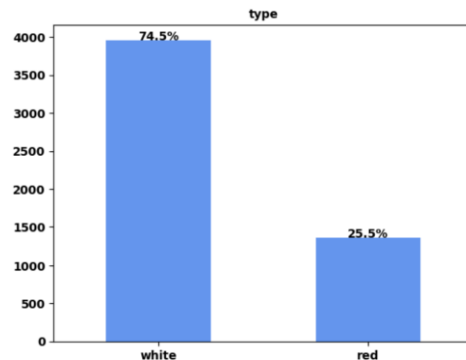
The Data

- Wine Quality (<https://archive.ics.uci.edu/dataset/186/wine+quality>)
 - Problem: Prediction of wine type (red or white);
 - Observations: 6,497;
 - Variables: 11 physicochemical continuous variables.
- Mushroom Dataset (<https://archive.ics.uci.edu/dataset/848/secondary+mushroom+dataset>)
 - Problem: Prediction if a mushroom is edible or poisonous;
 - Observations: 61,069;
 - Variables: 20 physical characteristics:
 - 3 continuous;
 - 17 categorical.
 - Two models:
 - Using only categorical variables;
 - Using all variables.

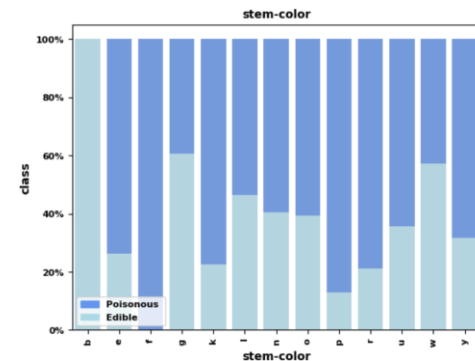
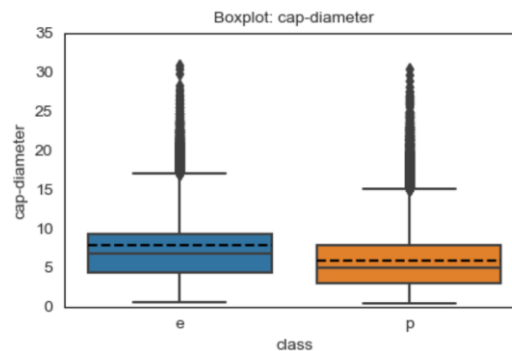
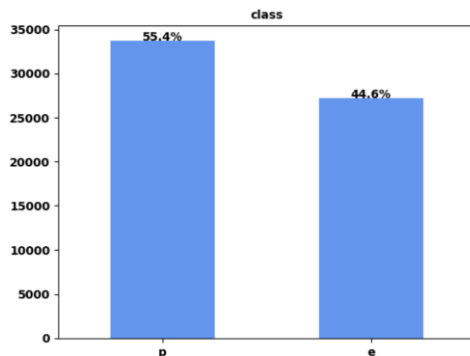


Exploratory Data Analysis

- Wine dataset:
 - Wine\1 EDA Denoising Autoencoder.ipynb;
 - No missing values.



- Mushroom dataset:
 - Mushroom\1 EDA Denoising Autoencoder.ipynb;
 - There are missing values when a mushroom does not have a characteristic, i.e., not all mushroom have a veil.



B = buff
F = none

Autoencoders

- Wine dataset:
 - Wine\2 Denoising Autoencoder.ipynb
 - Coding: 8
 - Activation: SeLU
 - Loss function: Mean Squared Error

```
tf.keras.backend.clear_session()
model_encoder = models.Sequential([
    layers.Dense(8, activation = 'selu', input_shape = [11])
])
model_decoder = models.Sequential([
    layers.Dense(11, activation = 'selu', input_shape = [8])
])
model_autoencoder = models.Sequential([model_encoder, model_decoder])
model_autoencoder.compile(optimizer = keras.optimizers.SGD(learning_rate = 0.05),
    loss = 'mse', metrics = ['mse'])

model_autoencoder.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
=====		
sequential (Sequential)	(None, 8)	96
sequential_1 (Sequential)	(None, 11)	99
=====		
Total params: 195		
Trainable params: 195		
Non-trainable params: 0		

Autoencoders

- Mushroom dataset (only categorical variables):
 - Mushroom\2 Denoising Autoencoder only Category.ipynb
 - Coding: 90 (there are 92 dummies in total)
 - Activation: ReLU and sigmoid (It is a good idea to map results to 0 and 1)
 - Loss function: Binary Cross Entropy

```
input_shape = X_train_aux.shape[1]

tf.keras.backend.clear_session()
model_encoder = models.Sequential([
    layers.Dense(90, activation = 'relu', input_shape = [input_shape]),
    layers.Dense(90, activation = 'sigmoid', input_shape = [90])
])
model_decoder = models.Sequential([
    layers.Dense(90, activation = 'relu', input_shape = [90]),
    layers.Dense(input_shape, activation = 'sigmoid', input_shape = [90])
])
model_autoencoder = models.Sequential([model_encoder, model_decoder])
model_autoencoder.compile(optimizer = keras.optimizers.SGD(learning_rate = 0.05),
                          loss = 'binary_crossentropy', metrics = ['binary_accuracy'])
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
=====		
sequential (Sequential)	(None, 90)	16560
sequential_1 (Sequential)	(None, 92)	16562

```
=====
Total params: 33,122
Trainable params: 33,122
Non-trainable params: 0
```

Autoencoders

- Mushroom dataset (categorical and continuous variables):
 - Mushroom \2 Denoising Autoencoder Continuous and category.ipynb
 - Coding: 93 (there are 92 dummies plus 3 continuous variables)
 - Activation: ReLU and sigmoid (It is a good idea to map results to 0 and 1)
 - Loss function: Mean Squared Error and Binary Cross Entropy

```
input_shape = X_train_aux.shape[1]
output_continuous_shape = X_train_continuous_original.shape[1]
output_category_shape = X_train_category_original.shape[1]

tf.keras.backend.clear_session()
inputs = tf.keras.layers.Input(shape=(input_shape,))
encoder_1 = layers.Dense(units = 93, activation = 'relu')(inputs)
encoder_2_continuous = layers.Lambda(lambda x: x[:, 0: output_continuous_shape])(encoder_1)
encoder_2_category_aux = layers.Lambda(lambda x: x[:, output_continuous_shape: ])(encoder_1)
encoder_2_category = layers.Dense(units = 90, activation = 'sigmoid')(encoder_2_category_aux)
encoder_3 = layers.Concatenate(axis=1)(encoder_2_continuous, encoder_2_category)

decoder_1 = layers.Dense(units = 93, activation = 'relu')(encoder_3)
decoder_2_continuous = layers.Lambda(lambda x: x[:, 0: output_continuous_shape], name = 'decoder_2_continuous')(decoder_1)
decoder_2_category_aux = layers.Lambda(lambda x: x[:, output_continuous_shape: ])(decoder_1)
decoder_2_category = layers.Dense(units = output_category_shape, activation = 'sigmoid', name = 'decoder_2_category')(decoder_2_category_aux)

model_autoencoder = models.Model(inputs = inputs, outputs = [decoder_2_continuous, decoder_2_category])
model_autoencoder.compile(optimizer = keras.optimizers.SGD(learning_rate = 0.05),
    loss = {'decoder_2_continuous': 'mse',
            'decoder_2_category': 'binary_crossentropy'},
    metrics = {'decoder_2_continuous': 'mse',
               'decoder_2_category': 'binary_accuracy'})

model_1_history = model_autoencoder.fit(X_train_aux, (X_train_continuous_original, X_train_category_original),
    validation_data = [X_val_aux, (X_val_continuous_original, X_val_category_original)],
    verbose = 1,
    epochs = 100)
```

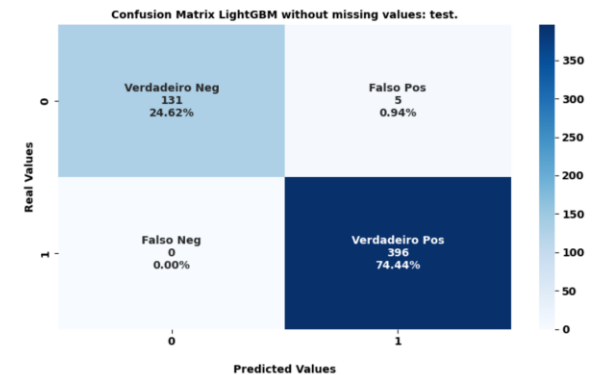
Model: "model"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 95)]	0	[]
dense (Dense)	(None, 93)	8928	['input_1[0][0]']
lambda_1 (Lambda)	(None, 90)	0	['dense[0][0]']
lambda (Lambda)	(None, 3)	0	['dense[0][0]']
dense_1 (Dense)	(None, 90)	8190	['lambda_1[0][0]']
concatenate (Concatenate)	(None, 93)	0	['lambda[0][0]', 'dense_1[0][0]']
dense_2 (Dense)	(None, 93)	8742	['concatenate[0][0]']
decoder_2_continuous (Lambda)	(None, 3)	0	['dense_2[0][0]']
decoder_2_category (Dense)	(None, 92)	8372	['lambda_1[0][0]']

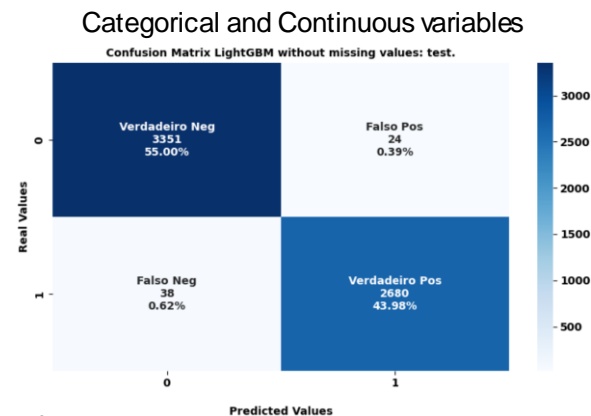
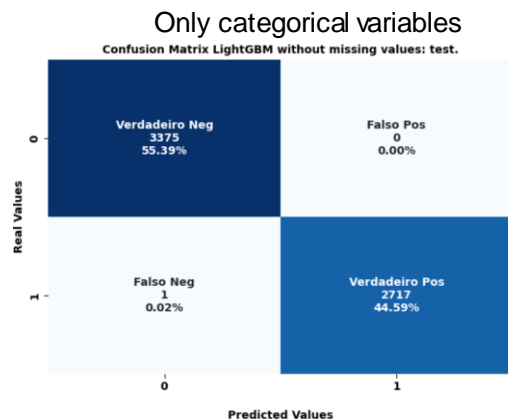
=====
Total params: 34,232
Trainable params: 34,232
Non-trainable params: 0

LightGBM without missing

- Wine:
 - Wine\2 Denoising Autoencoder.ipynb
 - Test set represent 10% of the data unseen by the model
 - Accuracy: 99%



- Mushroom (only categorical variables):
 - Mushroom\2 Denoising Autoencoder only Category.ipynb
 - Mushroom\2 Denoising Autoencoder Continuous and category.ipynb
 - Test set represent 10% of the data unseen by the model
 - Accuracy: 99%

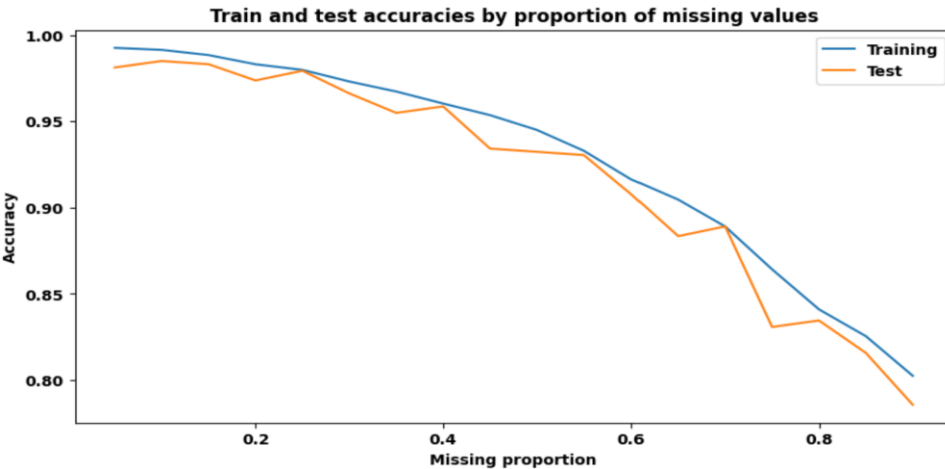


Missing Values Simulation

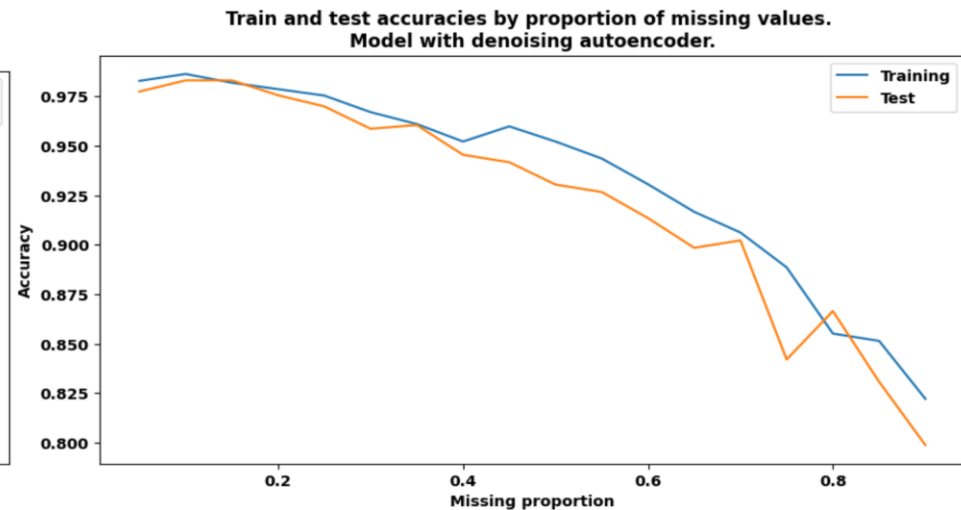
- 18 datasets for each problem were created
- Each dataset with a different proportion of randomly fabricated missing values. Proportions used: .05, .10, .15, .20, .25, .30, .35, .40, .45, .50, .55, .60, .65, .70, .75, .80, .85, .90
- For both mushroom studies, the missing values were only created in the categorical variables. First, the variables were one-hot encoded. After, each value was randomly set to zero.

Results: Wine Dataset

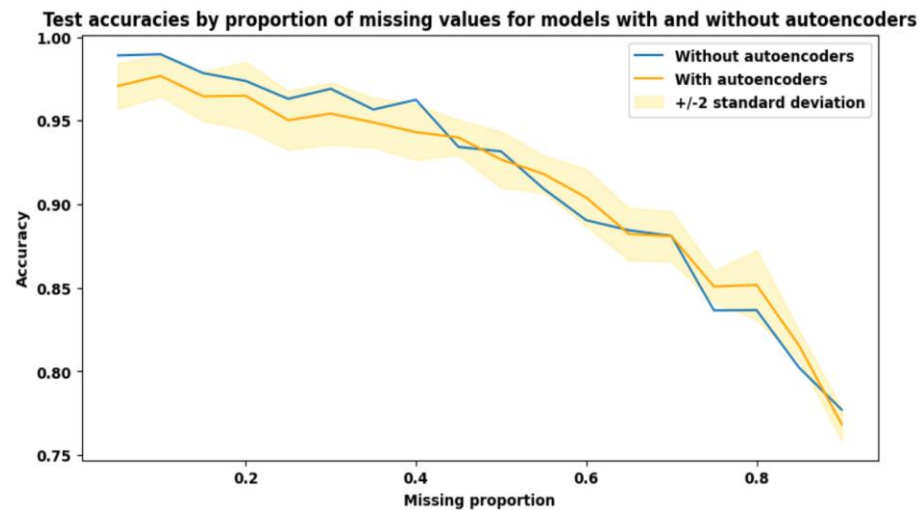
LightGBM without denoising autoencoder:
Missing values were filled with the median.



LightGBM with denoising autoencoder:



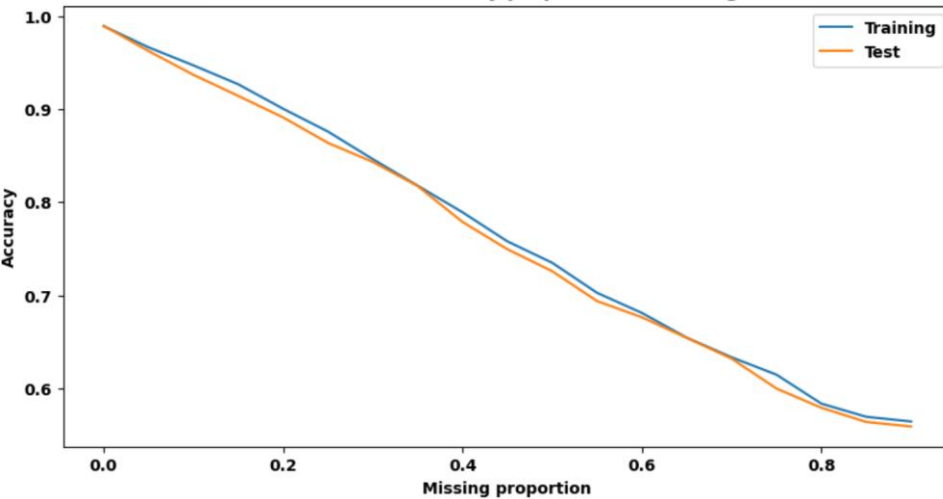
Comparison with and without autoencoder using bootstrap.



Results: Mushroom Only Categorical

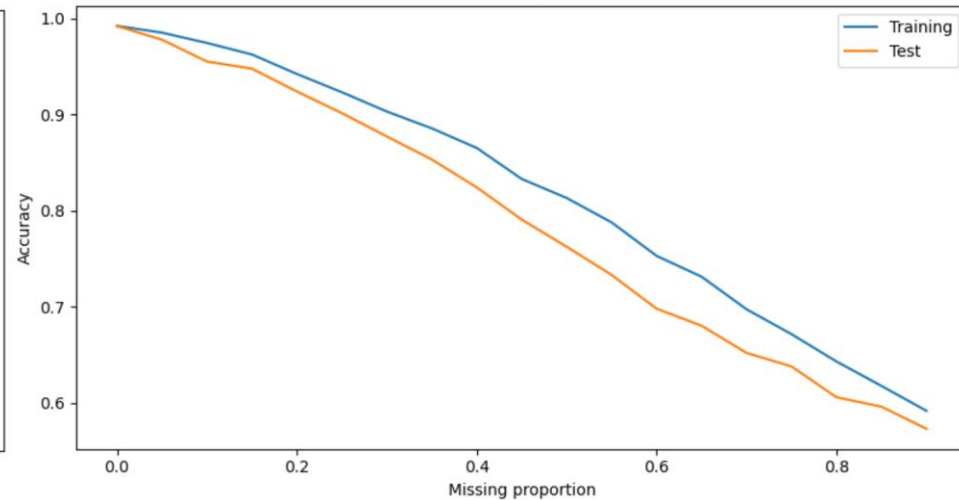
LightGBM without denoising autoencoder:
Missing values were filled with the median.

Train and test accuracies by proportion of missing values



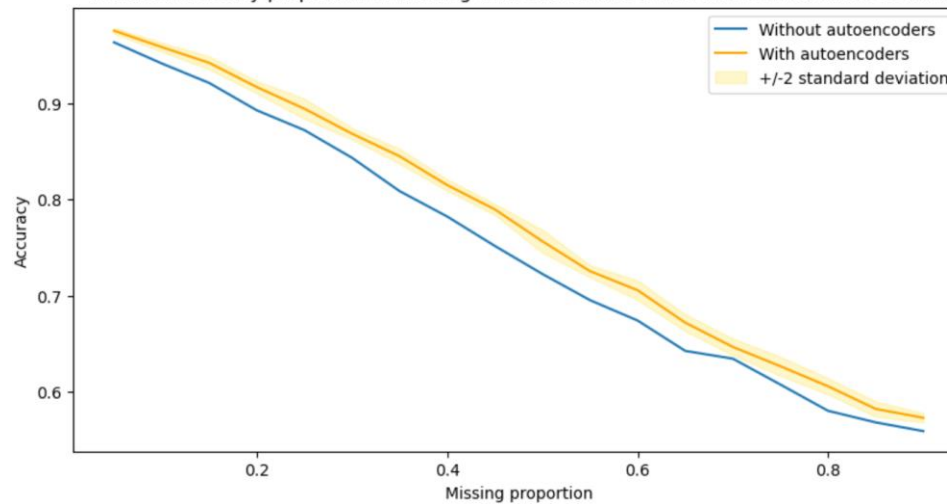
LightGBM with denoising autoencoder:

Train and test accuracies by proportion of missing values.
Model with denoising autoencoder.



Comparison with and without autoencoder.

Test accuracies by proportion of missing values for models with and without autoencoders

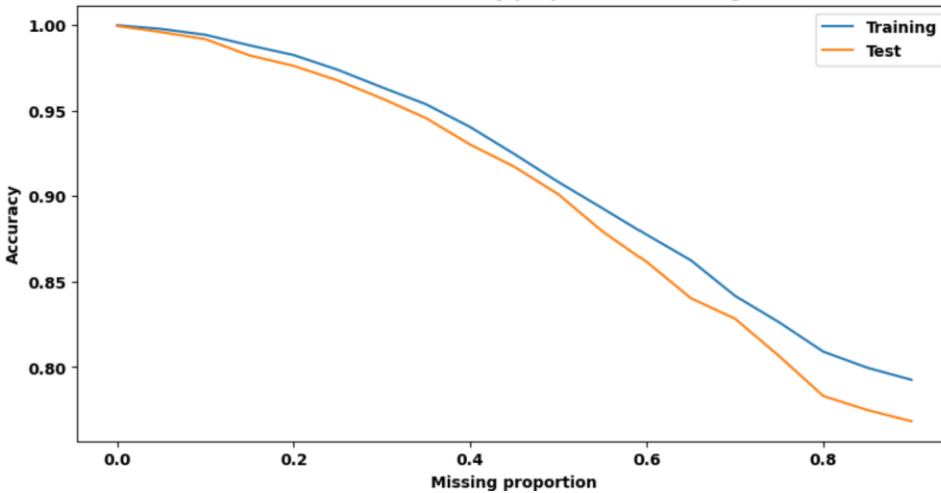


Results: Mushroom all variables

LightGBM without denoising autoencoder:

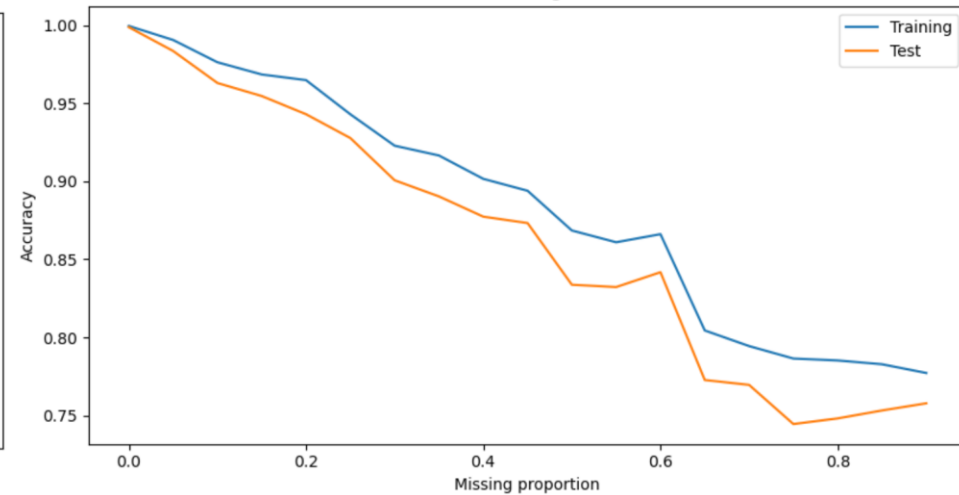
Missing values were filled with the median.

Train and test accuracies by proportion of missing values



LightGBM with denoising autoencoder:

Train and test accuracies by proportion of missing values.
Model with denoising autoencoder.



Comparison with and without autoencoder.

Test accuracies by proportion of missing values for models with and without autoencoders

