ChatGPT

Table of Contents

remember this chat as dl ass 4

4 assDL

```
import pandas as pd
import numpy as np
import pickle
import matplotlib.pyplot as plt
from scipy import stats
import tensorflow as tf
import seaborn as sns
from pylab import rcParams
from sklearn.model_selection import train_test_split
from keras.models import Model, load_model
from keras.layers import Input, Dense
from keras.callbacks import ModelCheckpoint, TensorBoard
from keras import regularizers

%matplotlib inline

sns.set(style='whitegrid', palette='muted', font_scale=1.5)

rcParams['figure.figsize'] = 14, 8

RANDOM_SEED = 42
LABELS = ["Normal", "Fraud"]



df =  pd.read_csv(r"creditcard.csv")



count_classes = pd.value_counts(df['Class'], sort = True)
count_classes.plot(kind = 'bar', rot=0)
plt.title("Transaction class distribution")
plt.xticks(range(2), LABELS)
plt.xlabel("Class")
plt.ylabel("Frequency");




from sklearn.preprocessing import StandardScaler

data = df.drop(['Time'], axis=1)

data['Amount'] =
StandardScaler().fit_transform(data['Amount'].values.reshape(-1, 1))
```

```
X_train, X_test = train_test_split(data, test_size=0.2,
random_state=RANDOM_SEED)
X_train = X_train[X_train.Class == 0]
X_train = X_train.drop(['Class'], axis=1)

y_test = X_test['Class']
X_test = X_test.drop(['Class'], axis=1)

X_train = X_train.values
X_test = X_test.values



input_dim = X_train.shape[1]
encoding_dim = 14



input_layer = Input(shape=(input_dim, ))

encoder = Dense(encoding_dim, activation="tanh",
                activity_regularizer=regularizers.l1(10e-5))
(input_layer)
encoder = Dense(int(encoding_dim / 2), activation="relu")
(encoder)

decoder = Dense(int(encoding_dim / 2), activation='tanh')
(encoder)
decoder = Dense(input_dim, activation='relu')(decoder)

autoencoder = Model(inputs=input_layer, outputs=decoder)




nb_epoch = 2
batch_size = 32

early_stop = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    min_delta=0.0001,
    patience=10,
    verbose=1,
    mode='min',
    restore_best_weights=True
)

autoencoder.compile(
    optimizer='adam',
    loss='mean_squared_error',
```

```
    metrics=['accuracy']
)

checkpointer = tf.keras.callbacks.ModelCheckpoint(
    filepath="model.keras",  # Changed to .keras
    verbose=0,
    save_best_only=True
)

tensorboard = tf.keras.callbacks.TensorBoard(
    log_dir='./logs',
    histogram_freq=0,
    write_graph=True,
    write_images=True
)

history = autoencoder.fit(
    X_train, X_train,
    epochs=nb_epoch,
    batch_size=batch_size,
    shuffle=True,
    validation_data=(X_test, X_test),
    verbose=1,
    callbacks=[checkpointer, early_stop]
).history
```

```
plt.plot(history['loss'])
plt.plot(history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right');
```

```
predictions = autoencoder.predict(X_test)
```

```
mse = np.mean(np.power(X_test - predictions, 2), axis=1)
error_df = pd.DataFrame({'reconstruction_error': mse,
                         'true_class': y_test})
```

```
error_df.describe()
```

```
threshold = 50
groups = error_df.groupby('true_class')
```

```
fig, ax = plt.subplots()

for name, group in groups:
    ax.plot(group.index, group.reconstruction_error,
marker='o', ms=3.5, linestyle='',
            label= "Fraud" if name == 1 else "Normal")
ax.hlines(threshold, ax.get_xlim()[0], ax.get_xlim()[1],
colors="r", zorder=100, label='Threshold')
ax.legend()
plt.title("Reconstruction error for different classes")
plt.ylabel("Reconstruction error")
plt.xlabel("Data point index")
plt.show();




from sklearn.metrics import
confusion_matrix,recall_score,accuracy_score,precision_sco
re




y_pred = [1 if e > threshold else 0 for e in
error_df.reconstruction_error.values]
conf_matrix = confusion_matrix(error_df.true_class, y_pred)

plt.figure(figsize=(12, 12))
sns.heatmap(conf_matrix, xticklabels=LABELS,
yticklabels=LABELS, annot=True, fmt="d");
plt.title("Confusion matrix")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()




error_df['pred'] = y_pred



# print Accuracy, precision and recall print(" Accuracy:

print("Accuracy:",accuracy_score
(error_df['true_class'],error_df['pred']))
print("
Recall:",recall_score(error_df['true_class'],error_df['pred']))
print("
Precision:",precision_score(error_df['true_class'],error_df['pr
ed']))
```

explain

This code performs credit card fraud detection using an autoencoder, a type of neural network that identifies anomalies based on reconstruction error. Here's an overview of each step:

1. **Data Loading and Visualization**: Loads the credit card dataset and visualizes the distribution of normal and fraudulent transactions using a bar chart.
2. **Data Preprocessing**:
   - Scales the 'Amount' column to standardize it.
   - Removes the 'Time' column, splits data into training and test sets, and separates the 'Class' column (labels) for evaluation. Only normal transactions are used to train the autoencoder.
3. **Autoencoder Model**:
   - Defines an autoencoder with layers for encoding (compressing) and decoding (reconstructing) the input data.
   - Compiles the model to minimize reconstruction error, optimizing for 'accuracy' and 'mean squared error'.
4. **Model Training**:
   - Trains the autoencoder using only normal transactions, with early stopping to avoid overfitting. The best model is saved, and TensorBoard is set up for logging.
5. **Performance Evaluation**:
   - Plots training and validation loss to check model convergence.
   - Predicts on the test data, calculates reconstruction error, and sets a threshold to classify transactions as fraud or normal.
6. **Confusion Matrix and Metrics**:
   - Computes a confusion matrix and visualizes it.
   - Calculates and displays accuracy, recall, and precision to assess the model's performance in identifying fraudulent transactions.

This process allows the autoencoder to learn the normal pattern of transactions and detect deviations, which likely indicate fraud.

explain code in detail

This code creates an anomaly detection model using an autoencoder to identify fraudulent credit card transactions. Here's a detailed breakdown of each part:

1. Import Libraries and Set Up Configuration

```python
import pandas as pd
import numpy as np
import pickle
import matplotlib.pyplot as plt
from scipy import stats
import tensorflow as tf
import seaborn as sns
from pylab import rcParams
from sklearn.model_selection import train_test_split
from keras.models import Model, load_model
from keras.layers import Input, Dense
from keras.callbacks import ModelCheckpoint, TensorBoard
from keras import regularizers
```

- Imports essential libraries for data handling, visualization, machine learning, and neural networks.
- `seaborn`, `matplotlib`, and `rcParams` are used to style plots and set figure dimensions.
- `tensorflow`, `keras` provide tools to build and train the neural network.

```python
%matplotlib inline
sns.set(style='whitegrid', palette='muted', font_scale=1.5)
rcParams['figure.figsize'] = 14, 8
RANDOM_SEED = 42
LABELS = ["Normal", "Fraud"]
```

- Sets the plotting style, figure size, and a random seed for reproducibility. `LABELS` are used to label the classes.

2. Data Loading and Initial Exploration

```python
df = pd.read_csv(r"creditcard.csv")
count_classes = pd.value_counts(df['Class'], sort=True)
count_classes.plot(kind='bar', rot=0)
plt.title("Transaction class distribution")
```

```
plt.xticks(range(2), LABELS)
plt.xlabel("Class")
plt.ylabel("Frequency")
```

- Loads the credit card dataset and visualizes the number of normal and fraudulent transactions in a bar plot.

3. Data Preprocessing

```python
from sklearn.preprocessing import StandardScaler
data = df.drop(['Time'], axis=1)
data['Amount'] = StandardScaler().fit_transform(data['Amount'].values.reshape(-1, 1))
```

- Drops the `Time` column since it's not relevant for anomaly detection.
- Scales the `Amount` column to ensure that all features are on a similar scale, which helps the neural network converge more efficiently.

4. Splitting Data for Training and Testing

```python
X_train, X_test = train_test_split(data, test_size=0.2, random_state=RANDOM_SEED)
X_train = X_train[X_train.Class == 0]
X_train = X_train.drop(['Class'], axis=1)
y_test = X_test['Class']
X_test = X_test.drop(['Class'], axis=1)
X_train = X_train.values
X_test = X_test.values
```

- Splits data into training and testing sets.
- The autoencoder is trained only on normal transactions (`Class == 0`), so we remove fraudulent transactions from the training set.
- Separates labels (`Class`) from features for test data to evaluate model performance later.

5. Building the Autoencoder Model

```python
input_dim = X_train.shape[1]
encoding_dim = 14
input_layer = Input(shape=(input_dim,))
encoder = Dense(encoding_dim, activation="tanh",
activity_regularizer=regularizers.l1(10e-5))(input_layer)
encoder = Dense(int(encoding_dim / 2), activation="relu")(encoder)
decoder = Dense(int(encoding_dim / 2), activation='tanh')(encoder)
decoder = Dense(input_dim, activation='relu')(decoder)
autoencoder = Model(inputs=input_layer, outputs=decoder)
```

- Defines an autoencoder architecture:

- **Encoder**: Compresses the input to a lower-dimensional representation using two dense layers with `tanh` and `relu` activations.
- **Decoder**: Reconstructs the input data from the compressed representation.

- `activity_regularizer` penalizes complex representations, encouraging a simpler model.

6. Model Compilation and Training

```python
nb_epoch = 2
batch_size = 32
early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=0.0001,
patience=10, verbose=1, mode='min', restore_best_weights=True)
autoencoder.compile(optimizer='adam', loss='mean_squared_error', metrics=
['accuracy'])
checkpointer = tf.keras.callbacks.ModelCheckpoint(filepath="model.keras", verbose=0,
save_best_only=True)
tensorboard = tf.keras.callbacks.TensorBoard(log_dir='./logs', histogram_freq=0,
write_graph=True, write_images=True)
```

- Sets hyperparameters for training, such as epochs and batch size.
- Adds callbacks:
  - **EarlyStopping** to stop training when validation loss stops improving.
  - **ModelCheckpoint** to save the best model.
  - **TensorBoard** for logging.
- Compiles the autoencoder with `adam` optimizer and mean squared error loss function.

```python
history = autoencoder.fit(X_train, X_train, epochs=nb_epoch, batch_size=batch_size,
shuffle=True, validation_data=(X_test, X_test), verbose=1, callbacks=[checkpointer,
early_stop]).history
```

- Trains the autoencoder with training data as both input and output (since it's unsupervised learning).
- Uses validation data to monitor performance and avoid overfitting.

7. Plot Training and Validation Loss

```python
plt.plot(history['loss'])
plt.plot(history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right')
```

- Plots training and validation loss over epochs to visualize model convergence.

8. Evaluate Model with Reconstruction Error

```python
predictions = autoencoder.predict(X_test)
mse = np.mean(np.power(X_test - predictions, 2), axis=1)
error_df = pd.DataFrame({'reconstruction_error': mse, 'true_class': y_test})
error_df.describe()
```

- Computes reconstruction error for each test instance. Higher errors indicate anomalies (potential fraud).
- Stores errors and true class labels in `error_df` for further analysis.

9. Set Threshold and Visualize Reconstruction Error

```python
threshold = 50
groups = error_df.groupby('true_class')
fig, ax = plt.subplots()
for name, group in groups:
    ax.plot(group.index, group.reconstruction_error, marker='o', ms=3.5, linestyle='',
label= "Fraud" if name == 1 else "Normal")
ax.hlines(threshold, ax.get_xlim()[0], ax.get_xlim()[1], colors="r", zorder=100,
label='Threshold')
ax.legend()
plt.title("Reconstruction error for different classes")
plt.ylabel("Reconstruction error")
plt.xlabel("Data point index")
plt.show()
```

- Sets a threshold to classify transactions as fraud if their reconstruction error exceeds 50.
- Plots reconstruction error to help visually identify which transactions fall above the threshold.

10. Confusion Matrix and Performance Metrics

```python
from sklearn.metrics import confusion_matrix, recall_score, accuracy_score,
precision_score
y_pred = [1 if e > threshold else 0 for e in error_df.reconstruction_error.values]
conf_matrix = confusion_matrix(error_df.true_class, y_pred)
plt.figure(figsize=(12, 12))
sns.heatmap(conf_matrix, xticklabels=LABELS, yticklabels=LABELS, annot=True, fmt="d")
plt.title("Confusion matrix")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
```

- Creates a confusion matrix to assess the model's performance in identifying fraudulent and normal transactions.

```python
error_df['pred'] = y_pred
print("Accuracy:", accuracy_score(error_df['true_class'], error_df['pred']))
print("Recall:", recall_score(error_df['true_class'], error_df['pred']))
print("Precision:", precision_score(error_df['true_class'], error_df['pred']))
```

- Calculates accuracy, recall, and precision to measure the model's effectiveness at fraud detection.

This pipeline trains an autoencoder to reconstruct normal transactions, with reconstruction error used to identify potentially fraudulent ones.

explain each line in detail

Here's a line-by-line explanation of the code:

---

1. Importing Required Libraries

```python
import pandas as pd
import numpy as np
import pickle
import matplotlib.pyplot as plt
from scipy import stats
import tensorflow as tf
import seaborn as sns
from pylab import rcParams
from sklearn.model_selection import train_test_split
from keras.models import Model, load_model
from keras.layers import Input, Dense
from keras.callbacks import ModelCheckpoint, TensorBoard
from keras import regularizers
```

- `pandas`: For data manipulation and analysis. It provides data structures and operations for manipulating numerical tables and time series.
- `numpy`: For numerical computations, providing support for large multi-dimensional arrays and matrices.
- `pickle`: Used for saving the trained model to disk (though not directly used here, it's imported possibly for model saving).
- `matplotlib.pyplot`: For data visualization, especially for creating plots.
- `scipy.stats`: For statistical functions; here, it's imported but not used directly.

- `tensorflow` **and** `keras` : For creating and training neural networks. `keras` is now part of TensorFlow.
- `seaborn` : For advanced data visualization, offering more aesthetically pleasing graphics than `matplotlib` .
- `pylab.rcParams` : Allows customization of plots, such as setting figure size.
- `sklearn.model_selection.train_test_split` : Used to split data into training and testing sets.
- `keras.models` **and** `keras.layers` : These are essential for defining and training the autoencoder model.
- `keras.callbacks` : Provides utility functions for saving the model's best state and managing training.

2. Plotting Configuration

```python
%matplotlib inline
sns.set(style='whitegrid', palette='muted', font_scale=1.5)
rcParams['figure.figsize'] = 14, 8
RANDOM_SEED = 42
LABELS = ["Normal", "Fraud"]
```

- `%matplotlib inline` : A Jupyter Notebook command that allows inline plotting within the notebook.
- `sns.set` : Configures `seaborn` for a specific plot style. Here, a `whitegrid` style is used with a muted color palette and increased font scale.
- `rcParams['figure.figsize']` : Sets the default size for all plots.
- `RANDOM_SEED` : Setting a seed ensures reproducible results.
- `LABELS` : Defines labels for the two classes in the dataset: Normal and Fraud.

3. Loading and Visualizing Data

```python
df = pd.read_csv(r"creditcard.csv")
count_classes = pd.value_counts(df['Class'], sort=True)
count_classes.plot(kind='bar', rot=0)
plt.title("Transaction class distribution")
plt.xticks(range(2), LABELS)
plt.xlabel("Class")
plt.ylabel("Frequency")
```

- `pd.read_csv` : Loads the credit card transaction dataset from a CSV file.
- `pd.value_counts` : Counts the occurrences of each class (0 for normal and 1 for fraud).
- `count_classes.plot` : Creates a bar plot to visualize the class distribution.
- `plt.title` : Sets the title for the plot.
- `plt.xticks` : Labels the x-axis values with class names.

- `plt.xlabel` and `plt.ylabel`: Label the x-axis and y-axis, respectively.

4. Data Preprocessing

```python
from sklearn.preprocessing import StandardScaler
data = df.drop(['Time'], axis=1)
data['Amount'] = StandardScaler().fit_transform(data['Amount'].values.reshape(-1, 1))
```

- `StandardScaler`: Scales the data to have a mean of 0 and standard deviation of 1.
- `df.drop`: Drops the `Time` column from the dataset, as it's not useful for anomaly detection.
- `StandardScaler().fit_transform`: Scales the `Amount` column. `reshape(-1, 1)` reshapes the `Amount` values into a column vector.

5. Splitting Data into Training and Testing Sets

```python
X_train, X_test = train_test_split(data, test_size=0.2, random_state=RANDOM_SEED)
X_train = X_train[X_train.Class == 0]
X_train = X_train.drop(['Class'], axis=1)
y_test = X_test['Class']
X_test = X_test.drop(['Class'], axis=1)
X_train = X_train.values
X_test = X_test.values
```

- `train_test_split`: Splits the dataset into training and testing sets, with 20% of the data for testing.
- `X_train[X_train.Class == 0]`: Keeps only normal transactions in the training set.
- `X_train.drop`: Drops the `Class` column from the training set since it's unsupervised.
- `y_test` and `X_test.drop`: Separates the `Class` column for evaluation and drops it from features.
- `X_train.values` and `X_test.values`: Converts data to numpy arrays for TensorFlow.

6. Autoencoder Architecture

```python
input_dim = X_train.shape[1]
encoding_dim = 14
input_layer = Input(shape=(input_dim,))
encoder = Dense(encoding_dim, activation="tanh",
activity_regularizer=regularizers.l1(10e-5))(input_layer)
encoder = Dense(int(encoding_dim / 2), activation="relu")(encoder)
decoder = Dense(int(encoding_dim / 2), activation='tanh')(encoder)
decoder = Dense(input_dim, activation='relu')(decoder)
autoencoder = Model(inputs=input_layer, outputs=decoder)
```

- `input_dim`: Sets the number of input features based on the shape of `X_train`.

- `encoding_dim` : Defines the size of the compressed representation in the middle layer.
- `Input(shape=(input_dim,))` : Specifies the input layer with `input_dim` features.
- `Dense(encoding_dim, activation="tanh", activity_regularizer=regularizers.l1(10e-5))` : First encoder layer with `tanh` activation and regularization.
- `Dense(int(encoding_dim / 2), activation="relu")` : Second encoder layer with `relu` activation.
- `decoder` **layers**: Decodes the compressed data back to the original feature size.
- `autoencoder` : Combines the encoder and decoder into an autoencoder model.

7. Model Compilation and Callbacks

```python
nb_epoch = 2
batch_size = 32
early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=0.0001,
patience=10, verbose=1, mode='min', restore_best_weights=True)
autoencoder.compile(optimizer='adam', loss='mean_squared_error', metrics=
['accuracy'])
checkpointer = tf.keras.callbacks.ModelCheckpoint(filepath="model.keras", verbose=0,
save_best_only=True)
tensorboard = tf.keras.callbacks.TensorBoard(log_dir='./logs', histogram_freq=0,
write_graph=True, write_images=True)
```

- `nb_epoch` **and** `batch_size` : Set the number of epochs and batch size.
- `early_stop` : Stops training if validation loss doesn't improve for 10 epochs.
- `autoencoder.compile` : Compiles the model with Adam optimizer and mean squared error loss.
- `checkpointer` : Saves the best model during training.
- `tensorboard` : Logs data for visualization in TensorBoard.

8. Training the Model

```python
history = autoencoder.fit(X_train, X_train, epochs=nb_epoch, batch_size=batch_size,
shuffle=True, validation_data=(X_test, X_test), verbose=1, callbacks=[checkpointer,
early_stop]).history
```

- `autoencoder.fit` : Trains the autoencoder by using training data as both input and output, as it's an unsupervised approach.
- `history` : Stores training and validation loss for plotting later.

9. Plotting Loss Curves

```python
plt.plot(history['loss'])
plt.plot(history['val_loss'])
plt.title('model loss')
```

```
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right')
```

- Plots training and validation loss over epochs, showing the model's learning progress.

10. Predicting and Calculating Reconstruction Error

```python
predictions = autoencoder.predict(X_test)
mse = np.mean(np.power(X_test - predictions, 2), axis=1)
error_df = pd.DataFrame({'reconstruction_error': mse, 'true_class': y_test})
error
```