Final Presentation

Member 1, Member 2, Member 3, Member 4

Introduction to project

Credit Risk Dataset:

People are classified by good/bad credit risk

Using this dataset to train models for loan application assessment

Why this specific dataset?

Relation to problem statement

Dataset consists of:

- Age
- Credit History/Amount
- Housing Situation

Models used:

- One-layer logistical regression model
- Single hidden layer neural network
- Multi hidden-layered deep neural network

Problem statement

- Importance of precision in risk assessment for loans
- Failure of traditional credit scoring methods in capturing complex relationships between applicant attributes and loan outcomes
- Looking for a more efficient and effective way to automatically assess loan applications
- Accurate classification of acceptance and rejection to aid institutions in more reliable and faster money lending decisions
- Logistic Regression
- Shallow Neural Network
- Deep Neural Network

Experiment Design: Research Questions

- How does the performance of a deep neural network compare to traditional machine learning models?
- How does the performance of a deep neural network compare to shallow neural networks?

Task assignments among team members

- **Member 1**
  - Dataset preprocessing
  - Review of Google Colab project
- Member 2
  - Model building
  - Model tuning
- Member 3
  - Introduction/Problem statement
  - Experiment design
- Member 4
  - Model evaluation, finalization, & conclusion

Experiment Design: Hardware/Software

- Hardware:
  - T4 GPU
- Software:
  - Google Colab
  - Python
  - TensorFlow and scikit-learn
  - Numpy, pandas, matplotlib

Experiment Design: Dataset

- Data collection:
  - Sourced from OpenML and published by Dr. Hans Hofmann in 1994
  - Classifies people described by a set of attributes as either good/bad credit risks for loans
  - Simulates real credit approval scenarios
- Dataset:
  - 21 columns (20 attributes, 1 label)
  - 1000 entities
  - 2 label classes (good/bad)
  - No missing or null values
  - 7 numeric columns, 5 ordinal category columns, 8 one-hot category columns

Data pre-processing methods

- Split into train/test sets
  - Uses stratified sampling
  - 80:20 ratio
- New ratio columns
  - Monthly credit burden
  - Installment per credit
  - Dependents per credit

Data pre-processing methods

- Numerical columns are replaced with their logs and are standardized
  - duration, credit_amount, installment_commitment, residence_since, age, existing_credits, num_dependents, monthly_credit_burden, installment_per_credit, dependents_per_credit
- Categorical columns with ordered categories are converted into ordinal columns
  - This is also applied to binary categorical columns
  - checking_status, savings_status, employment, own_telephone, foreign_worker
- Categorical columns with independent categories are one-hot encoded
  - job, housing, other_payment_plans, property_magnitude, other_parties, personal_status, credit_history, purpose
- Resulting dataset has 51 attributes, all of which are 64-bit floats
  - The right skew of the original numerical columns were also fixed

Selected Evaluation Metrics

- Models will be compared using accuracy, precision, recall, and F1-score
- Confusion matrices will be shown to better illustrate the classification performance of each model
    - Ideally, the models should prefer to avoid false positives over false negatives

Experiment Procedure

1. Download the dataset from OpenML
   - The dataset is downloaded as an ARFF file
   - The new file is then loaded into memory and converted into a Pandas dataframe
2. Preprocess the dataset
   - Address issues in the data, like right skew in numerical column distributions
   - Encode categorical columns
   - Compute new columns through feature engineering
3. Define tuners for each type of model (log. regression, shallow NN, deep NN)
   - Tuners will be kept as similar as possible
   - Log. Regression model is a sk-learn model, so it uses GridSearchCV instead of the Keras tuner RandomSearch
4. Observe the resulting models and their performance
   - Confusion matrices will be used in addition to other metrics

Experiment Procedure

```python
from pathlib import Path
import pandas as pd
from scipy.io import arff
import tarfile
import urllib.request

def load_dataset():
    file_path = Path("dataset_31_credit-g.arff")
    if not file_path.is_file():
        Path("datasets").mkdir(parents=True, exist_ok=True)
        url = "https://www.openml.org/data/download/31/dataset_31_credit-g.arff"
        urllib.request.urlretrieve(url, file_path)
    arff_file = arff.loadarff(file_path)

    return pd.DataFrame(arff_file[0])

default_dataset = load_dataset ()
```

```python
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

# Logistic Regression with GridSearchCV
log_reg = LogisticRegression(max_iter=1000)

param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],
    'solver': ['liblinear', 'lbfgs']
}

grid = GridSearchCV(log_reg, param_grid, cv=5)
grid.fit(X_train, y_train)

print("Best parameters for Logistic Regression:", grid.best_params_)

# Evaluation
y_pred_lr = grid.predict(X_test)

cm_lr = confusion_matrix(y_test, y_pred_lr)
ConfusionMatrixDisplay(cm_lr).plot()
plt.title("Confusion Matrix - Logistic Regression")
plt.show()

print(classification_report(y_test, y_pred_lr))
```

```python
def build_shallow_model(hp):
    model = keras.Sequential()
    model.add(layers.Input(shape=(X_train.shape[1],)))

    # Tune number of neurons
    neurons = hp.Int('units', min_value=32, max_value=512, step=16)
    activation = hp.Choice('activation', ['relu', 'tanh', 'sigmoid'])

    model.add(layers.Dense(neurons, activation=activation))

    model.add(layers.Dense(1, activation='sigmoid'))  # Binary classification output

    # Tune optimizer
    optimizer = hp.Choice('optimizer', ['adam', 'rmsprop', 'sgd'])

    model.compile(
        optimizer=optimizer,
        loss='binary_crossentropy',
        metrics=['accuracy']
    )
    return model

tuner_shallow = RandomSearch(
    build_shallow_model,
    objective='val_accuracy',
    max_trials=10,
    directory='shallow_nn_tuner',
    project_name='shallow_nn'
)

tuner_shallow.search(X_train, y_train, epochs=10, validation_split=0.2)

best_shallow_model = tuner_shallow.get_best_models(num_models=1)[0]

# Evaluate
y_pred_shallow = (best_shallow_model.predict(X_test) > 0.5).astype("int32")

cm_shallow = confusion_matrix(y_test, y_pred_shallow)
ConfusionMatrixDisplay(cm_shallow).plot()
plt.title("Confusion Matrix - Shallow Neural Network")
plt.show()

print(classification_report(y_test, y_pred_shallow))
```

Results and Discussions

- The end product of our experiment was very interesting, lets get into it!

Results Overview

- Models Evaluated:
- - Logistic Regression
- - Shallow Neural Network (SNN)
- - Deep Neural Network (DNN)
- Dataset: 1,000 instances (70% good, 30% bad), 80/20 train-test split
- Evaluation Metrics: Accuracy, Precision, Recall, F1-score
- Focus: Recall for 'bad' credit (minority class)

Confusion Matrices (Test Set)

Model	True Neg (Bad)	False Pos	False Neg	True Pos (Good)
Logistic Regression	22	38	13	127
Shallow Neural Network	23	37	15	125
Deep Neural Network	31	29	18	122

Evaluation Metrics Summary

Metric	Logistic Regression	Shallow NN	Deep NN
Accuracy	0.74	0.74	0.77
Precision (Bad)	0.37	0.38	0.52
Recall (Bad)	0.37	0.38	0.52
F1-score (Bad)	0.37	0.38	0.52
Recall (Good)	0.91	0.89	0.87
F1-score (Overall)	0.71	0.71	0.76

RQ-1: Deep NN vs. Logistic Regression

- DNN outperforms logistic regression in accuracy and minority class detection.
- - Accuracy: DNN (0.77) vs. Logistic Regression (0.74)
- - Recall for 'Bad' credit: DNN (0.52) vs. Logistic Regression (0.37)
- - Logistic Regression has higher recall for 'Good' credit (0.91), but lower for 'Bad'.
- DNN provides more balanced and real-world applicable performance.

RQ-2: Deep NN vs. Shallow NN

- Shallow NN slightly better than logistic regression, but DNN clearly stronger.
- - Recall for 'Bad': SNN (0.38) vs. DNN (0.52)
- - F1-score Overall: SNN (0.71) vs. DNN (0.76)
- DNN's deeper architecture helps capture more complex patterns.

Final Takeaways

- Logistic Regression: Strong baseline, but biased toward majority class.
- Shallow NN: Improved flexibility, but limited in recall.
- Deep NN: Best overall performance — balanced recall and accuracy.
- Conclusion: DNN is most effective model for identifying credit risk in this dataset.

A Demo

- You can run your code and show your results in real-time

END