Workshop: Not all AI is an LLM

Table of Contents

Introduction.

The aim of workshop is to train, evaluate and compare few different simple ML models to make predictions based on text input.

Steps to go through:

- Data preparation
- Model training
- Model evaluation
- Model selection

Models to go through:

- 'Classical' statistical models – random forest, support vector machine etc.
- Basic (simplest) neural networks: multi-layer perceptron network.
- Advanced neural networks: Long-Short Term Memory (LSTM) network.
- How good ML models in comparison with GPT?

Requirements:

- Any python-compatible IDE – VS Code, PyCharm, Spider etc.

Conventions :

	A short task that expected to be done by oneself. Mostly all tasks are expected to be done in succession
	Short hints, suggestions or code snippets to help complete the task.

Project setup.

Repo link: (https://github.com/ArturasKHyarchis/workshop)

Download the repo, install packages from requirements.txt – run in terminal

```
pip install -r requirements.txt
```

Project structure:
- data – folder with example dataset. Dataset source:
https://huggingface.co/datasets/mltrev23/financial-sentiment-analysis

- src – folder with skeleton scripts, to complete during the workshop

- sandbox – folder with code examples

Dataset preparation.

During the model training process three different datasets are used:

- Train data set. Usage – compute model parameters/ weights
- Validation set. Usage – initial model evaluation and calibrate/ fine-tune model
- Test set. Usage – final model evaluation

Ideally, these datasets should be collected independently. However, usually there is only one data source, and only one initial dataset exists. To create three datasets random splits of initial dataset is used.

General rules:

- Datasets do not overlap!
- Train set – largest. Test set – smallest (or the same size as validation set).

	Task 1. Split example dataset into three parts – train, validation (val) and test. Save each set as a separate file.
	`import pandas as pd` `df = pd.read_csv(data_path)` `test_data = df.sample(frac=0.2)` `data = df.drop(test_data.index)` `val_data = data.sample(frac=0.25)` `train_data = data.drop(val_data.index)` `test_data.to_csv(test_data_path, index=False)` `val_data.to_csv(val_data_path, index=False)` `train_data.to_csv(train_data_path, index=False)`

Model training

General info

Model training consists of several phases:

- Data preprocessing and feature extraction
- Model definition and parameter fitting
- Model evaluation and hyperparameters tweaking

Statistical models. Sklearn.

Main text vectorization options:

- TfidfVectorizer (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html)
- CountVectorizer (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html)

Main classification algorithms:

- RandomForestClassifier (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html)
- LogisticRegression (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html)
- SVM (https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html)

It's possible (recommended) to join all steps in one using `Pipeline()` (https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html#sklearn.pipeline.Pipeline)

Task 2. Try at least two different combinations (text vectorization + classification algorithm) to train a model. Save (best) model.

```
from sklearn.pipeline import Pipeline
from joblib import dump

model = Pipeline([
    ('vectorizer', {define vectorizer})
    ('classifier', {define algorithm})
])

model.fit(train_df['Sentence'], train_df['Sentiment'])

_, _, _ = calculate_metrics(val_labels, val_pred, print_metrics=True)

dump(model, model_path)
```

Neural networks (NN).

Simplest neural network architecture – several (many) connected Dense layers. Model input (i.e. features) is a list of numeric values, the output (i.e. predictions) – list of confidence scores. Often, data preprocessing (i.e. converting train data into features) is done outside the model. For multiclass classification this also includes pre-processing labels.

For text vectorization could be used keras TextVectorization layer. This layer needs to be provided with vocabulary in advance! (https://www.tensorflow.org/api_docs/python/tf/keras/layers/TextVectorization)

Model structure could be defined using `keras.models.Sequential` class.

Task 3. Train a simple neural network which consists of a few layers.

```
def create_model():

    tokenizer_layer = keras.layers.TextVectorization(
        max_tokens=2000,
        pad_to_max_tokens=True,
        output_mode='tf_idf',
        encoding='utf-8',
        standardize='lower_and_strip_punctuation',
    )

    model = keras.models.Sequential(
        [
            keras.layers.Input(shape=(1,), dtype=tf.string),
            tokenizer_layer,
            keras.layers.Dense(1024, activation='relu'),
            {define any other layers}
            keras.layers.Dense(3, activation='softmax'),
        ]
    )
    model.compile(
        optimizer=Adam(learning_rate=0.01),
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )
    return model, tokenizer_layer

.............
model, tokenizer_layer = create_model()
tokenizer_layer.adapt(features)

.............
model.fit(features, labels_categorical, epochs=100, batch_size=32)
```

	Task 4. Add validation set to the training process. Add callbacks to enhance model training. Save trained model
	`early_stopping = EarlyStopping(` `  monitor='val_accuracy',` `  patience=5,` `  restore_best_weights=True,` `  mode='max'` `)` `reduce_lr = ReduceLROnPlateau(` `  monitor='val_loss',` `  factor=0.1,` `  patience=3)` `..................` `model.fit(features, labels_categorical, validation_data = (val_features, val_labels_categorical), epochs=100, batch_size=32, callbacks=[early_stopping, reduce_lr])`
	`model.export(`*`model_path`*`)` `...................` `loaded_model = tf.saved_model.load(`*`model_path`*`)` `predictions = loaded_model.serve(`*`data`*`).numpy()`

It is possible to create many different model's architecture by using different layers. All possible layers are listed here: https://www.tensorflow.org/api_docs/python/tf/keras/layers

	Task 5. Change model structure to add LSTM layer. Train model. Save it.
	`tokenizer_layer = keras.layers.TextVectorization(` `  max_tokens=2000,` `  pad_to_max_tokens=True,` `  output_mode='int',` `  encoding='utf-8',` `  standardize='lower_and_strip_punctuation',` `)` `model = keras.models.Sequential(` `  [` `  keras.layers.Input(shape=(1,), dtype=tf.string),` `  tokenizer_layer,` `  keras.layers.Embedding(input_dim=2000, output_dim=512),` `  keras.layers.LSTM(512, return_sequences=True),` `  keras.layers.Dropout(0.2),` `  keras.layers.LSTM(128, return_sequences=False),` `  keras.layers.Dropout(0.2),` `  keras.layers.Dense(3, activation='softmax'),` `  ]` `)`

Model selection

Final model selection is made based on tradeoff between different metrics. These metrics might include (but not limited to) the following:

- How accurate is the model?
- How fast is the model?
- How 'heavy' is the model? (are there any specific software/ hardware requirements to run the model?)
- How cheap is the model?
- How maintainable is the model?
- Etc.

	Task 6. Run model_compare script to examine three models saved earlier. Which one would you choose?

A comparison with GPT.

	Task 7. Employ GPT model for the same classification task.
	``` def get_response(prompt): try: completion = client.chat.completions.create(model='gpt-4o-mini', messages=[{ "role": "system", "content": "You are an expect at analyzing financial sentiments." "Please evaluate the following sentence for its financial sentiment and respond with one word: 'negative', 'neutral', or 'positive'.", }, { "role": "user", "content": prompt, }],) return completion.choices[0].message.content except Exception as e: print(f'Error processing prompt: {prompt}. Error: {e["message"]}') return 'negative' ```