Machine Learning for social sciences

Machine Learning: Session 3

William Aboucaya

Introduction

In this course, we will focus on input classification. This type of method allows:

Introduction

In this course, we will focus on input classification. This type of method allows:

- To **identify predictors** for a certain class.

Introduction

In this course, we will focus on input classification. This type of method allows:

- To **identify predictors** for a certain class.
- To **produce automated labeling** based on input data.

Introduction

In this course, we will focus on input classification. This type of method allows:

- To **identify predictors** for a certain class.
- To **produce automated labeling** based on input data.

Classification is extremely useful in cases where you do not want to predict a value but rather the probability for a datapoint to be in a category among a set of pre-defined categories[1].

[1]There are ways to produce classifications without defining them at the time of training, this will be the topic of session 5 👀

Classification tasks and algoritms

Examples:

- Spam vs. Non-spam emails
- Disease vs. Healthy diagnosis
- Sentiment: Positive, Negative, Neutral

The goal is to learn a mapping from input features X to discrete labels y.

Methods:

- Logistic Regression
- k-Nearest Neighbors
- Decision Trees
- Random Forest
- Support Vector Machines
- Neural Networks

Classification tasks and algoritms

Examples:

- Spam vs. Non-spam emails
- Disease vs. Healthy diagnosis
- Sentiment: Positive, Negative, Neutral

The goal is to learn a mapping from input features X to discrete labels y.

Methods:

- Logistic Regression
- k-Nearest Neighbors
- Decision Trees
- Random Forest
- Support Vector Machines
- Neural Networks

For this class, we will focus on **multi-label logistic regression** and **neural network** for classification.

Classical ML Example: Forest Covertypes Dataset

- The **Forest Covertypes** dataset[2] contains:
  - ∼580,000 samples
  - 54 numerical features (e.g., elevation, slope, soil type)
  - 7 forest cover classes (tree species)
- Goal: Predict the forest cover type from cartographic variables.
- Typical algorithms:
  - Decision Trees
  - Random Forests
  - Multi-class logistic regression

[2] Jock Blackard. *Covertype*. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C50K5N. 1998.

Logistic Regression for Multi-class Problems

Extends binary logistic regression to handle more than two classes (**K-class problems**). Decision boundary is formed by **K linear functions**, one per class.

Logistic Regression for Multi-class Problems

Extends binary logistic regression to handle more than two classes (**K-class problems**). Decision boundary is formed by **K linear functions**, one per class.

Strategies:

One-vs-Rest (OvR): Train K binary classifiers

Softmax Regression: Single model predicting all classes simultaneously

Logistic Regression for Multi-class Problems

Extends binary logistic regression to handle more than two classes (**K-class problems**). Decision boundary is formed by **K linear functions**, one per class.

Strategies:

One-vs-Rest (OvR): Train K binary classifiers

Softmax Regression: Single model predicting all classes simultaneously

Works best when features are numeric and classes are linearly separable in feature space

Softmax Regression

Generalization of binary logistic regression to K classes
The **softmax** function converts raw scores (**logits**) into **probabilities**:

$$P(y = k \mid x) = \frac{e^{w_k^T x}}{\sum_{j=1}^{K} e^{w_j^T x}}$$

Variable Definitions:

- x: input feature vector
- w_k: weight vector associated with class k
- $w_k^T x$: logit (unnormalized score) for class k
- K: total number of classes
- $P(y = k \mid x)$: predicted probability for class k

Produces normalized probabilities across all classes.

Cross-Entropy Loss for Multi-class Classification

Training a multi-class classifier is done by minimizing **cross-entropy loss**. This loss function measures the **difference between true labels and predicted probability distribution**.

Defined as:

$$L_{\mathsf{CE}} = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{i,k} \log P(y = k \mid x_i)$$

Variable Definitions:

- N: number of training samples
- K: number of classes
- $y_{i,k}$: indicator (1 if sample i belongs to class k, otherwise 0)

Cross-Entropy Loss for Multi-class Classification

Training a multi-class classifier is done by minimizing **cross-entropy loss**. This loss function measures the **difference between true labels and predicted probability distribution**.

Defined as:

$$L_{\mathrm{CE}} = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{i,k} \log P(y = k \mid x_i)$$

Variable Definitions:

- N: number of training samples
- K: number of classes
- $y_{i,k}$: indicator (1 if sample i belongs to class k, otherwise 0)

Interpretation:

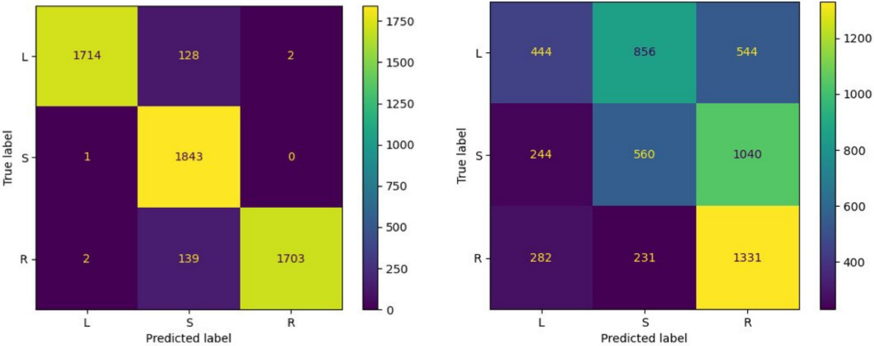
- Only the log probability of the correct class contributes.
- Loss is low if the model assigns high probability to the correct class.
- Loss grows sharply as predicted probability approaches 0.

Confusion Matrix

A very useful tool for classification evaluation: the **Confusion Matrix**.

Given a model's predictions and the actual labels for a given classification task and dataset, plot how often they are in agreement for each label.

Examples of good and bad confusion matrix[3]:



[3] Leyla Shojaeifard et al. "Left or right? Detecting driver's head movement on the road". In: *13th International Conference on the Internet of Things*. ACM, 2024.

Workflow for Numeric Multi-class Classification

1. Data preprocessing and normalization
2. Train/valid/test split
3. Train softmax logistic regression model
4. Evaluate using F1-score, precision, recall, confusion matrix

Let's dive into it 🤓

Another use case: text classification

So far, we have trained models based on purely numeric inputs (measurements, boolean values, etc.)

Another use case: text classification

So far, we have trained models based on purely numeric inputs (measurements, boolean values, etc.)

But what if we want to classify another type of input, such as text?

Another use case: text classification

So far, we have trained models based on purely numeric inputs (measurements, boolean values, etc.)

But what if we want to classify another type of input, such as text?

We need a method to produce a **numeric vector** from **a text**: that's **vectorization!**

Deep Learning Example: 20 Newsgroups Dataset

The **20 Newsgroups** dataset contains about 18,000 newsgroup documents across 20 categories.

Goal: Classify text documents into topics such as:

- Politics, Sports, Science, Religion, etc.

Each document is free text $\rightarrow$ requires **text preprocessing**.

Text vectorization

3 main methods for vectorization:

- **Bag-of-words:** Count the number of occurrences of each word in a given text without taking order into account.
- **TF-IDF:** Count the number of occurrences of each word in a given text mitigated by the number of texts in the whole corpus in which the text appears.
- **Word embeddings:** Map words as points in a vector space based on their semantics and surrounding words using a pre-trained embedding layer in a neural network.

Text vectorization

3 main methods for vectorization:

- **Bag-of-words:** Count the number of occurrences of each word in a given text without taking order into account.
- **TF-IDF:** Count the number of occurrences of each word in a given text mitigated by the number of texts in the whole corpus in which the text appears.
- **Word embeddings:** Map words as points in a vector space based on their semantics and surrounding words using a pre-trained embedding layer in a neural network.

Here, we will use word embedding vectorization, as most state-of-the-art methods rely on it for natural language processing.

Deep Learning for Text Classification

Pipeline typically includes:

- Text preprocessing (cleaning, normalization)
- Tokenization (word, subword, or character)
- Vectorization via embeddings
- Classification via neural architectures

Deep Learning for Text Classification

Pipeline typically includes:

- Text preprocessing (cleaning, normalization)
- Tokenization (word, subword, or character)
- Vectorization via embeddings
- Classification via neural architectures

Common model families:

- **RNN-based models (LSTM, GRU):** capture sequential dependencies
- **CNN-based models:** detect n-gram–like patterns in text
- **Transformers:** capture global context using self-attention

Tokenization

Transform text into a sequence of tokens:

- **Word-level**: “this class is fantastic” $\rightarrow$ [this, class, is, fantastic]
- **Subword-level**: BERT uses WordPiece ”unbelievable” $\rightarrow$ [”un”, ”##believable”]
- **Character-level**: Useful for noisy or multilingual text

Subword tokenization is widely used because it:

- Handles rare words
- Controls vocabulary size
- Works well across languages

Sequences, Padding, and Truncation

Neural networks expect **fixed-size input vectors**.

- Texts have variable length
- We choose a maximum sequence length (e.g., 128 tokens)
- Short texts $\rightarrow$ **padding** with zeros
- Long texts $\rightarrow$ **truncation**

Ensures that all input documents can be processed by the model in batches.

Word Embeddings

Transform **words** into dense, low-dimensional **numeric vectors.**

Solve limitations of one-hot encoding by **capturing semantic relationships.**

Approaches:

- **Static embeddings:** Word2Vec, GloVe
- **Contextual embeddings:** BERT, RoBERTa, GPT-based models

Word Embeddings

Transform **words** into dense, low-dimensional **numeric vectors.**

Solve limitations of one-hot encoding by **capturing semantic relationships.**

Approaches:

- **Static embeddings:** Word2Vec, GloVe
- **Contextual embeddings:** BERT, RoBERTa, GPT-based models

Properties learned include:

- Semantic similarity (e.g., king $-$ man $+$ woman $\approx$ queen)
- Syntactic patterns

Serve as input to deep learning models for improved understanding of language structure.

Transformer-based Models

Rely on **self-attention** to capture long-range dependencies without recurrence

Architecture components:

- Multi-head self-attention
- Feed-forward layers
- Positional encoding

Transformer-based Models

Rely on **self-attention** to capture long-range dependencies without recurrence

Architecture components:

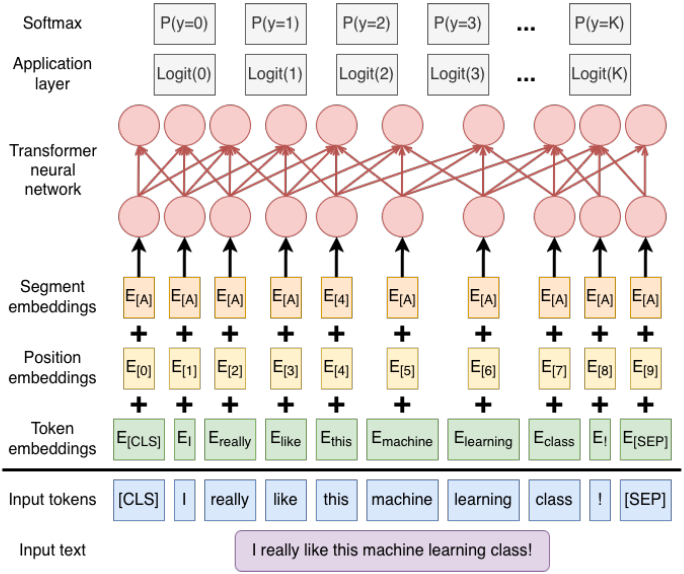
- Multi-head self-attention
- Feed-forward layers
- Positional encoding

Popular pretrained models:

- BERT (encoder-based, designed for classification)
- DistilBERT (lightweight)
- GPT-based models (autoregressive, more suitable for text generation)

Fine-tuning allows excellent performance on classification tasks with limited data

Transformer-based Model



Applying text classification

Now that we have seen how a model can be fine-tuned for a specific task, let's apply the method to the newsgroups dataset 🤓

Comparing the Two Approaches

Aspect	**Classical ML**	**Deep Learning**
Data Type	Numerical	Text / Image / Complex Data
Feature Engineering	Manual	Automated (learned)
Computation	Fast, light	Heavy, GPU needed
Interpretability	High	Lower
Example Dataset	Forest Covertypes	20 Newsgroups

Key Takeaways

Classification is about predicting discrete categories from input data.

Classical ML (e.g., logistic regressions, random forests):

- Excels with structured, tabular, numerical data.
- Usually requires careful feature engineering.
- Fast to train, interpretable, and computationally lightweight.

Key Takeaways

Classification is about predicting discrete categories from input data.

Classical ML (e.g., logistic regressions, random forests):

- Excels with structured, tabular, numerical data.
- Usually requires careful feature engineering.
- Fast to train, interpretable, and computationally lightweight.

Deep Learning (e.g., LSTMs, CNNs, Transformers):

- Learns representations directly from raw data (text, images, signals).
- Requires large datasets and GPUs but often yields state-of-the-art accuracy.
- Reduces need for manual feature engineering.

Key Takeaways

Classification is about predicting discrete categories from input data.

Classical ML (e.g., logistic regressions, random forests):

- Excels with structured, tabular, numerical data.
- Usually requires careful feature engineering.
- Fast to train, interpretable, and computationally lightweight.

Deep Learning (e.g., LSTMs, CNNs, Transformers):

- Learns representations directly from raw data (text, images, signals).
- Requires large datasets and GPUs but often yields state-of-the-art accuracy.
- Reduces need for manual feature engineering.

Practical Choice: Select the simplest model that works well for the data; prioritize interpretability vs. accuracy depending on the task.