Machine Learning Cheat sheet

https://www.linkedin.com/in/milostrifun/

1. Basic concepts: Linear regression, logistic regression, decision trees, random forests, gradient boosting, k-means clustering, PCA, neural networks

2. Popular libraries: NumPy, pandas, scikit-learn, TensorFlow, Keras

3. Data preprocessing: Handling missing data, feature scaling, encoding categorical variables, train-test split

4. Model evaluation: Metrics such as accuracy, precision, recall, F1 score, ROC AUC, k-fold cross-validation

5. Hyperparameter tuning: Grid search, random search, Bayesian optimization

6. Deep learning concepts: Convolutional neural networks, recurrent neural networks, LSTM, GRU

7. Commonly used functions: fit(), predict(), score(), confusion_matrix(), classification_report()

Basic Concepts

- **Linear Regression**: A statistical method for modeling the relationship between a dependent variable and one or more independent variables. Linear regression uses the relationship between the data points to draw a straight line through all of them.

- **Logistic Regression**: A classification algorithm that predicts a binary outcome (1 / 0, Yes / No, True / False) given a set of independent variables.

- **Decision Trees**: A flowchart-like tree structure where an internal node represents a feature(or attribute), the branch represents a decision rule, and each leaf node represents the outcome. The topmost node in a decision tree is known as the root node. It learns to partition based on the attribute value. Then, it partitions the tree recursively in a manner called recursive partitioning.

- **Random Forests**: A collection of decision trees, where each tree is trained on a random subset of the data. The final prediction is made by averaging the predictions of all the trees.

- **Gradient Boosting**: An ensemble method that trains weak models, such as decision trees, and combines them to create a more robust model. Gradient boosting works by iteratively adding models to the ensemble, where each new model attempts to correct the mistakes of the previous models.

- **K-Means Clustering**: An unsupervised learning algorithm that partitions a set of points into K clusters, where each topic belongs to the cluster with the nearest mean.

- **PCA (Principal Component Analysis)**: A technique for reducing the dimensionality of a dataset by transforming it onto a new set of orthogonal axes, known as principal components.

- **Neural Networks**: A set of algorithms modeled loosely after the human brain designed to recognize patterns. They interpret sensory data through machine perception, labeling, or raw clustering input. The ways they recognize are numerical, contained in vectors, into which all real-world data must be translated, be it images, sound, text, or time series.

Popular Libraries

- **NumPy**: A library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. Some key functions are:

    - **np.array()**: Creates an array
    - **np.arange()**: Returns evenly spaced values within a given interval
    - **np.zeros()**: Returns a new array of given shape and type, filled with zeros
    - **np.random.rand()**: Returns random floats in the half-open interval [0.0, 1.0)
    - **np.linalg.inv()**: Compute the (multiplicative) inverse of a matrix

- **Pandas**: A library providing easy-to-use data structures and data analysis tools. It's great for data wrangling and preparation. Some key functions are:

    - **pd.read_csv()**: Read a csv file and convert to a DataFrame
    - **df.head()**: Returns the first n rows of a DataFrame
    - **df.info()**: Provides the concise summary of a DataFrame
    - **df.describe()**: Generates descriptive statistics of a DataFrame
    - **df.groupby()**: Group data by one or more columns

- **Scikit-learn**: A machine learning library for Python that provides simple and efficient data mining and analysis tools. It is built on NumPy, pandas, and matplotlib. Some key functions are:

    - **from sklearn.linear_model import LinearRegression**: Import the linear regression model
    - **from sklearn.metrics import mean_squared_error**: Import the mean squared error metric
    - **from sklearn.model_selection import train_test_split**: Import the train_test_split function
    - **from sklearn.preprocessing import StandardScaler**: Import the StandardScaler function
    - **from sklearn.ensemble import RandomForestClassifier**: Import the RandomForestClassifier

- **TensorFlow**: A free and open-source software library for machine learning and neural networks. It provides a flexible ecosystem of tools, libraries, and community resources that enables researchers and developers to build, deploy, and experiment with a wide variety of models and applications. Some key functions are:

    - **tf.constant()**: Creates a constant tensor
    - **tf.Variable()**: Creates a variable tensor
    - **tf.add()**: Adds two tensors
    - **tf.nn.sigmoid()**: Computes sigmoid of x element-wise
    - **tf.train.GradientDescentOptimizer()**: Optimizer that implements the gradient descent algorithm

- **Keras**: A high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. Some key functions are:

    - **from keras.models import Sequential**: Import the Sequential model
    - **from keras.layers import Dense**: Import the Dense layer
    - **model.add()**: Add layers to the model
    - **model.compile()**: Compile the model before training
    - **model.fit()**: Fit the model to the training data
    - **model.evaluate()**: Evaluate the model on the test data
    - **model.predict()**: Make predictions on new data

Data Preprocessing

Data preprocessing is an important step in the machine learning pipeline and includes techniques to clean, transform and normalize the data so that it can be effectively used for building models.

- **Handling Missing Data**: Missing data can lead to bias and inaccurate results in the model. It's important to address missing data by either dropping the missing observations, imputing the missing values or using techniques such as mean/median/mode imputation or linear regression imputation. In python, this can be done using libraries such as pandas (**df.fillna()**, **df.dropna()**) or **sklearn.impute**.

- **Feature Scaling**: Feature scaling is used to normalize the range of independent variables or features of data. It is applied to normalize the data before applying machine learning algorithms. Some common techniques used for feature scaling are standardization and normalization. In python, this can be done using libraries such as **sklearn.preprocessing** (**StandardScaler**, **MinMaxScaler**).

- **Encoding Categorical Variables**: Categorical variables are variables that contain label values rather than numeric values. They need to be encoded as numeric variables before being used in a model. Some common techniques used for encoding categorical variables are One-Hot Encoding, Label Encoding. In python, this can be done using libraries such as **sklearn.preprocessing** (**OneHotEncoder**, **LabelEncoder**) or pandas **get_dummies()** function

- **Train-Test Split**: The data is split into a training set and a test set. The training set is used to train the model and the test set is used to evaluate the performance of the model. In python, this can be done using libraries such as **sklearn.model_selection** (**train_test_split()**)

Model Evaluation

Model evaluation is the process of assessing how well a model generalizes to an independent data set. It helps to understand how well the model is performing and identify the areas where the model needs improvement.

- **Metrics**: There are different metrics that can be used to evaluate a model's performance. Some common metrics are:
    - **Accuracy**: It is the ratio of correctly predicted observation to the total observations. It's computed as **(TP+TN)/(TP+TN+FP+FN)**
    - **Precision**: It is the ratio of correctly predicted positive observations to the total predicted positive observations. It's computed as **TP/(TP+FP)**
    - **Recall** (Sensitivity or True Positive Rate): It is the ratio of correctly predicted positive observations to all observations in actual class. It's computed as **TP/(TP+FN)**
    - **F1 Score**: It is the harmonic mean of precision and recall. The range for F1 Score is [0, 1]. It tells you how precise your classifier is (how many instances it classifies correctly), as well as how robust it is (it does not miss a significant number of instances). The greater the F1 Score, the better is the performance of our model. It's computed as **2*(Recall * Precision) / (Recall + Precision)**
    - **ROC AUC**: Receiver Operating Characteristic (ROC) curve is a plot of the true positive rate against the false positive rate. The Area Under the Curve (AUC) represents the degree or measure of separability. The higher the AUC, the better the model is at predicting 0s as 0s and 1s as 1s.
- **Cross-Validation**: Cross-validation is a technique used to evaluate a model's performance by splitting the data into training and test sets. One common method is k-fold cross-validation, where the data is split into k subsets, and the model is trained and tested k times, with a different subset as the test set each time. In python, this can be done using libraries such as **sklearn.model_selection** (**cross_val_score()**).
- **Hyperparameter tuning**: Hyperparameter tuning is the process of searching for the best set of hyperparameters for a model. This can be done by trying different combinations of hyperparameters and evaluating the model's performance using cross-validation. There are different techniques for hyperparameter tuning, such as grid search, random search, and Bayesian optimization. In python, this can be done using libraries such as **sklearn.model_selection** (**GridSearchCV**, **RandomizedSearchCV**)

Hyperparameter Tuning

Hyperparameter tuning is the process of searching for the best set of hyperparameters for a model. These parameters are not learned during the training process, unlike the model parameters. These are set before the training.

- **Grid Search**: Grid search is a technique for hyperparameter tuning where a grid of possible combinations of the hyperparameters is defined, and the model is trained and evaluated for each combination. The combination that leads to the best performance is selected as the final set of hyperparameters. In python, this can be done using **sklearn.model_selection** (**GridSearchCV**).

- **Random Search**: Random search is a technique for hyperparameter tuning where a random set of combinations of the hyperparameters is selected, and the model is trained and evaluated for each combination. The combination that leads to the best performance is selected as the final set of hyperparameters. In python, this can be done using **sklearn.model_selection** (**RandomizedSearchCV**).

- **Bayesian optimization**: Bayesian optimization is a probabilistic model-based optimization technique. This method uses Gaussian Processes to model the function and finds the best set of hyperparameters. In python, this can be done using libraries such as **scikit-optimize** or **hyperopt**.

Deep Learning Concepts

Deep learning is a subset of machine learning in which artificial neural networks, algorithms inspired by the structure and function of the brain's neural networks, are used. These models are capable of discovering hidden patterns from data.

- **Convolutional Neural Networks (CNN)**: CNNs are a class of deep, feed-forward artificial neural networks that are applied to analyzing visual imagery. They are designed to process data through multiple layers of arrays. These layers are designed to perform a specific function, such as edge detection or blurring.

- **Recurrent Neural Networks (RNN)**: RNNs are a class of artificial neural networks that are designed to process sequential data such as time series or natural language. These networks have a "memory" that can remember previous inputs, which allows them to process sequences of inputs.

- **LSTM (Long Short-Term Memory)**: LSTM is a type of RNN that is capable of learning long-term dependencies. It works by introducing a "memory cell" that can maintain its state over time, allowing the network to remember information for an extended period.

- **GRU (Gated Recurrent Unit)**: GRU is another type of RNN that is similar to LSTM but with fewer parameters, which makes it faster to train. Like LSTM, it uses gates to control the flow of information in the network, but it simplifies the gates used in LSTM by using update and reset gates, instead of input, output and forget gates.

Commonly used functions

- **fit()**: This function is used to train the model on the training data. It updates the model's parameters to minimize the loss function. For example, **model.fit(X_train, y_train)**

- **predict()**: This function is used to make predictions on new data. It returns the predicted output for the input data. For example, **y_pred = model.predict(X_test)**

- **score()**: This function is used to evaluate the model's performance on the test data. It returns a score based on the model's performance metric. For example, **accuracy = model.score(X_test, y_test)**

- **confusion_matrix()**: This function is used to compute the confusion matrix for a classification model. It returns a matrix that shows the count of true positives, true negatives, false positives, and false negatives. In python, this can be done using **sklearn.metrics** (**confusion_matrix(y_test, y_pred)**)

- **classification_report()**: This function is used to generate a report of the model's performance for a classification problem. It returns a report containing precision, recall, f1-score, and support for each class. In python, this can be done using **sklearn.metrics** (**classification_report(y_test, y_pred)**)