Week 1

Some Pointers

- Machine Learning is usually a small piece of a big project.
- Typically 10-15% of the time is spent on ML.
- A lot more time is spent in capturing and processing data for ML and taking decisions based on the ML output.

Steps in ML Project

1. Look at the big picture
2. Get the data
3. Discover and visualise the data to gain insights.
4. Prepare the data for Machine Learning Algorithms.
5. Select a model and train it.
6. Fine-tune your model.
7. Present your solution.
8. Launch, monitor and maintain your system.

Step 1: Look at the Big Picture

1. Frame the problem
    a. What are input and output?
    b. What is the business objective?
    c. What is the current solution?
2. Select a performance measure
    a. Regression
        i. Mean Squared Error (MSE)
        ii. Mean Absolute Error (MAE)
    b. Classification
        i. Precision
        ii. Recall
        iii. F1 Score
        iv. Accuracy
3. List and check the assumptions

Step 2: Get the Data

1. Check data samples
2. Understand the significance of all features

3. Data statistics
4. Create Test Set
    - Avoid data snooping bias (a *form of statistical bias manipulating data or analysis to artificially get statistically significant results***)**
    - Scikit learn provides a few functions to create test sets:
        - Random Sampling - *randomly selects k% points in the test set.*
        - Stratified Sampling - *samples test examples in such a way that they are representative of the overall distribution, avoids bias that may arise due to random sampling*

Step 3: Data Visualisation

- performed on the training set
- Standard correlation coefficient helps understand the relationship between features, visualise with heatmap

Step 4: Prepare Data for ML Algorithm

1. Separate features and labels from the training set.
2. Handling missing values and outliers
    a. Sklearn SimpleImputer Class can impute missing values

    ```
    from sklearn.impute import SimpleImputer
    imputer = SimpleImputer(strategy="median")

    imputer.fit(wine_features)
    tr_features = imputer.transform(wine_features)
    ```

3. Handle text and categorical attributes
    a. Ordinal encoder

    In ordinal encoding, each unique category value is assigned an integer value. For example, “*red*” is 1, “*green*” is 2, and “*blue*” is 3.

    ```
    from sklearn.preprocessing import OrdinalEncoder
    ordinal_encoder = OrdinalEncoder()
    ```

b. One hot encoder

```
from sklearn.preprocessing import OneHotEncoder
cat_encoder = OneHotEncoder()
```

The output is a SciPy sparse matrix rather than NumPy array. This enables us to save
space when we have a huge number of categories.
In case, we want to convert it to dense representation, we can do so with toarray() method.

4. Feature Scaling
    a. Min-max scaling or normalisation
        i. We subtract the minimum value of a feature from the current value and divide it by the difference between the minimum and the maximum value of that feature.
        ii. Values are shifted and scaled so that they range between 0 and 1.
        iii. Scikit-Learn provides *MinMaxScalar* transformer for this.
    b. Standardisation
        i. We subtract mean value of each feature from the current value and divide it by the standard deviation so that the resulting feature has a unit variance.
        ii. While normalization bounds values between 0 and 1, standardization does not bound values to a specific range.
        iii. Standardization is less affected by the outliers compared to the normalization.
        iv. Scikit-Learn provides *StandardScalar* transformation for feature standardization.

<u>Transformation Pipeline</u>

Scikit-Learn provides a Pipeline class to line up transformations in an intended order.

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
transform_pipeline = Pipeline([
          ('imputer', SimpleImputer(strategy="median")),
          ('std_scaler', StandardScaler()),])
wine_features_tr = transform_pipeline.fit_transform(wine_features)
```

- Each step in the sequence is defined by name, estimator pair.
- Each name should be unique and should not contain __ (double underscore).

<u>ColumnTransformer</u>

to transform a mix of categorical and numerical features

```
from sklearn.compose import ColumnTransformer

num_attribs = list(wine_features)
cat_attribs = ["place_of_manufacturing"]
full_pipeline = ColumnTransformer([
                    ("num", num_pipeline, num_attribs),
                    ("cat", OneHotEncoder(), cat_attribs),
                ])
wine_features_tr = full_pipeline.fit_transform(wine_features)
```

Step 5: Select and Train ML Model

1. Train and fit the model

```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(wine_features_tr, wine_labels)
```

2. Evaluate the model

```
from sklearn.metrics import mean_squared_error
quality_predictions = lin_reg.predict(wine_features_tr)
mean_squared_error(wine_labels, quality_predictions)
```

3. Cross-Validation

   Cross validation provides a separate MSE for each validation set, which we can use to get a mean estimation of MSE as well as the standard deviation, which helps us to determine how precise is the estimate.

```
from sklearn.model_selection import cross_val_score

def display_scores(scores):
  print("Scores:", scores)
  print("Mean:", scores.mean())
  print("Standard deviation:", scores.std())
```

```
scores = cross_val_score(lin_reg, wine_features_tr, wine_labels,
                          scoring="neg_mean_squared_error", cv=10)
lin_reg_mse_scores = -scores
display_scores(lin_reg_mse_scores)
```

4. Remedies for overfitting and underfitting
    a. Overfitting
        i. More data
        ii. Simpler model
        iii. More constraints/ regularisation
    b. Underfitting
        i. Model with more capacity
        ii. Less constraints/ regularisation

Step 6: Fine tune your model

Usually there are a number of hyperparameters in the model, which are set manually. Tuning these hyperparameters lead to better accuracy of ML models.

1. GridSearchCV

    We need to specify a list of hyperparameters along with the range of values to try. It automatically evaluates all possible combinations of hyperparameter values using cross-validation.

```
from sklearn.model_selection import GridSearchCV

param_grid = [
            {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
        {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3,
 4]},
            ]

grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                          scoring='neg_mean_squared_error',
                          return_train_score=True)

grid_search.fit(wine_features_tr, wine_labels)

grid_search.best_params_
```

2. RandomizedSearchCV

It selects a random value for each hyperparameter at the start of each iteration and repeats the process for the given number of random combinations.

```
from sklearn.model_selection import RandomizedSearchCV
```

Step 7: Present your Solution

Before launch,

1. We need to present our solution that highlights learnings, assumptions and systems limitation.
2. Document everything, create clear visualizations and present the model.
3. In case, the model does not work better than the experts, it may still be a good idea to launch it and free up bandwidths of human experts.

Step 8: Launch, monitor and maintain your system

- Launch
    - Plug in input sources
    - Write test cases
- Monitor
    - System outages
    - Degradation of model performance
    - Sampling predictions for human evaluation
    - Regular assessment of data quality, which is critical for model performance
- Maintain
    - Train model regularly every fixed interval with fresh data.
    - Production roll out of the model.

Sk-Learn

Types of sklearn objects

Transformers	Estimators	Predictors
• transforms dataset • `transform()` for transforming dataset. • `fit()` learns parameters. • `fit_transform()` fits parameters and `transform()` the dataset.	• Estimates model parameters based on training data and hyper parameters. • `fit()` method	• Makes prediction on dataset • `predict()` method that takes dataset as an input and returns predictions. • `score()` method to measure quality of predictions.

Data Preprocessing → Training → Inference

Sklearn APIs

Data API

Provides functionality for loading, generating and preprocessing the training and test data.

Module	Functionality
`sklearn.datasets`	Loading datasets - custom as well as popular reference dataset.
`sklearn.preprocessing`	Scaling, centering, normalization and binarization methods
`sklearn.impute`	Filling missing values
`sklearn.feature_selection`	Implements feature selection algorithms
`sklearn.feature_extraction`	Implements feature extraction from raw data.

Model API

Implements supervised and unsupervised models

Regression

- `sklearn.linear_model` (linear, ridge, lasso models)
- `sklearn.trees`

Classification

- `sklearn.linear_model`
- `sklearn.svm`
- `sklearn.trees`
- `sklearn.neighbors`
- `sklearn.naive_bayes`
- `sklearn.multiclass`

`sklearn.multioutput` implements multi-output classification and regression.

`sklearn.cluster` implements many popular clustering algorithms

Model Evaluation API

sklearn.metrics implements different APIs for model evaluation

- classification
- regression
- clustering

Model Selection API

sklearn.model_selection implements various model selection strategies like cross-validation, tuning hyper- parameters and plotting learning curves.

Model Inspection API

sklearn.model_inspection includes tools for model inspection.

Data Loading

General dataset API has three main kind of interfaces:

- The dataset **loaders** are used to **load** toy datasets bundled with sklearn.
- The dataset **fetchers** are used to **download and load** datasets from the internet.
- The dataset **generators** are used to **generate** controlled synthetic datasets.

Dataset API

Loaders

Load small standard datasets

Fetchers

Fetch and load larger datasets

Generator

Controlled synthetic datasets

return_X_y = True

Both loaders and fetchers return a Bunch object, which is a dictionary with two keys of our interest:

Key	Values
data	Array of shape (n, m)
target	Array of shape (n,)

Returns tuple $(\mathbf{X}, \mathbf{y})$ of numpy arrays:

- $\mathbf{X}$ has shape (n, m)
- $\mathbf{y}$ has shape $(n,)$

*load_**   *fetch_**   *make_**

Dataset loaders

Dataset Loader	# samples (n)	# features (m)	# labels	Type
`load_iris`	150	3	1	Classification
`load_diabetes`	442	10	1	Regression
`load_digits`	1797	64	1	Classification
`load_linnerud`	20	3	3	Regression (multi output)
`load_wine`	178	13	1	Classification
`load_breast_cancer`	569	30	1	Classification

Dataset Fetchers

Dataset Loader	# samples (n)	# features (m)	# labels	Type
`fetch_olivetti_faces`	400	4096	1 (40)	multi-class image classification
`fetch_20newsgroups`	18846	1	1 (20)	(multi-class) text classification
`fetch_lfw_people`	13233	5828	1 (5749)	(multi-class) image classification
`fetch_covtype`	581012	54	1 (7)	(multi-class) classification
`fetch_rcv1`	804414	47236	1 (103)	(multi-class) classification
`fetch_kddcup99`	4898431	41	1	(multi-class) classification
`fetch_california_housing`	20640	8	1	regression

Dataset generators

- **Regression**

  make_regression() produces regression targets as a sparse random linear combination of random features with noise. The informative features are either uncorrelated or low rank.

- **Classification**

  - **Single-label**

    make_blobs() and *make_classification()* create a bunch of normally-distributed clusters of points and then assign one or more clusters to each class thereby creating multi-class datasets.

  - **Multi-label**

    make_multilabel_classification() generates random samples with multiple labels with a specific generative process and rejection sampling.

- **Clustering**

  make_blobs() generates a bunch of normally-distributed clusters of points with specific mean and standard deviations for each cluster.

Loading external libraries

- ***fetch_openml()*** fetches datasets from openml.org, which is a public repository for machine learning data and experiments.

- ***pandas.io*** provides tools to read from common formats like CSV, excel, json, SQL.
- ***scipy.io*** specializes in binary formats used in scientific computing like .mat and .arff.
- ***numpy/ routines.io*** specializes in loading columnar data into NumPy arrays.
- ***dataset.load_files*** loads directories of text files where directory name is a label and each file is a sample.
- ***datasets.load_svmlight_files()*** loads data in svmlight and libSVM sparse format.
- ***skimage.io*** provides tools to load images and videos in numpy arrays.
- ***scipy.io.wavfile.read*** specializes reading WAV file into a numpy array.

Data transformation

Types of transformers

sklearn provides a library of transformers for

- Data cleaning (*sklearn.preprocessing*)
- Feature extraction (*sklearn.feature_extraction*)
- Feature reduction
- Feature expansion (*sklearn.kernel_approximation*)

Transformer methods

- **fit()** method learns model parameters from a training set.
- **transform()** method applies the learnt transformation to the new data.
- **fit_transform()** performs function of both fit() and transform() methods and is more convenient and efficient to use.

Transformers are combined with one another or with other estimators such as classifiers or regressors to build composite estimators.

Tool	**Usage**
Pipeline	Chaining multiple estimators to execute a fixed sequence of steps in data preprocessing and modelling.
FeatureUnion	Combines output from several transformer objects by creating a new transformer from them.
ColumnTransformer	Enables different transformations on different columns of data based on their types.