MLP Week 1 Notes

L1: Looking at the big picture

Steps in ML projects

1. Look at the big picture.
2. Get the data. → data viz.
3. Discover and visualize the data to gain insights.
4. Prepare the data for Machine Learning algorithms.
5. Select a model and train it. → EDA → Scaling, LR, Tree pipelines
6. Fine-tune your model. → hyper parameter tuning
7. Present your solution.
8. Launch, monitor and maintain your system.

- ML is usually a small piece in a big project. e.g. wine quality prediction is a small piece in setting up the manufacturing process.
- Typically 10-15% of time is spent on ML.
- A lot more time is spent on capturing and processing data needed for ML and taking decisions based on output of ML module.
- Needs strong collaboration with domain experts, product managers and eng-teams for successful execution.

- Is this a **supervised, unsupervised or a RL** problem?
- Is this a **classification, regression** or some other task?
- What is the nature of the output: **single** or **multiple** outputs?
- Does system need **continuous learning** or **periodic updates**?
- What would be the learning style: **batch** or **online**?

import pandas as pd

```
data = pd.read_csv("_____")
```

```
data.head()
```
→ 5 rows of data
→ how does it look like
→ training columns → features/attributes

```
X_train = train_df.drop('Price', axis=1)
y_train = train_df['Price']
```
→ final output

```
data.info()
```
→ info. of columns, how many total entries, empty entries

```
data.describe()
```
→ statistical attributes abt. numerical data

```
data['quality'].value_counts()
```
→ col name; 1 → 10, 2 → 20 ...

```
from sklearn.model_selection import train_test_split
X_t, X_e, Y_t, Y_e = train_test_split(X_train, Y_train, test_size = 0.2)
```

1) Data snooping ← process
- [x] Leads to biased estimation on test sets
- [x] Increases the risk of false positives → FP
- [ ] Leads to better estimation on training sets
- [ ] Reduces the risk of false positives

2) Which of the following statements are correct?
- [ ] Random sampling shuffles all of the data points and take it whole as a train set.
- [x] Random sampling randomly selects k number of samples from the data set
- [x] Stratified sampling samples test examples such that they are representative of overall distribution
- [ ] Stratified sampling samples test examples such that they are representative of the training dataset

→ any random k

3) Mention TRUE or FALSE : Stratified sampling gives us test distribution closer to the overall distribution as compared to the random sampling
- (•) TRUE
- ( ) FALSE

About Random and Stratified Sampling

sklearn
- k no. of random samples choose
- → represent overall distribution of the dataset.

L2: Data Visualization

validation, dataset, training, testing, training

- Performed on training set.
- In case of large training set -
  - Sample examples to form **exploration set**.
- Enables to understand features and their relationship among themselves and with output label.

In our case, we have a small training data and we use it all for data exploration. There is no need to create a separate exploration set.

```
data_c = data.copy()
```

It's a good idea to create a copy of the training set so that we can freely manipulate it without worrying about any manipulation in the original set.

Eg. seaborn → sns

```
sns.scatterplot(x='fixed acidity', y='density', hue='quality', data=exploration_set)
```

Eg. matplotlib as plt

```
exploration_set.plot(kind='scatter', x='fixed acidity', y='density', alpha=0.5, c="quality", cmap=plt.get_cmap("jet"))
```

- **Standard correlation coefficient** between features.
  - Ranges between -1 to +1
    - **Correlation = +1**: Strong positive correlation between features
    - **Correlation = -1**: Strong negative correlation between features
    - **Correlation = 0**: No linear correlation between features
  - Visualization with heat map
- Only captures linear relationship between features.
  - For non-linear relationship, use **rank correlation**

→ no.

```
corr_matrix['quality']
```

↓ fig.

```
plt.figure(figsize=(14,7))
sns.heatmap(corr_matrix, annot=True)
```

blue, red

1) A standard correlation coefficient value of +0.98 represents-
- (•) Strong positive correlation between features
- ( ) Weak positive correlation between features
- ( ) No correlation between features
- ( ) None of these

2) Why is it a good idea to create a copy of the training set?
- (•) So that we can freely manipulate it without altering original training data set
- ( ) So that changes in copy of the dataset are reflected in original dataset
- ( ) so that we can mix it with test dataset

3) *matplotlib* and *seaborn* are used for which of the following purposes?
- ( ) data loading
- ( ) data pre-processing
- (•) data visualization
- ( ) data manipulation

4) No linear correlation between the features is demonstrated by which of the following correlation values?
- (•) 0
- ( ) 1
- ( ) -1
- ( ) 2

5) Mention TRUE or FALSE: A heatmap can be used to visualize correlation matrix
- (•) TRUE
- ( ) FALSE

→ EDA (exploratory data analysis)

L3: Data Preparation

We often need to preprocess the data before using it for model building due to variety of reasons: (NaN)
- Due to errors in data capture, data may contain outliers or missing values.
- Different features may be at different scales.
- The current data distribution is not exactly amenable to learning.

Typical steps in data preprocessing are as follows:
1. Separate features and labels.
2. Handling missing values and outliers.
3. Feature scaling to bring all features on the same scale.
4. Applying certain transformations like log, square root on the features. (optional)

It's a good practice to make a copy of the data and apply preprocessing on that copy. This ensures that in case something goes wrong, we will at least have original copy of the data intact.

```
wine_features.isna().sum()
```
→ sum of all the nans of every column that has missing values.

dataset

- dropna()
- drop()

→ pandas (pd) → now missing values drop.

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

Do fit_transform

```
wine_features_tr = pd.DataFrame(tr_features, columns=wine_features.columns
```

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

4.5.1 Min-max scaling or Normalization

- We subtract 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.
- Values are shifted and scaled so that they range between 0 and 1.
- Scikit-Learn provides MinMaxScaler transformer for this.
- One can specify hyperparameter feature_range to specify the range of the feature.

4.5.2 Standardization

- 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.
- While *normalization* bounds values between 0 and 1, *standardization* does not bound values to a specific range.
- Standardization is less affected by the outliers compared to the normalization.
- Scikit-Learn provides StandardScalar transformation for feature standardization.
- Note that all these transformers are learnt on the **training data** and then applied on the training and test data to tranform them.
- **Never learn these transformers on the full dataset.**

```
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)
```

- The real world data has both categorical as well as numerical features and we need to apply different transformations to them.
- Scikit-Learn introduced ColumnTransformer for this purpose.

```
from sklearn.compose import ColumnTransformer
```

1) The function *.isna()* is used for which of the following purposes?
- [x] to detect missing (NaN) values
- [ ] to generate heatmap
- [ ] to generate scatter plot
- [ ] to plot standard deviation of available data in the place of NaN values

2) Which of the following methods from pandas may be used to drop rows containing missing values?
- [x] dropna()
- [x] drop()
- [ ] na_drop()
- [ ] nAn()

3) Mention TRUE or FALSE : Sklearn provides *SimpleImputer* class for filling up missing values.
- [ ] FALSE
- [x] TRUE

4) fit_transform() method on OrdinalEncoder object is called for which purpose?
- [ ] convert numbers to texts
- [x] convert text to numbers
- [ ] convert decimal to binary
- [ ] convert real numbers to rational numbers

5) Mention TRUE or FALSE: Scikit-Learn provides a Pipeline class to line up transformations in an intended order.
- [x] TRUE
- [ ] FALSE

L4: Selection and Training of ML Models

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

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

Score = ?

```
from sklearn.tree import DecisionTreeRegressor
tree_reg = DecisionTreeRegressor()
tree_reg.fit(wine_features_tr, wine_labels)
```

Note that the training error is 0, while the test error is 0.58. This is an example of an overfitted model.

We can use cross-validation (CV) for robust evaluation of model performance.

```
from sklearn.model_selection import cross_val_score
```

- 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.
- The additional cost we pay in cross validation is additional training runs, which may be too expensive in certain cases.

```
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)
```

- Random forest model builds multiple decision trees on randomly selected features and then average their predictions.
- Building a model on top of other model is called *ensemble learning*, which is often used to improve performance of ML models.

Model diagnosis	Remedy
Underfitting	Models with more capacity
	Less constraints/regularization
Overfitting	More data
	Simpler model
	More constraints/regularization

1) Which of the following can be used as an evaluation measure for regression models?
- [ ] pipeline
- [ ] confusion matrix
- [x] Mean square error
- [ ] None of the above

2) Mention TRUE or FALSE : Cross validation provides a separate MSE for each validation set.
- [x] TRUE
- [ ] FALSE

3) Building a model on top of another model is called-
- [ ] random forest
- [ ] deep learning
- [ ] stepwise learning
- [x] ensemble learning

4) What is the remedy for underfitting?
- [x] to have less constraints/regularization
- [ ] to have more constraints/regularization
- [ ] delete a significant portion of data
- [ ] None of the above

L5: Finetuning ML Models → Hyperparameter tuning (HPT)

- Usually there are a number of hyperparameters in the model, which are set manually.
- Tuning these hyperparameters lead to better accuracy of ML models.
- Finding the best combination of hyperparameters is a search problem in the space of hyperparameters, which is huge.

Grid search

- Scikit-Learn provives a class GridSearchCV that helps us in this pursuit.

```
from sklearn.model_selection import 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.

For example, there are number of hyperparameters in RandomForest regression such as:

- Number of estimators → n_estimators
- Maximum number of features → max_features

```
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)
```

Randomised Search CV → randomised search in all the parameters

```
grid_search.best_params_
```

```
grid_search.best_estimator_
```

1) The GridSearchCV class is a part of which library?
- Pillow
- Scikit-Learn
- NumPy
- BeautifulSoup

2) Which of the following are the hyperparameters in the RandomForest regression?
- [x] number of estimators
- [x] maximum number of features
- [ ] precision
- [ ] recall

} classification metrics

3) Mention TRUE or FALSE : Initializing GridSearchCV with *refit = True* option retains the best estimators on the full training set.
- TRUE
- FALSE

4) For a large hyperparameter space, it is desirable to try which of the following options-
- RandomizedSearchCV
- GridSearchCV
- MyCV
- StratifiedCV

L6: Intro to Scikit Learn

→ beginner friendly, easy to use, less memory

sklearn APIs are organized on the lines of our ML framework.

Scikit-learn	ML Framework
• Training data and preprocessing	Training data
• Model subsumes loss function and optimization procedure	Model
	Loss function
	Optimization
• Model selection and evaluation	Evaluation
• Model inspection	

→ standard principle

sklearn APIs are well designed with the following principles:

- **Consistency**: All APIs share a simple and consistent interface.
- **Inspection**: The learnable parameters as well as hyperparameters of all estimator's are accessible directly via public instance variables.
- **Nonproliferation of classes**: Datasets are represented as Numpy arrays or Scipy sparse matrix instead of pd custom designed classes.
- **Composition**: Existing building blocks are reduced as much as possible. pipeline, FU, CT
- **Sensible defaults** values are used for parameters that enables quick baseline building.

Types of sklearn objects

→ LR, Tree

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.

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.

Implements supervised and unsupervised models

Regression	Classification
• `sklearn.linear_model` (linear, ridge, lasso models) `sklearn.trees`	• `sklearn.linear_model` • `sklearn.svm` • `sklearn.trees` • `sklearn.neighbors` • `sklearn.naive_bayes` • `sklearn.multiclass`

simple / complex

`sklearn.multioutput` implements multi-output classification and regression. > 1

`sklearn.cluster` implements many popular clustering algorithms

Hint → module

```
import sklearn.linear_model import LogisticRegres
?LogisticRegression
```

? LR

1) Different types of sklearn objects are-
- [x] Estimators
- [x] Predictors
- [x] Transformers
- [ ] Random forest

2) Mark the correct statements:
- [x] predict() method takes a feature matrix as an input and returns predictions
- [x] score() method measures quality of predictions for a given test set
- [ ] fit_transform() is used on the test data
- [x] fit_transform() method fits the parameters and uses them to transform() the dataset

3) The function of sklearn.linear model is to implement-
- classical algorithms like logistic regression, linear regression etc
- the k-nearest neighbors algorithm
- multi-class classification models
- multioutput classification and regression

4) The function to implement the Naive-bayes classification is- 1 point

- sklearn.naive_Bayes
- sklearn.naive_bayes
- sklearn.naiveBayes
- sklearn.naive bayes

5) Suppose we use the following code for creating a random dataset of 10 samples with 2 features each. What is the 'y' or label vector? 1 point

```
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=10, centers=3, n_features=2, random_state=42)
```

- y = array([0,1,1,1,1,0,0,1,1,0])
- y = array([1,1,0,1,1,1,1,1,1,1])
- y = array([1,0,0,1,0,1,0,1,1,0])
- y = array([2,2,1,2,0,0,0,1,1,0])

6) Which of the following is a supervised learning problem? 2 points

- Recommending relevant product ads to a website visitor based on previous purchases or product search made on the website
- You have been given heights of different students of the class. You have to group them into three categories i.e. tall, medium and short.
- A bank looking to decide the amount of personal loan to its customers based on transactions in their account
- A robot has been placed in one corner of a grid-world with a 3x3 grid. The grid- world has an obstacle in the centre and goal in the other corner (diametrically opposite to the agent). There is a reward of +3, -1 and -10 for reaching the goal, one step in any direction and for hitting the obstacle.

7) Which of the following is a regression task? 2 points

- [ ] Recommending relevant product ads to a website visitor based on previous purchases or product search made on the website.
- [ ] You have been given heights of different students of the class. You have to group them into three categories i.e. tall, medium and short.
- [x] A bank looking to decide the amount of personal loan to its customers based on transactions in their account. It has already identified potential customers but now wants to ascertain loan amounts for each customer.
- [x] Predicting rainfall this year based on data available for the last 30 years.

8) Which one of the following statement is true? 1 point

I. A regression model that uses L1 regularization technique is called Ridge Regression
II. A regression model that uses L2 regularization technique is called Lasso Regression

- Statement I is true but II is false
- Statement II is true but I is false
- Both statements are true
- Both statements are false

9) We need to divide the sample data into train and test data because 2 points

- We do not need to train on the entire data
- The training happens faster on smaller set
- For ensuring that model has low bias and generalizes better on test/unseen data
- For ensuring the model has low variance and generalizes better on test/unseen data

10) Which function is used for fully describing the dataset?

- .shape
- .head
- .fulldescription
- .DESCR

Watch Practical Lectures of Data Loading and Demo of sklearn dataset API

1) In the iris dataset, what is the output of the following code?

```
from sklearn.datasets import load_iris
data = load_iris()
data.feature_names
```

- ['sepal_length (cm)', 'sepal_width (cm)', 'petal_length (cm)', 'petal_width (cm)']
- ['sepal_width (cm)', 'sepal_length (cm)', 'petal_width (cm)', 'petal_thickness (cm)']
- ['sepal_width (cm)', 'petal_length (cm)', 'petal_width (cm)', 'petal_thickness (cm)']
- ['petal_width (cm)', 'petal_length (cm)', 'petal_thickness (cm)', sepal_thickness (cm)]

2) With which of the following codes, we can examine the shape of the feature matrix?

- data=load_iris()
  data.shape
- data=load_iris()
  data.data.shape
- data=load_iris()
  data-data-shape
- data=load_iris()
  data-shape

3) The first four examples of the feature matrix can be accessed via which of the following codes?

- data.data[:5]
- data.data[:4]
- data.data[4:1]
- data.data[1:4]

4) Which of the following statements is true about openml.org?

- [x] It is a public repository for machine learning data and experiments
- [ ] It is a private completely restricted repository for machine learning data and experiments
- [x] openml.org allows everybody to upload open datasets.
- [ ] OpenML does not automatically analyze the data.

5) Mention TRUE or FALSE:
make_blobs enables us to generate random data for clustering.

- TRUE
- FALSE

MLP Week 2 Notes

L1: Data Preprocessing

The real world training data is usually not clean and has many issues such as missing values for certain features, features on different scales, non-numeric attributes etc.

Often there is a need to pre-process the data to make it amenable for training the model.

Sklearn provides a rich set of transformers for this job.

The same pre-processing should be applied to both training and test set.

Sklearn provides pipeline for making it easier to chain multiple transforms together and apply them uniformly across train, eval and test sets.

Once you get the training data, the first job is to explore the data and list down preprocessing needed.

Typical problems include

Missing values in features

Numerical features are not on the same scale.

Categorical attributes need to be represented with sensible numerical representation.

Too many features, reduce them.

Extract features from non-numeric data.

- Data cleaning (sklearn.preprocessing) such as standardization, missing value imputation, etc.
- Feature extraction (sklearn.feature_extraction)
- Feature reduction (sklearn.decomposition.pca)
- Feature expansion (sklearn.kernel_approximation)

Each transformer has the following 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.

`sklearn.feature_extraction` has useful APIs to extract features from data:

DictVectorizer FeatureHasher

DictVectorizer

Converts lists of mappings of feature name and feature value, into a matrix.

Original data

```
data =
[{'age': 4, 'height':96.0},
{'age': 1, 'height':73.9},
{'age': 3, 'height':88.9},
{'age': 2, 'height':81.6}]
```

Transformed feature matrix $\mathbf{X}'$

$$\mathbf{X}'_{4\times 2} = \begin{bmatrix} 4 & 96.0 \\ 1 & 73.9 \\ 3 & 88.9 \\ 2 & 81.6 \end{bmatrix}$$

```
dv = DictVectorizer(sparse=False)
df.fit_transform(data)
```

FeatureHasher

- High-speed, low-memory vectorizer that uses feature hashing technique.
- Instead of building a hash table of the features, as the vectorizers do, it applies a hash function to the features to determine their column index in sample matrices directly.
- This results in increased speed and reduced memory usage, at the expense of inspectability; the hasher does not remember what the input features looked like and has no inverse_transform method.
- Output of this transformer is scipy.sparse matrix.

- `sklearn.feature_extraction.image.*` has useful APIs to extract features from image data. Find out more about them in sklearn user guide at the following link: Feature Extraction from Images.
- `sklearn.feature_extraction.text.*` has useful APIs to extract features from text data. Find out more about them in sklearn user guide at the following link: Feature Extraction from Text.

Missing values occur due to errors in data capture such as sensor malfunctioning, measurement errors etc.

Many ML algorithms do not work with missing data and need all features to be present.

Discarding records containing missing values would result in loss of valuable training samples.

`sklearn.impute` API provides functionality to fill missing values in a dataset.

SimpleImputer KNNImputer

MissingIndicator provides indicators for missing values.

SimpleImputer

- Fills missing values with one of the following strategies: `'mean'`, `'median'`, `'most_frequent'` and `'constant'`.

Original feature matrix $\mathbf{X}$

```
si = SimpleImputer(strategy='mean')
si.fit_transform(X)
```

Transformed feature matrix $\mathbf{X}'$

$$\mathbf{X}_{4\times 2} = \begin{bmatrix} 7 & 1 \\ nan & 8 \\ 2 & nan \\ 9 & 6 \end{bmatrix} \quad \mathbf{X}'_{4\times 2} = \begin{bmatrix} 7 & 1 \\ 6 & 8 \\ 2 & 5 \\ 9 & 6 \end{bmatrix}$$

$$\frac{7+2+9}{3} = 6 \qquad \frac{1+8+6}{3} = 5$$

KNNImputer

- Uses k-nearest neighbours approach to fill missing values in a dataset.
  - The missing value of an attribute in a specific example is filled with the mean value of the same attribute of `n_neighbors` closest neighbors.
- The nearest neighbours are decided based on Euclidean distance.

Let's fill the missing value in first sample/row. $\mathbf{X}_{4\times 3} = \begin{bmatrix} 1. & 2. & nan \\ 3. & 4. & 3. \\ nan & 6. & 5. \\ 8. & 8. & 7. \end{bmatrix}$

Distance with $[1. \quad 2. \quad nan.]$

$$\begin{bmatrix} 3. & 4. & 3. \\ nan & 6. & 5. \\ 8. & 8. & 7. \end{bmatrix} \quad \begin{matrix} \sqrt{(1-3)^2+(2-4)^2} \approx 2.82 \\ \sqrt{(2-6)^2} = 4 \\ \sqrt{(1-8)^2+(2-8)^2} \approx 9.21 \end{matrix}$$

2 nearest neighbours

Values of the feature from 2 nearest neighbours

$$\frac{3+5}{2} = 4$$

# of neighbours

Original feature matrix $\mathbf{X}$

$$\mathbf{X}_{4\times 4} = \begin{bmatrix} 1. & 2. & nan. \\ 3. & 4. & 3. \\ nan & 6. & 5. \\ 8. & 8. & 7. \end{bmatrix}$$

Transformed feature matrix $\mathbf{X}'$

$$\mathbf{X}'_{4\times 4} = \begin{bmatrix} 1. & 2. & 4. \\ 3. & 4. & 3. \\ 5.5 & 6. & 5. \\ 8. & 8. & 7. \end{bmatrix}$$

```
knni = KNNImputer(n_neighbors=2, weights="uniform")
knni.fit_transform(X)
```

1) What will be the output of the following code:

```
data = [{'#courses_opted': 4, 'GPA': 6.7},{'#courses_opted': 4, 'GPA': 8.5},
    {'#courses_opted': 2, 'GPA': 8.1},{'#courses_opted': 1, 'GPA': 7.2},
    {'#courses_opted': 1, 'GPA': 9.1}]

from sklearn.feature_extraction import DictVectorizer
dv = DictVectorizer(sparse=False)
data_transformed = dv.fit_transform(data)
data_transformed[:2]
```

- [ ] [[4. 6.7]
  [4. 8.5]
  [2. 8.1]
  [1. 7.2]
  [1. 9.1]]
- [x] [[4. 6.7]
  [4. 8.5]]
- [ ] [[4. 6.7]
  [4. 8.5]
  [2., 8.1]]
- [ ] [[6.7 8.5]]

2) Which API provides functionality to fill missing values in a dataset?

- [x] sklearn.impute
- [ ] sklearn.FillMissingValue
- [ ] sklearn.extraction
- [ ] sklearn.DictVectorizer

3) Which of the following methods is used to replace the missing values of a feature by the average

- [ ] SimpleImputer(strategy='median')
- [ ] Imputer(strategy='median')
- [x] SimpleImputer(strategy='mean')
- [ ] Imputer(strategy='mean')

4) **import** numpy as np
from sklearn.impute **import** SimpleImputer
imp = SimpleImputer (missing values=np . nan , strategy=' median ')
X = [[4 , 1] , [np . nan , 5] , [8 , 0]]
imp . fit_transform (X)

What will be the missing value in X after the execution of the above code:

- [ ] 4
- [ ] 12
- [x] 6
- [ ] 5

L2: Numeric Transformers

Numerical features with different scales leads to slower convergence of iterative optimization procedures.

It is a good practice to scale numerical features so that all of them are on the same scale.

Let's learn how to scale numerical features with sklearn API.

Three feature scaling APIs are available in sklearn

StandardScaler

Transforms the original features vector $\mathbf{x}$ into new feature vector $\mathbf{x}'$ using following formula

$$\mathbf{x}' = \frac{\mathbf{x} - \mu}{\sigma}$$

Learns parameters μ and σ.

$$\mathbf{x}_{5\times1} = \begin{bmatrix}4\\3\\2\\5\\6\end{bmatrix} \rightarrow \texttt{ss = StandardScaler()}\ \texttt{ss.fit\_transform(x)} \rightarrow \mathbf{x}'_{5\times1} = \begin{bmatrix}0\\-1/\sqrt{2}\\-2/\sqrt{2}\\1/\sqrt{2}\\2/\sqrt{2}\end{bmatrix}$$

$\mu = 4, \sigma = \sqrt{2}$ $\quad\quad$ $\mu' = 0, \sigma' = 1$

Note that the transformed feature vector $\mathbf{x}'$ has mean (μ) = 0 and standard deviation (σ) = 1.

MinMaxScaler

It transforms the original feature vector $\mathbf{x}$ into new feature vector $\mathbf{x}'$ so that all values fall within range $[0,\ 1]$

$$\mathbf{x}' = \frac{\mathbf{x} - \mathbf{x}.min}{\mathbf{x}.max - \mathbf{x}.min}$$

where $\mathbf{x}.max$ and $\mathbf{x}.min$ are largest and smallest values of that feature respectively, of the original feature vector $\mathbf{x}$.

$$\mathbf{x}_{5\times1} = \begin{bmatrix}15\\2\\5\\-2\\-5\end{bmatrix} \rightarrow \texttt{mms = MinMaxScaler()}\ \texttt{mms.fit\_transform(x)} \rightarrow \mathbf{x}'_{5\times1} = \begin{bmatrix}1\\0.35\\0.5\\0.6\\0\end{bmatrix}$$

$\mathbf{x}.max$ = 15, $\mathbf{x}.min$ = -5

The largest number is transformed to 1 and the smallest number is transformed to 0.

MaxAbsScaler

It transforms the original features vector $\mathbf{x}$ into new feature vector $\mathbf{x}'$ so that all values fall within range $[-1,\ 1]$

$$\mathbf{x}' = \frac{\mathbf{x}}{\text{MaxAbsoluteValue}}$$

where $\text{MaxAbsoluteValue} = \max(\mathbf{x}.max, |\mathbf{x}.min|)$

$$\mathbf{x}_{5\times1} = \begin{bmatrix}4\\2\\5\\-2\\-100\end{bmatrix} \rightarrow \texttt{mas = MaxAbsScaler()}\ \texttt{mas.fit\_transform(x)} \rightarrow \mathbf{x}'_{5\times1} = \begin{bmatrix}0.04\\0.02\\0.05\\-0.02\\-1\end{bmatrix}$$

$\text{MaxAbsoluteValue} = \max(5, |-100|) = 100$

FunctionTransformer

Constructs transformed features by applying a user defined function.

$$\mathbf{X}_{4\times2} = \begin{bmatrix}128 & 2\\2 & 256\\4 & 1\\512 & 64\end{bmatrix} \rightarrow \texttt{ft = FunctionTransformer(numpy.log2)}\ \texttt{ft.fit\_transform(X)} \rightarrow \mathbf{X}'_{4\times2} = \begin{bmatrix}7 & 1\\1 & 8\\2 & 0\\9 & 6\end{bmatrix}$$

Polynomial transformation

Generates a new feature matrix consisting of all polynomial combinations of the features with degree less than or equal to the specified degree.

$$\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2] \rightarrow \texttt{pf=PolynomialFeatures(degree=2)}\ \texttt{pf.fit\_transform(X)} \rightarrow \mathbf{X}' = [\mathbf{x}_1,\ \mathbf{x}_2,\ \mathbf{x}_1\mathbf{x}_2, \mathbf{x}_1^2,\ \mathbf{x}_2^2]$$

degree =2

KBinsDiscretizer

- Divides a continuous variable into bins.
- One hot encoding or ordinal encoding is further applied to the bin labels.

$$\mathbf{x}_{9\times1} = \begin{bmatrix}0\\0.125\\0.25\\0.375\\0.5\\0.675\\0.75\\0.875\\1.0\end{bmatrix} \rightarrow \texttt{KBinsDiscretizer(n\_bins=5, strategy='uniform', encode = 'ordinal')} \rightarrow \mathbf{x}'_{9\times1} = \begin{bmatrix}0.\\0.\\1.\\1.\\2.\\3.\\3.\\4.\\4.\end{bmatrix}$$

1) Consider the following feature vector:

$$X = \begin{bmatrix}0\\3\\6\end{bmatrix}$$

What will be the transformed feature matrix X' after the **StandardScaler()** transformation?

- $\begin{bmatrix}0\\3/\sqrt{6}\\\sqrt{6}\end{bmatrix}$
- (selected) $\begin{bmatrix}-3/\sqrt{6}\\0\\3/\sqrt{6}\end{bmatrix}$
- $\begin{bmatrix}3/\sqrt{6}\\0\\3/\sqrt{6}\end{bmatrix}$
- $\begin{bmatrix}0\\-3/\sqrt{6}\\3/\sqrt{6}\end{bmatrix}$

2) The new feature matrix $\mathbf{x}'$ using the transformer **MinMaxScalar()** is obtained as

- $\mathbf{x}' = \frac{\mathbf{x}}{\mathbf{x}.\max - \mathbf{x}.\min}$
- $\mathbf{x}' = \frac{\mathbf{x}.\max - \mathbf{x}}{\mathbf{x}.\max - \mathbf{x}.\min}$
- (selected) $\mathbf{x}' = \frac{\mathbf{x} - \mathbf{x}.\min}{\mathbf{x}.\max - \mathbf{x}.\min}$
- $\mathbf{x}' = \frac{\mathbf{x}}{\mathbf{x}.\max + \mathbf{x}.\min}$

3) What will be the output of the following code:

```
from sklearn.preprocessing import MaxAbsScaler
Y = [[2], [8], [-4]]
mas= MaxAbsScaler()
print(mas.fit_transform(Y))
```

- [[0.2] [2.0] [1]]
- [[0.25] [2.0] [1]]
- [[0.2] [2.0] [-1]]
- (selected) [[0.25] [1] [-0.5]]

4) What will be the output of the following code:

```
from sklearn.preprocessing import FunctionTransformer
import numpy as np

X = np.array([[1], [2], [3]])
ft = FunctionTransformer(np.mean)
ft.fit_transform(X)
```

2

L3: Categorical Transformers

OneHotEncoder

- Encodes **categorical feature** or **label** as a one-hot numeric array.
- Creates one binary column for each of K unique values.
- Exactly one column has 1 in it and rest have 0.

$$\mathbf{x}_{4\times1} = \begin{bmatrix}1\\2\\3\\1\end{bmatrix} \rightarrow \texttt{ohe=OneHotEncoder()}\ \texttt{ohe.fit\_transform(x)} \rightarrow \mathbf{X}'_{4\times3} = \begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\\1 & 0 & 0\end{bmatrix}$$

unique values: $K = 3$ $\quad\quad$ # columns in transformed matrix = 3

LabelEncoder

Encodes target labels with value between 0 and $K-1$, where K is number of distinct values.

$$\mathbf{y}_{6\times1} = \begin{bmatrix}1\\2\\6\\1\\8\\6\end{bmatrix} \rightarrow \texttt{le = LabelEncoder()}\ \texttt{le.fit\_transform(y)} \rightarrow \mathbf{y}'_{6\times1} = \begin{bmatrix}0\\1\\2\\0\\3\\2\end{bmatrix}$$

Here $K = 4$: $\{1, 2, 6, 8\}$

1 is encoded as 0, 2 as 1, 6 as 2, and 8 as 3.

OrdinalEncoder

Encodes categorical features with value between 0 and $K-1$, where K is number of distinct values.

$$\mathbf{X}_{6\times2} = \begin{bmatrix}1 & \text{'male'}\\2 & \text{'female'}\\6 & \text{'female'}\\1 & \text{'male'}\\8 & \text{'male'}\\6 & \text{'female'}\end{bmatrix} \rightarrow \texttt{oe = OrdinalEncoder()}\ \texttt{oe.fit\_transform(X)} \rightarrow \mathbf{X}'_{6\times2} = \begin{bmatrix}0 & 1\\1 & 0\\2 & 0\\0 & 1\\3 & 1\\2 & 0\end{bmatrix}$$

OrdinalEncoder can operate multi dimensional data, while LabelEncoder can transform only 1D data.

LabelBinarizer

Several regression and binary classification can be extended to multi-class setup in one-vs-all fashion.

This involves training a single regressor or classifier per class.

For this, we need to convert multi-class labels to binary labels, and LabelBinarizer performs this task.

$$\mathbf{y}_{6\times1} = \begin{bmatrix}1\\2\\6\\1\\8\\6\end{bmatrix} \rightarrow \texttt{lb=LabelBinarizer()}\ \texttt{lb.fit\_transform(y)} \rightarrow \mathbf{Y}'_{6\times4} = \begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 1 & 0\\1 & 0 & 0 & 0\\0 & 0 & 0 & 1\\0 & 0 & 1 & 0\end{bmatrix}$$

estimator supports multiclass data, LabelBinarizer is not needed.

MultiLabelBinarizer

Encodes categorical features with value between 0 and $K-1$, where K is number of classes.

In this example $K = 4$, since there are only 4 genres of movies.

movie_genres =
[{'action', 'comedy' },
{'comedy'},
{'action', 'thriller'},
{'science-fiction', 'action', 'thriller'}]

$$\mathbf{X'}_{4\times4} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

```
mlb = MultiLabelBinarizer()
mlb.fit_transform(movie_genres)
```

add_dummy_feature

Augments dataset with a column vector, each value in the column vector is 1.

$$\mathbf{X}_{4\times2} = \begin{bmatrix} 7 & 1 \\ 1 & 8 \\ 2 & 0 \\ 9 & 6 \end{bmatrix} \xrightarrow{\text{add\_dummy\_feature(X)}} \mathbf{X'}_{4\times3} = \begin{bmatrix} 1 & 7 & 1 \\ 1 & 1 & 8 \\ 1 & 2 & 0 \\ 1 & 9 & 6 \end{bmatrix}$$

1) Consider the following feature matrix:

$$X = \begin{bmatrix} 4 \\ 3 \\ 2 \\ 3 \\ 1 \end{bmatrix}$$

What will be the dimension of transformed feature matrix X' after the OneHotEncoder() transformation?

- ○ 4×4
- ○ 4×5
- ○ 5×5
- ◉ 5×4

2) Consider the following categorical feature matrix X and its transformed feature matrix X' after the MultiLabelBinarizer() transformation:

$$X = \begin{bmatrix} \{\text{'Heart-disease'} & \text{'Diabetes'}\} \\ \{\text{'High-BP'}\} & \\ \{\text{'Heart-disease'} & \text{'High-BP'}\} \\ \{\text{'Diabetes'}\} & \end{bmatrix}$$

$$X' = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Which column in X' refers to the label 'Diabetes'?

3

3) What does the transformer **LabelBinarizer**() do?

- ◉ Converts multi-class labels to binary labels.
- ○ Converts multi-label feature to the binary label.
- ○ Encodes categorical features with value between 0 and $K-1$, where K is number of classes.
- ○ Encodes categorical features with values -1 and 1.

4) What will be the output of the following code:

```
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import add_dummy_feature

imp = SimpleImputer(missing_values=np.nan, strategy='mean')
X = [[3,8], [1, 4], [8,np.nan]]
print(add_dummy_feature(imp.fit_transform(X)))
```

- ○ [[3. 8.] [1. 4.] [8. 6.]]
- ○ [[1. 3. 8.] [1. 1. 4.] [1. 8. np.nan]]
- ◉ [[1. 3. 8.] [1. 1. 4.] [1. 8. 6.]]
- ○ [[1. 3. 8.] [1. 1. 4.] [1. 8. 4.]]

5) What will be the output of the following code:

```
from sklearn.preprocessing import LabelBinarizer
lb = LabelBinarizer()
print(lb.fit_transform(['yes', 'yes', 'no', 'yes']))
```

- ○ [[1] [0] [0] [1]]
- ◉ [[1] [1] [0] [1]]
- ○ [[0] [1] [1] [0]]
- ○ [[0] [0] [1] [0]]

L4: Filter Based Feature Selection

VarianceThreshold

Removes all features with variance below a certain threshold, as specified by the user, from input feature matrix

By default removes a feature which has same value, i.e. zero variance.

Univariate feature selection

Univariate feature selection selects features based on univariate statistical tests.

There are three APIs for univariate feature selection:

SelectKBest
Removes all but the k highest scoring features

SelectPercentile
Removes all but a user-specified highest scoring percentage of features

GenericUnivariateSelect
Performs univariate feature selection with a configurable strategy, which can be found via hyper-parameter search.

sklearn provides one more class of univariate feature selection methods that work on common univariate statistical tests for each feature:

SelectFpr selects features based on a false positive rate test.

SelectFdr selects features based on an estimated false discovery rate.

SelectFwe selects features based on family-wise error rate.

Univariate scoring function

- Each API need a scoring function to score each feature.
- Three classes of scoring functions are proposed:

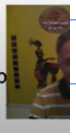
Mutual information (MI) | Chi-square | F-statistics

- MI and F-statistics can be used in both classification and regression problems.

mutual_info_regression | f_regression
mutual_info_classif | f_classif

- Chi-square can be used only in classification prob

chi2

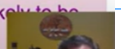


Mutual information (MI)

- Measures dependency between two variables.
- It returns a non-negative value.
  - MI = 0 for independent variables.
  - Higher MI indicates higher dependency.

Chi-square

- Measures dependence between two variables.
- Computes chi-square stats between non-negative feature (boolean or frequencies) and class label.
- Higher chi-square values indicates that the features and labels are likely to be correlated.



SelectKBest

```
skb = SelectKBest(chi2, k=20)
X_new = skb.fit_transform(X, y)
```

Selects 20 best features based on chi-square scoring function.

SelectPercentile

```
sp = SelectPercentile(chi2, percentile=20)
X_new  = sp.fit_transform(X, y)
```

Selects top 20 percentile best features based on chi-square scoring function.

GenericUnivariateSelect

```
transformer = GenericUnivariateSelect(chi2, mode='k_best', param=20)
X_new = transformer.fit_transform(X, y)
```

COPY

- Selects set of features based on a feature selection mode and a scoring function.
- The `mode` could be `'percentile'` (default), `'k_best'`, `'fpr'`, `'fdr'`, `'fwe'`.
- The `param` argument takes value corresponding to the `mode`.

1) Which of the followings is/ are the APIs for the feature selection based on the filter based methods?

- [ ] RFE
- [x] VarianceThreshold
- [ ] SelectFromModel
- [x] SelectPercentile

2) What task does the API VarianceThreshold do?

- (●) Removes all features with variance below a certain threshold from input feature matrix.
- ( ) Removes all features with variance above a certain threshold from input feature matrix.
- ( ) Only removes all features with zero variance.
- ( ) Normalize the features to reduce the variance.

3) What task does the **API SelectKBest** do?

- ( ) Removes all the k highest scoring features.
- (●) Removes all but the k highest scoring features.
- ( ) Removes all but a user-specified highest scoring percentage of features.
- ( ) Removes all user-specified highest scoring percentage of features.

4) What will the following code implement?

```
transformer = GenericUnivariateSelect(chi2, mode='k_best', param=20)
X_new = transformer.fit_transform(X, y)
```

- ( ) Selects top 20 percentile best features based on chi-square scoring function.
- ( ) Selects top 20 percent best features based on chi-square scoring function.
- (●) Selects 20 best features based on chi-square scoring function.
- ( ) Selects 20 best features based on F-statistics scoring function.

5) Which of the following options is/are correct about the scoring function in univariate feature selection methods.

- [x] Chi-square can be used only in classification problems.
- [ ] Chi-square can be used in both regression and classification problems.
- [ ] F-statistics can only be used in classification problems.
- [x] F-statistics can be used in both classification and regression problems.

6) What will be the output of the following code:

```
from sklearn.feature_selection import VarianceThreshold
import numpy as np

X = np.array([[1, 1, 1], [1,3,4], [1,2,4]])

vt = VarianceThreshold()
vt.fit_transform(X)
```

- ( ) [[1 1 1]
[1 3 4]
[1 2 4]]
- ( ) [[1 3 4]
[1 2 4]]
- ( ) [[3 4]
[2 4]]
- (●) [[1 1]
[3 4]
[2 4]]

L5: Wrapper Based Feature Selection

Recursive Feature Elimination (RFE)

- Uses an estimator to recursively remove features.
  - Initially fits an estimator on all features.
- Obtains feature importance from the estimator and removes the least important feature.
- Repeats the process by removing features one by one, until desired number of features are obtained.

- Use `RFECV` if we do not want to specify the desired number of features in `RFE`.
- It performs `RFE` in a cross-validation loop to find the optimal number of features.

SelectFromModel

Selects desired number of important features (as specified with max_features parameter) above certain threshold of feature importance as obtained from the trained estimator.

- The feature importance is obtained via `coef_`, `feature_importances_` or an importance_getter callable from the trained estimator
- The feature importance threshold can be specified either numerically or through string argument based on built-in heuristics such as `'mean'`, `'median'` and float multiples of these like `'0.1*mean'`.

Let's look at a concrete example of SelectFromModel

```
clf = LinearSVC(C=0.01, penalty="l1", dual=False)
clf = clf.fit(X, y)
clf.coef_

model = SelectFromModel(clf, prefit=True)
X_new = model.transform(X)
```

- Here we use a linear support vector classifier to get coefficients of features for `SelectFromModel` transformer.
- It ends up selecting features with non-zero weights or coefficients.

Sequential feature selection

Performs feature selection by selecting or deselecting features one by one in a greedy manner.

Uses one of the two approaches

Forward selection

Starting with a zero feature, it finds one feature that obtains the best cross validation score for an estimator when trained on that feature.

Repeats the process by adding a new feature to the set of selected features.

Backward selection

Starting with all features and removes least important features one by one following the idea of forward selection.

Stops when reach the desired number of features.

- The direction parameter controls whether forward or backward SFS is used.
- In general, forward and backward selection do not yield equivalent results.
- Select the direction that is efficient for the required number of selected features:
  - When we want to select 7 out of 10 features,
    - Forward selection would need to perform 7 iterations.
    - Backward selection would only need to perform 3.
  - Backward selection seems to be a reasonable choice here.

- SFS does not require the underlying model to expose a `coef_` or `feature_importances_` attributes unlike in `RFE` and `SelectFromModel`.
- SFS may be slower than `RFE` and `SelectFromModel` as it needs to evaluate more models compared to the other two approaches.

For example in backward selection, the iteration going from m features to $m-1$ features using k-fold cross-validation requires fitting $m \times k$ models, while

- `RFE` would require only a single fit, and
- `SelectFromModel` performs a single fit and requires no iterations.

1) Which of the following options are correct about the recursive feature elimination(RFE)?

- [ ] It is a filter based feature selection method.
- [x] It is a wrapper based feature selection method.
- [x] RFE works by searching for a subset of features by starting with all features in the training dataset and successively removing features until the desired number remains.
- [ ] Selects desired number of important features (as specified with max features parameter) above certain threshold of feature importance as obtained from the trained estimator

2) The argument of the parameter threshold in the class **SelectFromModel** can be

- [x] integer
- [x] 'mean'
- [x] 'median'
- [ ] Can not be a string like 'mean'.

3) Which approach is used in the class **SequentialFeatureSelection**()?

- ( ) Only forward selection approach
- ( ) Only backward selection approach
- (●) Both forward and Backward
- ( ) None of the above

4) If we want to select three features out of ten features using **SequentialFeatureSelection**(),which approach will be more reasonable?

- (●) Forward selection approach
- ( ) Backward selection approach

L6: Heterogenous Feature Transformations

Generally training data contains diverse features such as numeric and categorical.

Different feature types are processed with different transformers.

Need a way to combine different feature transformers seamlessly.

Composite Transformer

`sklearn.compose` has useful classes and methods to apply transformation on subset of features and combine them:

ColumnTransformer

- It applies a set of transformers to columns of an array or `pandas.DataFrame`, concatenates the transformed outputs from different transformers into a single matrix.
- It is useful for transforming heterogenous data by applying different transformers to separate subsets of features.
- It combines different feature selection mechanisms and transformation into a single transformer object.

- `ColumnTransformer()`
- Each tuple has format
  - `(estimatorName,estimator(...), columnIndices)`

```
column_trans = ColumnTransformer(
    [('ageScaler', CountVectorizer(), [0]]),
     ('genderEncoder', OneHotEncoder(dtype='int'), [1])],
    remainder='drop', verbose_feature_names_out=False)
```

COPY

Consider following feature matrix, which represent weight and gender of a class of students.

$$\mathbf{X}_{6\times 2} = \begin{bmatrix} 20.0 & 'male' \\ 11.2 & 'female' \\ 15.6 & 'female' \\ 13.0 & 'male' \\ 18.6 & 'male' \\ 16.4 & 'female' \end{bmatrix}$$

Here, first column is numeric, however, second column is categorical, therefore different transformers have to be applied on them.

In this example, lets apply MaxAbsScaler on the numeric column and OneHotEncoder on categorical column.

```
column_trans = ColumnTransformer(
    [('ageScaler', MaxAbsScaler(), [0]),
     ('genderEncoder', OneHotEncoder(dtype='int'), [1])],
    remainder='drop', verbose_feature_names_out=False)
column_trans.fit_transform(X)
```

$$\mathbf{X}_{6\times 2} = \begin{bmatrix} 20.0 & 'male' \\ 11.2 & 'female' \\ 15.6 & 'female' \\ 13.0 & 'male' \\ 18.6 & 'male' \\ 16.4 & 'female' \end{bmatrix} \quad \mathbf{X'}_{6\times 3} = \begin{bmatrix} 1. & 0. & 1. \\ 0.56 & 1. & 0. \\ 0.78 & 1. & 0. \\ 0.65 & 0. & 1. \\ 0.93 & 0. & 1. \\ 0.82 & 1. & 0. \end{bmatrix}$$

Transforming Target for Regression

TransformedTargetRegressor

- Transforms the target variable `y` before fitting a regression model.
- The predicted values are mapped back to the original space via an inverse transform.
- `TransformedTargetRegressor` takes `regressor` and `transformer` to be applied to the target variable as arguments.

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.compose import TransformedTargetRegressor
tt = TransformedTargetRegressor(regressor=LinearRegression(),
                                func=np.log, inverse_func=np.exp)
X = np.arange(4).reshape(-1, 1)
y = np.exp(2 * X).ravel()
tt.fit(X, y)
```

1) Which of the following modules in sklearn deals with applying transformation on subset of diverse features (such as numerical and categorical both) and combine them?

- sklearn.impute
- sklearn.feature_selection
- sklearn.model_selection
- **sklearn.compose** ✓

2) What will be the output of the following code:

```
import numpy as np
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import MinMaxScaler, OrdinalEncoder

X = np.array([[1, 'yes'], [2, 'no'], [3, 'no']])
ct = ColumnTransformer([('scaler', MinMaxScaler(),[0]),
                        ('pass','passthrough',[0]),
                        ('encoder', OrdinalEncoder(),[1])])

print(ct.fit_transform(X))
```

- [['0.1' '1' '1.0']
  ['0.5' '2' '0.0']
  ['0.0' '3' '0.0']]
- [['0.0' '1' '1.0']
  ['0.25' '2' '0.0']
  ['0.1' '3' '0.0']]
- [['0.0' '1' '1.0']
  ['0.5' '2' '1.0']
  ['1.0' '3' '0.0']]
- **[['0.0' '1' '1.0']
  ['0.5' '2' '0.0']
  ['1.0' '3' '0.0']]** ✓

L7: Dimensionality Reduction by PCA

Another way to reduce the number of feature is through unsupervised dimensionality reduction techniques.

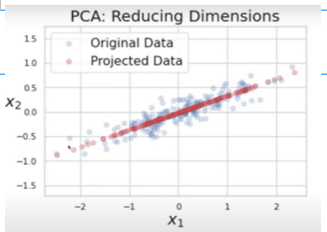
`sklearn.decomposition` module has a number of APIs for this task.

We will focus on how to perform feature reduction with principle component analysis (PCA) in sklearn.

PCA 101

- PCA, is a linear dimensionality reduction technique.
- It uses singular value decomposition (SVD) to project the feature matrix or data to a lower dimensional space.
- The first principle component (PC) is in the direction of maximum variance in the data.
  - It captures bulk of the variance in the data.
- The subsequent PCs are orthogonal to the first PC and gradually capture lesser and lesser variance in the data.
- We can select first k PCs such that we are able to capture the desired variance in the data.

`sklearn.decomposition.PCA` API is used for performing PCA based dimensionality reduction.



1) Which module in sklearn is used for dimensionality reduction techniques?

- sklearn.impute
- sklearn.feature_selection
- sklearn.compose
- **sklearn.decomposition** ✓

2) Which of the following statements is/ are correct about the principle component analysis (PCA)?

- [x] It is a linear dimensionality reduction technique.
- [x] It uses singular value decomposition (SVD) to project the feature matrix or data to a lower dimensional space.
- [ ] It uses stochastic methods to project the feature matrix or data to a lower dimensional space.
- [x] The subsequent PCs are orthogonal to the first PC.

L8: Chaining Transformers

The preprocessing transformations are applied one after another on the input feature matrix.

```
si = SimpleImputer()
X_imputed = si.fit_transform(X)
ss =StandardScaler()
X_scaled = ss.fit_transform(X_imputed)
```

It is important to apply exactly same transformation on training, evaluation and test set in the same order.

Failing to do so would lead to incorrect predictions from model due to distribution shift and hence incorrect performance evaluation.

The sklearn.pipeline module provides utilities to build a composite estimator, as a chain of transformers and estimators.

There are two classes: (i) Pipeline and (ii) FeatureUnion.

Class	Usage
Pipeline	Constructs a chain of multiple transformers 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.

`sklearn.pipeline.Pipeline`

Sequentially apply a list of transformers and estimators.

Intermediate steps of the pipeline must be 'transformers' that is, they must implement fit and transform methods.

The final estimator only needs to implement fit.

The purpose of the pipeline is to assemble several steps that can be cross-validated together while setting different parameters.

`Pipeline()`

- It takes a list of `('estimatorName', estimator(...))` tuples.
- The pipeline object exposes interface of the last step.

```
estimators = [
  ('simpleImputer', SimpleImputer()),
  ('standardScaler', StandardScaler()),
]
pipe = Pipeline(steps=estimators)
```

Accessing individual steps in Pipeline

```
estimators = [
  ('simpleImputer', SimpleImputer()),
  ('pca', PCA()),
  ('regressor', LinearRegression())
]
pipe = Pipeline(steps=estimators)
```

Total # steps: 3

1. SimpleImputer
2. PCA
3. LinearRegression

The second estimator can be accessed in following 4 ways:

- `pipe.named_steps.pca`
- `pipe.steps[1]`
- `pipe[1]`
- `pipe['pca']`

Accessing parameters of each step in Pipeline

Parameters of the estimators in the pipeline can be accessed using the `<estimator>__<parameterName>` syntax, note there are two underscores between `<estimator>` and `<parameterName>`

```
estimators = [
  ('simpleImputer', SimpleImputer()),
  ('pca', PCA()),
  ('regressor', LinearRegression())
]
pipe = Pipeline(steps=estimators)

pipe.set_params(pca__n_components = 2)
```

In above example `n_components` of `PCA()` step is set after the pipeline is created.

Performing grid search with pipeline

By using naming convention of nested parameters, grid search can implemented.

```
param_grid = dict(imputer=['passthrough',
                           SimpleImputer(),
                           KNNImputer()],
                  clf=[SVC(), LogisticRegression()],
                  clf__C=[0.1, 10, 100])
grid_search = GridSearchCV(pipe, param_grid=param_grid)
```

- `C` is an inverse of regularization, lower its value stronger the regularization is.
- In the example above `clf__C` provides a set of values for grid search.

- Combines multiple steps of end to end ML into single object such as missing value imputation, feature scaling and encoding, model training and cross validation.
- Enables joint grid search over parameters of all the estimators in the pipeline.
- Makes configuring and tuning end to end ML quick and easy.
- Offers convenience, as a developer has to call `fit()` and `predict()` methods only on a `Pipeline` object (assuming last step in the pipeline is an estimator).
- Reduces code duplication: With a `Pipeline` object, one doesn't have to repeat code for preprocessing and transforming the test data.

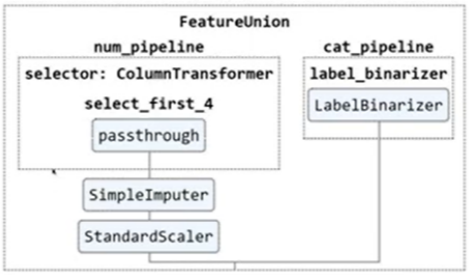
`sklearn.pipeline.FeatureUnion`

- Concatenates results of multiple transformer objects.
- Applies a list of transformer objects in parallel, and their outputs are concatenated side-by-side into a larger matrix.
- `FeatureUnion` and `Pipeline` can be used to create complex transformers.

Combining Transformers and Pipelines

- `FeatureUnion()` accepts a list of tuples.
- Each tuple is of the format:
  - `('estimatorName',estimator(...))`

```
num_pipeline = Pipeline([('selector',ColumnTransformer([('select_first_4',
                                                         'passthrough',
                                                         slice(0,4))])),
                         ('imputer', SimpleImputer(strategy="median")),
                         ('std_scaler', StandardScaler()),
                        ])
cat_pipeline = ColumnTransformer([('label_binarizer', LabelBinarizer(),[4]),
                                  ])
full_pipeline = FeatureUnion(transformer_list=
                             [("num_pipeline", num_pipeline),
                              ("cat_pipeline", cat_pipeline),])
```



1) What task does the module sklearn.pipeline do?

- (•) It provides utilities to build a composite estimator, as a chain of transformers and estimators.
- ( ) It provides the utilities to build the estimators only on test sets.
- ( ) It provides the utilities to build the estimators only on training sets.
- ( ) It provides utilities to reduce the dimension of training dataset.

2) What will be the output of the following code:

```
from sklearn.decomposition import PCA
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
estimators = [
    ('simpleImputer', SimpleImputer()),
    ('pca', PCA()),
    ('regressor', LinearRegression())
]
pipe = Pipeline(steps=estimators)
print(len(pipe.steps))
```

3

3) What are the two ways to create a pipeline object?

- ( ) Pipeline() and Create_Pipeline()
- ( ) Pipeline() and Pipe()
- ( ) Pipeline() and Join_Pipeline()
- (•) Pipeline() and make_pipeline()

4) Which of the following statements is/are correct about the class **sklearn.pipeline.FeatureUnion**?

- [ ] It provides utilities to build a composite estimator, as a chain of transformers and estimators.
- [x] It concatenates results of multiple transformer objects.
- [x] The list of transformer objects can be executed in parallel.
- [ ] It provides the utilities to build the estimators only on test sets.

L1: Linear Regression

How to build baseline regression model?

`DummyRegressor` helps in creating a baseline for regression.

```
from sklearn.dummy import DummyRegressor

dummy_regr = DummyRegressor(strategy="mean")
dummy_regr.fit(X_train, y_train)
dummy_regr.predict(X_test)
dummy_regr.score(X_test, y_test)
```

- It makes a prediction as specified by the strategy.
- Strategy is based on some statistical property of the training set or user specified value.

Strategy `mean` `median` `quantile` `constant`

How is Linear Regression model trained?

Step 1: Instantiate object of a suitable linear regression estimator from one of the following two options

Normal equation

```
from sklearn.linear_model import LinearRegression
linear_regressor = LinearRegression()
```

Iterative optimization

```
from sklearn.linear_model import SGDRegressor
linear_regressor = SGDRegressor()
```

Step 2: Call fit method on linear regression object with training feature matrix and label vector as arguments.

```
# Model training with feature matrix X_train a
# label vector or matrix y_train
linear_regressor.fit(X_train, y_train)
```

SGDRegressor Estimator

- Implements stochastic gradient descent
- Use for large training set up (> 10k samples)
- Provides greater control on optimization process through provision for hyperparameter settings.

- `loss= 'squared_error'`
- `loss = 'huber'`
- `penalty = 'l1'`
- `penalty = 'l2'`
- `penalty = 'elasticnet'`

SGDRegressor

- `learning_rate = 'constant'`
- `learning_rate = 'optimal'`
- `learning_rate = 'invscaling'`
- `learning_rate = 'adaptive'`

- `early_stopping = 'True'`
- `early_stopping = 'False'`

It's a good idea to use a random seed of your choice while instantiating SGDRegressor object. It helps us get reproducible results.

Set `random_state` to seed of your choice.

```
from sklearn.linear_model import SGDRegressor
linear_regressor = SGDRegressor(random_state=42)
```

Note: In the rest of the presentation, we won't set the random seed for sake of brevity. However while coding, always set the random seed in the constructor.

SGD is sensitive to feature scaling, so it is highly recommended to scale input feature matrix.

```
from sklearn.linear_model import SGDRegressor
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

sgd = Pipeline([
                ('feature_scaling', StandardScaler()),
                ('sgd_regressor', SGDRegressor())])

sgd.fit(X_train, y_train)
```

Note
- Feature scaling is not needed for word frequencies and indicator features as they have intrinsic scale.
- Features extracted using PCA should be scaled by some constant c such that the average L2 norm of the training data equals one.

```
from sklearn.linear_model import SGDRegressor
linear_regressor = SGDRegressor(shuffle=True)
```

- `learning_rate = 'constant'`
- `learning_rate = 'invscaling'`
- `learning_rate = 'adaptive'`

```
from sklearn.linear_model import SGDRegressor
linear_regressor = SGDRegressor(random_state=42)
```

What is the default setting?

- `learning_rate = 'invscaling'`
- `eta0 = 1e-2`
- `power_t = 0.25`

Learning rate reduces after every iteration:

```
eta = eta0 / pow(t, power_t)
```

Note: You can make changes to these parameters to speed up or slow down the training process.

```
from sklearn.linear_model import SGDRegressor
linear_regressor = SGDRegressor(learning_rate='adaptive',
                                eta0=1e-2)
```

- The learning rate is kept to initial value as long as the training loss decreases.
- When the stopping criterion is reached, the learning rate is divided by 5, and the training loop continues.
- The algorithm stops when the learning rate goes below 10^{-6}.

Set `max_iter` to desired #epochs. The default value is 1000.

```
from sklearn.linear_model import SGDRegressor
linear_regressor = SGDRegressor(max_iter=100)
```

Remember one epoch is one full pass over the training data.

Practical tip

SGD converges after observing approximately 10^6 training samples. Thus, a reasonable first guess for the number of iterations for n sampled training set is

$$\text{max\_iter} = \text{np.ceil}(10^6/n)$$

```
from sklearn.linear_model import SGDRegressor
linear_regressor = SGDRegressor(loss='squared_error',
                                max_iter=500,
                                tol=1e-3,
                                n_iter_no_change=5)
```

The SGDRegreesor stops
- when the training loss does not improve (loss > best_loss - `tol`) for `n_iter_no_change` consecutive epochs
- else after a maximum number of iteration `max_iter`.

```
from sklearn.linear_model import SGDRegressor
linear_regressor = SGDRegressor(average=True)
```

```
sgd_reg = SGDRegressor(max_iter=1, tol=-np.infty, warm_start=True,
                       penalty=None, learning_rate="constant", eta0=0.0005)

for epoch in range(1000):
    sgd_reg.fit(X_train, y_train)  # continues where it left off
    y_val_predict = sgd_reg.predict(X_val)
    val_error = mean_squared_error(y_val, y_val_predict)
```

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_m x_m = \mathbf{w}^T \mathbf{x}$$

The weights $w_1, w_2, \ldots, w_m$ are stored in `coef_` class variable.

```
linear_regressor.coef_
```

The intercept w_0 is stored in `intercept_` class variable.

```
linear_regressor.intercept_
```

Note These code snippets works for both LinearRegression and SGDRegressor, and for that matter to all regression estimators that we will study in this module. Why?

All of them are estimators.

Step 1: Arrange data for prediction in a feature matrix of shape (#samples, #features) or in sparse matrix format.

Step 2: Call predict method on linear regression object with feature matrix as an argument.

```
# Predict labels for feature matrix X_te
linear_regressor.predict(X_test)
```

Same code works for all regression estimators.

1) Which of the following are valid strategy parameter in DummyRegressor?

- [x] mean
- [x] median
- [x] quantile
- [ ] None of the Above

2) Which of the following objects can be used to solve linear regression using iterative optimization?

- ( ) LR=LinearRegression()
- (•) LR=SGDRegressor()
- ( ) LR=DummyRegressor()
- ( ) None of the Above

3) For performing Linear Regression using sklearn, which of the following module will be appropriate?

- ( ) sklearn.linear_regression
- (•) sklearn.linear_model
- ( ) sklearn.regression_model
- ( ) sklearn.linear_fit

4) Which of the following sklearn modules is used to do necessary transformation(s) on the training data before performing polynomial regression?

- ( ) sklearn.linear_model
- ( ) sklearn.metrics
- ( ) sklearn.model_selection
- (•) sklearn.preprocessing

L2: Model Evaluation

STEP 1: Split data into train and test

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

STEP 2: Fit linear regression estimator on training set.

STEP 3: Calculate training error (a.k.a. empirical error)

STEP 4: Calculate test error (a.k.a. generalization error)

Compare training and test errors

Using score method on linear regression object:

```
# Evaluation on the eval set with
# 1. feature matrix
# 2. label vector or matrix (single/multi-output)
linear_regressor.score(X_test, y_test)
```

The score returns R^2 or coefficient of determination

$$R^2 = \left(1 - \frac{u}{v}\right)$$

residual sum of squares: $u = (\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y})$
Sum of squared error (actual and predicted label)

total sum of square
Sum of squared error (actual and mean predicted $v = (\mathbf{y} - \hat{\mathbf{y}}_{\text{mean}})^T(\mathbf{y} - \hat{\mathbf{y}}_{\text{mean}})$

The score returns R^2 or coefficient of determination

$R^2 = \left(1 - \frac{u}{v}\right)$

When?

- The best possible score is 1.0. (u, sum of squared error = 0)
- A constant model that always predicts the expected value of y, would get a score of 0.0. ($u = v$)
- The score can be negative (because the model can be arbitrarily worse).

sklearn provides a bunch of regression metrics to evaluate performance of the trained estimator on the evaluation set.

mean_absolute_error

```
from sklearn.metrics import mean_absolute_error
eval_score = mean_absolute_error(y_test, y_predicted)
```

mean_squarred_error

```
from sklearn.metrics import mean_squarred_error
eval_score = mean_squarred_error(y_test, y_predicted)
```

r2_score Same as output of **score**

```
from sklearn.metrics import r2_score
eval_score = r2_score(y_test, y_predicted)
```

mean_squared_log_error

```
from sklearn.metrics import mean_squared_log_error
eval_score = mean_squared_log_error(y_test, y_predicted)
```

- Useful for targets with exponential growths like population, sales growth etc,
- Penalizes under-estimation heavier than the over-estimation.

mean_absolute_percentage_error

```
from sklearn.metrics import mean_absolute_percentage_error
eval_score = mean_absolute_percentage_error(y_test, y_predicted)
```

- Sensitive to relative error.

median_absolute_error

```
from sklearn.metrics import median_absolute_error
eval_score = median_absolute_error(y_test, y_predicted)
```

- Score is a metric for which higher value is better.
- Error is a metric for which lower value is better.

Convert error metric to score metric by adding neg_ suffix.

Function	Scoring
metrics.mean_absolute_error	neg_mean_absolute_error
metrics.mean_squared_error	neg_mean_squared_error
metrics.mean_squared_error	neg_root_mean_squared_error
metrics.mean_squared_log_error	neg_mean_squared_log_error
metrics.median_absolute_error	neg_median_absolute_error

In case we get comparable performance on train and test with this split, is this performance guaranteed on other splits too?

- Is test set sufficiently large?
  - In case it is small, the test error obtained may be unstable and would not reflect the true test error on large test set.
- What is the chance that the easiest examples were kept aside as test by chance?
  - This if happens would lead to optimistic estimation of the true test error.

We use cross validation for robust performance evaluation.

```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import linear_regression

lin_reg = linear_regression()
score = cross_val_score(lin_reg, X, y, cv=5)
```

- Uses KFold cross validation iterator, that divides training data into 5 folds.
- In each run, it uses 4 folds for training and 1 for evaluation.

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import LeaveOneOut
from sklearn.linear_model import linear_regression

lin_reg = linear_regression()
loocv = LeaveOneOut()
score = cross_val_score(lin_reg, X, y, cv=loocv)
```

```
from sklearn.linear_model import linear_regression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import ShuffleSplit

lin_reg = linear_regression()
shuffle_split = ShuffleSplit(n_splits=5, test_size=0.2, random_state=42)
score = cross_val_score(lin_reg, X, y, cv=shuffle_split)
```

It is also called random permutation based cross validation strategy.

- Generates user defined number of train/test splits.
- It is robust to class distribution.

In each iteration, it shuffles order of data samples and then splits it into train and test.

```
from sklearn.linear_model import linear_regression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import ShuffleSplit

lin_reg = linear_regression()
shuffle_split = ShuffleSplit(n_splits=5, test_size=0.2, random_state=42)
score = cross_val_score(lin_reg, X, y, cv=shuffle_split,
                        scoring='neg_mean_absolute_error')
```

scoring parameter can be set to one of the scoring schemes implemented in sklearn as follows

max_error **r2**
neg_mean_absolute_error **neg_mean_squared_error**
neg_mean_squared_log_error **neg_median_absolute_error**

- For trained estimator, set **return_estimator = True**
- For scores on training set, set **return_train_score = True**

```
from sklearn.model_selection import cross_validate
from sklearn.model_selection import ShuffleSplit

cv = ShuffleSplit(n_splits=40, test_size=0.3,
                  random_state=0)
cv_results = cross_validate(
    regressor, data, target,
    cv=cv, scoring="neg_mean_absolute_error",
    return_train_score=True,
    return_estimator=True)
```

STEP 1: Instantiate an object of learning_curve class with estimator, training data, size, cross validation strategy and scoring scheme as arguments.

```
from sklearn.model_selection import learning_curve

results = learning_curve(
    lin_reg, X_train, y_train, train_sizes=train_sizes, cv=cv,
    scoring="neg_mean_absolute_error")
train_size, train_scores, test_scores = results[:3]
# Convert the scores into errors
train_errors, test_errors = -train_scores, -test_scores
```

STEP 2: Plot training and test scores as function of the size of training sets. And make assessment about model fitment: under/overfitting or right fit.

STEP 1: Fit linear models with different number of features.

STEP 2: For each model, obtain training and test errors.

STEP 3: Plot #features vs error graph - one each for training and test errors.

STEP 4: Examine the graphs to detect under/overfitting.

1) Consider the following

1. Generalization error
2. Empirical error

A. Training error
B. Test error

Which among the following is correct match?

- [ ] 1-A
- [x] 1-B
- [x] 2-A
- [ ] 2-B

4) Which of the following evaluation metric is best suited for targets with exponential growth?

- ( ) Mean squared error
- ( ) Mean absolute error
- ( ) Mean absolute percentage error
- (•) Mean squared log error

2) What is the correct numerical range for the R^2 score (coefficient of determination)?

- ( ) [-1, 1]
- ( ) $(-\infty, +\infty)$
- (•) $(-\infty, 1]$
- ( ) [0, 1]

3) Which of the following python modules implements useful routines for model evaluation

- ( ) sklearn.model_selection
- ( ) sklearn.measurements
- (•) sklearn.metrics
- ( ) numpy

MLP Week 4 Notes

L1: Polynomial Regression

How is polynomial regression model trained?

Step 1: Apply polynomial transformation on the feature matrix.

Step 2: Learn linear regression model (via normal equation or SGD) on the transformed feature matrix.

Implementation tips: Make use of pipeline construct for polynomial transformation followed by linear regression estimator.

Set up polynomial regression model with normal equation

```
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures

# Two steps:
# 1. Polynomial features of desired degree (here degree=2)
# 2. Linear regression
poly_model = Pipeline([
                ('polynomial_transform', PolynomialFeatures(degree=2)),
                ('linear_regression', LinearRegression())])

# Train with feature matrix X_train and label vector y_train
poly_model.fit(X_train, y_train)
```

Set up polynomial regression model with SGD

```
from sklearn.linear_model import SGDRegressor
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures

poly_model = Pipeline([
                ('polynomial_transform', PolynomialFeatures(degree=2)),
                ('sgd_regression', SGDRegressor())])
poly_model.fit(X_train, y_train)
```

Notice that there is a single line code change in two code snippets.

How to use only interaction features for polynomial regression?

```
from sklearn.preprocessing import PolynomialFeatures
poly_transform = PolynomialFeatures(degree=2, interaction_only=True)
```

$[x_1, x_2]$ is transformed to $[1, x_1, x_2, x_1x_2]$.

Note that $[x_1^2, x_2^2]$ are excluded.

1) Which of the following API will Generate a new feature matrix consisting of all polynomial combinations of the features with degree less than or equal to the specified degree?

- PolynomialFeatures()
- TransformFeatures()
- Polynomial_Features()
- Transform_Features()

2) What will following code implement?

```
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures

poly_model = Pipeline([('poly_transformation', PolynomialFeatures(degree = 3)),
                       ('linear_regression', LinearRegression())])

poly_model.fit(X, y)
```

where **X** and **y** are the feature matrix and label-vector, respectively.

- It will train the linear regression model on the feature matrix **X**.
- It will only apply the polynomial transformation of degree 3.
- It will apply the polynomial transformation of degree 3 on feature matrix **X** and then will train the linear regression model.
- It will apply the polynomial transformation of degree 2 on feature matrix **X** and then will train the linear regression model.

3) What will be the transformed feature of $[x1, x2]$ after execution of following code?

```
from sklearn.preprocessing import PolynomialFeatures

poly_transform= PolynomialFeatures(degree=2, interaction_only=True)
```

- $[1, x_1, x_2, x_1^2, x_2^2]$
- $[1, x_1^2, x_2^2]$
- $[1, x_1, x_2, x_1x_2]$
- $[1, x_1, x_2, x_1^2, x_2^2, x_1x_2]$

L2: Regularization

How to perform ridge regularization with specific regularization rate?

[Option #1]

Step 1: Instantiate object of Ridge estimator

Step 2: Set parameter alpha to the required regularization rate.

```
from sklearn.linear_model import Ridge
ridge = Ridge(alpha=1e-3)
```

fit, score, predict work exactly like other linear regression estimators

[Option #2]

Step 1: Instantiate object of SGDRegressor estimator

Step 2: Set parameter alpha to the required regularization rate and penalty = l2.

```
from sklearn.linear_model import SGDRegressor
sgd = SGDRegressor(alpha=1e-3, penalty='l2')
```

How to search the best regularization parameter for ridge?

[Option #1]

Search for the best regularization rate with built-in cross validation in RidgeCV estimator.

[Option #2]

Use cross validation with Ridge or SGDRegressor to search for best regularization.

- Grid search
- Randomized search

Set up a pipeline of polynomial transformation followed by the ridge regressor.

```
from sklearn.linear_model import Ridge
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures

poly_model = Pipeline([
                ('polynomial_transform', PolynomialFeatures(degree=2)),
                ('ridge', Ridge(alpha=1e-3))])
poly_model.fit(X_train, y_train)
```

Instead of Ridge, we can use SGDRegressor, as shown on previous slide, to get equivalent formulation.

How to perform lasso regularization with specific regularization rate?

[Option #1]

Step 1: Instantiate object of Lasso estimator

Step 2: Set parameter alpha to the required regularization rate.

```
from sklearn.linear_model import Lasso
lasso = Lasso(alpha=1e-3)
```

fit, score, predict work exactly like other linear regression estimators

[Option #2]

Step 1: Instantiate object of SGDRegressor estimator

Step 2: Set parameter alpha to the required regularization rate and penalty = l1.

```
from sklearn.linear_model import SGDRegressor
sgd = SGDRegressor(alpha=1e-3, penalty='l1')
```

[Option #1]

Search for the best regularization rate with built-in cross validation in LassoCV estimator.

[Option #2]

Use cross validation with Lasso or SGDRegressor to search for best regularization.

- Grid search
- Randomized search

Set up a pipeline of polynomial transformation followed by the lasso regressor.

```
from sklearn.linear_model import Lasso
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures

poly_model = Pipeline([
                ('polynomial_transform', PolynomialFeatures(degree=2)),
                ('lasso', Lasso(alpha=1e-3))])
poly_model.fit(X_train, y_train)
```

Instead of Lasso, we can use SGDRegressor to get equivalent formulation.

How to perform both lasso and ridge regularization in polynomial regression?

Set up a pipeline of polynomial transformation followed by the SGDRegressor with `penalty = 'elasticnet'`

```
from sklearn.linear_model import Lasso
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures

poly_model = Pipeline([
                ('polynomial_transform', PolynomialFeatures(degree=2)),
                ('elasticnet', SGDRegressor(penalty='elasticnet',
                                            l1_ratio=0.3))])
poly_model.fit(X_train, y_train)
```

Remember elasticnet is a convex combination of L1 (Lasso) and L2 (Ridge) regularization.

- In this example, we have set l1_ratio to 0.3, which means l2_ratio = 1- l1_ratio = 0.7. L2 takes higher weightage in this formulation.

1) Which of the followings are correct ways to perform ridge regularization with regularization rate of 0.2?

- [x] `from sklearn.linear_model import Ridge` `ridge = Ridge(alpha=0.2)`
- [ ] `from sklearn.linear_model import Ridge_regression` `ridge = Ridge_regression(alpha=0.2)`
- [ ] `from sklearn.linear_model import SGDRegressor` `sgd = SGDRegressor(alpha=0.2, penalty='l1')`
- [x] `from sklearn.linear_model import SGDRegressor` `sgd = SGDRegressor(alpha=0.2, penalty='l2')`

2) Which penalty is used to perform both lasso and ridge regularization in polynomial regression using SGDRegressor()?

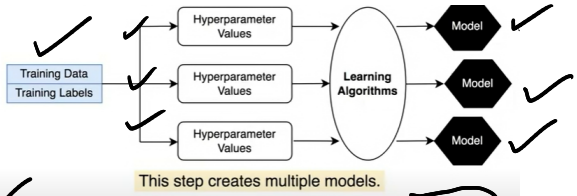
- l1
- l2
- elasticnet
- None of the above

3) Which of the following keyword parameter is used to set regularization type?

- penalty
- alpha
- eta0
- learning_rate

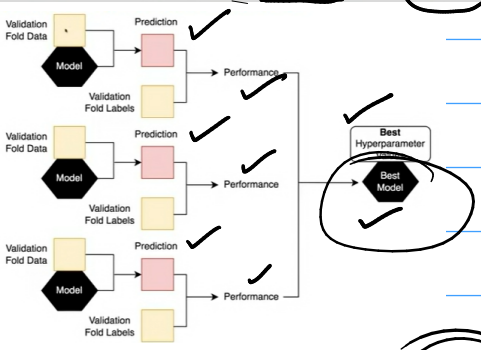
4) Consider the following code:
`sgdr = SGDRegressor(penalty = 'elasticnet', l1_ratio = 0.3)`
What is the **l2_ratio** for sgdr?

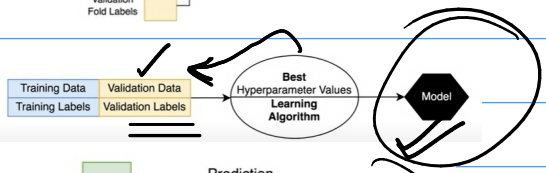
- 0.7
- 3.3
- 33.3
- 0.5

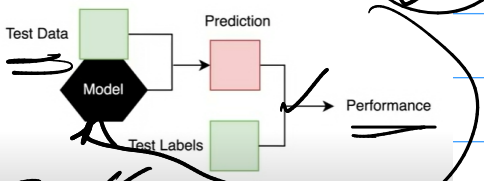


This step creates multiple models.

Tips • This step can be be run in parallel by setting n_jobs = -1







Note that the test set was not used in hyper-parameter search and model retraining .

L3: Hyper Parameter Tuning

- Hyper-parameters are parameters that are not directly learnt within estimators.
- In **sklearn**, they are passed as arguments to the constructor of the estimator classes.

For example,

- degree in PolynomialFeatures
- learning rate in SGDRegressor

Select hyperparameters that results in the best cross validation scores.

Hyper parameter search consists of

- an estimator (regressor or classifier);
- a parameter space;
- a method for searching or sampling candidates;
- a cross-validation scheme; and
- a score function.

Two generic HPT approaches implemented in sklearn are:

- **GridSearchCV** exhaustively considers all parameter combinations for specified values.

```
param_grid = [
  {'C': [1, 10, 100, 1000], 'kernel': ['linear']}
]
```

- **RandomizedSearchCV** samples a given number of candidate values from a parameter space with a specified distribution.

```
param_dist = {
  "average": [True, False],
  "l1_ratio": stats.uniform(0, 1),
  "alpha": loguniform(1e-4, 1e0)
}
```

Grid search	Randomized search
Specifies exact values of parameters in grid	Specifies distributions of parameter values and values are sampled from those distributions. Computational budget can be chosen independent of number of parameters and their possible values. The budget is chosen in terms of the number of sampled candidates or the number of training iterations. Specified in n_iter argument

- Some models can fit data for a range of values of some parameter almost as efficiently as fitting the estimator for a single value of the parameter.
- This feature can be leveraged to perform more efficient cross-validation used for model selection of this parameter.

- linear_model.LassoCV
- linear_model.RidgeCV
- linear_model.ElasticNetCV

```
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import POlynomialFeatures
from sklearn.linear_model import SGDRegressor

param_grid = [
    {'poly__degree': [2, 3, 4, 5, 6, 7, 8, 9]}
  ]

pipeline = Pipeline(steps=[('poly', PolynomialFeatures()),
                           ('sgd', SGDRegressor())])

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

grid_search.fit(x_train.reshape(-1, 1), y_train)
```

1) Which of the following statements is/are correct about the hyper-parameter?

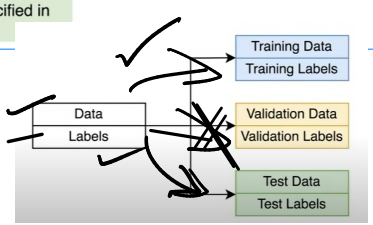
- [x] Hyper-parameter is a parameter whose value is used to control the learning process.
- [ ] Hyper-parameters are parameters that are directly learnt within estimators.
- [x] Hyper-parameters are parameters that are not directly learnt within estimators.
- [ ] In linear regression, weight vector is an example of hyper-parameter.

2) What are the two Hyper-parameter search approaches?

- [x] GridSearchCV
- [ ] GridCV
- [x] RandomizedSearchCV
- [ ] RandomizedCV

3) Which of the following **API** is used to find the names and current values for all parameters for a given estimator?

- estimator.get_param()
- estimator.find_param()
- estimator.get_parameter()
- [x] estimator.get_params()



4) Which of the following options is/are correct about **GridSearchCV**?

- [x] It exhaustively generates candidates from a grid of parameter values specified with the param grid parameter.
- [x] It evaluates all possible combination of parameter values when "fitting" on a dataset and retains the best combination.
- [ ] samples a given number of candidates from a parameter space.
- [ ] In **GridSearchCV**, a budget can be chosen independent of the number of parameters and possible values.

5) Some parameter settings may result in a failure to fit one or more folds of the data. Which keyword is used to avoid the same failure?

- [ ] fit_score=0
- [x] error_score=0
- [ ] error_value=1
- [ ] fit_value=0

L1: Classification Functions in Sklearn

- In this week we will study sklearn functionality for implementing classification algorithms.
- We will cover sklearn APIs for
  - Specific classification algorithms for least square classification, perceptron and logistic regression classifier.
    - with regularization
    - multiclass, multilabel and multi-output setting
  - Various classification metrics.
- Cross validation and hyper parameter search for classification works exactly like how it works in regression setting.
  - However there are a couple of CV strategies that are specific to classification

There are broadly two types of APIs based on their functionality:

Generic	Specific
• SGD classifier	• Logistic regression • Perceptron • Ridge classifier (for LSC) • K-nearest neighbours (KNNs) • Support vector machines (SVMs) • Naive Bayes
Uses gradient descent for opt Need to specify loss	Specialized solvers for opt

Model training

```
fit(X, y[, coef_init, intercept_init, …])
```

Prediction

`predict(X)` predicts class label for samples

`decision_function(X)` predicts confidence score for samples.

Evaluation

```
score(X, y[, sample_weight])
```

There a few common miscellaneous methods as follows:

- `get_params([deep])` gets parameter for this estimator.
- `set_params(**params)` sets the parameters of this estimator.
- `densify()`

Ridge classifier

- RidgeClassifier is a classifier variant of the Ridge regressor.

Binary classification:

- classifier first converts binary targets to {-1, 1} and then treats the problem as a regression task, optimizing the objective of regressor:
  - minimize a penalized residual sum of squares
    $\min_w ||Xw - y||_2^2 + \alpha||w||_2^2$
    sklearn provides different solvers for this optimization
    sklearn uses α to denote regularization rate
  - predicted class corresponds to the sign of the regressor's prediction

Multiclass classification:

- treated as multi-output regression
- predicted class corresponds to the output with the highest value

Step 1: Instantiate a classification estimator without passing any arguments to it. This creates a ridge classifier object.

```
from sklearn.linear_model import RidgeClassifier
ridge_classifier = RidgeClassifier()
```

Step 2: Call fit method on ridge classifier object with training feature matrix and label vector as arguments.

Note: The model is fitted using X_train and y_train.

```
# Model training with feature matrix X_train a
# label vector or matrix y_train
ridge_classifier.fit(X_train, y_train)
```

Set `alpha` to float value. The default value is 0.1.

```
from sklearn.linear_model import RidgeClassifi
ridge_classifier = RidgeClassifier(alpha=0.001)
```

- alpha should be positive.
- Larger alpha values specify stronger regularization.

How to solve optimization problem in RidgeClassifier?

Using one of the following solvers

`svd`	uses a Singular Value Decomposition of the feature matrix to compute the Ridge coefficients.
`cholesky`	uses `scipy.linalg.solve` function to obtain the closed-form solution
`sparse_cg`	uses the conjugate gradient solver of `scipy.sparse.linalg.cg`.
`lsqr`	uses the dedicated regularized least-squares routine `scipy.sparse.linalg.lsqr` and is fastest.
`sag`, `saga`	uses a Stochastic Average Gradient descent iterative procedure 'saga' is unbiased and more flexible version of 'sag'
`lbfgs`	uses L-BFGS-B algorithm implemented in `scipy.optimize.minimize`. can be used only when coefficients are forced to be positive.

- For large scale data, use `'sparse_cg'` solver.
- When both `n_samples` and `n_features` are large, use `'sag'` or `'saga'` solvers.
  - Note that fast convergence is only guaranteed on features with approximately the same scale.

```
ridge_classifier = RidgeClassifier(solver=auto)
```

`auto` chooses the solver automatically based on the type of data

If data is already centered, set fit_intercept as false, so that no intercept will be used in calculations.

Default:

```
ridge_classifier = RidgeClassifier(fit_intercept=True)
```

Use `predict` method to predict class labels for samples

Step 1: Arrange data for prediction in a feature matrix of shape (#samples, #features) or in sparse matrix format.

Step 2: Call predict method on classifier object with feature matrix as an argument.

```
# Predict labels for feature matrix X_test
y_pred = ridge_classifier.predict(X_test)
```

Perceptron classification

- It is a simple classification algorithm suitable for large-scale learning.
- Shares the same underlying implementation with `SGDClassifier`

```
Perceptron()
```

is equivalent to

```
SGDClassifier(loss="perceptron", eta0=1,
learning_rate="constant", penalty=None)
```

Perceptron uses SGD for training.

Step 1: Instantiate a Perceptron estimator without passing any arguments to it to create a classifier object.

```
from sklearn.linear_model import Perceptron
perceptron_classifier = Perceptron()
```

Step 2: Call fit method on perceptron estimator object with training feature matrix and label vector as arguments.

```
# Model training with feature matrix X_train and
# label vector or matrix y_train
perceptron_classifier.fit(X_train, y_train)
```

`penalty` (default = 'l2')	`l1_ratio` (default = 0.15)
`alpha` (default = 0.0001)	`early_stopping` (default = False)
`fit_intercept` (default = True)	`max_iter` (default = 1000)
`n_iter_no_change` (default = 5)	`tol` (default = 1e-3)
`eta0` (default = 1)	`validation_fraction` (default = 0.1)

- Perceptron classifier can be trained in an iterative manner with partial_fit method
- Perceptron classifier can be initialized to the weights of the previous run by specifying warm_start = True in the constructor.

LogisticRegression API

- Implements logistic regression classifier, which is also known by a few different names like logit regression, maximum entropy classifier (maxent) and log-linear classifier.

$$\arg\min_{w,C} \text{ regularization penalty } + C \text{ cross entropy loss}$$

- This implementation can fit
  - binary classification
  - one-vs-rest (OVR)
  - multinomial logistic regression

Step 1: Instantiate a classifier estimator without passing any arguments to it. This creates a logistic regression object.

```
from sklearn.linear_model import LogisticRegression
logit_classifier = LogisticRegression()
```

Step 2: Call fit method on logistic regression classifier object with training feature matrix and label vector as arguments

```
# Model training with feature matrix X_train and
# label vector or matrix y_train
logit_classifier.fit(X_train, y_train)
```

`solver`

- `'newton-cg'`
- `'lbfgs'`
- `'liblinear'`
- `'sag'`
- `'saga'`

- For small datasets, `'liblinear'` is a good choice, whereas `'sag'` and `'saga'` are faster for large ones.
- For unscaled datasets, `'liblinear'`, `'lbfgs'` and `'newton-cg'` are robust.
- For multiclass problems, only `'newton-cg'`, `'sag'`, `'saga'` and `'lbfgs'` handle multinomial loss.
- `'liblinear'` is limited to one-versus-rest schemes

By default, logistic regression uses lbfgs solver.

- *l2* - adds a L2 penalty term
- *l1* - adds a L1 penalty term

penalty

- elasticnet - both L1 and L2 penalty terms are added
- none - no penalty is added

Regularization is applied by default because it improves numerical stability.

- Not all the solvers supports all the penalties.
- Select appropriate solver for the desired penalty.
  - L2 penalty is supported by all solvers
  - L1 penalty is supported only by a few solvers.

Solver	Penalty
'newton-cg'	['l2', 'none']
'lbfgs'	['l2', 'none']
'liblinear'	['l1', 'l2']
'sag'	['l2', 'none']
'saga'	['elasticnet', 'l1', 'l2', 'none']

How to control amount of regularization in logistic regression?

- sklearn implementation uses parameter C, which is inverse of regularization rate to control regularization.
- Recall

$$\arg\min_{w,C} \text{ regularization penalty } + C \text{ cross entropy loss}$$

- C is specified in the constructor and must be positive
  - Smaller value leads to stronger regularization.
  - Larger value leads to weaker regularization.

LogisticRegression classifier has a class_weight parameter in its constructor.

What purpose does it serve?

- Handles class imbalanace with differential class weights.
- Mistakes in a class are penalized by the class weight.
  - Higher value here would mean higher emphasis on the class.

This parameter is available in classifier estimators in sklearn.

SGDClassifier

- SGD is a simple yet very efficient approach to fitting linear classifiers under convex loss functions
- This API uses SGD as an optimization technique and can be applied to build a variety of linear classifiers by adjusting the loss parameter.
- It supports multi-class classification by combining multiple binary classifiers in a "one versus all" (OVA) scheme.
- Easily scales up to large scale problems with more than 10^5 training examples and 10^5 features. It also works with sparse machine learning problems
  - Text classification and natural language processing

loss parameter

- 'hinge' - (soft-margin) linear Support Vector Machine
- 'log' - logistic regression
- 'modified_huber' - smoothed hinge loss brings tolerance to outliers as well as probability estimates
- 'squared_hinge' - like hinge but is quadratically penalized
- 'perceptron' - linear loss used by the perceptron algorithm
- 'squared_error', 'huber', 'epsilon_insensitive', or 'squared_epsilon_insensitive' - regression losses

- SGDClassifier implements a plain stochastic gradient descent learning routine.
  - the gradient of the loss is estimated with one sample at a time and the model is updated along the way with a decreasing learning rate (or strength) schedule.

Advantages:
- Efficiency.
- Ease of implementation

Disadvantages:
- Requires a number of hyperparameters.
- Sensitive to feature scaling.

- It is important
  - to permute (shuffle) the training data before fitting the model.
  - to standardize the features.

Step 1: Instantiate a SGDClassifer estimator by setting appropriate loss parameter to define classifier of interest. By default it uses hinge loss, which is used for training linear support vector machine.

```
from sklearn.linear_model import SGDClassifier
SGD_classifier = SGDClassifier(loss='log')
```

Here we have used `log` loss that defines a logistic regression classifier.

Step 2: Call fit method on SGD classifier object with training feature matrix and label vector as arguments.

```
# Model training with feature matrix X_train and
# label vector or matrix y_train
SGD_classifier.fit(X_train, y_train)
```

- Least square classification (RidgeClassifier)
- Perceptron (Perceptron)
- Logistic regression (LogisticRegression)

Alternatively we can use SGDClassifier with appropriate loss setting for implementing these classifiers:

- loss = `log` for logistic regression
- loss = `perceptron` for perceptron
- loss = `squared_error` for least square classification

Classification estimators implements a few common methods like fit, score, decision_function, and predict.

1) RidgeClassifier is a classifier variant of which regressor?
- (•) Ridge regressor
- ( ) Lasso regressor
- ( ) NoSQL regressor
- ( ) None of the above

2) The first step to apply Ridge classifier should be-
- ( ) Call fit method on ridge classifier object with training feature matrix and label vector as arguments
- (•) Instantiate a classification estimator without passing any arguments
- ( ) Set alpha to float value
- ( ) Use solvers to find the parameter weights that minimize the cost function

3) Which of the following solvers uses the $scipy.linalg.solve$ function to obtain the closed-form solution for minimizing the cost function?
- ( ) svd
- (•) cholesky
- ( ) lsqr
- ( ) lbfgs

4) For large scale data, which solver can be used in RidgeClassifier?
- (•) sparse_cg
- ( ) n_samples

5) To choose the solver automatically based on the type of data, we can use-
- ( ) ridge_classifier = RidgeClassifier(solver=sparse_cg)
- (•) ridge_classifier = RidgeClassifier(solver=auto)
- ( ) ridge_classifier = RidgeClassifier(solver=svd)
- ( ) ridge_classifier = RidgeClassifier(solver=cholesky)

6) How do we make predictions on data samples?
- (•) Use $predict$ method to predict class labels for samples
- ( ) Use $densify$ method to predict class labels for samples
- ( ) Use $sparsify$ method to predict class labels for samples
- ( ) None of the above

7) Mention TRUE or FALSE : LogisticRegression is a linear model for classification rather than regression.
- (•) TRUE
- ( ) FALSE

8) Which of the following codes can be used to set regularization using penalty in Logistic Regression classifier?
- (•) logit_classifier = LogisticRegression(penalty='l2')
- ( ) logit_classifier = LogisticClassifier(solver=auto)
- ( ) logit_classifier = LogisticClassifier(solver=svd)
- ( ) logit_classifier = LogisticRegression(penalty=0.12)

9) Which of the following solvers is used for a Logistic Regression classifier applying on a small dataset?
- (•) liblinear
- ( ) sag
- ( ) saga
- ( ) None of the above

10) Mention TRUE or FALSE : In LogisticRegression classifier, all the solvers support all the penalties.
- ( ) TRUE
- (•) FALSE

11) Which of the following statements are true about SGD?
- [x] It is an efficient approach to fitting linear classifiers under convex loss functions
- [x] It is an optimization technique
- [ ] It corresponds to a specific family of machine learning models
- [ ] None of the above

12) What does a SGDClassifier implement?
- (•) a plain stochastic gradient descent learning routine.

13) The disadvantages of SGDClassifier include-

- [ ] Efficiency
- [ ] Ease of implementation
- [x] Requires a number of hyperparameters
- [x] Sensitive to feature scaling

14) Which loss functions are supported in SGDClassifier?

- [x] hinge
- [x] modified huber
- [x] squared hinge
- [x] perceptron

15) How to specify maximum number of epochs for SGDClassifier?

- ( ) epoch ( )
- (•) max_iter
- ( ) max.iter()
- ( ) None of the above

L2: Multi Learning Classification

Basics of multiclass, multilabel and multioutput classification

- Multiclass classification has exactly one output label and the total number of labels > 2.
- For more than one output, there are two types of classification models:

Multilabel	Multiclass multioutput
total #labels = 2	total #labels > 2

We will refer both these models as multi-label classification models, where # of output labels > 1.

Multiclass, multilabel, multioutput problems are referred to as multi-learning problems.

problem types

- Multiclass classification (sklearn.multiclass)
  - OneVsRestClassifier
  - OneVsOneClassifier
  - OutputCodeClassifier
- Multilabel classification (sklearn.multioutput)
  - MultiOutputClassifier
  - ClassifierChain

meta-estimators

Inherently multiclass

- LogisticRegression (multi_class = 'multinomial')
- LogisticRegressionCV (multi_class = 'multinomial')
- RidgeClassifier
- RidgeClassifierCV

Multiclass as OVR

- LogisticRegression (multi_class = 'ovr')
- LogisticRegressionCV (multi_class = 'ovr')
- SGDClassifier
- Perceptron

Multilabel

- RidgeClassifier
- RidgeClassifierCV

- Classification task with more than two classes.
- Each example is labeled with exactly one class

In Iris dataset,

- There are three class labels: setosa, versicolor and virginica.
- Each example has exactly one label of the three available class labels.
- Thus, this is an instance of a multi-class classification.

In MNIST digit recognition dataset,

- There are 10 class labels: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- Each example has exactly one label of the 10 available class labels.
- Thus, this is an instance of a multi-class classification.

- Each example is marked with a single label out of k labels. The shape of label vector is $(n, 1)$.
- Use LabelBinarizer transformation to convert the class label to multi-class format.

```
from sklearn.preprocessing import LabelBinarizer
y = np.array(['apple', 'pear', 'apple', 'orange'])
y_dense = LabelBinarizer().fit_transform(y)
```

- The resulting label vector has shape of (n, k).

```
[[1 0 0]
 [0 0 1]
 [1 0 0]
 [0 1 0]]
```

- Use type_of_target to determine the type of the label.

```
from sklearn.utils.multiclass import type_of_target
type_of_target(y)
```

- In case, y is a vector with more than two discrete values, type_of_target returns multiclass.

target_type	y
'multiclass'	• contains more than two discrete values • not a sequence of sequences • 1d or a column vector
'multiclass multioutput'	• 2d array that contains more than two discrete values • not a sequence of sequences • dimensions are of size > 1
'multilabel-indicator'	• label indicator matrix • an array of two dimensions with at least two columns, and at most 2 unique values.
'unknown'	• array-like but none of the above, such as a 3d array, • sequence of sequences, or an array of non-sequence objects.

Apart from these, there are three more types, type_of_target can determine targets corresponding to regression and binary classification.

- continuous - regression target
- continuous-multioutput - multi-output target

All classifiers in scikit-learn perform multiclass classification out-of-the-box.

- Use sklearn.multiclass module only when you want to experiment with different multiclass strategies.
- Using different multi-class strategy than the one implemented by default may affect performance of classifier in terms of either generalization error or computational resource requirement.

OVR - OneVsRestClassifier

- Fits one classifier per class c - c vs not c.
- This approach is computationally efficient and requires only k classifiers.
- The resulting model is interpretable.

```
from sklearn.multiclass import OneVsRestClassifier
OneVsRestClassifier(LinearSVC(random_state=0)).fit(X, y)
```

- We need to supply estimator as an argument in the constructor.
- Support methods like other classifiers - fit, predict, predict_proba, partial_fit.

OneVsRest classifier also supports multilabel classification.

OVA - OneVsOneClassifier

- Fits one classifier per pair of classes. Total classifiers = $\binom{k}{2}$.
- Predicts class that receives maximum votes.
  - The tie among classes is broken by selecting the class with the highest aggregate classification confidence.

```
from sklearn.multiclass import OneVsOneClassifier
OneVsOneClassifier(LinearSVC(random_state=0)).fit(X, y)
```

- We need to supply estimator as an argument in the constructor.
- Support methods like other classifiers - fit, predict, predict_proba, partial_fit.

OneVsOne classifier processes subset of data at a time and is useful in cases where the classifier does not scale with

OneVsRestClassifier	OneVsOneClassifier
• Fits one classifier per class. • For each classifier, the class is fitted against all the other classes.	• Fits one classifier per pair of classes. • At prediction time, the class which received the most votes is selected.

MultiOutputClassifier	ClassifierChain
• Able to estimate a series of target functions that are trained on a single predictor matrix to predict a series of responses. • Allows multiple target variable classifications.	• Capable of exploiting correlations among targets. • For a multi-label classification problem with k classes, k binary classifiers are assigned an integer between 0 and $k-1$. • These integers define the order of models in the chain.

1) Mention TRUE or FALSE : Perceptron classification shares the same underlying implementation with SGDClassifier

- (•) TRUE
- ( ) FALSE

2) Mention TRUE or FALSE : Partial_fit method is a way to plot the iteration vs. loss curve.

- (•) TRUE
- ( ) FALSE

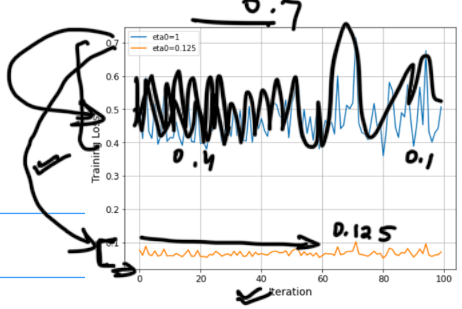


Figure 1: Iteration vs. Training Loss

3) The general trend of variation in training loss with number of iteration is-

- Training losses increase exponentially with the number of iterations.
- For $eta0 = 1$ and $eta0 = 0.125$, the mean of the training loss remains within a particular range of training loss with change in iteration
- For $eta0 = 1$, the training loss decreases with number of iteration and for $eta0 = 0.125$, the training loss increases with number of iteration.
- For $eta0 = 1$, the training loss increases with number of iteration and for $eta0 = 0.125$, the training loss decreases with number of iteration.

4) The mean value of training loss for $eta0 = 0.125$ is in the range-

- is less than 0.1
- 0.4 to 0.5
- mean is equal to 1
- is greater than or equal to 0.3

5) For what value of $eta0$, the graph of training loss is noisy?

- 0.125
- 1

6) Mention TRUE or FALSE: $GridSearchCV$ conducts exhaustive search over specified parameter values for an estimator

- TRUE
- FALSE

7) Single label multiclass classification have how many output labels?

- 2
- 1
- 3
- 4

8) Mention TRUE or FALSE: OVR classifier supports multilabel classification.

- TRUE
- FALSE

So far we learnt how to train classifiers for binary, multi-class and multi-label/output cases.

We will learn how to evaluate these classifiers with different scoring functions and with cross-validation.

We will also study how to set hyper-parameters for classifiers.

Many cross-validation and HPT methods discussed in the regression context are also applicable in classifiers.

- We will not repeat that discussion in this topic.
- Instead we will focus on only additional methods that are specific to classifiers.

Stratified cross validation iterators

There may be issues like class imbalance in classification, which tend to impact the cross validation folds.

The overall class distribution and the ones in folds may be different and this has implications in effective model training.

sklearn.model_selection module provides three stratified APIs to create folds such that the overall class distribution is replicated in individual folds.

sklearn.model_selection module provides the following three stratified APIs to create folds such that the overall class distribution is replicated in individual folds.

- StratifiedKFold
- RepeatedStratifiedKFold
- StratifiedShuffleSplit

LogisticRegressionCV

- Support in-build cross validation for optimizing hyperparameters
- The following are key parameters for HPT and cross validation

`cv` specifies cross validation iterator

`scoring` specifies scoring function to use for HPT

`cs` specifies regularization strengths to experiment with.

- Choosing the best hyper-parameters

`refit = True` Scores averaged across folds, values corresponding to the best score are selected and final refit with these parameters

`refit = False` the coefs, intercepts and C that correspond to the best scores across folds are averaged.

sklearn.metrics implements a bunch of classification scoring metrics based on true labels and predicted labels as inputs.

`accuracy_score` `balanced_accuracy_score` `top_k_accuracy_score` `roc_auc_score` `precision_score` `recall_score` `f1_score`

- `confusion_matrix` evaluates classification accuracy by computing the confusion matrix with each row corresponding to the true class.

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_true, y_predicted)
```

Example:

```
array([[2, 0, 0],
       [0, 0, 1],
       [1, 0, 2]])
```

Entry i, j in a confusion matrix

number of observations actually in group i, but predicted to be in group j.

- Confusion matrix

```
ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
```

- From estimators

```
ConfusionMatrixDisplay.from_estimator(clf, X_test, y_test)
```

- From predictions

```
ConfusionMatrixDisplay.from_predictions(y_test, y_
```

The `classification_report` function builds a text report showing the main classification metrics.

```
from sklearn.metrics import classification_report
print(classification_report(y_true, y_predicted))
```

```
from sklearn.metrics import precision_recall_curve
precision, recall, thresholds = precision_recall_curve(y_true, y_predicted)
```

```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = metrics.roc_curve(y_true, y_scores, pos_label=2)
```

- Treat data as a collection of binary problems, one for each class.
- Then, average binary metric calculations across the set of classes.
  - Can be done using `average` parameter.

`macro`	calculates the mean of the binary metrics
`weighted`	computes the average of binary metrics in which each class's score is weighted by its presence in the true data sample.
`micro`	gives each sample-class pair an equal contribution to the overall metric
`samples`	calculates the metric over the true and predicted classes for each sample in the evaluation data, and returns their average
`None`	returns an array with the score for each class

1) Which stratified APIs are provided by sklearn.model_selection module to create folds?

- StratifiedKFold
- RepeatedStratifiedKFold
- StratifiedShuffleSplit
- OVR

2) Mention TRUE or FALSE: confusion matrix evaluates classification accuracy by computing the confusion matrix with each row corresponding to the true class.

- TRUE
- FALSE

3) Confusion matrix can be displayed with which API in sklearn.metrics?

- ConfusionMatrixDisplay
- RepeatedStratifiedKFold
- ConfusionDisplay
- OVR

4) Which NB is used to implement the Gaussian Naive Bayes algorithm for classification?

- GaussianNB
- MultinomialNB
- ComplementNB of these

L1: Naive Bayes Classifier

Naive Bayes classifier

- Naive Bayes classifier applies Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable.

For a given class variable y and dependent feature vector x_1 through x_m, the naive conditional independence assumption is given by:

$$P(x_i|y, x_1, ..., x_{i-1}, x_{i+1}, ..., x_m) = P(x_i|y)$$

Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods.

List of NB Classifiers

- Implemented in sklearn.naive_bayes module

GaussianNB

BernoulliNB CategoricalNB

MultinomialNB ComplementNB

- Implements fit method to estimate parameters of NB classifier with feature matrix and labels as inputs.

The prediction is performed using predict method.

1) Which of the following is the correct assumption about the naive Bayes classifier?

- $P(x_i|x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_m) = P(x_i)$
- $P(x_1, x_2, \ldots, x_m) = P(x_i)$
- $P(y|x_1, x_2, \ldots, x_m) = P(y)$
- $P(x_i|y, x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_m) = P(x_i|y)$

2) Which of the following modules of sklearn contains the API GaussianNB?

- sklearn.linear_regression
- sklearn.model_selection
- sklearn.naive_bayes
- None of the above

3) Consider following code snippet and select the correct option.

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X, y)
```

where X and y are training data.

- gnb is best suited when features X are imbalanced.
- gnb is best suited when features X are categorical.
- gnb is best suited when X does not have categorical features.
- gnb is best suited for both categorical and numerical features.

4) Which of the following is best suited when the data is imbalanced?

- ComplementNB
- GaussianNB
- BernoulliNB
- None of the above

5) Which of the following Naive Bayes classifiers is suitable for classification of word counts for text classification ?

- GaussianNB
- CategoricalNB
- BernoulliNB
- MultinomialNB

MLP Week 7 Notes

L2: K Nearest Neighbours

Nearest neighbor classifier

- It is a type of instance-based learning or non-generalizing learning
  - does not attempt to construct a model
  - simply stores instances of the training data
- Classification is computed from a simple majority vote of the nearest neighbors of each point.
- Two different implementations of nearest neighbors classifiers are available.
  1. KNeighborsClassifier
  2. RadiusNeighborsClassifier

How are KNeighborsClassifier and RadiusNeighborsClassifier different?

KNeighborsClassifier	RadiusNeighborsClassifier
learning based on the k nearest neighbors	learning based on the number of neighbors within a fixed radius *r* of each training point
most commonly used technique	used in cases where the data is not uniformly sampled
choice of the value *k* is highly data-dependent	fixed value of *r* is specified, such that points in sparser neighborhoods use fewer nearest neighbors for the classification

Step 1: Instantiate a KNeighborsClassifer estimator without passing any arguments to it to create a classifer object.

```
from sklearn.neighbors import KNeighborsClassifier
kneighbor_classifier = KNeighborsClassifier()
```

Step 2: Call fit method on KNeighbors classifier object with training feature matrix and label vector as arguments.

```
# Model training with feature matrix X_train and
# label vector or matrix y_train
kneighbor_classifier.fit(X_train, y_train)
```

- Specify the number of nearest neighbors *K* from the training dataset using `n_neighbors` parameter.
  - value should be int.

```
kneighbor_classifier = KNeighborsClassifier(n_neighbors
```

- It is better to weight the neighbors such that nearer neighbors contribute more to the fit.

weights
- 'uniform' : All points in each neighborhood are weighted equally.
- 'distance' : weight points by the inverse of their distance.
  - closer neighbors of a query point will have a greater influence than neighbors which are further away.

algorithm
- 'ball_tree' will use BallTree
- 'kd_tree' will use KDTree
- 'brute' will use a brute-force search
- 'auto' will attempt to decide the most appropriate algorithm based on the values passed to the fit method.

For 'ball_tree' and 'kd_tree' algorithms, there are some other parameters to be set.

leaf_size
- can affect the speed of the construction and query, as well as the memory required to store the tree
- default = 30

metric
- Distance metric to use for the tree
- It is either string or callable function
  - some metrics are listed below:
    - "euclidean", "manhattan", "chebyshev", "minkowski", "wminkowski", "seuclidean", "mahalanobis"
- default = 'minkowski'

```
from sklearn.neighbors import RadiusNeighborsClassifi
radius_classifier = RadiusNeighborsClassifier()
```

The number of neighbors is specified within a fixed radius *r* of each training point using `radius` parameter.
- *r* is a float value.

```
radius_classifier = RadiusNeighborsClassifier(radius=1.0,
```

weights
- 'uniform'
- 'distance'
- [callable] function
- default = 'uniform'

algorithm
- 'ball_tree'
- 'kd_tree'
- 'brute'
- 'auto'
- default = *'auto'*

1) Which of the following sklearn modules implement relevant functions for the Nearest Neighbors algorithm?

- sklearn.neighbors
- sklearn.multiclass
- sklearn.model_selection
- sklearn.ensemble

2) Based on the following statements, select the correct option.

I. For every dataset, there exists a threshold b_1, such that increasing the value of k beyond this threshold does not significantly affect the accuracy of the model.

II. Let N denote the number of training data points. For every dataset, there exists a threshold N_1, such that increasing the value of N beyond this threshold does not significantly affect the accuracy of the model.

- Statement I is true, Statement II is true.
- Statement I is true, Statement II is false.
- Statement I is false, Statement II is false.
- Statement I is false, Statement II is true.

3) What does the value k in k-Nearest Neighbors algorithm refer to?

- Number of data points to be classified.
- Number of nearest neighbors to consider for each test data point.
- Number of dimensions of each data point.
- Maximum distance between a test data point and its nearest neighbor.

4) Which of the following statements are true? (Multiple options may be correct.)

- [x] kNN models with low values of k produce complex decision boundaries.
- [ ] kNN models with high values of k produce complex decision boundaries.
- [ ] kNN models with low values of k produce simple decision boundaries.
- [x] kNN models with high values of k produce simple decision boundaries.

5) Based on the following statements, select the correct option.

I. A given test data point in kNN can be thought of as being situated at the centroid of k training points nearest to it.

II. If $x_1, x_2, \cdots, x_k$ are the k nearest data points of x, then $\hat{y}(x) = \frac{1}{k} \cdot \sum_{i=1}^{k} y(x_i)$.

- Statement I is true, Statement II is true , Statement II is a correct explanation for statement I.
- Statement I is true, Statement II is true , Statement II is not a correct explanation for statement I.
- Statement I is true, Statement II is false.
- Statement I is false, Statement II is true.

6) Which of the following are false regarding kNN? (Multiple options may be correct.)

- [ ] kNN is a non-parametric supervised learning algorithm.
- [x] The number k in kNN is a parameter to be estimated from the training data.
- [x] The time complexity of classifying a test data point using kNN depends only upon the number of training instances, and not their dimension.
- [ ] KNN models suffer from the curse of dimensionality.

L6: SVM in Scikit-Learn

Support Vector Machines

- Support Vector Machines (SVM) are a set of supervised learning methods used for classification, regression and outliers detection.
- SVM constructs a hyper-plane or set of hyper-planes in a high or infinite dimensional space, which can be used for classification, regression or other tasks.
- In sklearn, we have three methods to implement SVM.

- SVC, NuSVC — These are similar methods but, accept slightly different sets of parameters. Implementation is based on libsvm.
- LinearSVC — Faster implementation of linear SVM classification with only linear kernel. Implementation is based on liblinear.

Array X : holding the training samples

shape → (n_samples, n_features)

```
X = [[0, 0], [1, 1]]
```

Array y : holding the class labels (strings or integers)

shape → (n_samples)

```
y = [0,1]
```

Step 1: Instantiate a SVC classifier estimator.

```
from sklearn.svm import SVC
SVC_classifier = SVC()
```

Step 2: Call fit method on SVC classifier object with training feature matrix and label vector as arguments.

```
# Model training with feature matrix X_train an
# label vector or matrix y_train
SVC_classifier.fit(X_train, y_train)
```

C → Regularization parameter
- float value

Default: `SVC_classifier = SVC(C=1.0)`

Note:
- strength of the regularization is inversely proportional to C
- strictly positive
- penalty is a squared l2 penalty

kernel
- 'linear'
- 'poly'
- 'rbf'
- 'sigmoid'
- 'precomputed'

Default: `SVC_classifier = SVC(kernel = 'rbf')`

- If `kernel = poly` , set `degree` (any integer value)
- If `kernel = callable` is given it is used to pre-compute the kernel matrix from data matrices

gamma → HPT

- 'scale' → value of gamma = $\frac{1}{\text{number of features} * X.Var()}$
- 'auto' → value of gamma = $\frac{1}{\text{number of features}}$
- float value

Default:
```
1 SVC_classifier = SVC(gamma = 'scale')
```

- If **kernel = 'poly'** or **'sigmoid'**, set **coef0** which is an independent term in kernel function (any integer value)

After the classifier is fit on the training data, there are few attributes which reveal the details of support vectors.

```
1 from sklearn.svm import SVC
2 SVC_classifier = SVC()
3 clf = SVC_classifier.fit(X_train, y_train)
4
5 #to view indices of the support vectors
6 clf.support_
7
8 #to view the support vectors
9 clf.support_vectors_
10
11 #to view the number of support vectors for each class
12 clf.n_support_
```

Step 1: Instantiate a NuSVC classifier estimator.

```
1 from sklearn.svm import NuSVC
```

Instead of C in SVC, ν is introduced in NuSVC to control the number of support vectors and margin errors.

ν is an upper bound on the fraction of margin errors and and a lower bound of the fraction of support vectors.

Value of ν should $\in (0, 1]$

Default: $\nu = 0.5$

Other parameters for NuSVC are same as that of SVC.

Step 1: Instantiate a LinearSVC classifier estimator.

```
1 from sklearn.svm import LinearSVC
2 LinearSVC_classifier = LinearSVC()
```

Step 2: Call fit method on SVC classifier object with training feature matrix and label vector as arguments.

```
1 # Model training with feature matrix X_train an
2 # label vector or matrix y_train
3 LinearSVC_classifier.fit(X_train, y_train)
```

penalty
- *l1* - adds a L1 penalty term → lasso
- *l2* - adds a L2 penalty term → ridge

l1 - leads to **coef_** vectors that are sparse.

Default:
```
1 LinearSVC_classifier = Linear_SVC(penalty = 'l2'
```

loss parameter
- 'hinge' - standard SVM loss
- 'squared_hinge' - square of the hinge loss

Default:
```
1 LinearSVC_classifier = Linear_SVC(loss = 'squared_hinge')
```

C
↓
Regularization parameter

dual → better results
↓
- Select the algorithm to either solve the dual or primal optimization problem.
- When n_samples >n_features, prefer dual=False.

fit_intercept → func^n
↓
To calculate the intercept for the model.

- SVC and NuSVC implement the "one-versus-one" approach for multi-class classification.

decision_function_shape
- 'ovo' → one vs one
- 'ovr' → one vs rest

- LinearSVC implements "one-vs-the-rest" approach for multi-class classification.

multi_class
- 'ovr'
- 'crammer_singer'

(out of syllabus)

Advantages of SVM

- Effective in high dimensional spaces.
- Effective in cases where number of dimensions is greater than the number of samples. (n features > n samples)
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different Kernel Functions can be specified for the decision function.

Disadvantages of SVM

(predict_probab.)

- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation.
- Avoid over-fitting in choosing Kernel functions if the number of features is much greater than the number of samples.

1) Which of the following code will throw an error?

- [ ] from sklearn import svm
- [x] from sklearn import SVC
- [ ] from sklearn.svm import SVC
- [x] from sklearn import svm.SVC

2) Which keyword parameter is used to set the Regularization parameter in svm.SVC?

- ( ) R
- ( ) H
- ( ) RC
- (•) C

3) Which of the following module of sklearn contains the API confusion matrix?

- ( ) sklearn.svm
- ( ) sklearn.datasets
- ( ) sklearn.model_selection
- (•) sklearn.metrics

4) What does the method svc.scores() return?

- ( ) It returns the mean precision on the given test data and labels.
- (•) It returns the mean accuracy on the given test data and labels.
- ( ) It returns the mean specificity on the given test data and labels.
- ( ) It returns the mean sensitivity on the given test data and labels.

5) Consider the following code for the classification:

```
svm_clf = Pipeline([
('scaler', StandardScaler()),
('linear_svc', LinearSVC(C=100, loss="hinge")),
])
```

If this leads to the overfitting, which of the following values may overcome the problem of overfitting?

- [ ] C = 110
- [x] C = 10
- [ ] C = 150
- [ ] C = 1

6) SVM classifiers do not output probabilities for each class.

- (•) True
- ( ) False

7) Consider the following code:

```
svm_clf = Pipeline([
('scaler', StandardScaler()),
('linear_svc', LinearSVC(C=1, loss="hinge")),
])
```

Which of the following code may be a alternate way to the same code?

- ( ) SVC(C=1)
- (•) SVC(kernel="linear", C=1)
- ( ) SVC(kernel="linear", C=100)
- ( ) SVC(kernel="rbf", C=1)

8) What will the following code implement? (Assume that all the necessary libraries are imported)

```
polynomial_svm_clf = Pipeline([
('poly_features', PolynomialFeatures(degree=3)),
('scaler', StandardScaler()),
('svm_clf', LinearSVC(C=10, loss="hinge"))])
```

- ( ) It will fit the training data using support vector machine.
- ( ) It will transform the training feature to degree 3 and then will fit the data using SVM
- ( ) It will normalize the training data using StandardScalar and then will fit the data using SVM.
- (•) It will transform the training data to degree 3, normalize the tansformed data using StandardScalar and then will fit the data using SVM.

9) Consider the following code: (Assume that all the necessary libraries are imported)

```
clf = make_pipeline(StandardScaler(), SVC(gamma='auto', kernel= 'poly'))
```

What will be the degree of the polynomial kernel function?

3

10) What is the default value of regualarization parameter used in SVC classifier?

- (•) 1.0
- ( ) 0.1
- ( ) 0.5
- ( ) 0.2

11) For a support vector machine model, let x_i be an input instance with label y_i. If $y_i.(w_0 + x_i^T W) > 1$ where w_0 and W are the estimated parameters of the model, then

- ( ) x_i is a support vector.
- (•) x_i is not a support vector.
- ( ) x_i is an outlier.
- ( ) Depending upon other data points, x_i may or may not be a support vector.

12) Which of the following methods is used to get the indices of support vectors?

- ( ) .support_vectors_
- (•) .support_

13) Which of the following methods is used to get the number of support vectors?

- ( ) .support_vectors_
- ( ) .support_
- (•) .n_support_
- ( ) .nsupport_

14) Consider the following code:

```
from sklearn import svm
clf = svm.NuSVC(gamma="auto")
clf.fit(X, Y)
```

where X and Y are the feature matrix and label vector, respectively. Which kernel is used in the above classification?

- ( ) linear
- ( ) polynomial of degree 3
- (•) rbf
- ( ) sigmoid

15) Which keyword parameter is used in NuSVC to set an upper bound on the fraction of margin errors and a lower bound of the fraction of support vectors.

- ( ) gamma
- ( ) sv
- (•) nu

MLP Week 8 Notes

L1: Decision Trees

Tree algorithms

- **ID3** = Iterative Dichotomiser 3
  - Creates a multiway tree
- **C4.5**
  - Successor to ID3
  - Converts the trained trees into sets of if-then rules
- **C5.0**
  - Quinlan's latest version release under a proprietary license
  - Uses less memory and builds smaller rulesets
- **CART**
  - Classification and Regression Trees
  - Supports numerical target variables (regression) and does not compute rule sets

scikit-learn uses an optimized version of the **CART algorithm**; however, it does not support categorical variables for now

Classification	sklearn.tree.DecisionTreeClassifier
Regression	sklearn.tree.DecisionTreeRegressor

Both these estimators have the same set of parameters except for `criterion` used for tree splitting.

`splitter` `max_depth` `min_samples_split` `min_samples_leaf`

`splitter` Strategy for splitting at each node. `best` `random`

`max_depth` Maximum depth of the tree.

`int` When `None`, the tree expanded until all leaves are pure or they contain less than `min_samples_split` samples.

`min_samples_split` The minimum number of samples required to split an internal node. `2`

`int` `float`

`min_samples_leaf` The minimum number of samples required to be at a leaf node. `1`

`int` `float`

`criterion` Specifies function to measure the quality of a split.

Classification	Regression
gini	squarred_error
entropy	friedman_mse
	absolute_error
	poisson

Visualize a tree

`sklearn.tree.plot_tree`

`decision_tree`	The decision tree to be plotted.
`max_depth`	The maximum depth of the representation. If `none`, the tree is fully generated.
`feature_names`	Names of each of the features. `none`
`class_names`	Names of each of the target classes in ascending numerical order. `none`
`label`	Whether to show informative labels for impurity.

Avoiding overfitting of trees

Pre-pruning

Uses hyper-parameter search like `GridSearchCV` for finding the best set of parameters.

Post-pruning

First grows trees without any constraints and then uses `cost_complexity_pruning` with `max_depth` and `min_samples_split`.

- Use `min_samples_split` or `min_samples_leaf` to ensure that multiple samples influence every decision in the tree, by controlling which splits will be considered.
  - A very small number will usually mean the tree will overfit.
  - A large number will prevent the tree from learning the data.

L5: Voting, Bagging and Random Forest

Voting estimators

Class: `sklearn.ensemble.VotingClassifier`

Class: `sklearn.ensemble.VotingRegressor`

Both these estimators take the following **common parameters**:

`base_estimator` `weights`

Both these estimators implement the following **functions**:

`fit` `predict` `fit_transform` `score`

VotingClassifier takes an **additional argument**:

`voting` `hard`

Class: `sklearn.ensemble.BaggingClassifier`

Class: `sklearn.ensemble.BaggingRegressor`

Class: `sklearn.ensemble.RandomForestClassifier`

Class: `sklearn.ensemble.RandomForestRegressor`

The parameters can be classified as:

- Decision tree parameters
- Bagging parameters

- The number of trees are specified by `n_estimators`.
  - Default #trees for classification = 10
  - Default #trees for regression = 100
- `bootstrap` specifies whether to use bootstrap samples for training.
  - `True`: bootstrapped samples are used.
  - `False`: whole dataset is used.
- `oob_score` specifies whether to use out-of-bag samples for estimating generalization error. It is only available when `bootstrap = True`.
- `max_samples` specifies the number of samples to be drawn while bootstrapping.
  - `None`: Use all samples in the training data.
  - `int`: Use `max_samples` samples from the training data.
  - `float`: Use `max_samples` * total number of samples from the training data. The value should be between 0 and 1.
- `random_state` controls randomness of features and samples selected during bootstrap.
- The number of features to be considered while splitting is specified by `max_features`.
  - `auto`, `sqrt`, `log2`, `int`, `float`
- The depth of the tree is controlled by `max_depth`. The default value is `None`, which means the tree will be grown until all leaf nodes are pure or until leaves contain less than `min_samples_splits` samples.
- `estimators_` member variable contains a collection of fitted estimators.
- `feature_importances_` member variable contains a list of important features.
- `fit` builds forest of trees from the training dataset with the specified parameters.
- `decision_path` returns decision path in the forest.
- `predict` returns class label in classification and output value in regression.
- `predict_proba` and `predict_log_proba` returns probabilities and their logs for classification set up.

L8: Boosting: Adaboost, GradientBoost, XGBoost

Class: sklearn.ensemble.AdaBoostClassifier

Class: sklearn.ensemble.AdaBoostRegressor

base_estimator
- Default estimator is DecisionTreeClassifier with depth = 1.

n_estimators
- Maximum number of estimators where boosting is terminated. The default value is 50.

learning_rate
- Weight applied to each classifier during boosting.
- Higher value here would increase contribution of individual classifiers.
- There is a trade-off between n_estimators and learning_rate.

base_estimator_ Base estimator of ensemble.

estimators_ Collection of fitted sub-estimators.

estimator_weights_ Weights for each estimator in ensemble.

estimator_errors_ Errors for each estimator in ensemble.

There are two most important parameters of these estimators:

- n_estimators
- learning_rates

sklearn.ensemble.GradientBoostingClassifier supports both binary and multiclass classification.

L3: Neural Networks: MLP

Multilayer Perceptron (MLP)

- It is a supervised learning algorithm.
- MLP learns a non-linear function approximator for either classification or regression depending on the given dataset.
- In sklearn, we implement MLP using:
  1. MLPClassifier for classification
  2. MLPRegressor for regression

- MLPClassifier supports multi-class classification by applying Softmax as the output function.
- It also supports supports multi-label classification in which a sample can belong to more than one class.
- MLPRegressor also supports multi-output regression, in which a sample can have more than one target.

otherwise binary classifies

Step 1: Instantiate a MLP classifier estimator.

```
from sklearn.neural_network import MLPClassifier
MLP_clf = MLPClassifier()
```

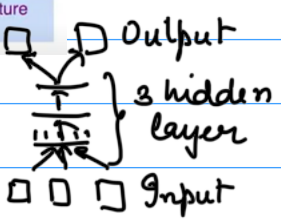
Step 2: Call fit method on MLP classifier object with training feature matrix and label vector as arguments.

Step 3: After fitting (training), the model can make predictions for new samples (X_test) using two methods:

```
MLP_clf.predict(X_test)
MLP_clf.predict_proba(X_test)
```

predict
- gives labels for new samples
- for example:
  `array([1, 0])`

predict_proba
- gives vector of probability estimates per sample
- for example:
  `array([1.967...e-04, 9.998...-01])`



MLP (hidden_layers)

hidden_layer_sizes

- This parameter sets the number of layers and the number of neurons in each layer.
- It is a **tuple** where each element in the tuple represents the number of neurons at the i th position where i is the index of the tuple.
- The length of tuple denotes the total number of hidden layers in the network.

To create a 3 hidden layer neural network with 15 neurons in first layer, 10 neurons in second layer and 5 neurons in third layer:

```
MLPClassifier(hidden_layer_sizes=(15,10,5))
```

= (15, 10, 5)

activation

'identity'	'logistic'	'tanh'	'relu' (Default)
no-op activation returns $f(x) = x$	logistic sigmoid function returns $f(x) = \frac{1}{(1+exp(-x))}$	hyperbolic tan function returns $f(x) = tanh(x)$	rectified linear unit function returns $f(x) = max(0, x)$

- MLPClassifier optimizes the log-loss function using LBFGS or stochastic gradient descent

solver

- lbfgs
- sgd
- adam (Default)

- If the `solver` is 'lbfgs', the classifier will not use minibatch.
- Size of minibatches can be set to other stochastic optimizers: `batch_size` (int)
- default batch_size is 'auto'.

```
batch_size=min(200, n_samp
```

intercepts_ → estimator of MLP

- It is a list of shape (n_layers - 1,)
- The i th element in the list bias vector corresponding to layer $i+1$.

Example:

- "Bias values for first hidden layer:"

```
print(MLP_clf.intercepts_[0])
```

- "Bias values for second hidden layer:"

```
print(MLP_clf.intercepts_[1])
```

third

[2]

---

learning_rate

- 'constant'
- 'invscaling'
- 'adaptive'

default: 'constant'

learning_rate_init

float value
default: 0.001

power_t

float value
default: 0.5

max_iter

int value
default: 500

HPT

- `learning_rate` and `power_t` are used only for `solver = 'sgd'`
- `learning_rate_init` is used when `solver='sgd'` or 'adam'.
- `shuffle` is used to shuffle samples in each iteration when `solver='sgd' or 'adam'`
- `momentum` is used for gradient descent update when `solver='sgd'`

- MLPRegressor trains using backpropagation with no activation function in the output layer.
- Therefore, it uses the square error as the loss function, and the **output is a set of continuous values**.

The parameters of MLPRegressor are the same as that of MLPClassifier.