Support Vector Machines

Prof.Dr. Bahadır AKTUĞ
Machine Learning with Python

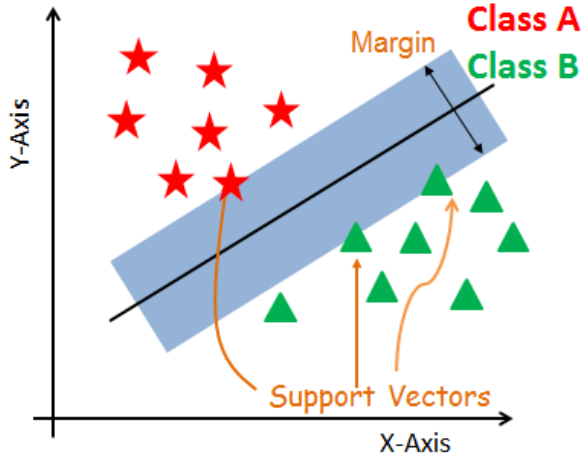
**Compiled from sources given in the references.*

Support Vector Machines

- SVM offers very high accuracy compared to other classifiers such as logistic regression, and decision trees.
- It is known for its kernel trick to handle nonlinear input spaces.
- It is used in a variety of applications such as face detection, intrusion detection, classification of emails, news articles and web pages, classification of genes, and handwriting recognition.
- Support Vector Machines is considered to be a classification approach, it but can be employed in both types of classification and regression problems. It can easily handle multiple continuous and categorical variables.

Support Vector Machines

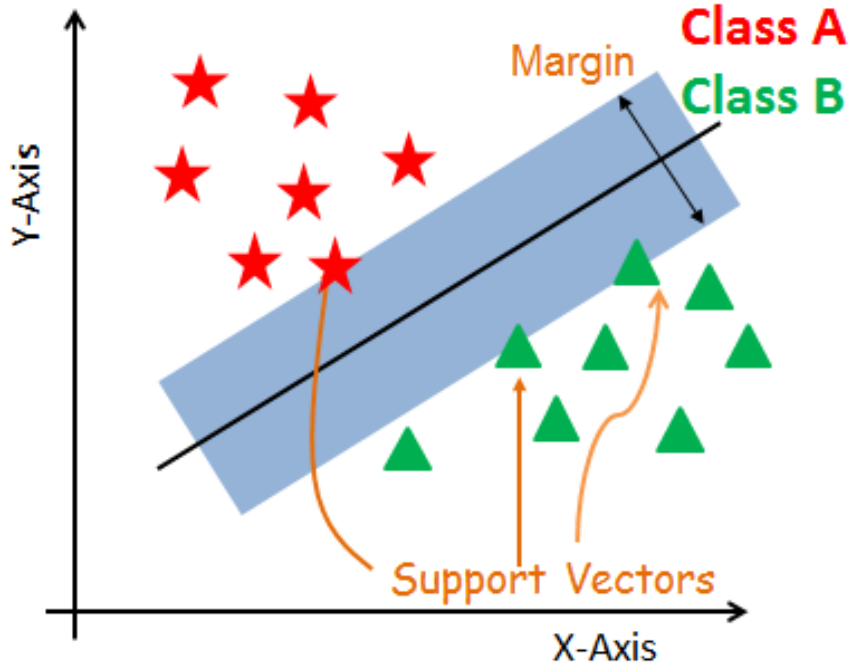
- SVM constructs a hyperplane in multidimensional space to separate different classes.
- SVM generates optimal hyperplane in an iterative manner, which is used to minimize an error.
- The core idea of SVM is to find a maximum marginal hyperplane(MMH) that best divides the dataset into classes.



Support Vector Machines

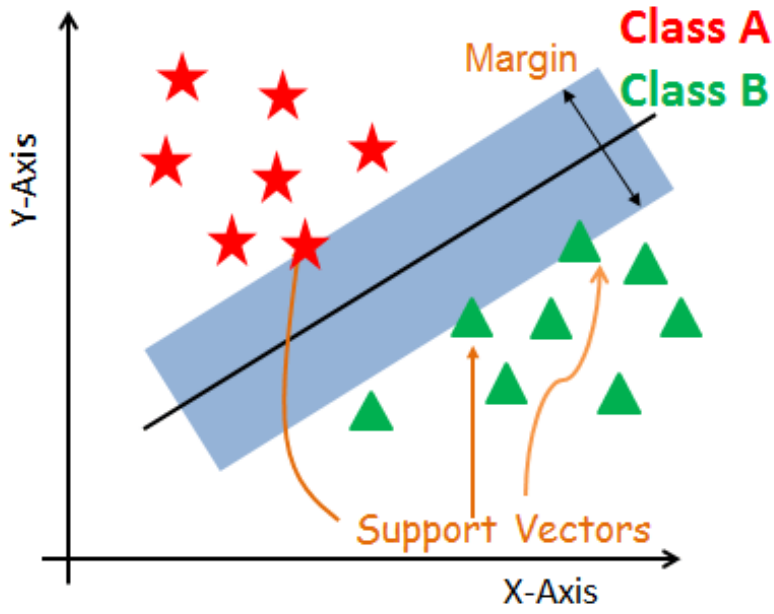
Support Vectors: the data points, which are closest to the hyperplane. These points will define the separating line better by calculating margins. These points are more relevant to the construction of the classifier.

Hyperplane:
A hyperplane is a decision plane which separates between a set of objects having different class memberships.



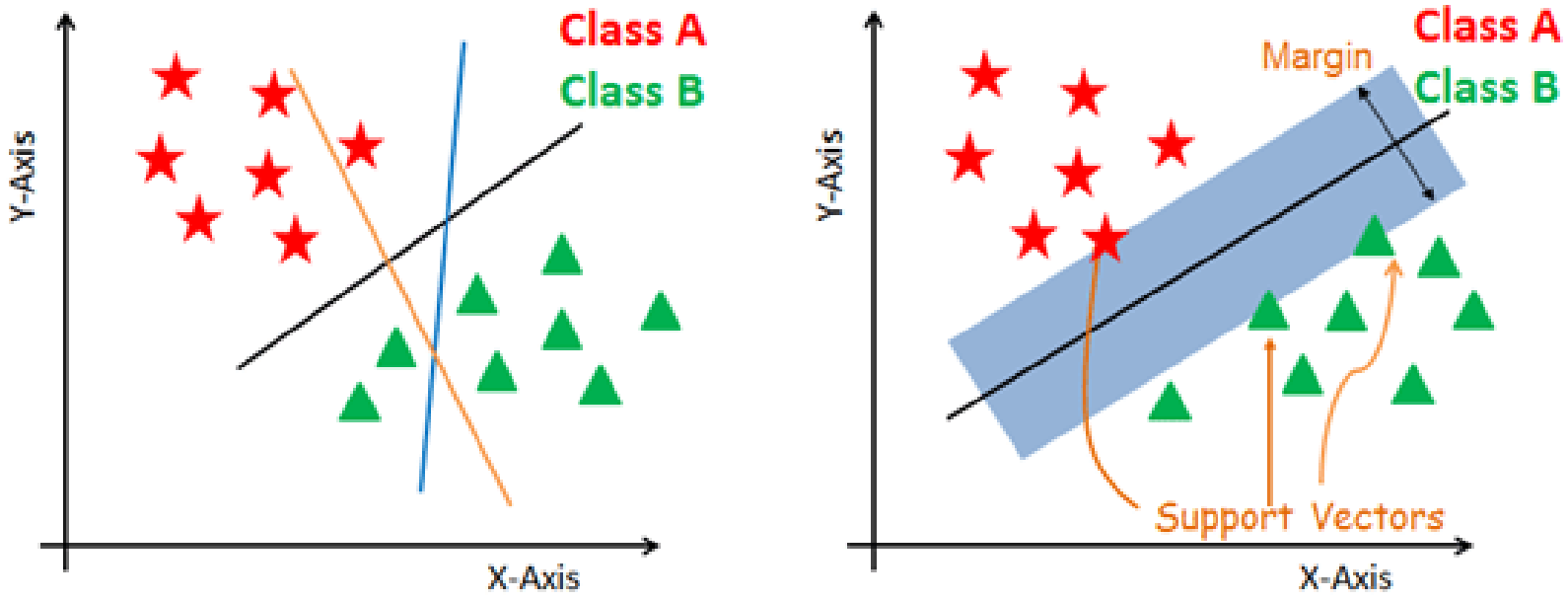
Support Vector Machines

Margin:
A margin is a gap between the two lines on the closest class points. This is calculated as the perpendicular distance from the line to support vectors or closest points. If the margin is larger in between the classes, then it is considered a good margin, a smaller margin is a bad margin.



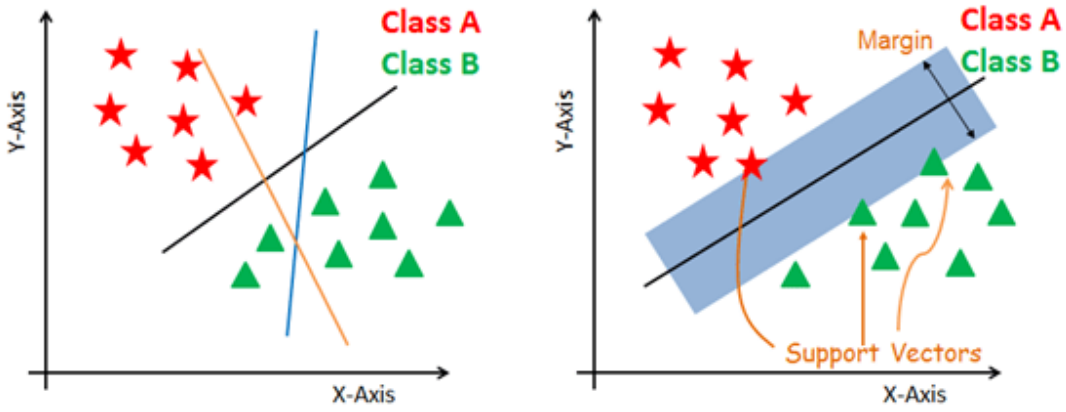
Support Vector Machines

- The main objective is to segregate the given dataset in the best possible way. The distance between the either nearest points is known as the margin.
- The objective is to select a hyperplane with the maximum possible margin between support vectors in the given dataset.



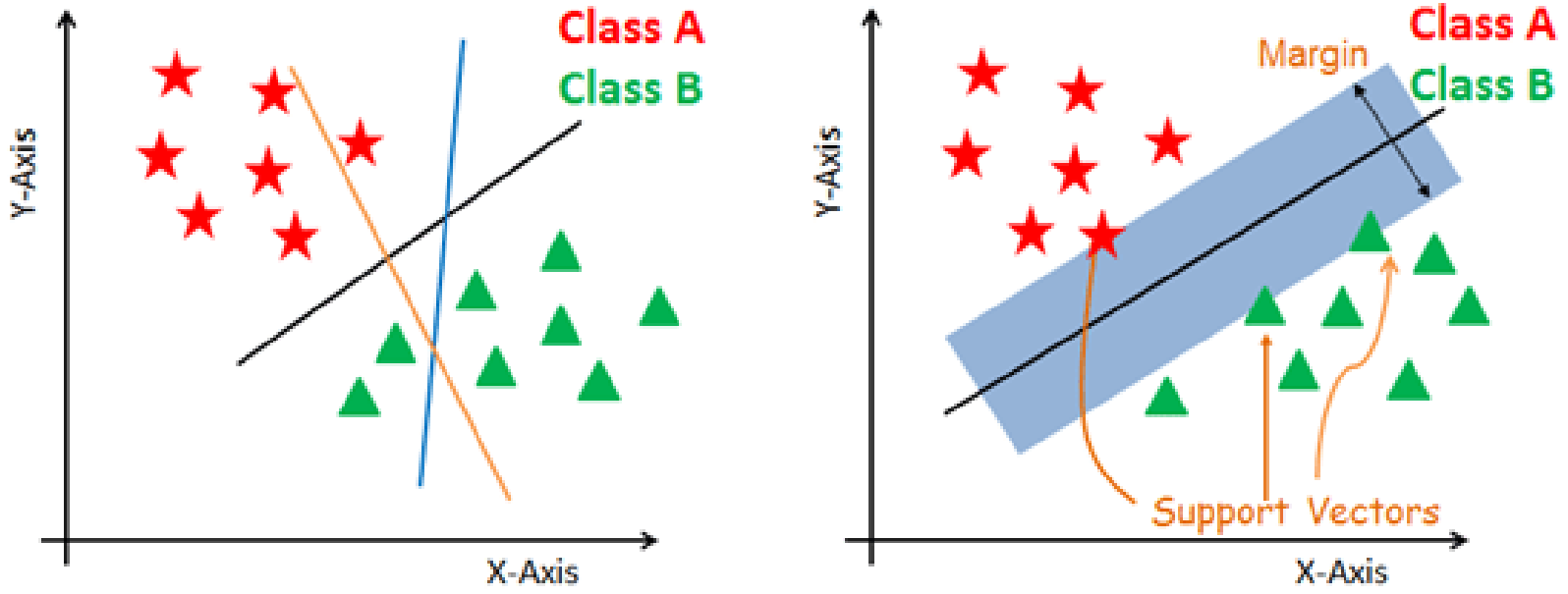
Support Vector Machines

- SVM searches for the maximum marginal hyperplane in the following steps:
  - Generate hyperplanes which segregates the classes in the best way. In the figure showing three hyperplanes black, blue and orange. Here, the blue and orange have higher classification error, but the black is separating the two classes correctly.



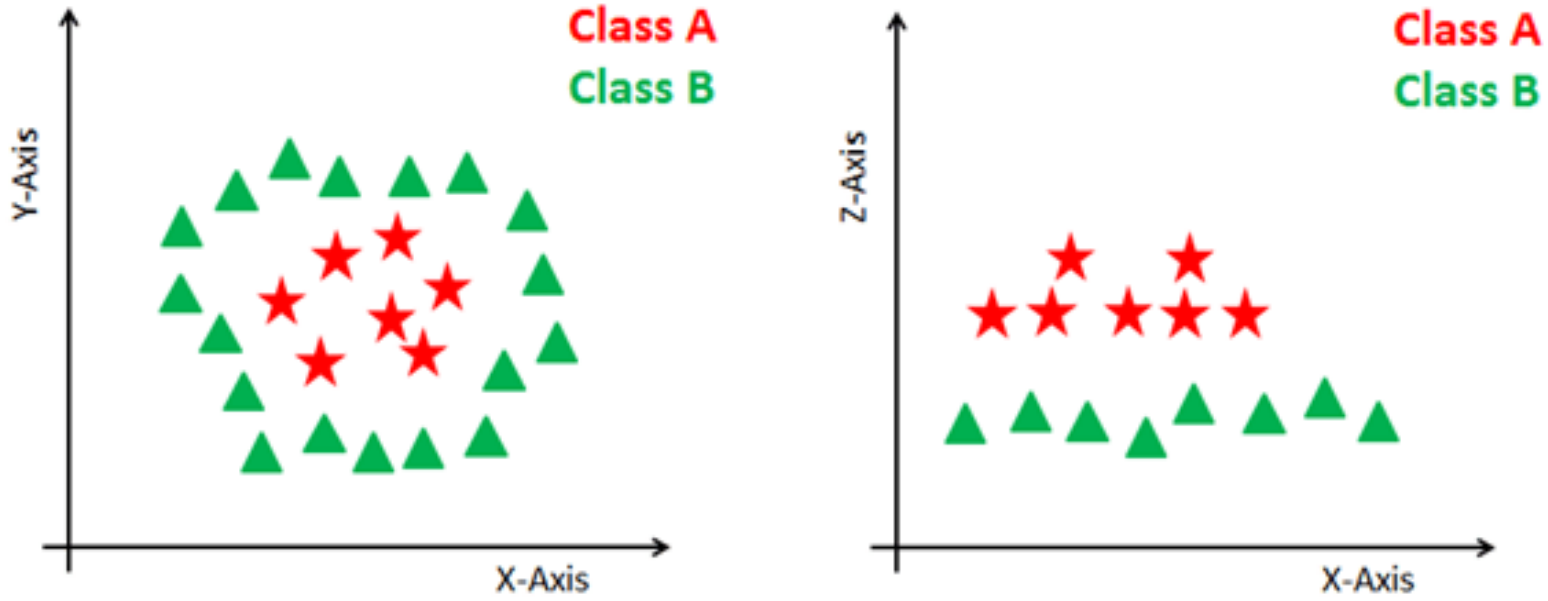
Support Vector Machines

- Select the right hyperplane with the maximum segregation from the either nearest data points as shown in the right-hand side figure.



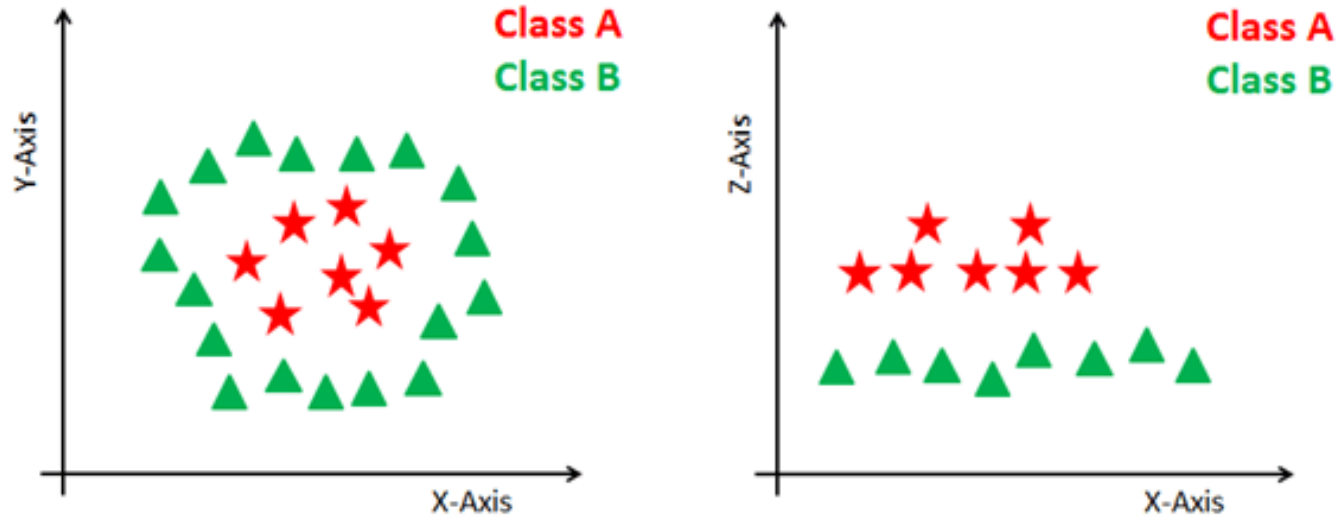
Support Vector Machines

- Some problems can't be solved using linear hyperplane, as shown in the figure below (left-hand side).



Support Vector Machines

- In such situation, SVM uses a kernel trick to transform the input space to a higher dimensional space as shown on the right.
- The data points are plotted on the x-axis and z-axis (Z is the squared sum of both x and y: z=x^2=y^2).
- Now you can easily segregate these points using linear separation.



Support Vector Machines

- The SVM algorithm is implemented in practice using a kernel. A kernel transforms an input data space into the required form. SVM uses a technique called the kernel trick.
- Here, the kernel takes a low-dimensional input space and transforms it into a higher dimensional space. In other words, you can say that it converts non-separable problem to separable problems by adding more dimension to it.
- It is most useful in non-linear separation problem. Kernel trick helps you to build a more accurate classifier.

SVM Kernels

- **Linear Kernel** A linear kernel can be used as normal dot product any two given observations. The product between two vectors is the sum of the multiplication of each pair of input values.
  - **K(x, xi) = sum(x * xi)**
- **Polynomial Kernel** A polynomial kernel is a more generalized form of the linear kernel. The polynomial kernel can distinguish curved or nonlinear input space.
  - **K(x,xi) = 1 + sum(x * xi)^d**
- **Radial Basis Function Kernel** The Radial basis function kernel is a popular kernel function commonly used in support vector machine classification. RBF can map an input space in infinite dimensional space.
  - **K(x,xi) = exp(-gamma * sum((x – xi^2))**
  - *Here gamma is a parameter, which ranges from 0 to 1. A higher value of gamma will perfectly fit the training dataset, which causes over-fitting. Gamma=0.1 is considered to be a good default value. The value of gamma needs to be manually specified in the learning algorithm.*

sklearn.svm.SVC

- class sklearn.svm.SVC(*,
  - C=1.0,
  - kernel='rbf',
  - degree=3,
  - gamma='scale',
  - coef0=0.0,
  - shrinking=True,
  - probability=False,
  - tol=0.001,
  - cache_size=200,
  - class_weight=None,
  - verbose=False,
  - max_iter=- 1,
  - decision_function_shape='ovr',
  - break_ties=False,
  - random_state=None)

SVM in Sci-kit Learn

C*float, default=1.0:* Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty.

kernel*{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'}, default='rbf':* Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape (n_samples, n_samples).

degree*int, default=3:* Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.

gamma*{'scale', 'auto'} or float, default='scale':* Kernel coefficient for 'rbf', 'poly' and 'sigmoid'. if gamma='scale' (default) is passed then it uses 1 / (n_features * X.var()) as value of gamma, if 'auto', uses 1 / n_features.

coef0*float, default=0.0:* Independent term in kernel function. It is only significant in 'poly' and 'sigmoid'.

shrinking*bool, default=True:* Whether to use the shrinking heuristic.

probability*bool, default=False:* Whether to enable probability estimates. This must be enabled prior to calling fit, will slow down that method as it internally uses 5-fold cross-validation, and predict_proba may be inconsistent with predict.

tol*float, default=1e-3:* Tolerance for stopping criterion.

SVM in Sci-kit Learn

cache_size*float, default=200:* Specify the size of the kernel cache (in MB).

class_weight*dict or 'balanced', default=None:* Set the parameter C of class i to class_weight[i]*C for SVC. If not given, all classes are supposed to have weight one. The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n_samples / (n_classes * np.bincount(y))

verbose*bool, default=False:* Enable verbose output. Note that this setting takes advantage of a per-process runtime setting in libsvm that, if enabled, may not work properly in a multithreaded context.

max_iter*int, default=-1:* Hard limit on iterations within solver, or -1 for no limit.

decision_function_shape*{'ovo', 'ovr'}, default='ovr':* Whether to return a one-vs-rest ('ovr') decision function of shape (n_samples, n_classes) as all other classifiers, or the original one-vs-one ('ovo') decision function of libsvm which has shape (n_samples, n_classes * (n_classes - 1) / 2). However, one-vs-one ('ovo') is always used as multi-class strategy. The parameter is ignored for binary classification.

break_ties*bool, default=False:* If true, decision_function_shape='ovr', and number of classes > 2, predict will break ties according to the confidence values of decision_function; otherwise the first class among the tied classes is returned. Please note that breaking ties comes at a relatively high computational cost compared to a simple predict.

random_state*int, RandomState instance or None, default=None:* Controls the pseudo random number generation for shuffling the data for probability estimates. Ignored when probability is False. Pass an int for reproducible output across multiple function calls.

1. Example IRIS Classification

```
dataLoaded = load_iris()
dataName = dataLoaded.filename[:-4]
print("Data Shape" , dataLoaded.data.shape)
print("Label Shape", dataLoaded.target.shape)
```

```
# Classification
classifier = svm.SVC()
methodName = type(classifier).__name__
classifier.fit(x_train, y_train)
predictions = classifier.predict(x_test)
```

1. Example IRIS Classification

```
Data Shape (150, 4)
Label Shape (150,)

Algorithm: SVC
Score : 0.9736842105263158

Confusion Matrix
[[13  0  0]
 [ 0 15  1]
 [ 0  0  9]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        13
           1       1.00      0.94      0.97        16
           2       0.90      1.00      0.95         9

    accuracy                           0.97        38
   macro avg       0.97      0.98      0.97        38
weighted avg       0.98      0.97      0.97        38
```

References

1 https://scikit-learn.org/
2 https://towardsdatascience.com/
3 McKinney, W. (2017). Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython 2nd Edition.
4 Albon, C. (2018). Machine Learning with Python Cookbook: Practical Solutions from Preprocessing to Deep Learning
5 Géron, A. (2017). Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems 1st Edition
6 Müller, A. C., Guido, S. (2016). Introduction to Machine Learning with Python: A Guide for Data Scientists
7 Burkov, A. (2019). The Hundred-Page Machine Learning Book.
8 Burkov, A. (2020). Machine Learning Engineering.
9 Goodrich, M.T., Tamassia, R., Goldwasser, M.H. (2013). Data Structures and Algorithms in Python, Wiley.
10 https://towardsdatascience.com
11 https://www.datacamp.com/
12 http://www.python-course.eu
13 https://developers.google.com/edu/python/
14 http://learnpythonthehardway.org/book/