EECS545 WN21 Machine Learning
Homework #3

Due date: 11:55pm, Tuesday Feb 22nd, 2022

Reminder: While you are encouraged to discuss problems in a small group (up to 5 people), you should write your own solutions and source codes independently. In case you worked in a group for the homework, please include the names of your collaborators in the homework submission. Your answers should be as concise and clear as possible. Please address all questions to `https://piazza.com/class/kxyu0owhjju7gu` with a reference to the specific question in the subject line (E.g. Homework 3, Q2(c): foobar).

Submission Instruction: You should submit both **solution** and **source code**. We will inspect your source code submission and run the code to check the results. Please be aware your points can be deducted if you don't follow the instructions listed below:

- Submit **writeup** to **Gradescope**: Your writeup should contain your answers to **all** questions (typed or hand-written). Include plots and results if asked. **For each subquestion, please select the associated pages from the pdf file. The graders are not required to consider pages other than the ones selected.**
  - Hand-writings that are very difficult to read may lead to some penalties. If you prefer hand-writing, please use scanners or mobile scanning apps that provide some correction for shading and projective transformations for better legibility; you should not just take photos and submit them.
- Code: Upload all codes to to **Gradescope**. We have underlined desired code output. Please do not upload any data files.

Code Submission Instructions Your solution to Q4 and Q5 should be written in a *single* python program `q4.py`, and `q5.py` respectively. The program should be runnable with the following command line: `python3 q4.py` (or `python3 q5.py`) and produce outputs into *standard output* and plots for *all* sub-problems. The program should read the data files relative to the current working directory: for example, `readMatrix('q5_data/MATRIX.TRAIN')`.

The program should print all necessary outputs (e.g. coefficients computed) into standard output (std-out). For plots, it is okay to save figures as multiple files (e.g. `q1-b.png`) as you want. There are no requirements on the filename or format, as long as it produces valid outputs. Be sure to include all outputs in your writeup.

Source Code Instruction: Your source code should run under an environment with following libraries:

- Python 3.6+
- Numpy (for implementations of algorithms)
- Matplotlib (for plots)
- Scikit-Learn (for Q5)

1 [10 points] MAP estimates and weight decay

Consider using a binary logistic regression model $p(y = 1|\mathbf{x};\mathbf{w}) = \sigma(\mathbf{w}^T\phi(\mathbf{x}))$ where σ is the sigmoid function, $\phi : \mathbb{R}^d \to \mathbb{R}^M$ is a feature transform. Assume that a training set $\{(\mathbf{x}^{(i)}, y^{(i)}); i = 1, ..., N\}$ with $\mathbf{x}^{(i)} \in \mathbb{R}^d, y^{(i)} \in \{0, 1\}$ is given as usual. The maximum likelihood estimate of the parameters $\mathbf{w}$ is given by

$$\mathbf{w}_{\mathrm{ML}} = \operatorname*{argmax}_{\mathbf{w}} \prod_{i=1}^{N} p(y^{(i)}|\mathbf{x}^{(i)};\mathbf{w}). \tag{1}$$

We regularize logistic regression by imposing a prior on the parameters. Suppose we choose the prior $\mathbf{w} \sim \mathcal{N}(0, \tau^2 I)$ (here, $\tau > 0$, and I is the $M \times M$ identity matrix), and obtain the MAP estimate of $\mathbf{w}$ as:

$$\mathbf{w}_{\mathrm{MAP}} = \operatorname*{argmax}_{\mathbf{w}} p(\mathbf{w}) \prod_{i=1}^{N} p(y^{(i)}|\mathbf{x}^{(i)};\mathbf{w}). \tag{2}$$

Prove that:

$$\|\mathbf{w}_{\mathrm{MAP}}\|_2 \leq \|\mathbf{w}_{\mathrm{ML}}\|_2. \tag{3}$$

2 [25 + 5 points] Direct construction of valid kernels

In class, we saw that by choosing a kernel $k(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T\phi(\mathbf{z})$, we can implicitly map data to a high dimensional space, and have the SVM algorithm work in that space. One way to generate kernels is to explicitly define the mapping ϕ to a higher dimensional space, and then work out the corresponding k.

However, in this question we are interested in direct construction of kernels. Suppose we have a function $k(\mathbf{x}, \mathbf{z})$ that gives an appropriate similarity measure (similar to inner product) for our learning problem, and consider plugging k into a kernelized algorithm (like SVM) as the kernel function. In order for $k(\mathrm{x}, \mathrm{z})$ to be a valid kernel, it must correspond to an inner product in some higher dimensional space resulting from some feature mapping ϕ. Mercer's theorem tells us that $k(\mathbf{x}, \mathbf{z})$ is a (Mercer) kernel if and only if for any finite set $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$, the matrix K is symmetric and positive semi-definite, where the square matrix $K \in \mathbb{R}^{N\times N}$ is given by $K_{ij} = k\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right)$.

Now here comes the question: Let k_1, k_2 be kernels over $\mathbb{R}^D \times \mathbb{R}^D$, let $a \in \mathbb{R}^+$ be a positive real number, let $f : \mathbb{R}^D \to \mathbb{R}$ be a real-valued function, let $\phi : \mathbb{R}^D \to \mathbb{R}^M$ be a function mapping from $\mathbb{R}^D$ to $\mathbb{R}^M$, let k_3 be a kernel over $\mathbb{R}^M \times \mathbb{R}^M$, and let $p : \mathbb{R} \to \mathbb{R}$ be a polynomial with positive coefficients.

For each of the functions k below, state whether it is necessarily a kernel. If you think it is a kernel, please prove it; if you think it is not, please give a counterexample.

(a) [2 points] $k(\mathbf{x}, \mathbf{z}) = k_1(\mathbf{x}, \mathbf{z}) + k_2(\mathbf{x}, \mathbf{z})$

(b) [2 points] $k(\mathbf{x}, \mathbf{z}) = k_1(\mathbf{x}, \mathbf{z}) - k_2(\mathbf{x}, \mathbf{z})$

(c) [2 points] $k(\mathbf{x}, \mathbf{z}) = ak_1(\mathbf{x}, \mathbf{z})$

(d) [2 points] $k(\mathbf{x}, \mathbf{z}) = -ak_1(\mathbf{x}, \mathbf{z})$

(e) [5 points] $k(\mathbf{x}, \mathbf{z}) = k_1(\mathbf{x}, \mathbf{z})k_2(\mathbf{x}, \mathbf{z})$

(f) [3 points] $k(\mathbf{x}, \mathbf{z}) = f(\mathbf{x})f(\mathbf{z})$

(g) [3 points] $k(\mathbf{x}, \mathbf{z}) = k_3(\phi(\mathbf{x}), \phi(\mathbf{z}))$

(h) [3 points] $k(\mathbf{x}, \mathbf{z}) = p\left(k_1(\mathbf{x}, \mathbf{z})\right)$

(i) [3 points] For this subproblem you don't have to check if k is a kernel or not. Instead, given a kernel $k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T\mathbf{z} + 1)^2$, find a feature map ϕ associated with $k(\mathbf{x}, \mathbf{z})$ such that $k(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T\phi(\mathbf{z})$. You may assume $D = 2$ for this subproblem.

(j) [5 points extra credit] Prove that the Gaussian Kernel, $k(\mathbf{x}, \mathbf{z}) = \exp\left(-\|\mathbf{x} - \mathbf{z}\|^2/2\sigma^2\right)$ can be expressed as $k(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T\phi(\mathbf{z})$, where $\phi(\cdot)$ is an infinite-dimensional vector. Specifically, find the closed form of an *infinite* dimensional feature vector ϕ.

[*Hint:* $\|\mathbf{x} - \mathbf{z}\|^2 = \mathbf{x}^T\mathbf{x} + \mathbf{z}^T\mathbf{z} - 2\mathbf{x}^T\mathbf{z}$. *It might be useful to consider Power Series:* $\exp(x) = \sum_{n=0}^{\infty} \frac{1}{n!}x^n$]

3 [20 points] Kernelizing the Perceptron

Consider a binary classification problem with labels $y \in \{-1, 1\}$. Assume that we have a (pre-processed) dataset $\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots (\mathbf{x}^{(N)}, y^{(N)})\}$ where $\mathbf{x}^{(n)} \in \mathbb{R}^{d+1}$ with $x_1^{(n)} = 1\ \forall n$. The *perceptron* is a linear classifier, given by $\text{sign}(h(\mathbf{x}; \mathbf{w}))$ where $h(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T\mathbf{x}$ and $\mathbf{w} \in \mathbb{R}^{d+1}$ denotes the weights of the perceptron classifier. The perceptron is trained via Algorithm 1 described below.

Essentially, in each iteration, the algorithm picks a random sample from the dataset $(\mathbf{x}^{(n)}, y^{(n)}) \in \mathcal{D}$ and checks if it is misclassified *i.e.,* $\text{sign}(h(\mathbf{x}^{(n)}; \mathbf{w}_t)) \neq y^{(n)}$. This is implemented by checking if $y^{(n)}h(\mathbf{x}^{(n)}; \mathbf{w}_t)$ is negative (Line 5). If a positive example is misclassified, then we update the weights with $+\mathbf{x}^{(n)}$. If a negative example is misclassified, then we update the weights with $-\mathbf{x}^{(n)}$ (Line 6). If the example is correctly classified, nothing is done. This process is repeated for T iterations.

Algorithm 1: Perceptron training algorithm

```
1 $\mathbf{w}_0 \leftarrow \mathbf{0}$;
2 for $t = 0$ to $T - 1$ do
3     Pick a random training example $(\mathbf{x}^{(n)}, y^{(n)})$ from $\mathcal{D}$ (with replacement)
4     $h \leftarrow \mathbf{w}_t^T\mathbf{x}^{(n)}$
5     if $y^{(n)}h < 0$ then
6         $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y^{(n)}\mathbf{x}^{(n)}$
7     end
8 end
9 return $\mathbf{w}_T$
```

Assume that T is large and $\mathcal{D}$ is linearly separable. In 1943, Frank Rosenblatt showed that this simple algorithm converges and outputs a $\mathbf{w}_T$ that is able to perfectly separate the training data $\mathcal{D}$ [1].

What if the dataset $\mathcal{D}$ is **not** linearly separable? In this problem, you'll show that we can *kernelize* the perceptron algorithm. This allows us to operate in a high-dimensional (possibly infinite-dimensional) feature space $\phi(\mathbf{x})$ where we can assume that the data is linearly separable.

Let $\phi : \mathbb{R}^{d+1} \to \mathbb{R}^M$ be a high-dimensional feature transform that is intractable to compute. Assume that $\mathcal{D}$ is separable in the feature space *i.e.,* $\mathcal{D}_\phi = \{(\phi(\mathbf{x}^{(1)}), y^{(1)}), \ldots (\phi(\mathbf{x}^{(N)}), y^{(N)})\}$ is linearly separable. Note that the perceptron classifier is now given by $\text{sign}(h(\phi(\mathbf{x}); \mathbf{w}))$ with $h(\phi(\mathbf{x}); \mathbf{w}) = \mathbf{w}^T\phi(\mathbf{x})$. Lastly, assume that we have a computationally-tractable kernel $\mathbf{k}$ such that $\mathbf{k}(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T\phi(\mathbf{x}')$.

(a) Define $\mathbf{\Phi} = \begin{bmatrix} \phi(\mathbf{x}^{(1)})^T \\ \vdots \\ \phi(\mathbf{x}^{(N)})^T \end{bmatrix} \in \mathbb{R}^{N \times M}$ as usual.

In this subproblem, we assume we that we use Algorithm 1 in the feature space for analysis. Thus, assume input dataset to Alg. 1 is $\mathcal{D}_\phi$.

(i) [3 points] Assuming $\mathbf{w}_t$ is of the form $\Phi^T\boldsymbol{\alpha}_t$ where $\boldsymbol{\alpha}_t \in \mathbb{R}^N$, show that $\mathbf{w}_{t+1}$ is also of the form $\Phi^T\boldsymbol{\alpha}_{t+1}$ for some $\boldsymbol{\alpha}_{t+1} \in \mathbb{R}^N$.

[1] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.335.3398&rep=rep1&type=pdf

(ii) [3 points] Show that all intermediate weights $\mathbf{w}_t$ where $0 \leq t \leq T$ (note that this includes the optimal weights $\mathbf{w}_T$), can be expressed as $\Phi^T \boldsymbol{\alpha}_t$ where $\boldsymbol{\alpha}_t \in \mathbb{R}^N$. *[hint: induction!]*

(b) Note that by means of the previous subproblem, we have showed that the weights of the perceptron are a linear combination of the features.

(i) [4 points] Observing your solution for part (a), can you provide an update rule for $\boldsymbol{\alpha}_{t+1}$ from $\boldsymbol{\alpha}_t$ and other known quantities. What is the maximum number of elements that differ between $\boldsymbol{\alpha}_t$ and $\boldsymbol{\alpha}_{t+1}$?

(ii) [4 points] Show that $h(\phi(\mathbf{x}^{(n)}), \mathbf{w}_t)$ can be expressed entirely in terms of $\boldsymbol{\alpha}_t$ and kernel $\mathbf{k}$ (doesn't involve ϕ or $\boldsymbol{\Phi}$).

(c) [6 points] Kernelize the perceptron algorithm. Write it exactly in the style of Alg. 1 *i.e.,* each line (wherever applicable) in Alg. 1 should be replaced with its "kernelized" version. The algorithm should be computationally tractable and thus must not contain any ϕ or $\boldsymbol{\Phi}$ terms.

You have trained the kernelized perceptron and are now given a new test data point $\mathbf{x}$; how would you classify it?

4 [30 points] Implementing Soft Margin SVM by Optimizing Primal Objective

Support Vector Machines (SVM) is a discriminative model for classification. Although it is possible to develop SVMs that do K class classifications, we are going to restrict ourselves to binary classification in this question, where the class label is either $+1$ (positive) or -1 (negative). SVM is not a probabilistic algorithm. In other words, in its usual construction, it does not optimize a probability measure as a likelihood. SVM tries to find the "best" hyperplane that maximally separates the positive class from the negative class.

Recall that the objective function for maximizing the soft margin is equivalent to the following minimization problem:

$$\begin{aligned} &\min_{\mathbf{w},b,\boldsymbol{\xi}} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i \\ &\text{subject to } y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) \geq 1 - \xi_i,\ \forall i = 1,\ldots,N \\ &\xi_i \geq 0,\ \forall i = 1,\ldots,N \end{aligned} \tag{4}$$

The above is known as the primal objective of SVM. Notice the two constraints on the slack variables ξ_i. It means that ξ_i must satisfy both of those conditions, while minimizing the sum of ξ_i 's times C. The constrained minimization is equivalent to following minimization involving the *hinge loss* term:

$$\min_{\mathbf{w},b} E(\mathbf{w},b), \quad E(\mathbf{w},b) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\max\left(0, 1 - y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right)\right) \tag{5}$$

You will be working with minimization of the above objective in this problem.

(a) [5 points] Briefly show that Eq.(4) and Eq.(5) are indeed equivalent optimization problems. **Clearly show your work for eliminating slack variables.**

(b) [6 points] Find the derivatives of the loss function $E(\mathbf{w}, b)$ with respect to the parameters $\mathbf{w}, b$. Show that:

$$\nabla_{\mathbf{w}} E(\mathbf{w},b) = \mathbf{w} - C\sum_{i=1}^{N}\mathbf{I}\left[y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) < 1\right] y^{(i)}\mathbf{x}^{(i)} \tag{6}$$

$$\frac{\partial}{\partial b}E(\mathbf{w},b) = -C\sum_{i=1}^{N}\mathbf{I}\left[y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) < 1\right] y^{(i)}, \tag{7}$$

where $\mathbf{I}[\cdot]$ is the indicator function. If $f(x) = \max(0, x)$, then assume that $\frac{\partial f}{\partial x} = \begin{cases} 1 & x > 0 \\ 0 & \text{otherwise} \end{cases}$

(c) [9 points] Implement the SVM algorithm using batch gradient descent. In previous assignments, you have implemented gradient descent while minimizing over one variable. Minimizing over two variables ($\mathbf{w}$, b) is not different. Both gradients are computed from current parameter values, and then parameters are simultaneously updated.[2]

Example pseudocode for optimizing $\mathbf{w}$ and b is given by Algorithm 2 below.

[2]Note: it is a common mistake to implement gradient descent over multiple parameters by updating the first parameter, then computing the derivative w.r.t second parameter using the updated value of the first parameter. In fact, updating one parameter then computing the gradient of the second parameter using the updated value of the first parameter, is a different optimization algorithm, known as Coordinate Descent.

Algorithm 2: SVM Batch Gradient Descent

```
1 $\mathbf{w}^* \leftarrow 0$;
2 $b^* \leftarrow 0$;
3 for $j = 1$ to NumIterations do
4     $\mathbf{w}_{grad} \leftarrow \nabla_{\mathbf{w}} E\left(\mathbf{w}^*, b^*\right)$;
5     $b_{grad} \leftarrow \frac{\partial}{\partial b} E\left(\mathbf{w}^*, b^*\right)$;
6     $\mathbf{w}^* \leftarrow \mathbf{w}^* - \alpha(j)\mathbf{w}_{grad}$;
7     $b^* \leftarrow b^* - 0.01\alpha(j)b_{grad}$;
8 end
9 return $\mathbf{w}^*$
```

The learning rate $\alpha(.)$ is now a function of time (iteration number) rather than a constant. This allows us to define a decaying learning rate as:

$$\alpha(j) = \frac{\eta_0}{1 + j\eta_0}$$

Throughout this question, set $\eta_0 = 0.5$ and the slack cost $C = 5$. Loading the file `q4_data.npy` loads the arrays `q4x_train`, `q4y_train`, `q4x_test`, and `q4y_test`. Run gradient descent over the training data 6 times, once for each of the NumIterations=$\{5, 50, 100, 1000, 5000, 6000\}$. For each run, please report your trained parameters $(\mathbf{w}, b)$ and the test classification accuracy. Note that the parameters are initialized with zeros and the learning rate for updating b^* is $0.01\alpha(j)$ (see Algorithm 2).

(d) [4 points] In stochastic gradient descent, we compute the gradients and update our parameters for every training example (rather than for every whole batch). To do stochastic gradient descent properly, we need to define a loss function per example (note that the loss function $E(\mathbf{w}, b)$ above is defined over the entire training set). We can rewrite the loss function above as:

$$\begin{aligned} E(\mathbf{w}, b) &= \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N} \max\left(0, 1 - y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right)\right) \\ &= \sum_{i=1}^{N}\left[\frac{1}{2N}\|\mathbf{w}\|^2 + C\max\left(0, 1 - y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right)\right)\right] \end{aligned}$$

Expressing the error function as one summation allows us to define an error term per training example like:

$$E^{(i)}(\mathbf{w}, b) = \frac{1}{2N}\|\mathbf{w}\|^2 + C\max\left(0, 1 - y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right)\right)$$

$$\text{then, } E(\mathbf{w}, b) = \sum_{i=1}^{N} E^{(i)}(\mathbf{w}, b)$$

Now derive the gradients $\nabla_{\mathbf{w}} E^{(i)}(\mathbf{w}, b)$ and $\frac{\partial}{\partial b} E^{(i)}(\mathbf{w}, b)$.

(e) [6 points] Implement the SVM algorithm using stochastic gradient descent (SGD). The pseudo-code looks like:

Algorithm 3: SVM Stochastic Gradient Descent

```
$\mathbf{w}^* \leftarrow 0$;
$b^* \leftarrow 0$;
**for** $j = 1$ *to NumIterations* **do**
    **for** $i = 1$ *to* $N$ **do**
        $\mathbf{w}_{grad} \leftarrow \nabla_{\mathbf{w}} E^{(i)}(\mathbf{w}^*, b^*)$;
        $b_{grad} \leftarrow \frac{\partial}{\partial b} E^{(i)}(\mathbf{w}^*, b^*)$;
        $\mathbf{w}^* \leftarrow \mathbf{w}^* - \alpha(j)\mathbf{w}_{grad}$;
        $b^* \leftarrow b^* - 0.01\alpha(j)b_{grad}$;
    **end**
**end**
**return** $\mathbf{w}^*$
```

Use the same $\alpha(.), \eta_0, C$ as part (b). Run your implementation of SVM Stochastic Gradient Descent over the training data 6 times, once for each of the NumIterations$= \{5, 50, 100, 1000, 5000, 6000\}$. For each run, report your trained parameters $(\mathbf{w}, b)$ and the test classification accuracy.

5 [15 points] SciKit-Learn SVM for classifying SPAM

Recall Q4 from HW2. We will repeat the same problem using SciKit-Learn SVM, and compare naive Bayes and SVM.

NOTE: For your convenience, this is the same information about the data from Q4 in HW2.

In our data, the text emails have been pre-processed so that they can be used for naive Bayes. The pre-processing ensures that only the email body and subject remain in the dataset; email addresses (EMAILADDR), web addresses (HTTPADDR), currency (DOLLAR) and numbers (NUMBER) were also replaced by the special tokens to allow them to be considered properly in the classification process. (In this problem, we'll going to call the features "tokens".)

We have done the feature extraction works for you, so you can just load the data matrices (called document-term matrices in text classification) which contain all the data. In a document-term matrix, the i^{th} row represents the i^{th} document/email, and the j^{th} column represents the j^{th} distinct token. Thus, the (i, j) entry of this matrix represents the number of occurrences of the jth token in the ith document.

For this problem, we chose the set of tokens (vocabulary) to only contain the medium frequency tokens, as the tokens that occur too often or too rarely do not have much classification value. (Examples: tokens that occur very often are terms like "the," "and," and "of," which occur in any spam and non-spam emails.) Also, terms were stemmed using a standard stemming algorithm; basically, this means that "price," "prices" and "priced" have all been replaced with "price," so that they can be treated as the same token.

Since the document-term matrix is sparse (has lots of zero entries), we store it in an efficient format to save space. We provide a starter code **q5.py** which contains the **readMatrix** function that reads in the document-term matrix, the correct class labels for all emails, and the full list of tokens.

(a) [**5 points**] Implement an SVM classifier with **linear kernel** for spam classification using the corresponding package from SciKit-Learn[3]. You may need to run `pip install scikit-learn` to install the package.

You should use the **LinearSVC** class[4] in this package (already imported in **q5.py**) to train your parameters with the training dataset **MATRIX.TRAIN**. Then use these parameters to classify the test dataset **MATRIX.TEST** and report the error using the **evaluate** function. Implement your code in **q5.py**.

(b) [**6 points**] Repeat part (a), but with training sets of size ranging from 50, 100, 200, . . . , up to 1400, by using the files `MATRIX.TRAIN.*`. Plot the test error each time (use `MATRIX.TEST` as the test data) to obtain a learning curve (test set error v.s. training set size). Also, report the number of support vectors for each of the learned models. Which training set size gives the best test set error?

(c) [**4 points**] Reference instructor results for Q4 in HW2 (see Canvas → Homework → HW2 → solution → hw2_sol.pdf). How do naive Bayes and support vector machines compare (in terms of test error) as a function of the training set size?

Credits

Some questions adopted/adapted from http://www.stanford.edu/class/cs229/ and from Bishop PRML.

[3]See `https://scikit-learn.org/stable/` for more information about the package.

[4]See `https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html` for more information about the **LinearSVC** class.