

COMP90051 Statistical Machine Learning

Project 2 Description

Due date: 6:00pm Thursday, 19 October 2023

Weight: 25%; forming combined hurdle with Proj1

Copyright statement: All the materials of this project—including this specification and code skeleton—are copyright of the University of Melbourne. These documents are licensed for the sole purpose of your assessment in COMP90051. You are not permitted to share or post these documents online.

Academic misconduct: You are reminded that all submitted Project 2 work is to be your own individual work. Automated similarity checking software will be used to compare all submissions. It is University policy that academic integrity be enforced. For more details, please see the policy at <http://academichonesty.unimelb.edu.au/policy.html>. Academic misconduct hearings can determine that students receive zero for an assessment, a whole subject, or are terminated from their studies. You **may not** use software libraries or code snippets found online, from friends/private tutors, or anywhere else. You can only submit your own work.

In this project, we return to the classic: the *support vector machine* (SVM). Although you have learned the deep learning techniques in recent lectures, SVM is still a powerful and theoretically guaranteed framework for classification, formulated around the idea of maximum margin of separation which ensures strong generalisation performance. After this project, you can understand the code framework of maximum-margin algorithms better, enabling you to code deep-network-based SVM on your own.

We have seen the equivalent primal and dual formulations of the SVM in lectures, and how these lead to different time complexities of inference, as well as the ability to incorporate kernels when using the dual. In this project, you will work **individually** to implement several SVM algorithms. These go beyond what was covered in class, and includes real research papers that you can read and understand. By the end of the project you should have developed:

ILO1. A deeper understanding of the SVM and its primal and dual formulation;


ILO2. An appreciation of how SVMs are applied; and

ILO3. Demonstrable ability to implement ML approaches in code.

Overview

The project consists of three tasks.


1. Solving the primal problem with stochastic online updates [7 marks]
2. Solving the dual problem with Kernel-Adatron [10 marks]
3. Incorporating kernels [8 marks]

All tasks are to be completed in the provided Python Jupyter notebook `proj2.ipynb`.¹ Detailed instructions for each task are included in this document. These tasks may require you to consult one or more academic papers. We provide helpful pointers (with margin symbol ) throughout this project spec to guide your reading and to correct any ambiguities.

¹We appreciate that while some have entered COMP90051 feeling less confident with Python, many workshops so far have exercised and built up basic Python and Jupyter knowledge. Both are industry standard tools for the data sciences.

SVM algorithms. The project’s tasks require you to implement SVM training algorithms by completing provided skeleton code in `proj2.ipynb`. All of the SVM training algorithms must be implemented as sub-classes of the provided base `SVM` class. This ensures all SVM training algorithms inherit the same interface, and your implementations must conform to this interface. Your implementations **must** conform to this interface. You may implement functionality in the base `SVM` class if you desire—*e.g.*, to avoid duplicating common functionality in each sub-class. Your classes may also use additional private methods to better structure your code. And you may decide how to use class inheritance.

Python environment. You must use the Python environment used in workshops to ensure markers can reproduce your results if required. We assume you are using Python ≥ 3.8 , numpy $\geq 1.19.0$, scikit-learn $\geq 0.23.0$ and matplotlib $\geq 3.2.0$.

Other constraints. You may not use functionality from external libraries/packages, beyond what is imported in the provided Jupyter notebook highlighted here with margin marking . You must preserve the structure of the skeleton code—please only insert your own code where specified. You should not add new cells to the notebook. You may discuss the SVM learning slide deck or Python at a high-level with others, but do not collaborate with anyone on direct solutions. You may consult resources to understand SVM conceptually, but do not make any use of online code whatsoever. (We will run code comparisons against online partial implementations to enforce these rules. See ‘academic misconduct’ statement above.)

Submission Checklist

You must complete all your work in the provided `proj2.ipynb` Jupyter notebook. When you are ready to submit, follow these steps. You may submit multiple times. We will mark your last attempt. **Hint** (💡): *It is a good idea to submit early as a backup. Try to complete Task 1 in the first week and submit it; it will help you understand other tasks and be a fantastic start!*

1. Restart your Jupyter kernel and run all cells consecutively.
2. Ensure outputs are saved in the `ipynb` file, as we may choose not to run your notebook when grading.
3. Rename your completed notebook from `proj2.ipynb` to `username.ipynb` where `username` is your university central username².
4. Upload your submission to the Project 2 Canvas page.

Marking

Projects will be marked out of 25. Overall approximately 50%, 25%, 25% of available marks will come from correctness, code structure & style, and experimentation. Markers will perform code reviews of your submissions with **indicative** focus on the following. We will endeavour to provide (indicative **not exhaustive**) feedback. The purpose of our feedback is to help you learn, and not to justify marks.

1. **Correctness:** Faithful implementation of the algorithm as specified in the reference or clarified in the specification with possible updates in the Canvas changelog. It is important that your code performs other basic functions such as: raising errors if the input is incorrect, working for any dataset that meets the requirements (*i.e.*, not hard-coded), etc.

²Canvas/UniMelb usernames look like `fenliu`, not to be confused with email such as `feng.liu`.

2. **Code structure and style:** Efficient code (*e.g.*, making use of vectorised functions, avoiding recalculation of expensive results); self-documenting variable names and/or comments; avoiding inappropriate data structures, duplicated code and illegal package imports.
3. **Experimentation:** Each task you choose to complete directs you to perform some experimentation with your implementation, such as evaluation, tuning, or comparison. You will need to choose a reasonable approach to your experimentation, based on your acquired understanding of the SVM learners.

Late submission policy. Late submissions will be accepted to 4 days at -2.5 penalty per day or part day. Weekends and holidays will also be counted towards the late penalty.

Task Descriptions

Task 1: Primal problem with stochastic gradient update [7 marks total]

Your first task is to implement a soft-margin SVM in its primal formulation, using *stochastic gradient descent* (SGD) for training. This is an online algorithm which is similar to the perceptron training algorithm (see week 6 workshop), where training involves an outer loop run several times, and an inner loop over each instance in the training set, where the parameters are updated using the gradient of the loss for a single example. The stopping criterion here for the outer is to finish `iterations` iterations.

At each step, the algorithm will update the weights \mathbf{w} and bias b , with update rules:

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t - \eta \nabla_{\mathbf{w}} L_i(\mathbf{w}, b) \\ b_{t+1} &= b_t - \eta \nabla_b L_i(\mathbf{w}, b),\end{aligned}$$

where η is the learning rate, and $L_i(\mathbf{w}, b)$ is the loss function for the i^{th} example. You will first have to figure out the per-instance loss, $L_i(\mathbf{w}, b)$, based on the total loss of the training set,

$$L(\mathbf{w}, b) = \sum_{i=1}^n \max(0, 1 - y_i(\mathbf{w}'\mathbf{x}_i + b)) + \frac{1}{2}\lambda\|\mathbf{w}\|$$

and then derive the relevant gradients needed for the SGD updates. You will then need to implement a `PrimalSVM` class with the following functions:

1. `fit`, which trains SVM using SGD as described above
2. `predict`, which classifies a new instance \mathbf{x} based on sign of $\mathbf{w}'\mathbf{x} + b$
3. `__init__`, to retain state or precompute values to support the above methods.

Experiments. Once your `PrimalSVM` class is implemented, it is time to perform some basic experimentation.

(a) Include and run an evaluation on the given dataset:

```
psvm = PrimalSVM(eta = 0.1, lambda0 = 0.1)
psvm.fit(X,y,iterations = 100)
print(f"Accuracy is {round(psvm.evaluate(X,y),4)}")
psvm.visualize(X,y)
```

Print the training accuracy and plot the dataset and decision surface.

(b) Now you will need to tune your `PrimalSVM`'s hyperparameters. Based on your understanding of the λ value, test a range of values (consider equally spaced points in logarithmic space) and run experiments to find the best value.³ You must set `eta` to 0.1 and `iterations` to 100. Output the result of this strategy—which could be a graph, number, etc. of your choice.

Task 2: Dual SVM formulation with stochastic online update [10 marks total]

In this task, you are to implement the dual formulation of the soft-margin SVM. Your implementation should be done in the provided `DualSVM` skeleton code, and be based on SGD training algorithm in Table 7.1 of Chapter 7 of the following book:

Cristianini, N., & Shawe-Taylor, J. (2000). An Introduction to Support Vector Machines and Other Kernel based Learning Methods (pp. 125-148). Cambridge: Cambridge University Press.⁴

You will need to implement the following functions:

1. `__init__`

2. `fit` with the training algorithm described above. For the training loop the same stopping criterion as Task 1, that is stopping after a given number of iterations (*e.g.*, 100). The algorithm listed in the paper assumes a fixed bias, however for this project, you should implement `get_bias` function to compute b . The standard method for doing so uses the equation:

$$b = y_i - \sum_{j=1}^n \alpha_j y_j k(\mathbf{x}_i, \mathbf{x}_j)$$

for an arbitrary instances i with $0 < \alpha_i < C$. However, this assumes training has converged and the KKT conditions have been satisfied, which may not be true with SGD training. For this reason, you should compute the bias estimate for all candidate instances i , and use the average estimate as the bias.⁵

3. `predict`, which should classify a new instance \mathbf{x} based on sign of $b + \sum_{j=1}^n \alpha_j y_j k(\mathbf{x}_j, \mathbf{x})$, where k is the kernel function (linear for now, but your implementation should be general to support Task 3, below).

4. `primal_weights`, which computes the equivalent primal weights using a linear kernel, *i.e.*, $\mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i$.

Experiments. Evaluate your model with a given dataset, use $\eta = 0.1$, $C = 100$, `iterations` = 100 and a linear model. Perform the following experiments, with `dsvm = DualSVM(eta = 0.1)` and keep the other hyperparameter values unchanged except C value:

(a) Tune the value of C , to optimise for training accuracy.

(b) Report the equivalent weights for the dual solution, and compare to the primal trained with equivalent loss. Hint: consider how C and λ are related, such that you compare an equivalent model. Test to check the

³Best here means the setting with the best training accuracy. This is used here as a sanity check to test whether the model can fit the data, but of course it is not a good measure of the model's generalisation accuracy.

⁴Access via e-book in the library catalogue, *e.g.*, [using this link](#) and the 'Get access' option.

⁵This isn't always the most reliable method of setting the bias. It could be improved by optimising the training objective for the best bias scalar given α . But for the purpose of the project, we will keep with the simple method as described.

weights are similar, using the norm of the difference.

(c) Identify the support vectors and point where $\alpha = C$. Display your results either on a plot or by printing the index of the relevant instances.

Task 3: Kernel [8 marks total]

In this task, you are going to use the kernel trick. Specifically, you need to use the kernelised version of SVM in combination with a few different kernels. **In this section, you cannot use any libraries other than ‘numpy’.**

First, implement polynomial and RBF kernels. The linear kernel is simply a dot product of its inputs and has been provided. Polynomial and RBF kernels should be implemented as defined in the lecture slides. For the polynomial kernel, you should include a scalar offset and degree, *i.e.*, $(\mathbf{u}'\mathbf{v} + c)^d$, and for the RBF include a width parameter, σ . Your kernels should support evaluation with vector (instance) or matrix (dataset) inputs in both the first and second position, as documented in the `__call__` function, and return a scalar, vector, or matrix as appropriate.

Experiments. Here run `dsvm` with the three different kernels, you should use $\eta = 0.1$, $C = 100$ and `iterations = 100`. You don’t need to show how you tune the hyperparameters in this task, but you should run experiments with reasonable values. Use the `visualize` function to display the decision boundary for each of the three kernels. In a few sentences, explain which kernel you think is best suited to this problem, and why.

Task 4: Sequential minimal optimization [optional, NOT required]

In the above sections, you have implemented SVM on your own. In this section, you can read the following paper to know sequential minimal optimization (SMO), a fast algorithm to train SVM:

Platt, J. (1998). Sequential minimal optimization: A fast algorithm for training support vector machines.

See in particular the pseudocode in §2.5 of this paper. You can also find useful information from the book cited in Task 2. The algorithm is somewhat complex, so we leave it as reading material instead of tasks to be completed in the assignment.