RESEARCH

Assignment 4

Suresh Kumar Choudhary, Emilio Kuhlmann and Sofya Laskina

Full list of author information is available at the end of the article

Abstract

The goal of the project: Implement soft-margin Support Vector Machine algorithm, need to train and test on Iris Dataset and clean the Dataset provided for excercise-2.

The main result of the project: The soft margin linear SVM classifier has been built and tested on Iris dataset and cleaned the dataset for excercise2

Personal key learnings: We used the solvers first time and understood the vital aspects of the algorithm

Estimated working time::8

Project evaluation:1

Number of words: 1324

1 Scientific Background

Support vector machines are a supervised learning method used to perform binary classification on data. In the SVM algorithm [1], we take each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well. Support Vectors are simply the co-ordinates of individual observation. The original support vector machines (SVMs) were invented by Vladimir Vapnik [2], were designed to address a longstanding problem with logistic regression.

Results

Soft-Margin SVM Task

Task 1

We implemented A linear soft-margin SVM with the help of the quadratic equation solving python-library CVXOPT. Note that we did not use the suggested solver QPSOLVER as we liked CVXOPT interface better. Recall that to implement a soft-margin SVM we need to solve the following quadratic equation:

$$\min_{w\in\mathcal{H},\xi\in\mathbb{R}^m} \tau(w,\xi) = \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i,$$

subject to

$$y_i(\langle x_i, w\rangle) + b) \geq 1 - \xi_i, \quad \text{with} \quad \xi_i \geq 0,\ i = 1,\ldots,m.$$

Since we are solving this optimization problem using the CVXOPT library in python so we need to match the solver's API which, according to the documentation is of

the form:

$$\min \frac{1}{2} x^T P x + q^T x,$$

subject to

$$Gx \leq h, Ax = b$$

The equivalent Standard form of SVM quadratic optimization problem is:

$$\min \frac{1}{2} \alpha^T H \alpha + -1^T \alpha,$$

subject to

$$-\alpha_i \leq 0, \alpha_i \leq C, y^T \alpha = b$$

We are now ready to convert our numpy arrays into the cvxopt format, using the same notation as in the documentation this gives:

$$H_{i,j} = y^i y^j < x^{(i)} x^{(j)} >$$

- P:= H a matrix of size mxm
- q:=-1 a vector of size m×1
- G:=[-diag[1],diag[1]] a stacked diagonal matrix of -1s of size m×m and 1s of size mxm
- h:=[vector(0),vector(C)] a stacked vector of zeros of size m×1 and C of size mx1
- A:=y the label vector of size m×1
- b:=0 a scalar

This was essentially tweaking the standard form of the hard-margin linear SVM to include the slack variables ξ_i. We then "unwrapped" the solution of the standard form to get the vector $w \in \mathbb{R}^{m-1}$ and $b \in \mathbb{R}$ that together span the separating hyperplane by which the decision function decides. Technically, just w spans a linear subspace and adding b makes it an affine subspace. This concludes the training part of the soft-margin SVM. The decision function was a simple matter of implementing the one-liner

$$f_{w,b}(x) = sgn(\langle w, b \rangle + b).$$

As a toy Problem we first experimented with a dataset provided by the Landgraf team, our implementation worked rather well, reaching an accuracy of approximately 95%. For a nice visual representation this, check out our figure 1 .

Task 2

We then moved onto the the Iris dataset(150 measurements).This is not a binary classification problem, there are three labels: "setosa", "versicolor" and "virginica".

Since we do not have a multi-class algorithm, we have simply relabel the measurements into "setosa" and "non setosa", thus gaining a binary problem. So, for it we created two dataset one with class label "setosa" and "All remaining" named as Dataset1 and other with class label "versicolor" and "All remaining" named as Dataset2.

We tried our soft-margin classifier on the dataset1 (test size was 30%) and concluded that the algorithm worked better the harder the margin. We calculated F-score (weighted avaerage of precision and recall) as accuracy measure as dataset was not balanced after relabel. With all C values, Where C is used to control slack variables, we achieved an accuracy of 100%, that can be visualized in figure 2 . For confusion matrices and classificaton report, you can refer figure 6 and 4 respectively. In similar way, we tried our soft-margin classifier on the dataset2. With $C = 0.01$, we achieved an accuracy of approx. 83%, that can be visualized in figure 3 . For confusion matrices and classificaton report, you can refer figure 7 and 5 respectively.

Task 3

To make a multi-class classifier we could simply make a one vs the rest classification(binary classification) for all the classes. E.g. for four classes: 1 vs 2-3-4, then 2 vs 1-3-4 then 3 vs 1-2-4 and finally 4 vs 1-2-3. and can pick the class with highest confidence for prediction purpose among these binary classification models.

Data cleansing task

We performed data cleansing task in python using Pandas package. After reading and looking at the relatively short table we were able to discover a few errors with a naked eye:

1. line 24 had another delimiter, rather than ',’. We fixed this by accessing the row directly(since it was just one such a row in a data set) with pandas ".at" method and setting it to the value of first column separated by delimiter used in this raw, namely':'
2. columns actor and firstname together with lastname seem redundant. We first checked for those rows, where name + space + lastname does not equal the column actor to check for possible pseudo , for instance. There were only three rows where columns did not much and they were all missing values or typos. We fetched this columns from the column actor.
3. all the numbers in row 18 were enclosed in quotes, we replace the quotes by empty string with 'str.replace' method
4. since in each numeric column there was a string present, this column were read in in 'string' type. We cast them to 'int' or 'float' depending on the appearing values with 'astype' method
5. column Price has different notations. There is also one nan. We did not touch nan value, then converted all values with an 's' at the end of the word into 'yes', all the rest into 'no'. After counting occurrences we realized, nan should also be a 'no' so we changed that.
6. after calculating sum column-wise turned out our "Gross" column was faulty. Although we did not quite understand the meaning of a column, it was clear we should only keep absolute values.

7 finally we checked for duplicates in data, which did not occur, dropped column "actor", and sorted the data frame, so that firstname and lastname are the first columns to appear there.

Figure 1: Soft-Margin Linear SVM Classification

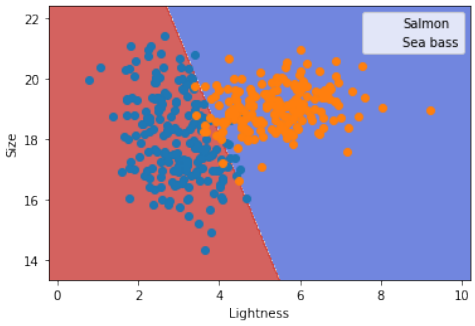
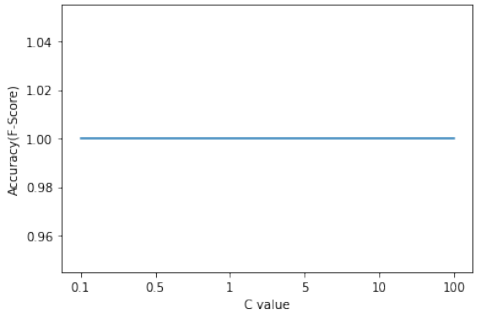


Figure 2: Plot for C value vs F1-Score(Accuracy) for Target Class Setosa vs Non Setosa



Discussion

Personally we would have like more guidance on how to get the standard form for the Soft-Margin linear SVM. This was quite tricky to figure out, and would have required little extra work from the professor.

code: To write code, We took help from [3]

Author details

References

1. Boser, B.E., Guyon, I.M., Vapnik, V.N.: A training algorithm for optimal margin classifiers. In: Proceedings of the Fifth Annual Workshop on Computational Learning Theory, pp. 144–152 (1992)
2. Chapelle, O., Vapnik, V.: Model selection for support vector machines. Advances in neural information processing systems **12**, 230–236 (1999)
3. SVM. https://pythonprogramming.net/soft-margin-kernel-cvxopt-svm-machine-learning-tutorial/

Figure 3: Plot for C value vs F1-Score(Accuracy) for Target Class Versicolor vs Non Versicolor

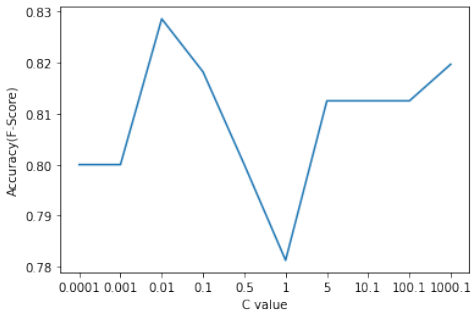


Figure 4: Precision Recall Measures for SVM Model on Target class Setosa vs others with Test DataSet

```
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()


# argmax returns the index of the max value in a row
cm = confusion_matrix(y_test, y_pred)

plot_confusion_matrix(cm, cm_plot_labels, title='Confusion Matrix')
```

```
              precision    recall  f1-score   support

      Setosa       1.00      1.00      1.00        15
  non Setosa       1.00      1.00      1.00        30

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

Figure 5: Precision Recall Measures for SVM Model on Target class Versicolor vs others with Test DataSet

```
# Generate a classification report
cm_plot_labels = ['versicolor', 'non versicolor']
# For this to work we need y_pred as binary labels not as probabilities
#y_pred_binary = np.where(predictions > 0.5, 1, 0)

report = classification_report(y_test, y_pred, target_names=cm_plot_labels)

print(report)

# argmax returns the index of the max value in a row
cm = confusion_matrix(y_test, y_pred)

plot_confusion_matrix(cm, cm_plot_labels, title='Confusion Matrix')
```

```
                precision    recall  f1-score   support

    versicolor       0.80      0.27      0.40        15
non versicolor       0.72      0.97      0.83        30

      accuracy                           0.73        45
     macro avg       0.76      0.62      0.61        45
  weighted avg       0.75      0.73      0.69        45
```

Figure 6: Confusion Matrix for SVM Model on Target class Setosa vs others with Test DataSet

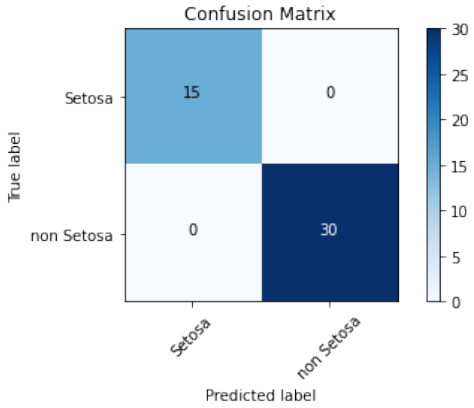


Figure 7: Confusion Matrix for SVM Model on Target class Versicolor vs others with Test DataSet