Real World Machine Learning

Kevin Fu

October 2019

1 Introduction

Last week, we saw how support vector machines use hyperplanes to solve binary classification problems of linearly-separable data. We also saw that with soft-margins, SVMs can handle error; with kernels, SVMs can deal with non-linearly separable data; and with one-vs-rest (OVR) or one-vs-one (OVO), we can create multi-class SVMs. In this lecture, we'll introduce some useful packages to code an SVM, and give hints on the SVM competition.

2 Setup

The first step to tackling a machine learning problem is getting a sense of the data. For 2D data, graphing packages like `matplotlib` or `seaborn` can be useful. For the SVM competition, with its eleven features, using Microsoft Excel or Apple's Numbers to view the input `.csv` is good for figuring out which features are useful and which are not. However, once we start using `scikit-learn` for the SVM itself, we'll need to get this `.csv` in a form Python can work with. The Data I/O code from the Decision Trees competition is good, but using `pandas` with a Jupyter notebook is better.

2.1 Conda

To use `pandas` and Jupyter notebooks, we'll need to install Conda. Conda is an open-source Python package manager designed for data scientists. If you've pip installed a package before, you've used a package manager like Conda. (If you haven't, look into running Python from the command line: Terminal on Mac/Linux, Command Prompt on Windows.) Data scientists generally choose Conda over pip because it groups package installations into separate environments, which prevents conflicts between packages intended for separate projects.

One way to install and use Conda is through Anaconda, which bundles the package manager with 150+ data science packages for convenience. However, this makes the installation time longer, for packages that we may not need. A better way to install Conda is with Miniconda, which will just install Conda and everything it needs to run, letting you decide what packages to install.

Google "install miniconda" and follow the instructions for your operating system. Once you create and activate a conda environment:

```
conda create --name [environment-name]
conda activate [environment-name]
```

installing packages is the same as with pip.

```
conda install [package-name]
```

After installing `numpy` and `pandas`, reading in a `.csv` file in Python becomes this easy:

```
import numpy as np
import pandas as pd

filename = \say{yogurt_nutrition_facts.csv}
nutr_facts = pd.read_csv(filename, index_col=0)
```

2.2 Jupyter Notebooks

It'd be useful to see the `nutr_facts` variable in the example above, which is a Pandas DataFrame object. Printing it out to a normal terminal gives us this:

```
             serving_size  calories  fat  sat_fat  cholesterol  sodium  carbs  sugars  protein greek_or_not
brand_name
Chobani               150       110  0.0        0          5.0      60     15      13       12      greek
Dannon                150       140  4.5        3         15.0      70     20      15        5        not
Yoplait               150       150  2.0        1         10.0     105     26      19        6        not
Yoplait_Greek         150       100  0.0        0          2.5      60     10       7       15      greek
Dannon_Oikos          150       110  0.0        0         10.0      45     16      15       12      greek
La_Yogurt             170       160  5.0        3         20.0      80     23      19        6        not
Fage                  227       190  0.0        0          0.0      70     27      25       20      greek
Lala                  170       150  1.5        1         10.0      95     29      24        6        not
Stonyfield            227       200  8.0        5         25.0     110     26      22        7        not
Voskos                150       130  0.0        0          0.0      45     21      16       11      greek
```

which is hard to read. But with a Jupyter notebook, we get a nice table, like we would out of Excel:

brand_name	serving_size	calories	fat	sat_fat	cholesterol	sodium	carbs	sugars	protein	greek_or_not
Chobani	150	110	0.0	0	5.0	60	15	13	12	greek
Dannon	150	140	4.5	3	15.0	70	20	15	5	not
Yoplait	150	150	2.0	1	10.0	105	26	19	6	not
Yoplait_Greek	150	100	0.0	0	2.5	60	10	7	15	greek
Dannon_Oikos	150	110	0.0	0	10.0	45	16	15	12	greek
La_Yogurt	170	160	5.0	3	20.0	80	23	19	6	not
Fage	227	190	0.0	0	0.0	70	27	25	20	greek
Lala	170	150	1.5	1	10.0	95	29	24	6	not
Stonyfield	227	200	8.0	5	25.0	110	26	22	7	not
Voskos	150	130	0.0	0	0.0	45	21	16	11	greek

To create a Jupyter notebook, first install the `jupyter` package with Conda, then, in the command line, type:

```
jupyter notebook
```

This will open the notebook in your default browser. Create a new Python3 notebook by clicking the `New` dropdown in the top right corner, and copy-paste the same code as before into a block, then append the line

```
nutr_facts
```

to see the table above.

If you've ever run Python in interactive mode (by typing `python3` into the command line), seeing a printout after `nutr_facts` will feel familiar. That's because Jupyter notebooks run "cells" of Python code in interactive mode. Breaking code into cells is useful: you can run cells individually or call variables from cells above the current one. Also, Google Colab notebooks are Jupyter notebooks connected to Google's servers, so learning how to use local Jupyter notebooks will help with that; see our Google Colab lecture for more info.

Aside from making `.csv` files easier to read in, `pandas` also allows us to convert data to `numpy` arrays easily. `pandas` DataFrames can be converted to 2D `numpy` arrays:

```
my_arr = data_frame.to_numpy()
```

and any subsequent `numpy` operations, like `np.delete()` or `np.reshape()` can be performed on the array.

3 Scikit-Learn

To train our SVM on the data, we'll need to use `scikit-learn` (`sklearn`), a collection of data science tools and models.

3.1 Preprocessing

In the `nutr_facts` DataFrame, we'll want to separate the last column, containing the output classes, from the input feature data. We'll also want to make it a numeric binary classification problem so `scikit-learn`'s SVM models can handle it. Formatting data to work with a given model is called preprocessing. In code, that's:

```
X = np.delete(nutr_facts.to_numpy(), -1, 1)
y = np.where(nutr_facts['greek_or_not']=='not',0,1)
```

where X is the input features and y is the output classes. By printing these `numpy` arrays, you'll see they have the same data as the `pandas` DataFrame, only without the qualitative labels.

Without a Kaggle competition to do split the data into training and testing data, and do the evaluation for us, we'd have to do both manually. `Scikit-learn` (installed as `sklearn`) makes splitting data easy:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size = 0.20)
```

`test_size` here is the proportion of our overall data that will go to the testing set; in this case, $0.20 = 20\%$ testing data. After a model is run, `sklearn` also makes evaluating that data with a confusion matrix simple.

However, since we primarily do machine learning competitions from Kaggle for this club, we have access to pre-split training and testing data and an evaluator. Thus, we read in the testing `.csv` data and preprocess it the same way we processed the training data. Your code for the SVM competition will be similar to this:

```
# train.csv and test.csv already read and preprocessed

X_train = train_df.to_numpy().astype(float)
y_train = train_df['Survived'].astype(float).to_numpy()
X_test = test_df.to_numpy().astype(float)
```

Here, the training and testing DataFrames are converted to `numpy` arrays, and the data in them is converted to floats. We'll get a final array, `y_pred`, from our SVM model.

At last, with `sklearn` and our properly formatted input/output data, the multivariable calculus behind an SVM can be reduced to three lines of code:

```
from sklearn import svm
model = svm.SVC()
model.fit(X_train, y_train)
```

The SVC in `svm.SVC()` stands for Support Vector Classifier, to differentiate it from sklearn's Support Vector Regression model (`svm.SVR()`). Getting `y_pred` is even simpler:

```
y_pred = model.predict(X_test)
```

3.2 Hyperparameter Tuning

Of course, running a raw SVM on any data is unlikely to succeed. For example, the 0.63838 score under "Kevin Fu" on the SVM competition is a raw SVM on minimally preprocessed data. The `SVC` model in `sklearn` has dozens of hyperparameters to tune; for the SVM competition, the important ones to focus on are `C`, `kernel`, and `gamma`. Refer to the docs and last week's lecture for what each one means.

Tuning hyperparameters can feel like guess-and-check, because it largely is. In fact, the process of "guess-and-check" is built into `sklearn.model_selection` as `RandomSearchCV()`. A more systematic approach would be `GridSearchCV()`, but like OvO vs OvR, in real-world testing both give similar results. Implement either if you want to avoid manually guessing at hyperparameters.

4 Results

Now that we have a working SVM model and `y_pred`, all that's left is to submit and send our output `.csv` to Kaggle.

4.1 Writing to .csv

Again, `pandas` makes this simple. After using `np.reshape()` and `np.concatenate()` to merge our `y_pred` array with a corresponding `id` row, writing to a `.csv` file is as simple as creating an output DataFrame and saving it:

```
out_df = pd.DataFrame(columns=['id', 'solution'], \
            data=out_data).astype(int)
outfile = \say{solution.csv}
out_df.to_csv(outfile, index=False)
```

In the last line, `index=False` tells `pandas` to avoid numbering the rows, since our `id` column does that already.

4.2 Real-World Evaluation

Without a Kaggle competition's automated evaluation, evaluating an SVM becomes harder. After splitting your data into training and testing data with `sklearn`, as referenced at the top of subsection 3.1 Preprocessing, then a confusion matrix is a simple way to gauge a model, once you know how to read one. A confusion matrix for a binary classification problem looks like this:

n=165	**Predicted: NO**	**Predicted: YES**
Actual: NO	50	10
Actual: YES	5	100

True positives and true negatives are cases where our model is correct: where the predicted class lines up with the actual class. In a confusion matrix, these are the bottom-right and top-left squares, respectively. A false positive (top-right) is when the predicted value is yes but the actual value is no, and a false negative (bottom-left) is the opposite. In each case, the second word ("positive"

vs "negative") corresponds to the model's prediction, and the first word ("true" vs "false") tells us if the model's prediction is correct or not.

Because the importance of each of these cases varies depending on the problem—for example, false negatives in disease detection are extremely problematic—we have different evaluation metrics based on these four cases. The most common two are precision and recall. Recall is calculated as follows (true positives are abbreviated as TP, false negatives are abbreviated as FN):

$$\frac{TP}{TP + FN} = \frac{TP}{actual\ yes} \tag{1}$$

Recall is also known as True Positive Rate, because it's simply the ratio of correctly predicted positives to actual positives, as shown in the formula.

The precision equation is:

$$\frac{TP}{TP + FP} = \frac{TP}{predicted\ yes} \tag{2}$$

Recall and precision tend to be inversely correlated; higher precision leads to lower recall, and vice-versa. Recall can be thought of as a model's ability to identify the positive cases, which is important when the positive cases are extremely infrequent, like in disease identification. Precision is how accurate the model is when it finds positive cases, differentiating it from pure accuracy, which is simply:

$$\frac{TP + TN}{total} \tag{3}$$

Finally, to weigh the trade-off between recall and precision, we can find a model's f1-score, which is the harmonic mean of recall and precision:

$$2 * \frac{precision * recall}{precision + recall} \tag{4}$$

which we use instead of an arithmetic mean of recall and precision because it punishes extremely low values for either. For example, with a precision of 0.0 and a recall of 1.0, the arithmetic mean would be 0.5, but the f1-score is 0.0.

With sklearn, we can get all this information easily. This code:

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

will show the confusion matrix as a 2x2 numpy array and a weighted summary of the precision, recall, and f1-score.

5 Competition

The SVM competition is up at `https://tjmachinelearning.com/competitions/1920` and ends at 11:59 PM on Tuesday, November 5 (a week from today). You should use `numpy`, `pandas`, and `scikit-learn`. This will count toward the rankings, and as usual, the top three winners will receive candy, and the top winner will get a free t-shirt.

As it says on the website, this competition is more about manipulating incomplete datasets and tuning hyperparameters than SVM coding ability; `pandas` and `sklearn` provide methods to accomplish both.

6 References

6.1 Images

- Confusion matrix: `https://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/`
- All other images: `https://github.com/tjmachinelearning/yogurt-svm`

6.2 Documentation

- Conda: `https://docs.conda.io/projects/conda/en/latest/index.html`
- numpy: `https://docs.scipy.org/doc/numpy/reference/`
- pandas: `https://pandas.pydata.org/pandas-docs/stable/`
- scikit-learn: `https://scikit-learn.org/stable/modules/svm.html`

6.3 Other

Grid Search vs Random Search:

- `https://scikit-learn.org/stable/auto_examples/model_selection/plot_randomized_search.html`
- `https://blog.usejournal.com/a-comparison-of-grid-search-and-randomized-search-using-scikit-learn-29823179bc85`

Confusion Matrix:

- `https://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/`
- `https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c`

General sklearn:

- `https://stackabuse.com/implementing-svm-and-kernel-svm-with-pythons-scikit-learn/`