

# Lab 14

Today, we will start exploring another method for classification **Support Vector Machines** or SVM. Before we dig into SVMs, we will start with how to evaluate classification results.

Today's goals are:

0. Explain how to evaluate models in the classification task
1. Define Maximal Margin
2. Motivate SVM

## Thoughts on Cross Val

Before starting with today's topics, let's talk about cross val. Consider two questions:

- Which is better:  $k$ -fold or LOOCV? Why?
- Is there a comparison to the *process* of cross val to gradient descent (ie. mini-batch and stochastic)?

## Evaluating Methods for Classification

Classification and prediction are often seen as two different and distinct parts of supervised machine learning. One of the biggest differences has to do with the notion of "close". In prediction, a model can be *close* but not exactly on truth to be considered "good enough." However for classification, we are placing objects into distinct groups and there is no notion of "close enough." For example, one either plays for the Washington Nationals or they don't. In professional baseball, there is no notion of playing part time for two teams.

With this difference in mind, how do we evaluate our methods for classification tasks? Let's start with a specific evaluation metric MSE and adjust this for classification. We begin by recalling the equation for MSE for prediction:

$$MSE_{\text{prediction}} = \text{avg}(|\text{truth} - \text{guess}|^2)$$

In classification, we have *predicted classes* and the *true classes* of each data point. How might we adjust the above for classification?

Take a minute to jot down your thoughts and we'll come back together to discuss:

$$MSE_{\text{classification}} = ????$$

(this area is intentionally left blank)

## Indicator functions

One trick that we use in machine learning over and over again are **indicator functions**. Simply indicator functions *indicate* something. We often use them in a similar way to boolean functions. With an indicator function, we *indicate* that a condition is met with a 1 and if not, the indicator function results in a 0. We write indicator functions as:  $I(\text{condition})$ .

For example, we can reimagine MSE as the average of the indicator function over the whole function, where the indicator function indicates when the true class does not equal the predicted class:

$$MSE_{\text{classification}} = \text{avg}(I(\text{trueclass} \neq \text{predclass}))$$

**How is this related to MSE for prediction?!?**

In the case of classification, we are either correct or not. So in this MSE, we want to know the average number of times that we are wrong. So we add up all the times that we are wrong (using the indicator function) and divide by the total number of datapoints.

This is very similar to the MSE for prediction (ie. linear regression) where we added up how off we were (squared) and divided by the number of data points.

## Motivating SVM

Today, we are going to start looking at Support Vector Machines (or SVMs). SVM is a method for classification. As with kNN, we will work on examples that split our data into two classes.

Last week, we focused on the motivating the differences between the train and test phases in supervised learning from the perspective of reaching for the truth underlying our data. In classification, we can use the same lens; we are seeking not to just divide the data that we have into two groups, but rather to **find the true boundary** that exists between the territory of the two groups. For example, instead of just trying to classify people into what state they live in based on their location, can we use the locations of where people live to find the state boundaries?

## Imports for Today

Today we are working with two new (constructed) datasets.

We begin as usual, importing the packages and data that we need. Plot the first dataset `new_data` and note at least 2 observations about the data.

```
In [ ]: ## Import block
        %matplotlib inline

        import matplotlib.pyplot as plt
        import seaborn as sns; sns.set()

        import numpy as np
        from numpy import linalg as LA
        import pandas as pd

        import random

        from sklearn import linear_model
```

```
In [ ]: ## Functions for later use

        def assign_class(datapoint, slope, intercept):
            pass

        def classification_mse(class_truth, pred_class):
            pass
```

```
In [ ]: # For function testing
```

```
In [ ]: ## Import Data
```

```
new_data = pd.read_csv("lab14data.csv", sep = ",")
new_data["group"] = -1*new_data["group"]
new_data_np = np.genfromtxt("lab14data.csv", delimiter=',', skip_header=1)

extra_data = pd.read_csv("extra_lab14data.csv", sep = ",")
extra_data["group"] = -1*extra_data["group"]
extra_data_np = np.genfromtxt("extra_lab14data.csv", delimiter=',', skip_header=1)

## x-values for plotting
x = np.linspace(-3.1, 3.15, 1000)
```

```
In [ ]: # Plot new_data
#
# What do you notice?
```

## Example boundaries

With the goal of dividing the space into two territories. Which of the below options do you think is best?

```
In [ ]: # Possible boundaries:

# Plot the data
plt.scatter(new_data["myst"], new_data["mysty"])

# Plot one possible option to divide the two groups
# In this case  $y = 0.6x$ 
plt.plot(x, 0.6*x, linestyle='dashed', c="r")

# Plot another possible option to divide the two groups:
# In this case  $y = 1.6x$ 
plt.plot(x, 1.6*x, linestyle='dashed', c="g")
```

**Question** - What do you like about each of these lines? Is one better than another?

(Notes for you!)

Let's consider one more option:

```
In [ ]: plt.scatter(new_data["myst"], new_data["mysty"])

# Plot a third possible option to divide the two groups:
```

```
# In this case y = x
plt.plot(x,x,linestyle='dashed',c="k")
```

**Question** - What do you like about this line? Is this one better than the others?

(Notes for you!)

## Maximal Margin

Recall that our goal is dividing a data space into two "territories." We don't know how much noise exists in our dataset, and so we want to consider which line would create the widest boundary between our two data groups.

I think of this as *how much can we inflate a line before the inflated line touches the data*. To visualize this idea, we will use the method `fill_between` :

```
In [ ]: # Inflating the red boundary:
        inflate_width = 0.7

        # Plot the data points
        plt.scatter(new_data["myst"],new_data["mysty"])

        # Plot the boundary
        plt.plot(x,0.6*x,'-r')

        #Plot the "inflation" around the line
        plt.fill_between(x, 0.6*x - inflate_width, 0.6*x + inflate_width,
                        edgecolor='none', color='#AAAAAA', alpha=0.4)
```

Does this margin touch/include any data points?

Play with the `inflate_width` variable above to find a good value for this parameter that is as big as possible without touching the data.

Repeat this process for the green and black lines below. According to your experiments, which has the widest possible inflation?

```
In [ ]: # Inflating the green boundary:
        inflate_width = ?????

        plt.scatter(new_data["myst"],new_data["mysty"])
        plt.plot(x,1.6*x,'-g')
        plt.fill_between(x, 1.6*x - inflate_width, 1.6*x + inflate_width,
                        edgecolor='none', color='#AAAAAA', alpha=0.4)
```

```
In [ ]: # Inflating the green boundary:
        inflate_width = ?????

        plt.scatter(new_data["myst"],new_data["mysty"])
        plt.plot(x,x,'-k')
        plt.fill_between(x, x - inflate_width, x + inflate_width,
```

```
edgecolor='none', color='#AAAAAA', alpha=0.4)
```

## Maximal Margin Definition

The maximal margin is simply the biggest margin that we can place **between** our two groups. We build this margin by finding a central line and then inflating it equally in both directions. Finding this central line uses Lagrange multipliers, which while really cool, are beyond the scope of this course.

We can define the maximal margin for higher dimensions beginning with a plane (or hyperplane) and inflating it in the same manner.

## Testing our Maximal Margins

It turns out that we have a bit of extra data, and true to the train/test paradigm, let's see how our best line does with this extra data:

```
In [ ]: # Training data
plt.scatter(new_data["myst"], new_data["mysty"],
            c=new_data["group"], cmap = "spring")

# Testing data
plt.scatter(extra_data["myst"], extra_data["mysty"],
            c=extra_data["group"], cmap = "spring")

## Your line and inflated part here -
# You need to fill in which line r ( $y = 0.6x$ ), g ( $y = 1.6x$ ),
# k ( $y = x$ ) is the best one based on your thoughts from above
inflate_width = ???
plt.plot(x, ???, '-?')
plt.fill_between(x, ??? - inflate_width, ??? + inflate_width,
                 edgecolor='none', color='#AAAAAA', alpha=0.4)
```

## Violations to the Margin

Notice that we have some colors on the wrong side of the line. This tells us that perhaps our margin does not generalize to new data very well. Let's count how well it did on **just** the test data.

The below plot plots only the test datapoints with the decision boundary and the associated inflated part. Using the below plot:

- Count how many data points are on the wrong "side" of the line
- Count how many data points are on the right "side" of the line BUT are within the margin

Both of these are violations to the maximal margin, but of different intensities.

```
In [ ]: # Counting violations

# Testing data
```

```
plt.scatter(extra_data["myst"],extra_data["mysty"],
            c=extra_data["group"], cmap = "spring")

## Your line and inflated part here -
# Again, please fill in which line r (y = 0.6x), g (y = 1.6x),
# or k (y = x) is the best one based on your thoughts from above
inflate_width = ???
plt.plot(x, 1.6*x, '-?')
plt.fill_between(x, ??? - inflate_width, ??? + inflate_width,
                 edgecolor='none', color='#AAAAAA', alpha=0.4)
```

**Question:** Given these counts and this plot, do you think this is a good margin?

(Notes for you)

**Question:** Can you alter the width of the margin to have less violations within the margin?  
(Try this below.)

```
In [ ]: # Varying the margin width

# Testing data
plt.scatter(extra_data["myst"],extra_data["mysty"],
            c=extra_data["group"], cmap = "spring")

## Your line and inflated part here -
# Again, please fill in which line r (y = 0.6x), g (y = 1.6x),
# or k (y = x) is the best one based on your thoughts from above
inflate_width = ???
plt.plot(x, ???, '-?')
plt.fill_between(x, ??? - inflate_width, ??? + inflate_width,
                 edgecolor='none', color='#AAAAAA', alpha=0.4)
```

### Trying other boundaries

Returning to our other examples from earlier, is either of the other two options a "better" margin? In other words, if we use either of the other lines as the boundary, can we have a wider margin with the same (or fewer) violations in the test set?

```
In [ ]: # Varying the margin width for a first other option

# You used one option for the line above, choose a different
# one from our list: r (y = 0.6x), g (y = 1.6x), or k (y = x)
# as the second one that you will explore.

# Vary margin (ie. inflate_width) and count the violations:

# Testing data
plt.scatter(extra_data["myst"],extra_data["mysty"],
            c=extra_data["group"], cmap = "spring")

## Your line and inflated part here -
inflate_width = ???
plt.plot(x, ???, '-?')
```

```
plt.fill_between(x, ??? - inflate_width, ??? + inflate_width,
                 edgecolor='none', color='#AAAAAA', alpha=0.4)
```

```
In [ ]: # Varying the margin width for second other option

# You have used two options for the lines above. Now select
# the last one from our list: r (y = 0.6x), g (y = 1.6x),
# or k (y = x) to explore.

# Vary margin (ie. inflate_width) and count the violations:

# Testing data
plt.scatter(extra_data["myst"], extra_data["mysty"],
            c=extra_data["group"], cmap = "spring")

## Your line and inflated part here -
inflate_width = ???
plt.plot(x, ???, '-?')
plt.fill_between(x, ??? - inflate_width, ??? + inflate_width,
                 edgecolor='none', color='#AAAAAA', alpha=0.4)
```

### Best Margin?

Which one is the best? What makes you say that?

**Note:** In this exploration we have used the *test* set to inform our discussion about which is the "best" margin. We did this for the purpose of *motivating* SVM. In the usual set-up, the test set would **not** be involved with deciding the maximal margin.

## Maximal Margins --> SVM

SVM is **relaxation** of the maximal margin. Recall that maximal margins are between two classes, which means that they do not allow for any violations, either in the margin or crossing the middle boundary. In SVM, we seek to balance 1) finding the widest margin and 2) allowing (but limiting) violations to the boundary.

For SVM, we follow the usual procedure for supervised learning. We first divide our data into training and testing data. Then we find the best margin based on the training data and evaluate how well it did on the test set.

## Assigning classes

In preparation for implementing SVM, we create a function that assigns each data point to a class based on its relationship to the boundary line. If the datapoint is above the boundary assign it to class 1 (the positive class) and if it is below assign it to class -1 (the negative class).

**Relationship to the boundary** - To determine whether a point is above or below the boundary, we follow the below procedure:



1. Convert our boundary line from an equation (like  $y = mx + b$ ) to an expression (such as  $y - (mx + b)$ ).
2. Create relationship function  $R(p)$  as being equal to this expression. That is for a point  $p = (p_x, p_y)$ , then  $R(p) = p_y - (m \cdot p_x + b)$ .
3. Define a classification function  $C(p)$  such that if  $R(p) \geq 0$ , then  $C(p) = 1$ . Otherwise  $C(p) = -1$ . In effect  $R(p)$  tells us whether our point  $p$  belongs to the positive or negative class.

At the top of this notebook, there is a beginning for the function above called `assign_class` that seeks to assign points to the positive and negative classes. Use the above outline to fill in the details for `assign_class`

```
In [ ]: # Code block for you
```

## Next time

Next time, we will implement SVM and discuss how **kernels** are central to SVM.

## Final Thoughts

To finish up this lab, complete definitions for `assign_class` and `classification_mse`. Test your implementations on the three potential lines from this lab by computing the classification test MSE for each of them.

Then apply your implementation to a new line of your choosing. Make a plot with the line of your choosing and state the classification test MSE. Is your new line a good one? Justify your answer with both a picture and the classification test MSE. Share your plot and your answer in a post on **#lab14\_submission** channel on slack with your answer.

If you have questions from this lab, post them to **#lab\_questions** with the same preamble (i.e. starting with **Lab14**). If you have the same question, please use one of the emoji's to upvote the question. If you would like to answer someone's question, please use the thread function. This will tie your answer to their question.

## Resources consulted

0. [ISLR \(https://hastie.su.domains/ISLR2/ISLRv2\\_website.pdf\)](https://hastie.su.domains/ISLR2/ISLRv2_website.pdf)
1. [SDS 293 Notes by R. Jordan Crouser \(http://www.science.smith.edu/~jcrouser/SDS293/\)](http://www.science.smith.edu/~jcrouser/SDS293/)
2. [In-Depth: Support Vector Machines \(https://jakevdp.github.io/PythonDataScienceHandbook/05.07-support-vector-machines.html\)](https://jakevdp.github.io/PythonDataScienceHandbook/05.07-support-vector-machines.html)

