Support Vector Machines in Business

“Company leaders are becoming storytellers” (Hotz, 2020) is a quote said by George Hotz in one of his crazy streams. In this quote Geo refers to the importance of making decisions through data and the impact that it will have in the future. It highlights that everyday leaders will make less decisions on their own. The role of the leader will shift to more of an influencer than a decision maker. I am not saying that leaders will make no decisions. However, I do mean that the fast speed at which sensors are getting integrated in our lives, the yet not stopping Moore's law, and the not stooping discoveries on algorithms, will lead the future into very competitive decision making. One of the current biggest constraints in business when it comes to decision making is time. I often come to the scenario where I have to make a product or strategic decision within a certain amount of time. I know that the most optimal decision can be made through the use of data, but time does not allow it, so I am forced to guess what we believe is best.

In this paper I want to dive deep into a somewhat new classification algorithm used for decision making. I want to explain at a conceptual, and graphical level, and then implement it in Python from scratch. We will follow a top to bottom approach. We will look at Support Vector Machine (SVM for short) from the computing perspective since this is how it is being applied in the business field. This means that knowing Python, and basic algebra is a prerequisite.

This paper will follow a question, answer and example structure.The intention of this structure is to hopefully fulfill the questions that you are probably already asking yourself with an answer and an example. The example will hopefully click in what was not clear in the answer.

### **What are Support Vector Machines?**

Support Vector Machine (SVM for short) is a classification algorithm. This means that given data about two groups, this algorithm will determine whether a new data point belongs to one data group or the other. The algorithm learns from data and creates boundaries. It will determine whether a data point falls into one group or the other depending on where the data point falls in the boundaries. The process is shown in *figure 1* at a high level.



### **How do they work?**

The way that the algorithm is able to determine if one data point is in one group or the other is by drawing a decision boundary in a graph that includes both groups. Depending on which side of the decision boundary the data point falls, the data point is determined to be in one group or the other. For the graph shown in *figure 2* we will use a data set that is small and only has two dimensions for the sake simplicity. As shown in the image, we have 3 lines that divide the two data sets. The equation of a line is the following:

Equation 1

In SVM the slope and x values are vectors with the same number of dimensions. That means when you apply the inner product between these two, the result is a scalar value. SVMs are an optimization problem where we want to minimize the magnitude of the vector that represents the slope and maximize b. This means we need to iterate through the combination of these two values until we find a slope vector value that is greater than 0, but that after computing it in *equation 2* combined with the *b* value will be bigger or equal to 1. The equation of the line is the following:

Equation 2



### 

### **What is happening when it is used in libraries?**

Now that we have conceptual and graphical understanding of what SVMs do, we can look at how this algorithm is implemented in popular libraries like tensorflow, scikit learn, or pytorch. In the code block below we are not using any library. We are looking at an SVM that I wrote from scratch with comments explaining what is happening at each step.

First we loop through a dictionary that contains the data that we are going to pass through the algorithm and dump it into an array. Then we choose the maximum value and the minimum value among all of the feature values. We create another array named “step\_sizes” and populate it with the max value times .1, the max value times .01 and the max value times .001. We then create an array with two values as the maximum value feature and call it “latest optimum”.

|  |
| --- |
| def fit(self, data):  # self data will store the data that will be passed through the  # SVM  self.data = data   # { ||w||: [w, b]}  opt\_dict = {}   # this is the transformation matrix for trying all possible  # combinations  transform = [[1, 1], [-1, 1], [-1, -1], [1, -1]]   all\_data = []  # in this loop we are extracting the features into an array  for yi in self.data:  print("what yi represents in data")  print(yi)  for featureset in self.data[yi]:  print("what featurset represents in data")  print(featureset)  for feature in featureset:  print(10 \* "#")  print("what feature represents in featureset")  print(feature)  all\_data.append(feature)  print(  "this is the final list of features that were appended"  )  print(all\_data) |
|  |
|  |

After we have all these values we start the optimization. For the optimization we start iterating through the step\_sizes array, and create a numpy array named “w” and populated with the biggest value in the featureset twice. Then we set a boolean variable named optimized equal to false and start a while loop that will run turns true. Inside the while loop we start iterating through a numpy array that is created with the “arrange” function. The arrange function does the same as the range function in the built in functions from python. It creates a range of values given a sequence of rules. This numpy function is given 3 values. It is given the lower starting point, then the increment or decrement amount, and the ending point. In this case the range represents the options that we will choose as “b” as we are trying to optimize. It is important to remember that we are trying to maximize b and minimize w in sign(xi w + b) 1. Once the dot product of x and w plus b is greater or equal to one we know that we have found a candidate combination of values so we store these values in a dictionary. Then we compare the original max value that we used for this run and if it is greater than 0 we have found the most optimal values else we make w equal to w minus a step (step is max feature value either times .1, .01 or .001 in this instance). We iterate through this process until we find the values that best optimize our objective function.

|  |
| --- |
| # then we gab the maximum and minimum value from all features and then  # we set it to none  self.max\_feature\_value = max(all\_data)  self.min\_feature\_value = min(all\_data)  all\_data = None   # we create an array with the max feature value and multiply it by  # .1, .01, and .001. These values represent the the size of the  # steps that will be taken. The smaller the steps the more times  # that it will be needed to take  # support vectors yi(xi,w+b) = 1  step\_sizes = [  self.max\_feature\_value \* 0.1,  self.max\_feature\_value \* 0.01,  # this is the point where it starts to become very  # expensive  self.max\_feature\_value \* 0.001,  ]   # this one is extremely expensive  b\_range\_multiple = 5  b\_multiple = 5  latest\_optimum = self.max\_feature\_value \* 10  # for max value in the step size array  for step in step\_sizes:  # w = an np array with the lates optimum twice  w = np.array([latest\_optimum, latest\_optimum])  # set optimized to equal false  optimized = False  # start iterating until optimized gets flipped to false  while not optimized:  # arrange is similar to the function range in python. It  # returns a list created by given parameters arrange  for b in np.arange(  -1 \* (self.max\_feature\_value \* b\_range\_multiple),  self.max\_feature\_value \* b\_range\_multiple,  step \* b\_multiple,  ):  print(  "This is every element of the array created by ranges"  )  print(b)  for transformation in transform:  print("this is each transformation in transform array")  print(  "w\_t = array with the latest optimum twice times the transformation array"  )  w\_t = w \* transformation  print(w\_t)  found\_options = True  # weakest link in the SCM fundamentally SMO attempts  # to fix this a bit  # yi(xi.w + b) >= 1  for i in self.data:  for xi in self.data[i]:  yi = i  if not yi \* (np.dot(w\_t, xi) + b) >= 1: # xi = feature set,   # w\_t = (max featureset \* 10) \*  # transformation  # matrix  # b =   found\_options = False   if found\_options:  opt\_dict[np.linalg.norm(w\_t)] = [w\_t, b]   if w[0] < 0:  optimized = True  print("Optimized a step")  else:  w = w - step   norms = sorted([n for n in opt\_dict])  # ||w|| : [w, b]  opt\_choice = opt\_dict[norms[0]]  self.w = opt\_choice[0]  self.b = opt\_choice[1]  latest\_optimum = opt\_choice[0][0] + step \* 2   for i in self.data:  for xi in self.data:  yi = i  print(xi, ":", yi \* (np.dot(self.w, xi) + self.b)) |
|  |

There are other functionalities that SVMs need to be properly used like graphing packages and kernels - To deal with multidimensional data sets. This paper explains in a nutshell what SVMs are doing in the background when they are called from libraries like tensorflow or scikit-learn.

### **So now we ask ourselves how does this algorithm get used from a business and product perspective?**

The primary applications of SVMs are around image annotation. SVMs are especially good at object and text identification. Let’s remember that SVM is a classification algorithm. It classifies images into the searched object (1) and the rest of the image (-1). The searched object can be anything from human to not living objects. Then it extracts features from each pixel. It uses the pixel values (0 - 255) to extract features from the faces of humans or details of objects and then is able to classify them. At a higher level this can be useful from criminal identification at airports, to identification of cancer cells in the skin, to translating images of text to actual text among other applications. When we look at products, SVMs are being used in writing tablets because they are especially good for the translation of handwriting to text, google lens for the identification of objects, google photos for the identification of people among others. The bottom line is that SVMs help us classify data points whether they are attributes of cancer cells to pixel values.

**Bibliography**

Hotz, G. (2020, 06 30). *Geor*. George Hotz | Just Chatting | talking hour | Science & Technology. Retrieved 09 24, 2020, from https://www.youtube.com/watch?v=9sls9V4m5Vc&t=15s&ab\_channel=commaaiarchive

Ray, S. (2017, 09 13). *Understanding Support Vector Machine(SVM)*. Understanding Support Vector Machine(SVM) algorithm from examples (along with code). Retrieved 09 24, 2020, from https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/#:~:text=%E2%80%9CSupport%20Vector%20Machine%E2%80%9D%20(SVM,mostly%20used%20in%20classification%20problems.

sentdex. (2016, April Sentex). *Machine Learning with Python*. Practical Machine Learning Tutorial with Python intro P.1. Retrieved 9 24, 2020, from https://www.youtube.com/watch?v=OGxgnH8y2NM&list=PLQVvvaa0QuDfKTOs3Keq\_kaG2P55YRn5v&ab\_channel=sentdex

Winston, P. (2020, 01 14). *Learning: Support Vector Machines*. Support Vector Machines. Retrieved 09 25, 2020, from https://www.youtube.com/watch?v=\_PwhiWxHK8o&ab\_channel=MITOpenCourseWare