**Author**: Peter Olson

**Recommended Course**: APCSA

Introduction to Machine Learning using Image Classification

**Notes from the author:**

I *highly* recommend this lesson to be scheduled following the AP test. This is a very cool area of study in computer science, and it is also one that primarily studied by graduate computer science students. As a prerequisite for this series of lessons, students should be proficient and well-practiced in using 2D arrays and nested loops. The goal of these lessons is to teach students a new way of thinking… not necessarily the complex mathematics related to optimization of algorithms and techniques. If the students can come through these lessons with an interest and basic understanding of machine learning, and with a renewed mind to the possibilities of coding so that programs can improve upon previous results, then you should consider it a job well-done.

This lesson revolves around **image classification** **in** **Java**. Why? Am I a monster? Well, I don’t think so, but the idea here was that standard linear regression analyses are too complex, even though they are the simpler of algorithms for machine learning processes. There’s too much stats, too much calculus. I have students that take my course who never reach Precalculus or Calculus, so this set of lessons was designed to simplify the machine learning design to its bare minimum, perhaps even to a fault. I don’t want a *good* machine-learning program, I want one that can be easily understood by students without having high learning-curves dependent upon the student’s ability to decontextualize higher mathematics from within a programming environment.

In order to recompense these shortcomings in design, I chose to lean into visualization through pixelization rather than relying upon the eye-glazing monotony of spreadsheet data. For some reason, most examples or projects you will find online lean heavily into teaching the interpretation of the feature data rather than finding the feature data itself. This lesson intends for students to actually code tools for scraping in feature data into files and focuses less on the students understanding a series of higher-level statistical measures.

The following plans are categorized as follows, along with their goals, which will be expanded upon in their relevant sections:

1. Day 1 – Identifying Features and the ML Process
   1. Goals:

* Use I/O with an existing (powerful) ML program
* Practice identifying features
* Compare and contrast linear algorithms vs learning algorithms

1. Day 2 – Designing Features and Experimenting with Training Data
   1. Goals:

* Create a feature within the program and implement it
* Add or remove features
* Run program using training data, test accuracy using test data

1. Day 3 – Machine Learning Proposal
   1. Goals:

* Consider real world uses for machine learning
* Plan out the steps to implement a ML project

Lesson Plan – Day 1

**Introduction**:

Students need to get familiar with machine learning programs before seeing any code at all. The goal on the first day is to get students thinking ‘*how does that work?’* and leading them towards those answers. To do this, a series of activities will introduce students to core concepts, such as the differences between linear and learning algorithms, supervised vs unsupervised vs reinforcement learning, and features.

**Goals**:

* Use I/O with an existing (powerful) ML program
* Practice identifying features
* Compare and contrast linear algorithms vs learning algorithms

**Activity Schedule**:

1. **Icebreaker Activity** – **Throwing a rock at students**

*Time required: 15 minutes*

Goal:

Get students thinking about identifying features and how computers can learn

Instructions:

Did I get your attention? First, there’s a catch, so don’t freak out. This activity requires some props, namely 2 or 3 real rocks and one fake rock. Here’s one that works, or you can find something similar:

* <https://www.amazon.com/ALPI-River-Stone-Stress-Toy/dp/B00E46M2D4/ref=sr_1_7?crid=2UB8L9TK4SUU3>

Demonstrate that your rocks are real by dropping one on the ground or perhaps knocking on it with your fist, or something similar. Depending on how believable your fake heavy object is (which should be lightweight and squishy), you may want to hide your fake object before throwing it. Yell out to your most easily-frazzled student, ‘Catch Jim!’, or whatever your student’s name is, and toss the **fake rock** at him or her. If all goes well, then several students will freak out a bit at the unexpected flying rock in the air.

Step 2, yell out ‘Whoops! My mistake. Can I have that back?’ and have students throw you your fake rock. Now hold your real and fake rock, perhaps apologetically talk about how slippery your rocks are, before once again tossing your **fake rock** at the same student (technically, we would want a different shaped or textured fake rock here for a more perfect analogy). Most likely, they will this time try and catch it or otherwise won’t freak out as bad (repeat this process if needed). Then it’s time to discuss why the student(s) reacted like they did.

Questions:

1. Why didn’t [Jim] freak out when the rock ‘slipped’ out of my hands?
2. How did he/she know it was a fake rock?
3. How can you tell the difference between a real and a fake rock? Why?
4. How would a computer tell the difference between a real and a fake rock?
5. If you showed a computer a picture of one fake rock and one real rock and told the computer which one was fake and which was real, and then showed the computer a different looking fake rock, would it be able to tell if was real or fake? What would help it get better at determining the difference between real and fake rocks?
6. **Cat identifying Activity**

*Time required: 20 minutes*

Goal:

Get students to learn about Type 1 and Type 2 errors when identifying features

Instructions:

Open the PowerPoint ‘IsItACat.ppt’ and play a game where students identify whether the picture is a cat or not. Have all students vote by raising their hands if they think the image is a cat and tell them that not raising your hand means you think it is not a cat. Record the total votes on the board as you go.

After slide #5, ask students what features would help determine whether the picture is a cat or not. For slides #6-10, ask students if these features helped them correctly identify whether the picture is a cat.

After slide #10, tell students that now the program is designed to identify whether it is a green light or not. Ask what features the program should look for first. For slides #11-15, ask students if these features helped them correctly identify whether it was a green light or not. Ask what the big difference is in a program that identifies whether a picture is a cat and whether a streetlight is green or not.

Questions from above:

1. After slide #5:
   1. What features would help determine whether the picture is a cat or not?
      1. Pick 2 or 3 features from their suggestions
   2. What does it mean for a picture to be a cat verses having a cat in the picture?
2. During slides #6-10:
   1. Do these features help correctly determine that the picture is a cat?
3. After slide #10:
   1. What features should we look for to identify that the streetlight is green?
      1. Pick 2 or 3 features from their suggestions
4. During slides #11-15:
   1. Do these features help correctly determine that the streetlight is green or not?
5. After slide #15:
   1. What is the big difference between a program that identifies whether a picture is a cat or not and a program that identifies whether a streetlight is green or not?
6. **Cat and Dog Recognizer ML Program**

*Time required*: *20 minutes*

Source: <https://www.dropbox.com/s/4vr2ez8l1ecnqbf/CatAndDogRecognizer.zip?dl=0>

Goal:

Get students to practice using a machine learning program and thinking about how programs improve after increasing quantities of training data

Instructions:

First, students need to download the program from the DropBox link above. The program can be run through the batch file if the program is run from a windows computer, or if running on a mac, find the CatAndDogRecognizer executable jar file and run that.

The program runs by selecting photos on your computer and pressing the button to have the program identify whether the photo is a cat or a dog. If it is neither, or if the program doesn’t know, it will say it is unsure. Have students try and find…

* 2 cats that are identified correctly
* 1 cat that is identified as a dog
* 1 cat that is identified as a cat that is not a cat
* 2 dogs that are identified correctly
* 1 dog that is identified as a cat
* 1 dog that is identified as a dog that is not a dog
* 1 cat or dog that is a cat or dog, but the program is unsure about

Then have students continue on to the worksheet, where they will answer questions about what they have learned from the activities and from this program

1. **Day 1 Worksheet** – Formative assessment

*Time required:* *35 minutes*

Goal:

Get students writing and thinking about non-linear thinking, features, making errors, and training vs test data.

Instructions:

If you are on a block schedule, students should be able to complete this worksheet in class. If you are not on a block schedule, this worksheet would be assigned as homework. This first assessment is designed to get students thinking and writing about the goals of this day’s lesson.

*See Day 1 Worksheet.docx*

Lesson Plan – Day 2

**Introduction**:

It’s coding time. Day 2 is all about getting your hands dirty *in the code*. Reading documentation, interpreting algorithms, designing features, implementing features, running training data, running test data, comparing results. In particular, students should get familiar with the machine learning structure of optimizing the weights of features, and ways in which features can be added or removed from the program. By the end of the day, students can compete to have the most accurate program when running a set of test data.

**Note from the author:**

The ‘PixLab’ program is really two programs built into one. The first part is an image-processing type of program. The second part uses the first to do machine learning using image classification. I highly recommend that you use the first program earlier in the year so that by the time your students reach this series of lesson plans, they are already familiar with the image-processing part of the program.

**Goals**:

* Create a feature within the program and implement it
* Add and remove features
* Run program using training data and compare results
* Run program using test data

**Activity Schedule**:

1. **Designing Features Brainstorm** – Ice Breaker

*Time required: 15 minutes*

Goal:

Get students thinking about how to design algorithms to identify features and create data. Get students thinking about confounding variables and overfitting.

Instructions:

Open ‘Day 2.ppt’ and look at slide #1. The two images are pictures of handwritten numbers that are in black and white. Tell students that these images are black and white pixels that are arranged in a two-dimensional array. For slides #1-3, each pair of images are differently drawn 0s and 1s. Ask students the following questions in a guided group conversation:

Questions:

1. What objects are in these images? How do you know?
2. What makes a one different from a zero?
   1. Write down all ideas from students
3. (Opt) How are they different based on their pixels?
   1. On their size? Shape? Distribution? Darkness / lightness? Sharp edges?
   2. Write down all ideas from students
4. *Pick a feature from list*. How would you program this feature? What would it return to you to tell you about the success of that feature?
   1. Try and steer students away from binary (two option) features
5. *Go to slide #2 or #3 if you haven’t yet*. How do some features work for the previous images that don’t work for these? What features are consistent across different styles of images of zeros and ones?
6. **Introduction to Machine Learning with Images**

*Time required: 7 minutes*

Goal:

Get students thinking about the different ways that images can be broken down and classified

Instructions:

Watch the video on neural networks first. Explain that this is the most complicated machine learning, aka ‘deep learning’ there is, and that they won’t be attempting that design today. Watch the second video, and after the video, explain that the goal today will be to use simple image classification techniques using a series of image segmentation techniques

Sources:

1. Video #1: <https://www.youtube.com/watch?v=2hMuYi2Wudw>
2. Video #2: <https://www.youtube.com/watch?v=taC5pMCm70U>

3. **Introduction to the PixLab Program**

*Time required: 20 minutes*

Goal:

Get students familiar with the tools and image viewing and creating process of the PixLab program

Instructions:

The PixLab program was originally designed to apply simple effects to images and to view those images. I adapted the program and added additional functionality to allow the user to use segmentation to clean up images in order to more easily draw data from those images. Most features are derived from binary-colored images, black and white, which can then more easily be processed. For this particular activity, the goal is simply to get students using the program and experimenting with the available methods, applying the methods to various pictures. Additionally, it is worth noting that there are multiple main functions in the class, so you will want to use the Picture.java class for experimenting with applying effects to pictures, and the MLRunner.java file for classifying whether the given image is one image type or the other. The MLDetector.java file will be explained further in a later activity.

For this activity, demonstrate using the program first. To do this, open Picture.java and find the main function at the bottom. There are a series of commented out methods, followed by a separate ‘explore()’ method, and a ‘write’ method. The first section of commented out methods are effects that can be applied to the picture that is stored in the Picture reference object. Note that some of these, the ones marked by ‘BW’ can only be applied to already black and white images. These can be uncommented for the effects to be applied. The explore() method shows the results of the method(s) and their effects. You can click around the image to grab the different colors of individual pixels. Lastly, the write method will save a new image of this image with the image name provided in the parameter.

Have students experiment using the program for the first 15 minutes, and for the last 5 minutes, have students send you their favorite image, and you can show the rest of the class the student’s images from your computer / a projector.

4. **Introduction to MLDetector.java**

*Time required: 18 minutes / 35 minutes*

Goal:

Get students acquainted with the procedures used to classify two different images, how the data is stored across iterations, and how the program learns over time through the process of a simplified Support Vector Machine (SVM) algorithm

Instructions:

Depending on whether your class is a block schedule or not, this is either the last activity or is the last activity before student work time on their first coding assignment. Students that finish early can begin on the first activity on the Lesson Plan for Day #3

MLDetector.java does a few things. It will clean up the images using various effect methods from the Picture.java class, run two or more feature methods for pulling data from the images that informs the program regarding the strength of those features, runs a few simple normalization and averaging methods, makes a guess about the classification of the image, asks for feedback on how well it did, and then adjusts weights for the next iteration of the algorithm.

This is where things are a little scrappy. The program does not use a true SVM algorithm, but rather draws on core ideas while avoiding complex mathematics and statistics. I don’t want the goal here to be to teach students advanced mathematics, they will learn that another time. The goal is to teach students how to think differently as programmers!

Please read and review the comments at the top of MLDetector.java and the comments on the runDetector method. These comments will explain how the machine learning works, what setup is needed, and how to reset between different runs of the program. Note that MLRunner.java is the executable class, not MLDetector.java.

Students can practice running the program with different inputs from the training data set folder. Note that before students begin using this program, the instructor should consider how many (and which) features they want to give students and which they want to hide. Don’t give students all 6 pre-written features—using these alone will achieve a near 100% success rate for the MNIST data set. It is recommended that instead the instructor includes 2 or 3 features, and asks the students to improve upon the program from there. You will want to cut these extra feature methods and store them in another file before handing the java file to your students. Like before, you should demonstrate how to use this part of the program, and explain the different parts of MLDetector.java, along with how they work.

Technical Notes:

*Wait, how is this a SVM algorithm?!*

Good question. It’s not. It would be more accurate to call it something else, but I liken it most closely to the SVM concept. I stripped the algorithm of its math and made many simplifications. Let me explain what I did, how it relates to SVM, how the math is reduced while maintaining the qualities of separation binary instances, and how the algorithm works without having to think about hyperplanes or additional dimensionality.

Note that this technical section is for the instructor, not for students. They really do not need to know this stuff, but it may be of interest to you.

In SVM, the idea is that there is a predictable, separable boundary that a computer can learn to adjust based on a set of features that ‘graphs’ the locations of objects. If the object lies on one side of the boundary, the program guesses that it is of one type, and if it is on the other side of the boundary, it guesses that it is of the other type. Here, we are using binary labels and binary classification (using reinforcement learning), and we want to be able to use a decent number of features, so SVM is a good fit.

I had two goals in finding the right algorithm for the machine learning design:

1. Allow for scalable feature addition and removal
2. Remove vector-based mathematics

Taking out the V in SVM does change things up quite a bit. This algorithm uses no KKT conditions (calculus) and no Lagrange multipliers, and instead focuses on the margin between instances. In order to reduce complexity, I turned to averages, something easily understandable by a high school student. In order to reduce dimensionality and the graphical nature of multi-feature instances, features are normalized and then averaged to produce a single value. This value is produced based on the weights of the features that are saved over multiple iterations. From there, you have three numbers: the current instance averaged feature value, and the weighted values that have been learned over past iterations. The ‘margin’ is the differences between the current instance value and the binary weighted values—whichever is closest to the feature value wins out the guess from the program, and the weights are adjusted by averaging the feature value into the weights, according to the ‘weight’ (proportional to the total iterations) of the current iteration. Thus, the program is not learning where to put the line, like a typical SVM program, but rather it is adjusting two lines, one on either side of the feature instance, and seeing which one the feature instance is closest two (we’re talking about 1 dimension though, not 2D). So, the algorithm draws on some neural network design without doing any crazy backward/forward propagation.

No calculus needed, no dimensionality, no vectors, and students don’t even need much algebraic knowledge outside of averages and weighted averages.

This algorithm, like any, has pros and cons:

Pros:

* Simplifies mathematics
* Allows for large feature set
* Less overfitting with larger feature sets than SVM
* Outliers are less likely to result in overfitting
* No confusing multi-dimensional abstraction
* Can be used to scale from binary to non-binary classification
* Doesn’t require a large number of training data images

Cons:

* Requires reinforcement learning
* Features have equal weighting (none are more important than another)
* Loss of dimensionality could result in imprecise classifications (tight margins) if certain feature pairs mirror one another in value
* Greater imprecision with lower numbers of features (although you could argue this is a positive)

The code itself is more readable than SVM (since it has been purged of most mathematics) and can be easily adjusted by the removal or addition of feature methods. There is additional flexibility for the data set being used as well—one simply needs to follow the reset instructions in the MLDetector.java file and make sure the training and test data sets following the image name instructions—so this detecting algorithm can work for any two binary classifications of images, as it should, so long as you have a decent data set to work with.

Lastly, it is worth discussing that this algorithm can be expanded into a non-binary classification system by continuing to add in new binary comparisons, and establishing a vocabulary of recognizable objects. This system would be pseudo-non-binary… if the vocabulary of the program becomes too great, it will lead to more and more imprecision. What this limit would be, I am not sure, but it would be a fun project to look into.

Lesson Plan – Day #3

**Introduction**:

This lesson is all about optimization and production. How can an already existing ML program get better, and secondly, how else can these new ways of thinking be applied in the world? This day includes both a formative assessment (a coding assignment) and a summative assessment (a report/proposal), the latter of which may have to be assigned as homework to be done outside of class if time is short.

Several tasks are asked of students within these activities:

1. Create a new feature method and implement that feature into the SVM algorithm
2. Scale up or down the number of features used in the algorithm
3. Train the program using training data and settle on the features producing the best results
4. Get an accuracy score using the test data and share your results on the board
   1. The student with the highest score will have to prove it on new test data
5. Envision a proposal as a businessman or woman who is looking to create a new project designed to meet some need in the world using machine learning

**Goals**:

* Create new feature methods
* Add or remove features
* Train the program and try out test data
* Consider real world uses for machine learning
* Plan out the steps to implement a ML project

**Activity Schedule**:

1. **Implementing Features**

*Time: 30-60 minutes*

Goal:

Get students practicing writing algorithms and data using segmentation techniques

Instructions:

Well, we want students to code, yes? It could be a doozy, depending on what the student tries to implement. Let’s say a student tries to write a feature that gets the ratio of white to black pixels. No problem. You would hope they could do that in 30 minutes or less. Let’s say a student tries to write a feature that finds the number of acute angles within the interior of the number. Oof. They might need more like 3 hours, not 30 minutes.

The goal here is to get students to get creative, to design an algorithm and implement it. Whether they find a happy conclusion in that algorithm working or not is another story, but I would encourage you as the teacher to be lenient in the expectations in this step, while also encouraging more *simple* feature methods rather than complicated ones.

The feature method should output some kind of fractional result, or one that can be converted into a decimal. Students should then switch out their new feature for an existing one. From there, students can try different combinations of features, or add additional features.

1. **Finalizing and Testing Project**

*Time: 15 minutes*

Goal:

Get students to practice using training data and test data

Instructions:

Once the feature methods and algorithms are set, have students stop running the training data, and instead run the test data. Students should record what percentage of the 30 test data images the program got correct (no, this percentage should not correlate with their grade), and then write their percentage on the board. The top percentage will be tested at the end of the period (by you), and if confirmed, this student can win some kind of prize! (You have a prize in mind, right?)

As part of their second formal assessment, students will then answer a few questions about their code in a document before submitting this document and the coding files that were altered… which should be MLDetector.java and/or Picture.java, depending on whether the latter was added to.

Questions in assignment:

1. What feature did you write and implement?
2. How did it work?
3. How well did it work? Did it improve the program?
4. Out of the 30 test data images, how many did the program get correct?
5. What features did your program use and what were the programs final weights after all the training and test data images?

*See Day 3 Assignment.docx*

1. (*Opt*) **Envisioning a Machine Learning Proposal**

*Time: 60 minutes*

Goal:

Get students thinking about the design process of building a machine learning program and the use-cases of machine learning programs. Get students practicing business skills, engineering skills, and critical thinking regarding the machine learning project steps

Instructions:

While this last assignment may not be necessary, it is still a great idea to get students exposed to real-world applications and design, as well as familiarizing students with the processes behind what it takes to get such a momentous project up and running.

First, go over these steps for creating and implementing a reliable machine learning project:

Step 0 – 1: Ask the right questions

Step 0: Begin with labels

Step 1: Set Objective

Step 2 – 4: Get the right data

Step 2: Getting data

Step 3: Split data

Step 4: Explore data

Step 5 – 7: Use algorithms to make recipes from patterns in data

Step 5: Get tools

Step 6: Train models

Step 7: Debugging AI

Step 8 – 9: Check that the recipes work on new data

Step 8: Validate your model

Step 9: Test your model

Step 10: Build a production ready system

Step 10: Productionization

Step 11: Make sure that launching it is a good idea

Step 11: Make a launch decision

Step 12: Keep your production machine learning system reliable over time

Step 12: Monitor and maintain system

The goal here is not for students to memorize this model, but to understand that a successful ML project requires a substantial amount of planning, materials, personnel, time, and maintenance.

Ask student the following questions to get them thinking about what they have learned and in order to prepare them for writing their proposals:

Questions:

1. What did you learn on Day 1?
2. What did you learn on Day 2?
3. What is machine learning? How is it different than regular ‘*linear*’ programming?
4. How is machine learning more powerful than linear programming?
5. What are some ways that machine learning is used now? Take a guess?
   1. Write student examples on board
6. What are some ways that machine learning could be used to improve everyday life?
   1. Write student examples on board

Lastly, students can follow the instructions and the rubric for their machine learning project proposal, the details of which can be found in the word document ‘Machine Learning Project Proposal.docx’.

*See Machine Learning Project Proposal.docx*