Preparing CS Teachers for Teaching Machine Learning in AP Classrooms

Author's Name: Peter Olson

Coach Name: I-Heng McComb

Host Organization: Stanford

ETP Type: Professional Development

Subject/Grade: APCSA / 11-12

Abstract

Major advancements in computer programming are pushing machine learning practices to the forefront of big tech industry standards. Extending on the series of lessons called 'Introduction to Machine Learning using Image Classification', teachers themselves need to be educated and equipped to understand these topics and lead lessons like these in the classroom. The teacher leading a lesson on these topics needs a thorough understanding of machine learning practices, algorithms, and standards so that they are prepared to use machine learning applications and address issues that arise when implementing new solutions.

The series of lessons developed for the teacher must be adequately prepared and organized so that a teacher can easily adopt such a curriculum into their own practice, and to do so requires its own set of instructions so that the lessons are amenable to CS teachers of any skill level. This ETP includes daily instructional materials, formative assessments, an image-processing program and machine learning program written in Java, instructional videos, and formative and summative assessments for the instructor.

Focal Content & Supporting Practices

- 9-12S.DA.8 Use data analysis tools and techniques to identify patterns in data representing complex systems
- 9-12S.AP.11 Implement an algorithm that uses artificial intelligence to overcome a simple challenge
- CSTP 3.5 Using and adapting resources, technologies, and standards-aligned instructional materials, including adopted materials, to make subject matter accessible to all students

21st Century Skills and Applications

- Critical Thinking
- Technological Literacy

Measurable Objective(s)

The objective is for computer science teachers to learn the core components of machine learning design, including algorithms, data, modeling, and training/testing, in order to feel ready and equipped to guide students through the 'Introduction to Machine Learning using Image Classification' set of lesson plans.

Formative Assessment(s)

Teachers will receive an organized series of resources, including video and document formats, that will be used to understand major machine learning topics, the image editor and machine learning image classification program written in Java, and the algorithm and feature design used within the program. A series of guided 'quizzes' are given to assure that the instructor knows all of the critical components of each of these subtopics along the way, including the organizational structure of the lesson plans and the key fundamentals of machine learning.

Summative Assessment(s)

After the teacher has gotten through all of the instructive materials and after they have completed all of the guided quizzes to check their understanding, a final summative assessment is provided to make sure that they understand the program and are familiar with its internal structures. This assessment is provided in the form of a coding assignment given to the teacher, for their own practice in working with the image editing methods, the feature methods, and the iterative process of using training data and testing data to produce a ‘learned’ program. For the task given, the teacher is given a minimal score they can attempt to achieve in their machine learning program, which if achieved, determines that they are ready for teaching the machine learning lesson to their APCSA class.

Fellowship Description

Tengyu Ma and his PhD student Jeff HaoChen are researchers at Stanford University specializing in machine learning and algorithms, including deep reinforcement learning, representation learning, robustness, non-convex optimization, and distributed optimization. Tengyu teaches machine learning courses at Stanford concurrently with his research studies.

During this fellowship, I learned about machine learning algorithms and project design, instances and their connections to features, how to use data to train and test the program, common fail-points of machine learning programs such as overfitting, and optimization techniques. These skills are highly relevant today, since many software engineering careers are increasingly looking for competencies in machine learning and artificial intelligence design. Furthermore, learning to work with data and using algorithms to learn about that data are highly relevant in careers such as data science and various engineering-focused careers, particularly data engineering.

The goals of my fellowship with them are fairly flexible—they are looking to adapt the series of lessons on machine learning from last year and make it more presentable and adoptable to APCSA teachers. The goal of this project is to develop a series of instructional documents and assessments that would teach instructors new to machine learning the fundamentals and prepare instructors to use the program and set of lesson plans in their own classroom setting.

This opportunity is meant to be a follow-up to the previous project that I designed with Tengyu and Jeff, whose original design and goal was to develop a set of machine learning lessons that would be appropriate for students at the high school level. In order to achieve this goal, an entirely new infrastructure was developed, in Java, using a pre-existing image processing program developed by Barbara Ericson as the base layer of interacting with and displaying images. This new machine learning algorithm was designed to reduce the complexity of the mathematics involved by removing all semblance of calculus and advanced statistics, such that a student in 11th or 12th grade would be able to pick up the machine learning program without needing to learn additional non-computer science material. The algorithm was based on the Support Vector Machine algorithm, but was simplified and combined with features of neural networks to create a weighted average-type algorithm, called the ‘simplified SVM’ algorithm, or sSVM for short. This new algorithm had the benefits of easy feature addition or reduction, without suffering significant processing time gain or loss, allowing for easy to integrate collaborative feature design, which was implemented into the assessment process of the lesson design, such that students could be directly involved in writing the machine learning algorithm, individually or in groups.

Fellowship Connection to School/Classroom

This project is developed and characterized as a professional development series of lessons designed to educate and instruct APCSA teachers on machine learning fundamentals, the image processing and machine learning algorithm ‘sSVM’, and the organization layout and lesson layout for the ‘Introduction to Machine Learning using Image Classification’ series of lessons. This series of lessons is fully developed, and includes several components that the teacher is expected to learn about and by the end of this instructional set of lessons, be able to teach to their own APCSA classroom. These components include three sets of documents spanning three days of material, including 1) Day 1 Worksheet, 2) Day 2.ppt, 3) Day 3 Assignment, 4) Intro to ML Lesson Plans Olson, 5) IsItACat.ppt, 6) Machine Learning Project Proposal Assignment, 7) Machine Learning Vocabulary Reference Sheet, 8) CatAndDogRecognize

Machine Learning Program, and 9) the PixLab Binary Image Classification Machine Learning program

The set of instructional documents and videos prepared for this project are designed to prepare the APCSA teacher to use and implement the machine learning lesson plan outlined above. Teachers are expected to learn the core components of machine learning and its programming fundamentals, going beyond the scope of the machine learning lesson they will give, such that they will have a well-rounded understanding of the machine learning field in computer science and data science. Teachers are also prepared for the individual documents and the organizational structure of the machine learning lesson, such that they are aware of each piece of the series of lesson plans, why they are important, what the answers are to the questions that they will be guiding students through, and how to implement and test student understanding using the included formal and summative assessments provided within the series of lesson plans. Finally, teachers are taught on the image processing and machine learning program, with the final summative assessment testing their own programming skills and understanding by writing their own image processing functions and their own machine learning features, in order to try and self-test their own readiness to implement the series of lesson plans on machine learning in their own APCSA classroom.

Following this project, I will attempt to test this series of instructional documents and assessments with another APCSA teacher, and see how well they are able to pick up the concepts and complete the tasks given to them in order to learn the series of lesson plans.

Instructional Plan

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 is 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
    a. Goals:
        - Use I/O with an existing (powerful) ML program
        - Practice identifying features
        - Compare and contrast linear algorithms vs learning algorithms
2. Day 2 – Designing Features and Experimenting with Training Data
    a. Goals:
        - Create a feature within the program and implement it
        - Add or remove features
        - Run program using training data, test accuracy using test data
3. Day 3 – Machine Learning Proposal
    a. 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

    <u>Goal</u>:

    Get students thinking about identifying features and how computers can learn

    <u>Materials</u>:

    - Some kind of fake, squishy object that looks like a heavy, hard object. If this is hard to get, see the alternative activity below this one

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

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 it was real or fake? What would help it get better at determining the difference between real and fake rocks?

1b. **Alternative to Icebreaker Activity — Identifying the Fake Apple**

Time required: 15 minutes

Goal:

Get students thinking about identifying features and how computers can learn

Materials:

- 1 fake green apple. I was able to find some really convincing fake fruit at *Michaels*
- 2 or more real green apples

Instructions:

First, select as many student volunteers as apples that you have. These volunteers should all be on the same side of the room. Tell them that they are not allowed to say anything for this activity. Then tell each volunteer to hold up their apple. Pick on a new participant who is on the opposite side of the room. They should be at least 10 feet away for the best conversations that will happen afterward.

Now you can let the class know that one of the apples is fake and the others are real. If you were able to find a good fake apple, and unblemished or similar real apples, this will work better. Ask the participant not holding an apple which of the apples is the fake one. Most likely, they will be able to correctly identify the fake apple (but do not allow them to get closer to see the apples or touch them). Then, have the participants holding the apples reveal which are the real apples and which are the fake ones, and students can pass around the apples if they want to see them.

Have a conversation about the activity by asking and discussing the following questions:

1) How did the participant (ask them) know that the apple was fake? What features of the fake apple helped identify the fake apple? If they got it wrong, have them look at the fake apple and identify how it's features are distinguishable from the real apples
2) How did the participant know that the fake apple's features were not 'accurate'?
3) Would a baby be able to tell the differences between the fake apple and real one easily (if they had to point at the one they wanted to eat)? Why?
4) How could a computer learn like a baby learns what makes an apple, an 'apple'?

2. **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 'Day 1 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:
   a. What features would help determine whether the picture is a cat or not?
      i. Pick 2 or 3 features from their suggestions
   b. What does it mean for a picture to be a cat versus having a cat in the picture?

2. During slides #6-10:
    a. Do these features help correctly determine that the picture is a cat?
3. After slide #14:
    a. What features should we look for to identify that the streetlight is green?
        i. Pick 2 or 3 features from their suggestions
4. During slides #15-19:
    a. Do these features help correctly determine that the streetlight is green or not?
5. After slide #19:
    a. 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?

3. **Cat and Dog Recognizer ML Program**
   Time required: *30 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

3b. **Alternative to Cat and Dog Recognizer ML Program — ChatGPT Prompt Engineering Lesson**
   Time required: 30 minutes

Goal:

Teach students how to use ChatGPT effectively and how to make good use of its versatility

Instructions:

Open ‘Day 1 ChatGPT Prompt Engineering.ppt’. First, have students make a ChatGPT account if they have not made one already. Once everyone has an account and has accessed ChatGPT, you can begin the lesson on ChatGPT. While the first few slides are about what ChatGPT is and how it works, the majority of the slideshow examples are of the different kinds of tasks that ChatGPT can handle, showing examples (gifs) of prompts being entered and the responses that ChatGPT gives. For each slide, encourage students to experiment with each one of the prompt types by asking ChatGPT their own questions or giving ChatGPT their own tasks.

The last few slides teach about prompt engineering and how to be efficient and effective with using prompts for ChatGPT or other AI language-based bots, and how these prompts are different from other search engines, such as Google.

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

   Time required: 25 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 MNIST Numbers.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?
      a. Write down all ideas from students
   3. (Opt) How are they different based on their pixels?
      a. On their size? Shape? Distribution? Darkness / lightness? Sharp edges?
      b. Write down all ideas from students
   4. *Pick a feature from the list*. How would you program this feature? What would it return to you to tell you about the success of that feature?
      a. 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?

2. **Introduction to Machine Learning with Images**
   Time required: 10 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: 30 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 and MLFeatures.java**

   Time required: 35 min

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

MLFeatures.java contains the feature methods that are used to produce weights for the algorithm. You can comment out or remove methods completely, and you can add in any number of methods. Unlike other ML algorithms, adding in feature methods does not exponentially increase the processing time (unless you write an extremely inefficient feature method that is individually very slow). It is up to you on whether to share how MLDetector.java works or not, but all students need to understand is how to write feature methods in MLFeatures.java for everything they need to do. In my experience using these lesson plans, I think it would be easier just focusing on MLFeatures.java.

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 algorithm 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 to (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
    a. 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: 50 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.

   Activity:
   Put students into groups of two. The goal of each pair will be to design one feature method that can be added to MLFeatures.java, and to adjust MLFeatures.java in order to have the most accurate ML program in the class. Students are allowed to remove feature methods or add new ones, but they must have a minimum of two feature methods. If students are able to write a feature method within this time, that would be excellent, but ultimately the competition for testing each student pair's program will take more time than is available for this segment, and will have to be completed at a later time.

   Each student pair can run their program against the test images once the program has been trained by all 80 training images. Note that the parameters for MLRunner.java need to be adjusted to run test images. Whichever team correctly identifies the most test images correctly, wins.

   Assignment: 'Day 3 Writing Features.docx'

2. **Finalizing and Testing Project**
   Time: 20 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?

Assignment:

Day 3 Writing Features.docx

3. (*Optional*) **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: Productionisation

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 from 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?
    a. Write student examples on board
6. What are some ways that machine learning could be used to improve everyday life?
    a. 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

Additional Supports

1. Distance learning
    a. Provide students online equal access to materials through links, screen-sharing, and personalized comments in the video conference app
2. ESL / ELL
    a. Turn on captions for videos, provide vocab sheet starting on Day 2
3. SPED
    a. Have students sit in groups and work together. Focus on groups sharing and learning from each other, and running the programs using training data

Materials

- https://drive.google.com/drive/folders/1rnGoqRu4SBB809MEbZGdyYl5hFJdnjF8?usp=sharing Ignited Folder. Contents are separated into four zipped files due to their large size. Unzip all files to see contents once downloaded.

- https://drive.google.com/drive/folders/1ezLy3Ubo14Q8ewhUW-mTzaGEgoHmzvlK?usp=sharing Personal Folder. These are the same materials as above, and are viewable and copyable by anyone.

References

- YouTube. (2019). *What are neural networks? YouTube*. Retrieved July 25, 2022, from https://www.youtube.com/watch?v=2hMuYi2Wudw.
- codebasics. (2020). *Image classification vs Object detection vs Image Segmentation | Deep Learning Tutorial 28*. *YouTube*. Retrieved July 25, 2022, from https://www.youtube.com/watch?v=taC5pMCm70U.
- Ramo, K. (n.d.). Version (January 6th 2018). *CatAndDogRecognizer*. github. Retrieved from https://github.com/klevis/CatAndDogRecognizer.

- Ericson, B. (n.d.). PixLabVersion (an early version, which was amended - originally used for just image processing). Barbara Ericson.

Keywords

Machine learning, image classification, programming