CS 6240: Final Project

Goal: Use machine learning for foreground-background classification in high-resolution brain scans.

After learning about various classification and prediction techniques, we will now apply our expertise to a real machine learning challenge. This challenge is organized as a friendly competition between teams, each having two members. If you have not done so yet, talk to your class mates and use the online forum to finalize team formation ASAP.

Each team has to create all deliverables from scratch. In particular, it is not allowed to copy another team's code or text and modify it. (If you use publicly available code/text, you need to cite the source in your code and report!) You also cannot just replicate another team's approach. Every team has to develop their own solution independently.

The project has two separate assignments with different due dates: a report and the presentation slides. **Make sure you submit each to the appropriate assignment.**

Project report: Please submit your solution through Blackboard by the due date shown online. For late submissions you will lose one percentage point per hour after the deadline. This HW is worth 100 points and accounts for 15% of your overall homework score. To encourage early work, you will receive a 10-point bonus if you submit your solution on or before the early submission deadline stated on Blackboard. (Notice that your total score cannot exceed 100 points, but the extra points would compensate for any deductions.)

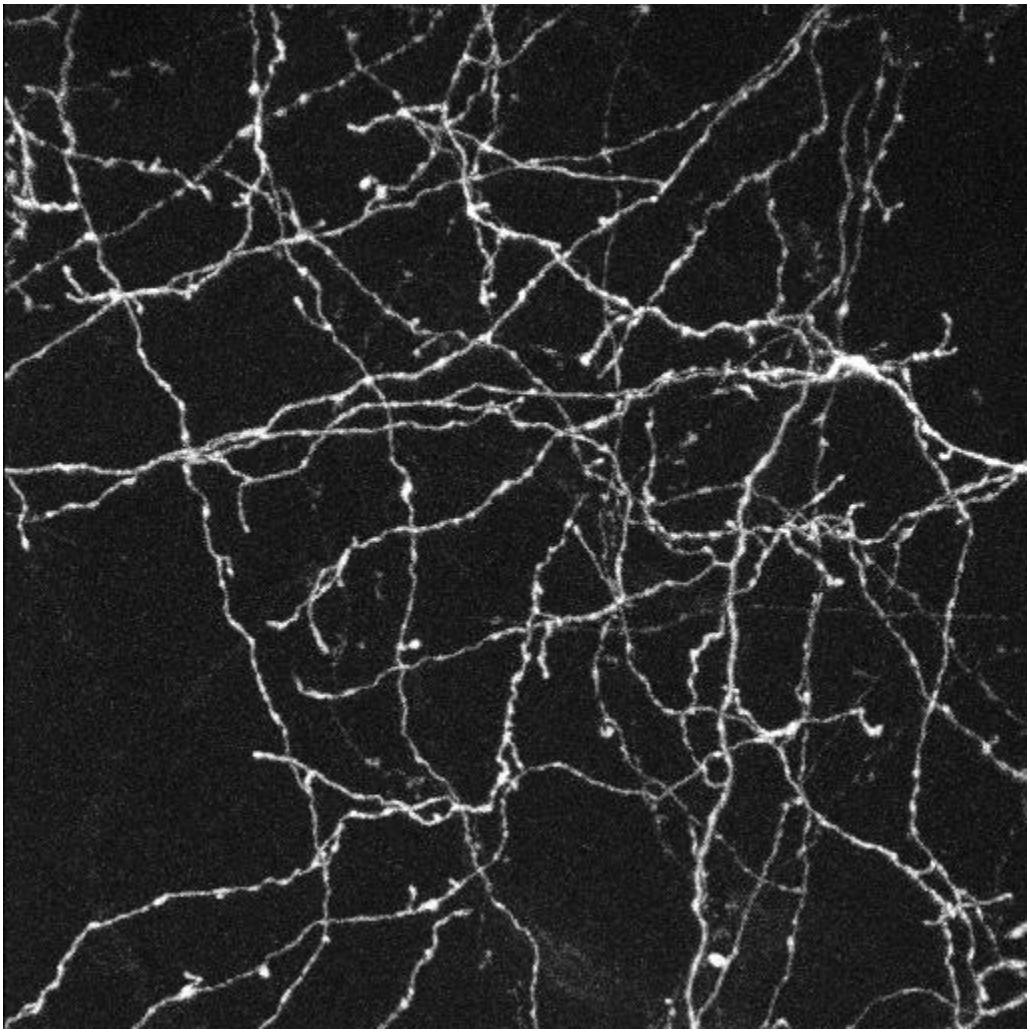
To enable the graders to run your solution, make sure you include a standard Makefile with the same top-level targets (e.g., *alone* and *cloud*) as the one Joe presented in class (see the Extra Material folder in the Syllabus and Course Resources section). You may simply copy Joe's Makefile and modify the variable settings in the beginning as necessary. For this Makefile to work on your machine, you need Maven and make sure that the Maven plugins and dependencies in the pom.xml file are correct. Notice that in order to use the Makefile to execute your job elegantly on the cloud as shown by Joe, you also need to set up the AWS CLI on your machine. (If you are familiar with Gradle, you may also use it instead. However, we do not provide examples for Gradle.)

As with all software projects, you must include a README file briefly describing all of the steps necessary to build and execute both the standalone and AWS Elastic MapReduce (EMR) versions of your program. This description should include the build commands and fully describe the execution steps. This README will also be graded.

Presentation slides: Please submit your solution through Blackboard by the due date shown online. For late submissions you will lose one percentage point per hour after the deadline. This HW is worth 100 points and accounts for 10% of your overall homework score. Make sure your presentation is a **PDF** file.

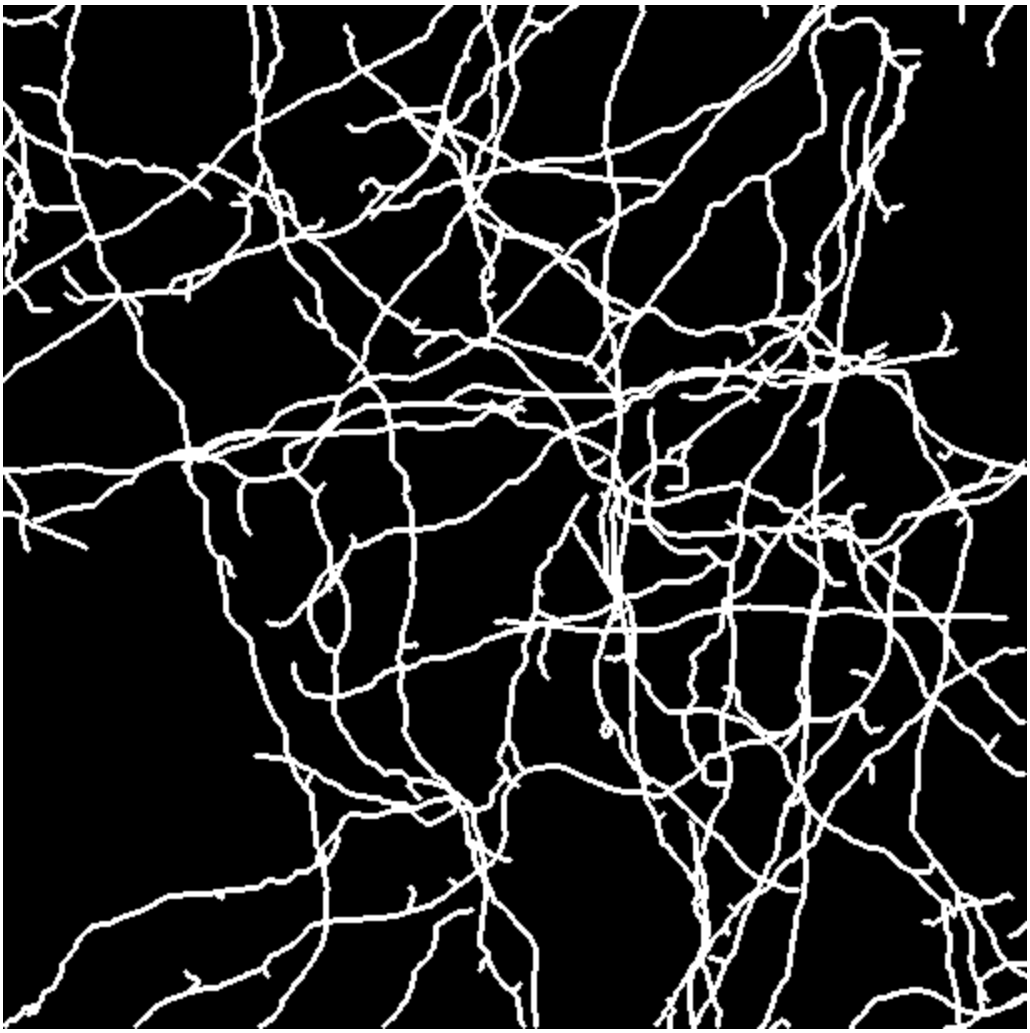
Problem Background

We are working with scientists who are interested in turning high-resolution brain scans like the one below into a graph representing nerve connections indicated by the bright lines. Note that the original image is 3-dimensional, but for better intuition we show the 2-dimensional projection on the X-Y plane.



As you can see, the image is noisy, but axons (the lines going across the image) are clearly visible. To improve the quality of algorithms that automatically trace these axons in an image, we want to classify each pixel as foreground (belongs to an axon) or background (does not belong to an axon).

The scientists have put in a lot of effort, manually tracing the axons. Intuitively, the traced data looks like the image below, where white indicates foreground and black indicates background. (In reality, there is also a 2-pixel wide "zone of uncertainty" between white lines and background. We do not show it separately to avoid clutter.)



Labeled Input Data

Using the manually traced image, we generated labeled data as follows:

- Select a pixel (i, j, k) in the image.
- Extract a neighborhood vector centered around (i, j, k).
- Extract the label of (i, j, k) from the trace. Here 1 indicates foreground; 0 indicates background.
- Save the record as the neighborhood vector, followed by the label.

Assume you selected pixel (i=10, j=10, k=10) and are working with a neighborhood of size 3x3x3. We would store these 27 pixels as a vector n with 27 elements, where n[0] stores the brightness value of pixel (9, 9, 9), n[1] the brightness of (9, 9, 10), n[2] of (9, 9, 11), n[3] of (9, 10, 9), n[4] of (9, 10, 10), and so on. The domain experts recommend neighborhoods of size 21x21x7, which is what our labeled data sets contain.

Increasing Training Data Diversity

This part is completely optional. It increases the number of labeled data records, which may help in achieving greater prediction accuracy.

We can use the following insights to increase the number and diversity of training records: if we rotate a neighborhood around a center pixel with label L, then we obtain a "new" neighborhood whose center still has label L. Similarly, if we mirror the neighborhood along a plane going through the center pixel, then we obtain a "new" neighborhood without affecting the label of the center pixel. Hence, given a training record as created above, we can produce as many additional versions of it as we want by rotating and mirroring it.

In practice, the Z-dimension of the brain images behaves differently than the X and Y dimensions. This is caused by the imaging process, which is very reliable in X and Y, but produces inaccurate shapes in Z. For this reason, we recommend applying rotation and mirroring only in the X-Y plane. Also, notice that rotating by arbitrary angles requires non-trivial interpolation to assign the resulting brightness values to (discrete) pixels. Furthermore, rotating a mirrored image is equivalent to rotating followed by mirroring. As a consequence, we recommend you only consider the following transformations:

1. Rotate by 0 degrees, with and without mirroring along the Y-axis.
2. Rotate by 90 degrees, with and without mirroring along the Y-axis.
3. Rotate by 180 degrees, with and without mirroring along the Y-axis.
4. Rotate by 270 degrees, with and without mirroring along the Y-axis.

Intuitively, each original training record produces 8 new training records (one of them being identical to the initial one). Mathematically, the operations can be expressed as shown below. (Note that for simplicity, the formulas below are for center-point coordinates x=0 and y=0, rotation around this point in the X-Y plane, and mirroring along the Y-axis. You need to adjust the formulas accordingly when the center point has different coordinates.):

- Mirroring turns pixel (x, y, z) into (-x, y, z).
- Rotation by 90 degrees turns pixel (x, y, z) into (y, -x, z).
- Rotation by 180 degrees turns pixel (x, y, z) into (-x, -y, z).
- Rotation by 270 degrees turns pixel (x, y, z) into (-y, x, z).

Data Set and Classification Task

You are given several labeled data sets, each consisting of rows that contain an input vector of 21x21x7 brightness values from an image, together with the center pixel's foreground-vs-background label. These data sets can be found at https://drive.google.com/drive/folders/1EJBgJFmp-FQf2czw9LGImoOhEO2OvOoo

Use the labeled data from images 1, 2, 3, 4, and 6 to train the most accurate model for predicting the labels in image 5.

Hints and Suggestions

The standard way of training and testing a classification model in machine learning is as follows:

1. Partition the labeled data into 3 separate sets: training data, validation data, and test data. This is done randomly, such that most data goes into training (typically 60-80%), and less into the other two (typically 20-10% for each).
2. Repeat until satisfied with model accuracy
    a. Set model parameters to some values.
    b. Train model on the training data using this setting.
    c. Evaluate model accuracy on the validation data.
3. The winning model is the one with the greatest accuracy on the validation data. Evaluate its accuracy on the test data to get realistic estimate of accuracy on unseen future data.

Since we have a separate test set for final evaluation, you can skip step 3. This leaves more data for training and validation set.

While uniform random partitioning into training and validation data often works well, there are applications where it is not sufficient. Ours is one of those tricky cases. Assume we have two labeled records with centers (x, y, z) and (x+1, y+1, z+1), respectively. Uniform sampling might assign (x, y, z) to training, and (x+1, y+1, z+1) to the validation data. Clearly, the two neighborhoods and the labels of these two pixels are highly correlated. Hence a model that overfits to the training data will also do very well on validation pixel (x+1, y+1, z+1). To prevent overly optimistic validation accuracy caused by such correlations, we have to be more careful when partitioning the labeled data. A safe procedure for ensuring independent training and validation data is to partition by image. For example, training data come from images 1, 2, 3, and 6; while 4 is used for validation. Alternatively, you can also partition each image into two separate "blocks", using one block for training data and the other for validation data.

Given a model type (Random Forest, SVM, k-NN, etc.), the secret for achieving the best possible accuracy is to explore a lot of parameter combinations. In some cases there are few parameters and it is relatively easy to navigate to the best settings; in other cases you have to cast a wide net. Find recommendations on the Web or discuss your questions and findings in the discussion forum. Either way, make sure you fully understand the meaning and potential impact of a parameter so that you can make educated decisions about limiting the huge parameter space.

For classification accuracy, a high value does not necessarily mean that you found a good model. For example, assume 99% of the data has label 0, and only 1% of the data has label 1. Then a "dumb" model that always predicts label 0 for every input will have 99% accuracy. This means that if your model achieves less than 99%, it does not even beat this "dumb" model!

Requirements

Model Training Program

1. Use the labeled data in any way you deem appropriate to train, tune, and test your model.
2. Use a MapReduce or Spark parallel processing program upon distributed data. This should be a single program composed of one or more jobs. Make sure your program clearly improves over

the single-processor solution by overcoming a computation or memory bottleneck. To evaluate your design, ask yourself: "Does it achieve good speedup?" and/or "Is it able to use more of the training data (rows and/or columns) compared to an approach that has to fit all training data into memory on a single machine before being able to train a model?"

3. Choose your favorite classification or prediction technique, including ensemble models, to build a prediction model. (You may also implement your own, but that might be too time-consuming.) Note that we are solving a classification problem, hence the labels predicted by your model should be either 0 (background) or 1 (foreground). If you use a technique that returns probabilities, make sure you threshold them accordingly when producing the final predictions.
4. You may use any data-mining related open source libraries, languages, or environments as long as they are called from your single program. An example is to use Weka libraries in a Hadoop MapReduce program as discussed in the modules.
5. You must write the program so that it is out-of-scope to use a pre-packaged solution (e.g., H2O). If you are uncertain whether an external component is allowed, ask on the online forum. A good test is to ask yourself: "Did I solve any non-trivial distributed computation problem myself, e.g., determine how input is partitioned or how load is assigned?"
6. The resulting model must be persisted for (possibly repeated) use by the prediction program.

Make sure you discuss the following in your report: (1) How many tasks are created during each stage of the model training process? (2) Is data being shuffled? (3) How many iterations are executed during model training (for methods that have multiple iterations)? (4) You also need to find out, how to control performance. In particular, change the number of partitions or the actual partitioning itself and report how this affects running time.

Prediction Program

1. Use MapReduce or Spark to implement a parallel prediction program that uses the previously generated, persisted model to predict for each record in the unlabeled data whether the center pixel is foreground (1) or background (0). Stated differently, this program should replace the "?" label in the unlabeled data by either "0" or "1", based on the model's prediction.

Validation

1. As part of the model training process, most data mining techniques require tuning of parameters. Use the given labeled data in any way you deem appropriate to do this.
2. We will use accuracy (ACC) for the final evaluation. In the final evaluation test set there are about 0.0057% foreground and 99.9943% background pixels.

Testing

1. Once you found a satisfactory model, execute your prediction program on the unlabeled test data (image 5) to produce your final predictions.
2. Your output file must be named as specified in the deliverables. It should be a single file with a single column and as many rows as there are test points. Conceptually, you need to replace all

"?" labels with either 1 or 0, and remove all columns except for the label column. (Do not change row order or merge/delete any rows etc.) Make sure all predictions are stored in plain text, i.e., exactly as strings "1" or "0" (without the quotation marks, of course).

These formatting rules are strict for both grading your homework and determining the competition winner. To assist you in validating and tuning your model and also making sure your prediction file format is correct, a simplified version of the scoring program will be made available on Blackboard. You may use and modify this code but, as written, it represents the de facto scoring algorithm.

Final Project Report

The final report should fully describe your prediction methodology as well as the design of your programs. This should be concise (see page limit in deliverables section), but comprehensive, as it serves as project documentation to assist graders in evaluating its quality. Also report on your validation results using (and describing) your quality measurement. Make sure your report covers the following points:

- Type of model used and parameters explored for it.
- Pseudo-code, possibly accompanied by examples and images, illustrating the steps of the distributed computation. (For Spark this is often not necessary; here a short Scala program can be shown.) It should make clear how data and computation are partitioned and assigned to the workers. This needs to be done for both model training and prediction. Discuss:
    - How many tasks are created during each stage of the model training process?
    - Is data being shuffled?
    - How many iterations are executed during model training (for methods that have multiple iterations)?
    - How did changes of parameters controlling partitioning affect the running time? E.g., for bagging, was it better to partition the model file, the test data, or both?
- If you performed any pre-processing, briefly summarize what you did and why you did it.
- Accuracy numbers for different parameter settings you explored. Use your own judgement in choosing, which results to include.
- Running time and speedup results for the model training and the prediction phase. You do not need to include numbers for every model or model instance you tried. Use good judgement to select a few "representative" numbers.

Presentation

Each team will present highlights of their solution to the class during final exam week. The allotted time is 6 minutes with another minute for questions. Make sure you submitted a PDF version of your slides by the deadline. (See also the file naming convention in the deliverables section.) This will enable the instructors to download all of them to a designated presentation laptop, avoiding time loss due to transition problems. For compatibility, all presentations have to be in PDF format. Note that this might

prevent certain fancy animations. Consider the following recommendations when preparing your presentation:

- Think of this as a hiring talk. You are interviewing for a high-paying job with your favorite Big Data company and the class is a group of experts from this company who will decide if they should hire you or not. How do you show this technically versed audience in a few minutes that you are capable of solving a Big Data problem? However, avoid over-selling!
- Since everybody knows the data and the problem, you do not need to talk about it. Dive right into the *solution* you found.
- Focus on a few carefully selected results that best showcase your expertise as a MapReduce or Spark programmer. The presentation should give an overview of your approach, but you want to spend most time on the interesting aspects that you think make your solution special or highlight your analytical and creative thinking.
- Present graphs or tables showing concrete measurements. Think about which ones you need in order to make a case that you (1) solved the problem well and (2) your code is scalable or in other ways improves over the single-processor solution.
- Very briefly present your validation accuracy results for comparison to the test results, which will be published by the instructors.

Points will be awarded based on slide design and quality of oral presentation. Exceeding the 6-minute time limit will result in automatic deduction. It is important that you practice presentation timing.

All students have to be present for all presentations. Absence will result in significant point loss on the presentation assignment.

Competition (extra credit)

To raise the stakes and add some fun, there will be a competition for the best predictions. The members of the winning team will receive a 30-point credit on the presentation assignment. Second to eighth place will receive 20, 15, 10, 5, 5, 5, and 5 points in credit. Total score is not capped, i.e., the credit might result in some teams with more than 100% on the presentation—with the corresponding impact on the final course grade. Team ranking will be determined based on prediction **accuracy** on the unlabeled data:

Accuracy = (TruePositives + TrueNegatives) / TotalTestRecords.

We might ask the top 4 winners to demonstrate their project programs performing the model building and prediction. This would take place outside of class in a scheduled 10-15 minute meeting during finals week. At least one team member must attend and demonstrate the project from source code build through predicting the unlabeled data. In the unfortunate case that a top team is unable to demonstrate their project, they will vacate their position and all lower scoring teams will rise one place. Being unable to demonstrate your project does not affect your presentation score, just your qualification to receive extra credit points.

Deliverables for Project Report

Submit the following in a **single zip file**:

1. The project report as discussed above. The report cannot have more than 6 pages in the font size and layout of this assignment. The shorter, the better—just make sure all the important information is included in a concise manner. (1 PDF file; 40 points)
2. The syslog (or similar) files for a successful run of your model training program on AWS. (5 points)
3. The syslog (or similar) files for a successful run of your prediction program on AWS for the provided file of unlabeled data records. (5 points)
4. Final prediction output file for the full unlabeled data, as specified in the requirements. (Include the entire file, even if it exceeds 1MB! Just make sure you removed all columns except for the label column.) **Make sure you name the result file according to your team members as follows**: use the last names of all team members, sorted alphabetically, as the file name. For instance, the two-person team Joe Smith and Mary Miller would use file name MillerSmith.csv. (10 points)

Please name your submitted ZIP file "**CS6240_lastName1_lastName2_report**", with the last names of all team members in alphabetical order. Never include input data, output data over 1 MB, or any sort of binaries such as JAR or class files.

Make sure the following is easy to find in your **CCIS Github** repository:

5. The source code of your programs, including an easily-configurable Makefile that builds your programs for local execution. **Make sure your code is clean and well-documented. Messy and hard-to-read code will result in point loss.** In addition to correctness, efficiency is also a criterion for the code grade. However, *low prediction accuracy* does not result in any deduction. (40 points)

Deliverables for Presentation

1. The presentation slides as discussed above. **Make sure you name the file according to your team members as follows**: use the last names of all team members, sorted alphabetically, as the file name. For instance, the two-person team Joe Smith and Mary Miller would use file name MillerSmith.pdf. (1 PDF file)