Project 2 (25% of your final grade)

Image Classification

Deadline: December 5th, 11:59pm, 2025.
Perfect score: 100.

Assignment Instructions:

Teams: By default, the project should be completed by teams of two students. Maximum of three students in a group is allowed (in case we couldn't get all teams with 2 students). You are also free to choose to work on your own, in which case you can get a 8 point bonus (for this project, it means 8*25% = 2 for your final grade).

Submission Rules: Submit your reports electronically as a PDF document through Canvas of the course. For programming questions, you also need to submit a compressed file via Canvas, which contains your code. Do not submit Word documents, raw text, or hardcopies etc. Make sure to generate and submit a PDF instead. Each team of students should submit **only a single copy of your solutions** and indicate all team members on their submission. Failure to follow these rules may result in lower grade for the project.

Project demonstrations: You will need to demonstrate your program to me in real time on a date after the deadline. The schedule will be coordinated between your team and me. During the demonstration, you have to use the file submitted on Canvas BEFORE the deadline and execute it on your laptop computer. You will also be asked to describe the architecture of your implementation and key algorithmic aspects of the project. **You need to make sure you can complete the demonstration and answer my questions in the 20-30 minutes allotted for each team**. If your program is not running properly on the computer you are using and you have to spend time to configure your computer, this counts against your allotted time.

Late Submissions: You are supposed to finish your work before the deadline specified at the top. Submissions after the deadline are not acceptable.

Extra Credit for LaTeX: You will receive 4 extra credit points (4*25%=1) if you submit your answers as a typeset PDF (using LaTeX, in which case you should also submit electronically your latex source code). Please keep in mind that we will not accept hardcopies and scanned document (handwritten then scanned).

Collusion, Plagiarism, etc.: Each team must prepare its solutions independently from other teams, i.e., without using common notes, code or worksheets with other students or trying to solve problems in collaboration with other teams. You MUST indicate any external sources you have used in the preparation of your solution. Do not plagiarize online sources and in general make sure you do not violate any of the academic standards of the department or the university. Also refer to the Academic Honesty, Usage of AI tools, Plagiarism Policy in the course syllabus on Canvas to make sure you follow throughout the project. Failure to follow these rules may result in failure in this project.

Project Description:

Acknowledgement: This project is based on the one created by Dan Klein and John DeNero that was given as part of the programming assignments of Berkeley's CS188 course.

In this project, you will have the chance to work on image classification problems, which have been researched and addressed rapidly in recent years since the AI evolution. Specifically, you will implement two classifier algorithms we have covered in class

1. **Naive Bayes**
2. **Perceptron**

to classify two image data sets (which are already simplified for you)

1. a set of **digit images**
2. a set of **face images** in which edges have already been detected

Digit images classification falls under the tech category of Optimal Character Recognition (OCR), which focuses on extracting text from image sources. The first data set on which you will run your classifiers is a collection of handwritten numerical digits (0-9). This is a very commercially useful technology, similar to the technique used by the US post office to route mail by zip codes. There are systems that can perform with over 99% classification accuracy (e.g.,LeNet-5).

Face detection is the task of localizing faces within video or still images. The faces can be at any location and vary in size. There are many applications for face detection, including human computer interaction and surveillance. You will attempt a simplified face detection task in which your system is presented with an image that has been pre-processed by an edge detection algorithm. The task is to determine whether the edge image is a face or not.

The two algorithms you will be implementing on digit image classification and face detection - **Naive Bayes** and **Perceptron** belong to the category of supervised machine learning approaches, which have been proved to be pretty effective in solving image classification problems, a field where typical search methods with logic reasoning fail to solve. A general pipeline for such a supervised learning algorithm has been demonstrated and discussed during the class (Fig. 1).

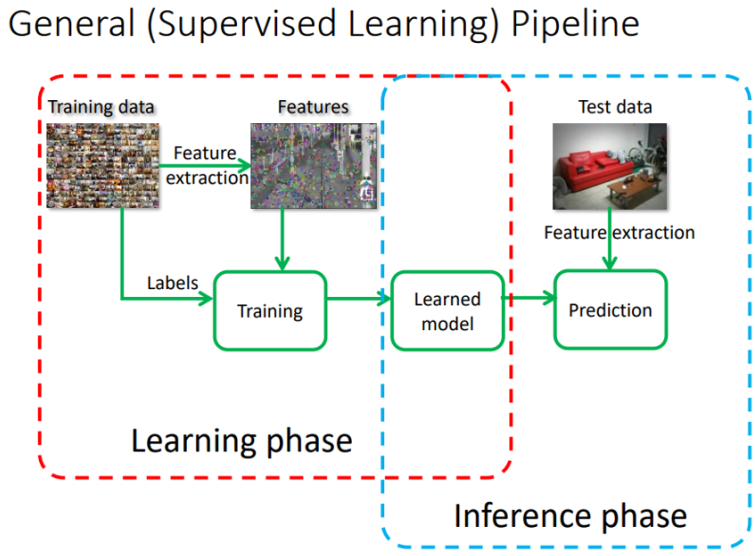


Figure 1: General Pipeline for Supervised Machine Learning

From the pipeline shown above, you can now see the major steps involved in building a supervised machine-learning system (such as Naive Bayes or the Perceptron) that learns patterns from data—like image data (digits, faces) in this project.

1. **Data acquisition** - To train any supervised learning algorithm, you must first collect sufficient labeled data for **training** and a separate set of data for **testing**. For example, think about how many cats you have seen in your life—those experiences form the "training data" your brain uses to recognize a new cat, which acts like the "test data." In real machine learning tasks, raw data is often noisy or imperfect (e.g., point-cloud data from a LiDAR sensor), so sometimes you may need to perform pre-processing to clean or normalize it before use.

2. **Feature Extraction** - Unlike humans, a computer does not understand an image holistically. It only processes numerical information—such as the RGB values of each pixel in a color image. Therefore, we must convert raw data into **a set of meaningful features** that both capture important information about the data and can be understood by the learning algorithm. In classical machine learning, this step is critical because the model's performance heavily depends on how well the features represent the underlying patterns.

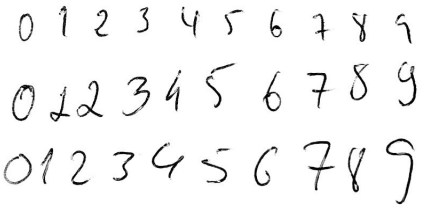
3. **Training the model** - Once you have pre-processed data with well-defined features, you can feed it into a chosen learning algorithm. Different algorithms learn in different ways, specifically in this project

   - **Naive Bayes** learns a probability model based on the frequency of features.
   - **Perceptron** learns a set of weights that define a decision boundary

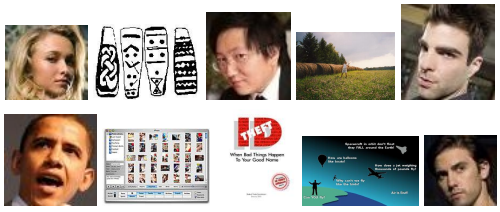
   During training, the model makes predictions on the training samples and compares them with the known ground-truth labels. Whenever it predicts incorrectly, the model updates its internal parameters to improve future predictions. This feedback loop—guided by correct labels—is why the paradigm is called supervised learning.

4. **Inference and Evaluation** - After training, you evaluate the model on data it has never seen before. A **validation set** is often used to tune the model and check that it generalizes well. Then, a separate **test set** is used to measure the final accuracy in an objective and unbiased way. Since the test data has no overlap with the training data, the performance on this set gives a realistic estimate of how well the model will perform on new, unseen inputs.

Get enough knowledge and feel excited about this project so far? Let's get into this!!



Which Digit?



Which are Faces?

Which Digit? (Simplified)

Face or not face? (Simplified)

Figure 2: Examples of the data points in the data set. The bottom ones are the simplified data you will be using in this project

1. **[10pts] Data Processing** - The data are attached to the project as a zip file called cs4346-data, which contains two datasets - a dataset for digits and the other for faces (Examples of image data (digits and faces) are shown in the bottom line of Fig. 2). And for each dataset, there are three type of data

   - **Training data** - The data you use for training your algorithms.
   - **Validation data** - The data you use to validate your algorithms so that you can see how good they perform and make corresponding adjustments.

- **Test data** - The data you use to test your algorithms, calculate and report the accuracy of learning on your report. You statistics for accuracy should come from the experiments on these test data.

And for each of them, the ground truth labels are provided to indicate what digit it is, or is it a face or not, which will be used for training, validation, and testing, perspectively.
The data of the same category (training, validation, or test) are put together in a text file (done intentionally). You first job is to parse these data information correctly in these files, so that you know the dimension of each individual image and its corresponding label. This step should facilitate your subsequent training, validation and test step.

2. **[15pts] Feature Extraction from Data** - You need to design the features for each of the two datasets - face and digits, and write a program that is capable of extracting the features for each image in the dataset. In class, We have talked about two possible features $\phi(x) = (\phi_1(x), \phi_2(x), ..., \phi_l(x))$ that you can use

1. **just pixels (very raw)** - $\phi(x) =$ (pixel 1 of x, pixel 2 of x, ..., pixel l of x) where each feature element $\phi_i(x)$ can be a binary value indicating whether that pixel is occupied by a symbol (hash tag, or "+" symbol, etc.)
2. **counting (very naive)** - $\phi(x) =$ (number of whitespace of x, number of symbols (hash, "+" symbols) of x).

You need to use **two** features for each of the classification problem (digits and faces). Feel free to use the above two. You can use the same two features for both problems. **You are also welcome to define your own features**. This is where your creativity comes into. For instance, I can give you one more feature option - for the raw pixel features above, you can modify it in a way that you just divide the image into a regular grid (say 10x10). Each square in the grid defines a binary feature that indicate whether there is anything marked inside the square.
My suggestion is - don't spend too much time on designing complicated features that require a lot of coding. You would be surprised that even pretty simple features could be great predictors. Your classifiers will be able to do quite well on these tasks when given enough training data.

3. **[50pts] Implement Classification algorithms** - Once you have processed your data and defined your feature for the data, you need to implement **two classification algorithms** (a) Naive Bayes and (b) Perceptron on **both datasets** (detecting faces and classifying digits). For each algorithm, and for each dataset, train the two algorithms on the part of the data set that is reserved for training. First, use only 10% of the data points that are reserved for training, then 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, and finally 100%. In this case, when it comes to the evaluation part (which will come next as part 4), you can see how the results change with different percentages of the training data used to train your models. **Validation data** is provided to you to be used to validate your algorithms internally as a way to check if your implementation is correct or not. You don't need to report any evaluation with validation data and all the performance evaluation is related to test data. Think in this way - when you release a product - say a cellphone, validation is about testing internally before you release to make sure it is all functional and ready to be released. When the users are using the released product, they are actually testing it and potentially write a review about its performance and quality.

4. **[25pts] Performance Evaluation of the trained models** - Evaluation results you put in the report should **ONLY** be from the test data. The evaluation we care about is straightforward - (1) the **prediction accuracy (or error)** of the models you train, and (2) the **runtime** of each algorithm with respect to each percentage of the training data you used. Therefore, there are two manifold of performance comparison

- Comparison within each algorithm - In part 3, we have used different percentage of the training data to train our model. You should provide evaluation results showing how the prediction accuracy or error, as well as the runtime change with respect to the increased percentage of the training data.
- Compare two algorithms - For each percentage of the training data you used, compare the performance of the two algorithms - both in terms of prediction accuracy (or error) and the runtime needed for training.

How to compute the average prediction error (or accuracy) and its standard deviation? What we want is a metric for how consistent our prediction accuracy will be if we use a certain number of training points. Thus, we need to vary which training points are used and see the spread of results. The following pseudocode would accomplish this:
For i=1:5 (you can use more iterations, if time permits)
    Train on 10% of randomly selected data points
    Test on test data and save the accuracy in acc[i]

End
Return mean(acc) and std(acc).
Repeat the same for 20%, 30%, etc.
You may find the random module in Python to be especially convenient in choosing your random samples.

Report expectation: For the above 4 parts, write a report describing how you handle the original data, how you design and extract features for the provided image data, how you implement the two algorithms, then demonstrate and discuss the results, the learned lessons, as well as future improvement if any. During the demo, you need to demonstrate that your understanding and implementation of the project, and can run the code without issues (i.e., reads an image from the a data file, and returns a predicted label, extract features you defined, run the algorithms on the fly, etc.).

Please keep in mind that:

- You can use existing libraries (such as Random, NumPy, etc.), **but not for the learning algorithms**. You should implement yourself the learning algorithms as well as the feature extraction.
- Your algorithm should not look at the testing data before the training is over. If you use any testing data point for training, that would be considered as cheating.
- There is no standard definition of "good enough" here. Typically, if you have less than 60% accuracy even when you use 100% of your training data then that means that you could do a better job on the features design or that something about your algorithm implementation could be incorrect. For this project, **as long as you have 60% accuracy for digit data and 70% accuracy for face data, your algorithms are considered good enough**.

Have fun!