# CS 475 Machine Learning: Homework 1
# Supervised Classifiers 1

Due: Monday September 26, 2016, 11:59pm
50 Points Total Version 1.0

**Make sure to read from start to finish before beginning the assignment.**

## 1 Introduction

The goal of the programming homeworks in this course is to build a machine learning library. In each homework assignment you will expand your learning library by implementing and evaluating new algorithms. Most, but not all, programming assignments will build upon previous assignments by comparing algorithms on common data. The purpose of the first assignment will be to introduce the data and build the foundations of the learning library.

### 1.1 Components of Assignments

Assignments consist of two parts.

1. **Programming:** You will implement learning algorithms and test them on provided data.
2. **Analytical questions:** These questions will ask you to consider questions related to the topics covered by the assignment. You will be able to answer these questions without relying on your programming.

Assignments are worth various points. Typically, the first and sixth (last) homework are each worth 50 points, and the others are worth 100 each. The point totals will be indicated in the assignment.

Each assignment will contain a version number at the top. While we try to ensure every homework is perfect when we release it, small errors do happen. When we correct these, we'll update the version number, post a new PDF and announce the change. Each homework starts at version 1.0 (no beta).

## 2 Data

The first part of the semester will focus on supervised classification. We consider several real world binary classification datasets taken from a range of applications. Each dataset is in the same format (described below) and contains a train, development and test file. You will train your algorithm on the train file and use the development set to test that your algorithm works. The test file contains unlabeled examples that we will use to test your algorithm. It is **a very good idea** to run on the test data just to make sure your code doesn't crash. You'd be surprised how often this happens.

### 2.1 Biology

Biological research produces large amounts of data to analyze. Applications of machine learning to biology include finding regions of DNA that encode for proteins, classification of gene expression data and inferring regulatory networks from mRNA and proteomic data.

Our biology task of characterizing gene splice junction sequences comes from molecular biology, a field interested in the relationships of DNA, RNA, and proteins. Splice junctions are points on a sequence at which "superfluous" RNA is removed before the process of protein creation in higher organisms. Exons are nucleotide sequences that are retained after splicing while introns are spliced out. The goal of this prediction task is to recognize DNA sequences that contain boundaries between exons and introns. Sequences contain exon/intron (EI) boundaries, intron/exon (IE) boundaries, or do not contain splice examples.

For a binary task, you will classify sequences as either EI boundaries (label 1) or non-splice sequences (label 0). Each learning instance contains a 60 base pair sequence (ex. ACGT), with some ambiguous slots. Features encode which base pair occurs at each position of the sequence.

### 2.2 Finance

Finance is a data rich field that employs numerous statistical methods for modeling and prediction, including the modeling of financial systems and portfolios.[1]

Our financial task is to predict which Australian credit card applications should be accepted (label 1) or rejected (label 0). Each example represents a credit card application, where all values and attributes have been anonymized for confidentiality. Features are a mix of continuous and discrete attributes and discrete attributes have been binarized.

### 2.3 NLP

Natural language processing studies the processing and understanding of human languages. Machine learning is widely used in NLP tasks, including document understanding, information extraction, machine translation and document classification.

Our NLP task is sentiment classification. Each example is a product review taken from Amazon kitchen appliance reviews. The review is either positive (label 1) or negative (label 0) towards the product. Reviews are represented as uni-gram and bi-grams; each one and two word phrase is extracted as a feature.

### 2.4 Speech

Statistical speech processing has its roots in the 1980s and has been the focus of machine learning research for decades. The area deals with all aspects of processing speech signals, including speech transcription, speaker identification and speech information retrieval.

Our speech task is spoken letter identification. Each example comes from a speaker saying one of the twenty-six letters of English alphabet. Our goal is to predict which letter was spoken. The data was collected by asking 150 subjects to speak each letter of the alphabet twice.

[1] For an overview of such applications, see the proceedings of the 2005 NIPS workshop on machine learning in finance. http://www.icsi.berkeley.edu/~moody/MLFinance2005.htm

Each spoken utterance is represented as a collection of 617 real valued attributes scaled to be between -1.0 and 1.0. Features include spectral coefficients; contour features, sonorant features, pre-sonorant features, and post-sonorant features. The binary task is to distinguish between the letter M (label 0) and N (label 1).

### 2.5 Vision

Computer vision processes and analyzes images and videos and it is one of the fundamental areas of robotics. Machine learning applications include identifying objects in images, segmenting video and understanding scenes in film.

Our vision task is image segmentation. In image segmentation, an image is divided into segments are labeled according to content. The images in our data have been divided into 3x3 regions. Each example is a region and features include the centroids of parts of the image, pixels in a region, contrast, intensity, color, saturation and hue. The goal is to identify the primary element in the image as either a brickface, sky, foliage, cement, window, path or grass. In the binary task, you will distinguish segments of foliage (label 0) from grass (label 1).

### 2.6 Synthetic Data

When developing algorithms it is often helpful to consider data with known properties. We typically create synthetic data for this purpose. To help test your algorithms, we are providing two synthetic datasets. These data are to help development.

#### 2.6.1 Easy

The easy data is labeled using a trivial classification function. Any reasonable learning algorithm should achieve near flawless accuracy. Each example is a 10 dimensional instance drawn from a multi-variate Gaussian distribution with 0 mean and a diagonal identity covariance matrix. Each example is labeled according to the presence one of 6 features; the remaining features are noise.

#### 2.6.2 Hard

Examples in this data are randomly labeled. Since there is no pattern, no learning algorithm should achieve accuracy significantly different from random guessing (50%). Data is generated in an identical manner as *Easy* except there are 94 noisy features.

## 3 Programming (35 points)

In this assignment you will implement a simple linear classifier: Perceptron. We have provided Python code that performs some basic operations for the learning library. You will fill in the details. Search for comments that begin with `TODO`; these sections need to be written. You may change the internal code as you see fit but the behavior for the given command lines **cannot be changed**. **Do not change the name or package of any of the provided code.**

### 3.1 Python Libraries

We will be using Python 2.7.x. We are *not* using Python 3, and will not accept assignments written for this version. We recommend using a recent release of Python 2.7.x, such as Python 2.7.6 or higher, but anything in this line should be fine.

For each assignment, we will tell you which Python libraries you may use. We will do this by providing a `requirements.txt` file. We *strongly* recommend using `Virtualenv` to ensure compliance with the permitted libraries. By strongly, we mean that unless you have a very good reason not to, and you really know what you are doing, you should use `Virtualenv`. You can read an introduction to the tool here `https://virtualenv.pypa.io/en/stable/`. For each assignment, start by creating a new `Virtualenv` environment. Activate the environment and install the libraries listed in the `requirements.txt` file using pip:

```
pip install -r requirements.txt
```

Modern versions of `Virtualenv` will default to disallowing access to global site-packages from within the environment. For older versions of virtualenv, you will need this flag: `--no-site-packages`.

We will be running your code within a `Virtualenv` created with *only* the libraries in `requirements.txt`. If you use a library not included in this file, your code will fail when we run it. We are asking you to use `Virtualenv` to ensure that you do not mistakenly include other libraries.

Make sure you are using the correct `requirements.txt` file from the current assignment. We may add new libraries to the file in subsequent assignments, or even remove a library (less likely). If you are using the wrong assignments `requirements.txt` file, it may not run when we grade it. For this reason, we suggest creating a new `Virtualenv` for each assignment.

It may happen that you find yourself in need of a library not included in the `requirements.txt` for the assignment. You may request that a library be added by posting to Piazza. This may be useful when there is some helpful functionality in another library that we omitted. However, we are unlikely to include a library if it either solves a major part of the assignment, or includes functionality that isn't really necessary.

In this and future assignments we will allow you to use `numpy` and `scipy`.

### 3.2 How to Run the Library

The library operates in two modes: train and test. Both stages are in the main method of `classify.py`.

The command line for train mode is:

```
python classify.py --mode train --algorithm algorithm_name --model-file model_file --data train_file
```

The `mode` option indicates which mode to run (train or test). The `algorithm` option indicates which training algorithm to use. Each assignment will specify the string argument for an algorithm. The `data` option indicates the data file to load. Finally, the `model-file` option specifies where to save the trained model.

The test mode is run in a similar manner:

```
python classify.py --mode test --model-file model_file --data test_file --predictions-file predictions_file
```

The `model_file` is loaded and run on the `data`. Results are saved to the `predictions_file`.

As an example, the following trains a perceptron classifier on the speech training data:

```
python classify.py --mode train --algorithm perceptron --model-file speech.perceptron.model \
                   --data speech.train
```

To run the trained model on development data:

```
python classify.py --mode test --model-file speech.perceptron.model --data speech.dev \
                     --predictions-file speech.dev.predictions
```

As we add new algorithms we will also add command line flags using the `argparse` library to specify algorithmic parameters. These will be specified in each assignment.

### 3.3 Data Formats

The data are provided in what is commonly known as SVM-light format. Each line contains a single example:

```
0 1:-0.2970 2:0.2092 5:0.3348 9:0.3892 25:0.7532 78:0.7280
```

The first entry on the line is the label. The label can be an integer (0/1 for binary classification) or a real valued number (for regression.) The classification label of $-1$ indicates unlabeled. Subsequent entries on the line are features. The entry `25:0.7532` means that feature 25 has value 0.7532. Features are 1-indexed.

Model predictions are saved as one predicted label per line in the same order as the input data. The code that generates these predictions is provided in the library. The script `compute_accuracy.py` can be used to evaluate the accuracy of your predictions for classification:

```
compute_accuracy.py data_file predictions_file
```

We provide this script since it is exactly how we will evaluate your output.

### 3.4 Components

The foundations of the learning framework have been provided for you. You will need to complete this library by filling in code where you see a `TODO` comment. You are free to make changes to the code as needed provided you do not change the behavior of the command lines described above. We emphasize this point: *do not change the command line flags or algorithm names.* We use these command lines to test your code. If you change their behavior, we cannot test your code.

The classes of interest in the library are:

- **FeatureVector**- The data representing an instance are stored as a feature vector. A feature vector is a vector of doubles, where the value of the $i$th dimension of the feature vector corresponds to the value of the $i$th feature. A FeatureVector must support operations such as `get(index)`, which returns the value of the feature at `index` and `add(index, value)`, which sets the value of the feature at `index`.

  Since many learning applications encode instances as sparse vectors, `FeatureVector` should be **a sparse vector**. A sparse vector efficiently encodes very high dimensional data by not maintaining values for features with 0 values. Some common implementations of sparse vectors include hash maps and lists of index/value pairs. If you fail to do this correctly, your code will run very slowly. You will need to add a method(s) to iterate over the non-empty positions of the vector. How you chose to do this is up to you.

  You are also welcome to use data structures within `Scipy`: `http://www.scipy.org/`.

- **Label**- A label object encodes the label for a learning example. The `Label` class is abstract and you will implement a `ClassificationLabel`, which contains an int to indicate a class (binary prediction will be 0 or 1). `ClassificationLabel` should implement `__str__` so that it returns its int value as a string.
- **Instance**- An instance represents a single learning example. An instance contains a data object and a label. The data object is a `FeatureVector` and the label will be a `Label` object. For classification, the label will be a `ClassificationLabel` object. When the label is unknown (test data) the label will be `None`.
- **Predictor**- This is an abstract class that will be the parent class for all learning algorithms. Learning algorithms must implement the `train` and `predict` methods. Predictors must be serializable using `Pickle` so that they can be saved after training.
- **classify.py**- This file contains the main method used to run the learning library. This is where `Predictor` objects are created and trained.

### 3.5 Perceptron

In this assignment you will implement the Perceptron algorithm for binary classification.

Perceptron is a mistake-driven online learning algorithm. It takes as input a vector of real-valued inputs $\mathbf{x}$ and makes a prediction $\hat{y} \in \{-1, +1\}$ (for this assignment we consider only binary labels). Predictions are made using a linear classifier: $\hat{y} = \text{sign}(\mathbf{w} \cdot \mathbf{x})$. The term $\mathbf{w} \cdot \mathbf{x}$ is the dot product of $\mathbf{w}$ and $\mathbf{x}$ computed as $\sum_i x_i w_i$. Updates to $\mathbf{w}$ are made only when a prediction is incorrect: $\hat{y} \neq y$. The new weight vector $\mathbf{w}'$ is a function of the current weight vector $\mathbf{w}$ and example $\mathbf{x}$, $y$. The weight vector is updated so as to improve the prediction on the current example. Note that Perceptron naturally handles continuous and binary features, so no special processing is needed.

The basic structure of the algorithm is:

1. Initialize $\mathbf{w}$ to $\mathbf{0}$, set learning rate $\eta$ and number of iterations $I$
2. For each training iteration $k = 1 \dots I$:
   (a) For each example $i = 1 \dots N$:
      i. Receive an example $\mathbf{x}_i$
      ii. Predict the label $\hat{y}_i = \text{sign}(\mathbf{w} \cdot \mathbf{x}_i) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x}_i \geq 0 \\ -1 & \text{otherwise} \end{cases}$
      iii. If $\hat{y}_i \neq y_i$, make an update to $\mathbf{w}$: $\mathbf{w}' = \mathbf{w} + \eta y_i \mathbf{x}_i$

Note that there is no bias term in this version and you should *not* include one in your solution. Also observe the definition of "sign" to account for 0 values. While sign returns $-1$ and 1, you must output as predictions the actual label values, which are 0 and 1.

### 3.6 Averaged Perceptron

One of the drawbacks to the Perceptron is that the learned parameters can change quite a bit over each iteration, making the highly dependent on the final examples it observes during training. The Averaged Perceptron tries to correct this by making the final predictor depend on all of the weight vectors, not just the final weight vector.

Specifically, the training algorithm for Averaged Perceptron remains identical to the one described above. However, instead of keeping the final $\mathbf{w}$ as the predictor, it keeps

the average of all predictors used during training. The set of weights $\mathbf{w}^{\text{final}}$ that is kept for test predictions is defined as: $\mathbf{w}^{\text{final}} = \sum_i \mathbf{w}_i$, where $\mathbf{w}_i$ is the predictor that resulted from round $i$ of training. Note that in an efficient implementation, you need not keep around every $\mathbf{w}$ until the very end, but you can keep around summary statistics required to compute the final $\mathbf{w}^{\text{final}}$.

#### 3.6.1 Deliverables

You need to implement the Perceptron and Averaged Perceptron algorithm as well as all other basic functions and classes of the library described above. Your predictors will be selected by passing the string `perceptron` or `averaged_perceptron` as the argument for the algorithm parameter.

#### 3.6.2 Learning Rate

Perceptron uses a learning rate $\eta$, where $0 < \eta \leq 1$. Your default value for $\eta$ should be 1. You *must* add a command line argument to allow this value to be adjusted via the command line.

Add this command line option by adding the following code to the `get_args` function in `classify.py`.

```
parser.add_argument("--online-learning-rate", type=float, help="The learning rate for perceptron",
                    default=1.0)
```

Be sure to add the option name exactly as it appears above. You can then use the value read from the command line in your main method by referencing it as `args.online_learning_rate`. Note that the dashes have been replaced by underscores.

### 3.7 Number of training iterations

Usually we can iterate multiple times over the data. This can improve performance by increasing the number of updates each algorithm makes. We will define the number of times each algorithm iterates over all of the data by the parameter `online_training_iterations`. You *must* define a new command line option for this parameter. Use a default value of 5 for this parameter.

You can add this option by adding the following code to the `get_args` function of `classify.py`.

```
parser.add_argument("--online-training-iterations", type=int,
                    help="The number of training iterations for online methods.", default=5)
```

You can then use the value read from the command line in your main method by referencing it as `args.online_training_iterations`.

During training, you should not change the order of examples. You must iterate over examples exactly as they appear in the data file, i.e. as provided by the data loader.

### 3.8 Grading Programming

The programming section of your assignment will be graded using an automated grading program. Your code will be run using the provided command line options, as well as other variations on these options (different parameters, data sets, etc.) The grader will consider the following aspects of your code.

1. **Exceptions:** Does your code run without crashing?

2. **Output:** Some assignments will ask you to write some data to the console. Make sure you follow the provided output instructions exactly.

3. **Accuracy:** If your code works correctly, then it should achieve a certain accuracy on each data set. While there are small difference that can arise, a correctly working implementation will get the right answer.

4. **Speed/Memory:** Efficiency largely doesn't matter, except where lack of efficiency severely slows your code (so slow that we assume it is broken) or the lack of efficiency demonstrates a lack of understanding of the algorithm. For example, if your code runs in two minutes and everyone else runs in 2 seconds, you'll lose points. Alternatively, if you require 2 gigs of memory, and everyone else needs 10 MB, you'll lose points. In general, this happens not because you did not optimize your code, but when you've implemented something incorrectly.

### 3.9 Code Readability and Style

In general, we do not care about code style or that it conforms to Python standards for naming variables, methods, etc. However, your code should be readable, which means minimal comments and clear organization. If your code works perfectly then you will get full credit. However, if it does not we will look at your code to determine how to allocate partial credit. If we cannot read your code or understand it, then it is very difficult to assign partial credit. Therefore, it is in your own interests to make sure that your code is reasonably readable and clear.

### 3.10 Code Structure

Your code must support the command line options and the example commands listed in the assignment. Aside from this, you are free to change the internal structure of the code, write new classes, change methods, add exception handling, etc. However, do not change the name or packages of any code you have been provided. We suggest you remember the need for clarity in your code organization.

### 3.11 Knowing Your Code Works

How do you know your code really works? That is a very difficult problem to solve. Here are a few tips:

1. Check results on `easy` and `hard`. They should be close to 100% and 50% respectively.

2. Use Piazza. While you **cannot** share code, you can share results. It is acceptable to post your results on dev data for your different algorithms. A common result will quickly emerge that you can measure against.

3. Output intermediate steps. Looking at final predictions that are wrong tells you little. Instead, print output as you go and check it to make sure it looks right. This can also be helpful when sharing information on the bulletin board.

4. Debug. Find a Python debugger that you like and use it. It's incredibly helpful.

### 3.12 Debugging

The most common question we receive is "how do I debug my code?" The truth is that machine learning algorithms are very hard to debug because the behavior of the algorithm is unknown. In these assignments, you won't know ahead of time what accuracy is expected for your algorithm on a dataset. This is the reality of machine learning development, though in this class you have the advantage of your classmates, who may post the output of their code to the bulletin board. While debugging machine learning code is therefore harder, the same principles of debugging apply. Write tests for different parts of your code to make sure it works as expected. Test it out on the easy datasets to verify it works and, when it doesn't, debug those datasets carefully. Work out on paper the correct answer and make sure your code matches. Don't be afraid of writing your own data for specific algorithms as needed to test out different methods. This process is part of learning machine learning algorithms and a reality of developing machine learning software.

## 4 Analytical (25 Points)

In addition to completing the analytical questions, your assignment for this homework is to learn Latex. All homework writeups must be PDFs compiled from Latex. Why learn latex?

1. It is incredibly useful for writing mathematical expressions.
2. It makes references simple.
3. Many academic papers are written in latex.

The list goes on. Additionally, it makes your assignments much easier to read than if you try to scan them in or complete them in Word.

We realize learning latex can be daunting. Fear not. There are many tutorials on the Web to help you learn. We recommend using pdflatex. It's available for nearly every operating system. Additionally, we have provided you with the tex source for this PDF, which means you can start your writeup by erasing much of the content of this writeup and filling in your answers. You can even copy and paste the few mathematical expressions in this assignment for your convenience. As the semester progresses, you'll no doubt become more familiar with latex, and even begin to appreciate using it.

Be sure to check out this cool latex tool for finding symbols. It uses machine learning! `http://detexify.kirelabs.org/classify.html`

**1) Hypothesis Class (3 points)** True/False (and why): You are using an algorithm that selects a hypothesis from a class that you know contains the optimal hypothesis for a given problem. In this case, there is no benefit to using regularization.

**2) Loss Function (4 points)** For each of the following, state if the function is a valid loss function. If it is, state whether it would make a suitable loss function for binary classification. If it is not a valid loss function, why not? $\hat{y}$ is the predicted label and $y$ is the correct label.

1. $\ell(y, \hat{y}) = y - \hat{y}$

2. $\ell(y, \hat{y}) = \frac{1}{3}(y - \hat{y})^2$
3. $\ell(y, \hat{y}) = |(y - \hat{y})|/\hat{y}$
4. $\ell(y, \hat{y}) = \max(0, 1 - y \cdot \hat{y})$

**3) Ranking (7 points)** Ranking is a common supervised machine learning problem, such as in ranking search engine results. In a ranking task, the predictor is given a set of instances and returns an ordering over the instances. We haven't discussed how to design or train ranking algorithm, but some of the algorithms we have learned about can be used within a ranking task.

1. How would you use a trained regression or classification algorithm to rank a set of instances? Assume that the model has already been trained for this purpose. You only need to describe how it will be used at test time.
2. Provide a loss function suitable for ranking. This loss function need not be related to your answer in the first part of the question.

**4) Regularization and Overfitting. (11 points)** Statisticians love linear models because these models are very simple and interpretable. Many variants of linear models have been proposed, and most of them are formulated as a (penalized) least squares objective. Consider these three least squares objectives:

$$\hat{\beta}_0 = \operatorname*{argmin}_{\beta_0} \|y - X_1\beta_0\|_2^2, \tag{1}$$

$$(\hat{\beta}_1, \hat{\beta}_2) = \operatorname*{argmin}_{\beta_1, \beta_2} \|y - X_1\beta_1 - X_2\beta_2\|_2^2, \tag{2}$$

$$\hat{\beta}_3 = \operatorname*{argmin}_{\beta_3} \|y - X_1\beta_3\|_2^2 + \lambda\|\beta_3\|_2^2, \tag{3}$$

where $\lambda > 0$, $y \in \mathbb{R}^n$, $X_1 \in \mathbb{R}^{n \times d_1}$, and $X_2 \in \mathbb{R}^{n \times d_1}$. (3) is well known as the ridge regression. The square norm acts as a penalty function to reduce overfitting. Prove

$$\|y - X_1\hat{\beta}_3\|_2^2 \geq \|y - X_1\hat{\beta}_0\|_2^2 \geq \|y - X_1\hat{\beta}_1 - X_2\hat{\beta}_2\|_2^2. \tag{4}$$

## 5 What to Submit

In each assignment you will submit two things.

1. **Code:** Your code as a zip file named `code.zip`. **You must submit source code (.py files)**. We will run your code using the exact command lines described above, so make sure it works ahead of time. Remember to submit all of the source code, including what we have provided to you. We will include the libraries specific in `requirements.txt` but nothing else.
2. **Writeup:** Your writeup as a **PDF file** (compiled from latex) containing answers to the analytical questions asked in the assignment. Make sure to include your name in the writeup PDF and use the provided latex template for your answers.

Make sure you name each of the files exactly as specified (code.zip and writeup.pdf).

To submit your assignment, visit the "Homework" section of the website (http://www.cs475.org/.)

## 6 Questions?

Remember to submit questions about the assignment to the appropriate group on Piazza: https://piazza.com/class/it1vketjjo71l1.