messmaster_1

January 29, 2025

Note: if you are running this locally, make sure you have cloned the repo with `git clone https://github.com/nyu-dl/DS-GA-1003-Machine-Learning-2025`.

Now, onto the magical journey!

1 MessMaster: the Curse of the Muffin-Faced Dog (part 1)

If your room ever gets messy, you can blame the second law of thermodynamics because the room's entropy is always increasing. No matter how much you try to tidy it up, the room will always get messy again.

An inventor named Boltzmann has a similar problem. He invented an **AutoSorter 3025** to sort similar objects into different bins. However, there is a room that the AutoSorter does not work well. It is a room full of chihuahuas and blueberry muffins.

The auto-sorter cannot tell them apart because this is what it sees:

Each object is reduced to just two features:

1. How likely there are two large black dots (potentially eyes).
2. How likely there is a brown circular shape (muffin top or Chihuahua fur).

Unfortunately, these features don't fully capture the difference between a muffin and a tiny, confused dog—leading to some interesting classification errors…

In his desperation, Boltzmann uncovered a dusty copy of 'Machine Learning: A Lecture Note,' rumored to be written by a wise scholar known only as 'The Guardian of Patterns.' Now, Boltzmann needs help, and that's where you come in. As apprenticies of the Guardian of Patterns, your mission is to train a better classifier and fix the AutoSorter 3025.

- Understand how the energy function related to classification
- Compare and contrast zero-one loss, margin loss, and cross-entropy loss to see how different training objectives affect learning.
- Observe how the decision boundary evolves as we train a Perceptron, SVM, and Logistic Regression classifier.

By mastering these principles, you will help Boltzmann restore order to the Muffin-Chihuahua crisis—before entropy wins again!

1.0.1 1. Energy Function and Classification

First, Boltzmann wants you to prove you are an apprentice of the Guardian of Patterns. He has seen 'energy function' in the book, and he wants you to explain it in the context of his chihuahua and muffin dilemma.

> An energy function e assigns a real value to an observed instance and a latent instance (x, z) and is parameterized by a multi-dimensional vector θ.

$$e : \mathcal{X} \times \mathcal{Z} \times \Theta \mapsto \mathbb{R}.$$

Q1a. What does the energy function do in the context of classifying chihuahuas (positive) and muffins (negative)?

Answer:

Q1b. Given some examples of what x, z, θ are in the context of chihuahuas v.s. muffins from AutoSorter 3025's perspective.

Answer: - Observed instance x: - Latent instance z: - Parameter θ:

For now we will assume there is no latent variable and that observed instance comes in pairs (x, y), where x is the feature and y is the label.

So the energy function is $e : (\mathcal{X} \times \mathcal{Y}) \times \emptyset \times \Theta \mapsto \mathbb{R}$.

Q1c. Given a new observation x', how do you infer if it's Chihuahua or muffin (y') based on your energy function?

Answer:

Q1d. How do we learn the parameters θ'?

Answer:

Q1e. What if learning a desirable θ is hard and we want to penalize some undesirable outcomes? How does this change our minimization objective and what is this called in ML?

Answer:

1.0.2 2. Optimizing Parameters: from Energy to Loss and Probabilities

"Thanks for explaining energy function to me!" Boltzmann says. "Now I want to learn the parameters θ for my energy function, but I have a few more questions."

Q2a. >

$$\hat{y}(x) = \operatorname*{argmin}_{y} e((x, y), \emptyset, \theta).$$

The inference of label involves a for loop and seems to be slow. How can we parallelize it?

Answer:

Boltzmann said, "I'll start with a simple feature extractor. Let's say it's linear. So $f(x, \theta) = Wx+b$." How I learn the parameters for f?

You learned the simplest way is using zero-one loss. That is, trying to update θ whenever the lowest-energy label is not the true label.

However, this objective is not differentiable, so it's hard to optimize. So we use other loss functions such as margin loss and cross-entropy loss.

Q2b. What is margin loss?

Answer:

Q2c. What happens if we set the margin to 0?

Answer:

Boltzmann heard that cross-entropy loss is the most popular. But it is related to probabilities. He is confused about the connection and how this loss function reduces energy for the correct outcome.

Q2d. Help Boltzmann understand cross-entropy loss.

i. On a high level, how do we connect the energy function to categorical probabilities?

ii. Write down its definition.

iii. Derive its gradient with respect to θ.

iv. What does the gradient tell us about the update rule?

Answer:

1.0.3 3. The Subtle Boundary of Chihuahua and Muffin: Classification in Action

You step into the room filled with chihuahuas and muffins. You will use three kinds of loss function to reduce the energy assigned to chihuahua and increase the energy assigned to muffin. You are determined to fix Boltzmann's AutoSorter 3025.

First, we will write a plotting function to visualize the training process.

Exercise: Read the code and try to connect it with the mathematical concepts we discussed. Write comments to connect the mathematical concepts to the code. Where is:

- energy function e
- observed data x
- learnable parameter θ
- learned parameter θ'
- loss function L
- the inferenced label $\hat{y}(x)$?

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.colors import Normalize
def plot_training_evolution(X, y, model, iterations=10, batch_size=1,␣
 ↪title="Model Evolution"):
    """
```

```
    Plot decision boundary evolution over training, showing only seen data at␣
↪each iteration.

    Args:
        X (ndarray): Input features.
        y (ndarray): Labels.
        model: The model (Perceptron, Logistic Regression, or SVM). # NOTE: ???
        iterations (int): Number of iterations to train.
        batch_size (int): Number of examples seen in each iteration.
        title (str): Plot title.
    """
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100), np.linspace(y_min,␣
↪y_max, 100))

    plt.figure(figsize=(10, 4.5))

    num_samples = X.shape[0]
    for i in range(iterations):
        # Select a subset of the data (seen data)
        start_idx = (i * batch_size) % num_samples
        end_idx = start_idx + batch_size
        X_seen = X[start_idx:end_idx] # NOTE: ???
        y_seen = y[start_idx:end_idx]

        # Train the model on the seen data
        # Exercise: replace this line with manual_partial_fit(model, X_seen,␣
↪y_seen, classes)
        # You need to implement the following steps:
        # 1. compute model prediction
        # 2. compute error
        # 3. compute parameter update
        # 4. update model parameters
        # NOTE: ???
        model.partial_fit(X_seen, y_seen, classes=np.unique(y))
        # NOTE: ???

        # Compute the decision boundary
        if hasattr(model, "decision_function"):
            Z = model.decision_function(np.c_[xx.ravel(), yy.ravel()])
        else:
            Z = model.predict_proba(np.c_[xx.ravel(), yy.ravel()])[:, 1]
        Z = Z.reshape(xx.shape)

        # Plot the decision boundary and seen data
        plt.subplot(2, 5, i + 1)
```

```
        plt.contourf(xx, yy, Z > 0, alpha=0.8, cmap="coolwarm") # NOTE: Z > 0␣
↪is the label
        # X_all = X[:end_idx]
        # y_all = y[:end_idx]
        # plt.scatter(X_all[:, 0], X_all[:, 1], c=y_all, edgecolor="k",␣
↪label="Seen Data")
        X_new = X[start_idx:end_idx]
        y_new = y[start_idx:end_idx]
        X_old = X[:start_idx]
        y_old = y[:start_idx]
        norm = Normalize(vmin=0, vmax=1)
        if len(X_old) > 0:
            plt.scatter(X_old[:, 0], X_old[:, 1], c=y_old, edgecolor="k",␣
↪cmap=plt.cm.viridis, label="Previous Data", norm=norm)
        plt.scatter(X_new[:, 0], X_new[:, 1], c=y_new, edgecolor="red",␣
↪cmap=plt.cm.viridis, label="New Data", norm=norm)
        plt.title(f"{title} Iteration {i + 1}")
        plt.xlabel("Likely Ear")
        plt.ylabel("Likely Leg")

    # add legend
    plt.tight_layout()
    plt.show()
```

Next, we will load the generated data and standardize it.

[2]:
```
%pip install pandas
import pandas as pd
from sklearn.preprocessing import StandardScaler
df = pd.read_csv("secret/data/margin_classifier_alternate.csv")
X = df[["x1", "x2"]].values
y = df["y"].values
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

df_2 = pd.read_csv("secret/data/svm_vs_logistic_regression.csv")
X_2 = df_2[["x1", "x2"]].values
y_2 = df_2["y"].values
X_2_scaled = scaler.transform(X_2)
```

```
Requirement already satisfied: pandas in /opt/anaconda3/lib/python3.11/site-
packages (2.1.4)
Requirement already satisfied: numpy<2,>=1.23.2 in
/opt/anaconda3/lib/python3.11/site-packages (from pandas) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in
/opt/anaconda3/lib/python3.11/site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/opt/anaconda3/lib/python3.11/site-packages (from pandas) (2023.3.post1)
```

```
Requirement already satisfied: tzdata>=2022.1 in
/opt/anaconda3/lib/python3.11/site-packages (from pandas) (2023.3)
Requirement already satisfied: six>=1.5 in /opt/anaconda3/lib/python3.11/site-
packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

Finally, we will define the training functions for the three models.

[3]:
```
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier


def train_perceptron(X, y, iterations=10, class_weight=None, batch_size=1):
    perceptron = Perceptron(max_iter=1, tol=None, warm_start=True,␣
 ↪random_state=42, class_weight=class_weight)
    plot_training_evolution(X, y, perceptron, iterations, batch_size,␣
 ↪title="Perceptron")


def train_svm(X, y, iterations=10, margin=3, batch_size=1):
    assert margin > 0, "Margin must be positive. If margin is 0, it becomes a ??
 ↪?." # TODO: fill in the ???
    alpha = 1 / (margin **2)
    svm = SGDClassifier(loss="hinge", max_iter=1, tol=None, alpha=alpha,␣
 ↪warm_start=True, random_state=42)
    plot_training_evolution(X, y, svm, iterations, batch_size, title="SVM")


def train_logistic_regression(X, y, iterations=10, batch_size=1):
    logistic = SGDClassifier(loss="log_loss", max_iter=1, alpha=0.01, tol=None,␣
 ↪warm_start=True, random_state=42)
    plot_training_evolution(X, y, logistic, iterations, batch_size,␣
 ↪title="LogReg")
```

Q3a. Perceptron v.s. SVM

Let's first start classifying using two models with similar loss functions. Observe the training process of Perceptron and SVM.

i. How does the decision boundary change?

ii. What is the difference between the final decision boundary of the two models?

iii. What causes this difference?

iv. (Exercise) What happens if you increase or decrease the margin for SVM? Why does this happen?

Answer:

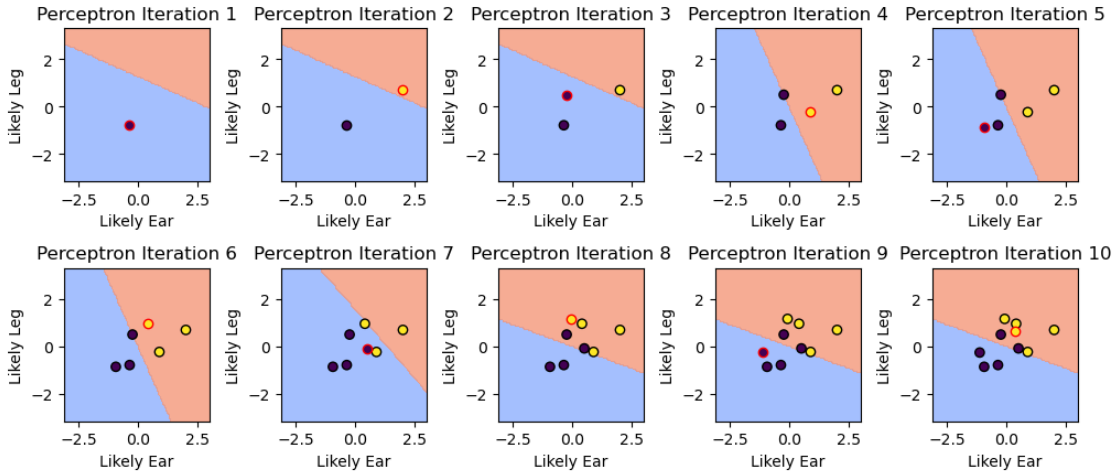
i.

ii.

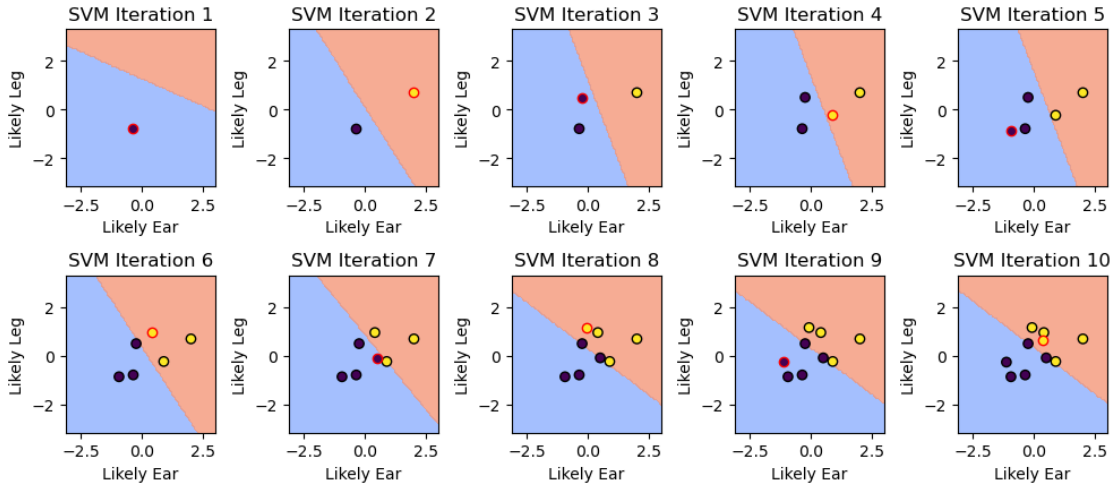
iii.

iv.

[4]:
```
train_perceptron(X_scaled, y, iterations=10) #, class_weight="balanced")#{0: 1,␣
↪1: 0})
```



[5]:
```
train_svm(X_scaled, y, iterations=10, margin=3)
```



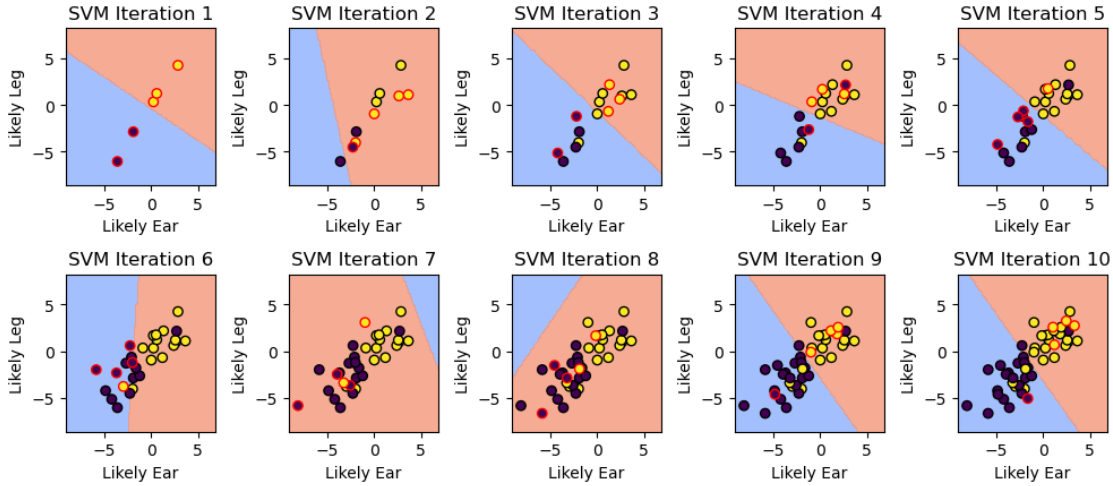
Q3b. SVM v.s. Logistic Regression

Next let's compare Max Margin Classifier (SVM) and Logistic Regression Classifier.

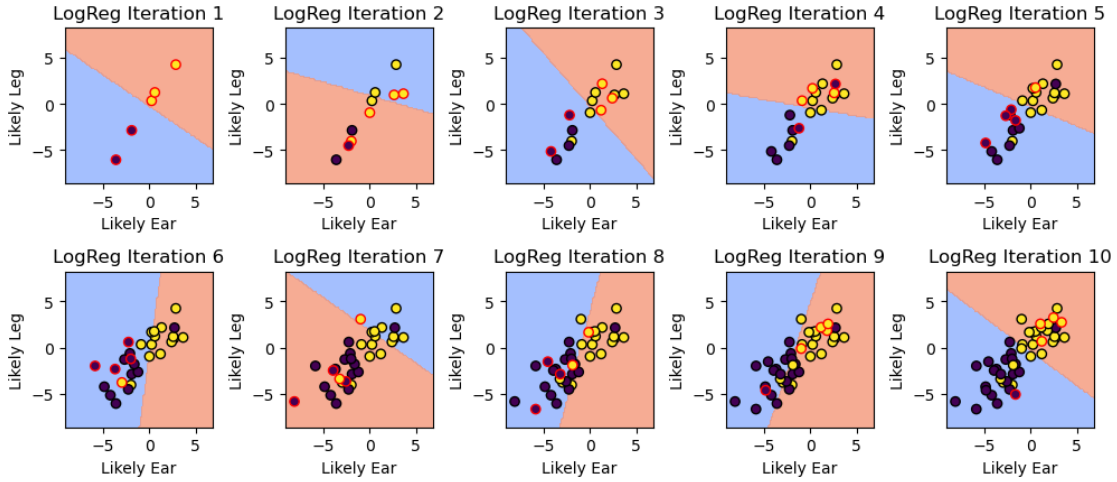
How does the changes in decision boundary differ between the two models? Why?

Answer:

[6]:
```
train_svm(X_2_scaled, y_2, iterations=10, margin=3, batch_size=5)
```



[7]:
```
train_logistic_regression(X_2_scaled, y_2, iterations=10, batch_size=5)
```



So far, our decision boundary has been linear. Next time, we will see use MLP to learn a non-linear decision boundary. This would enable us to better classify the chihuahua and muffin!