Lecture Three: Supervised Learning: Classification

COMP3032 Machine Learning

Binary classification problems

Binary classification problems-there are two possible classes

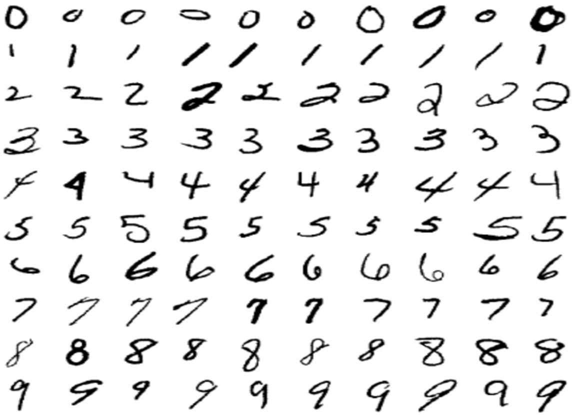
- Will the house sell? (yes/no)
- Email: spam / no spam
- Credit card transaction - fraud or real?
- Cat or Dog? (Input: picture, output: Cat / dog)
- Medical diagnosis for a particular disease (output yes/no)
- Terrain: drivable (yes/no)

Multiclass classification problems

There are more than two possible classes

- Classify hand-written digits (Input: image of a digit, output: number 0-9)
- Speech recognition / speaker recognition (Input: sound, Output: words / identified speaker)
- Medical diagnosis (input: symptoms, output: disease)
- Traffic sign recognition
- Automatically grading assignments into HD/D/Credit/Pass/Fail etc

Classifying hand-written digits



What is logistic regression

- Logistic regression is a way of using some regression algorithms for classification
- Logistic regression outputs probability margins, between 0 and 1, for estimating membership to a particular class
- What is the probability that a particular email is a spam?

Logistic regression: Main ideas

- A logistic regressor returning a value between 0 and 1, used as probabilities
- Not directly outputting the weighted sum in linear regression
- The number gets transformed via a function $\sigma(\theta^T \cdot \mathbf{x})$
- $0 < \sigma(\theta^T \cdot \mathbf{x}) < 1$
- For any instance $\mathbf{x}$, $\sigma(\theta^T \cdot \mathbf{x})$ outputs a probability
- Classification is thus based on this probability

Sigmoid function

- $\sigma(t) = \frac{1}{1+e^{-t}}$
- the value only goes between 0 and 1
- It approaches 1 as t approaches infinity, and approaches 0 as t approaches negative infinity
- It takes the value of 0.5 when t is 0

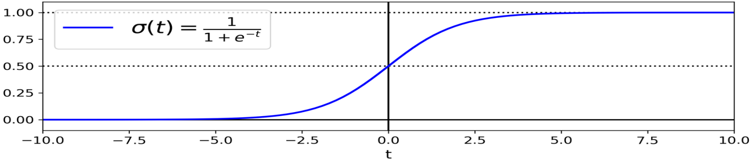


Figure 4-21. Logistic function

Geron, A. (2019). Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow

Probability using sigmoid function

Use sigmoid function to generate the probability

- In normal linear regression, the estimate $\hat{y}$ is given by: $\hat{y} = h_\theta(\mathbf{x}) = \theta^T \cdot \mathbf{x}$
- To transform this into a probability, use the sigmoid function (the logistic function)
- The usual linear regression becomes
- $\hat{p} = h_\theta(\mathbf{x}) = \sigma(\theta^T \cdot \mathbf{x}) = \frac{1}{1+e^{-\theta^T \cdot \mathbf{x}}}$
- $\hat{p}$ is between 0 and 1
- Interpret $\hat{p}$ as the probability of y $= 1$, for a given example $\mathbf{x}$, with the parameters θ.

Prediction

Make prediction by setting the thresholds:

- Predict whether an instance $\mathbf{x}$ belongs to a particular class by setting probability thresholds:
- e.g.

$$\hat{y} = \begin{cases} 0 & \text{if } \hat{p} < 0.5 \\ 1 & \text{if } \hat{p} \geq 0.5 \end{cases}$$

Training and cost function

- As with linear regression, we use data to estimate the parameters.
- The objective of training is to set the parameter vector θ so that the model estimates
  - high probabilities for positive instances ($y = 1$) and
  - low probabilities for negative instances ($y = 0$)
- To train the logistic regression model, we need
  - a cost function
  - a solver that tries to find the optimal coefficients to minimize this function

Cost function for a single training instance

- Measures how far the predicted probability ($\hat{p}$) is from the actual class label (y) using the following formula
- $$c(\theta) = \begin{cases} -log(\hat{p}) & \text{if } y = 1 \\ -log(1 - \hat{p}) & \text{if } y = 0 \end{cases}$$
- $-log(t)$ becomes very large as $t \to 0$
- $-log(t)$ closes to 0 as $t \to 1$
- Exactly what we want

Log loss function

- The cost function over the whole training set is the average cost over all training instances
- It can be written in a single expression known as the log loss function
- Cost function in general form
- $J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)} log(\hat{p}^{(i)}) + (1 - y^{(i)}) log(1 - \hat{p}^{(i)})]$
- Need to find the θ which make $J(\theta)$ smallest
- Why this function? It is convex, and so guaranteed to find the global minimum
- Can be derived from Maximum Likelihood Estimation

Find the optimal coefficients

- How to find the optimal coefficients to minimize the loss function?
- No known closed form exists
- A naive approach would be to try all the possible combinations of the coefficients until the minimal loss is found
- An exhaustive search is not feasible given the infinite combinations
- There are different solvers to efficiently search for the best coefficients and scikit-learn implements some of them

Use gradient descent

- To find the θ which makes $J(\theta)$ smallest, we can use gradient descent
- The partial derivative with respect to a parameter θ_j:
- $\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} [\sigma(\theta^T \cdot x) - y^{(i)}] x_j^{(i)}$
- For each instance it computes the prediction error, multiplies it by the jth feature value, then computes the average over all training instances

Training and prediction

- Once you have the gradient vector containing all the partial derivatives you can use it in the Batch Gradient Descent algorithm
- For Stochastic GD you would take one instance at a time
- For Mini-batch GD you would use a mini-batch at a time
- Compare it with the linear regression

An example

- iris is a popular dataset containing sepal and petal lengths and widths of 150 iris flowers
- there are three species: iris-setosa, iris-versicolor, iris-virginica
- try to build a classifier to detect the iris-virginica type based only on the petal width feature

An example

Steps:

- load the data
- train a logistic regression model
- predict probabilities for flowers with petal widths varying from 0 to 3 cm

The code: loading data

```
import numpy as np
import matplotlib.pyplot as plt

#load data

from sklearn import datasets
iris = datasets.load_iris()
list(iris.keys())

#['data','target', 'frame', 'target_names', 'DESCR',
                          'feature_names', 'filename']

X = iris["data"][:, 3:] # petal width
y = (iris["target"] == 2).astype(np.int)
#  1  if Iris-Virginica, else 0
```

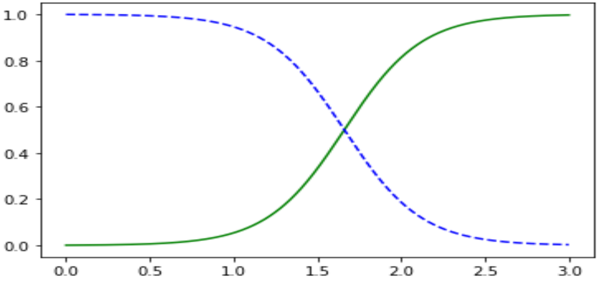
The code: training

```
#training

from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression()
log_reg.fit(X, y)
```

The code: predicting

```
#predict probabilities for flowers

X_new = np.linspace(0, 3, 1000).reshape(-1, 1)
y_proba = log_reg.predict_proba(X_new)
plt.plot(X_new, y_proba[:, 1],"g-",
                          label="Iris-Virginica")
plt.plot(X_new, y_proba[:, 0],"b--",
                          label="Not Iris-Virginica")
# + more Matplotlib code to make the image look pretty
```

The plot



Decision Boundary

To predict the class, the classifier will return the most likely class based on the probability.

- There is a decision boundary where both probabilities are equal to 50%
- It is at around 1.6 cm
- If the petal width is greater than 1.6 cm, it will predict Iris virginica
- Otherwise it will predict not

The plot: more details

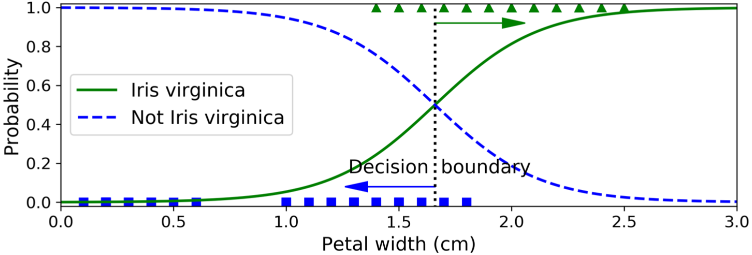


Figure 4-23. Estimated probabilities and decision boundary

Geron, A. (2019). Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow

Confusion Matrix

The general idea of a confusion matrix is to count the number of times instances of class A are classified as class B, for all A/B pairs.

- First need to have a set of predictions so that they can be compared to the actual targets
- Each row in a confusion matrix represents an actual class
- Each column represents a predicted class
- A perfect classifier would only have true positives and true negatives
- Its confusion matrix would have nonzero values only on its main diagonal (top left to bottom right)

Confusion Matrix

	Class 1 Predicted	Class 2 Predicted
Class 1 Actual	TP	FN
Class 2 Actual	FP	TN

[1]

[1] From https://www.geeksforgeeks.org/confusion-matrix-machine-learning/

Computing Confusion Matrix

Computing the confusion matrix

1. Need a set of *predictions* to compare with actual *targets*

```
y_pred = log_reg.predict(X)
```

2. Get the comfusion matrix

```
from sklearn.metrics import confusion_matrix
```

3. Compute the confusion matrix. Simply pass the *target classes* (`y`) and *predicted classes* (`y_pred`) to `confusion_matrix()`

```
confusion_matrix(y, y_pred)
```

4. Result (class1: Not Virginica, class2: Virginica)

```
[[98 2]
[4 46]]
```

Precision and Recall

Confusion matrix provides a lot of information, but sometimes one prefers a more concise metric. The accuracy of the positive predictions is called the precision of the classifier.

- *precision*:

$$\text{precision} = \frac{\text{TP}}{\underbrace{\text{TP} + \text{FP}}_{\text{total predicted positives}}}$$

- *recall* (also called sensitivity or the true positive rate (TPR))

$$\text{recall} = \frac{\text{TP}}{\underbrace{\text{TP} + \text{FN}}_{\text{total actual positives}}}$$

Precision and Recall

Use scikit-learn functions to compute precision and recall:

```
from sklearn.metrics import precision_score, recall_score
```

```
precision_score(y, y_pred)
```

```
0.9583333333333334
```

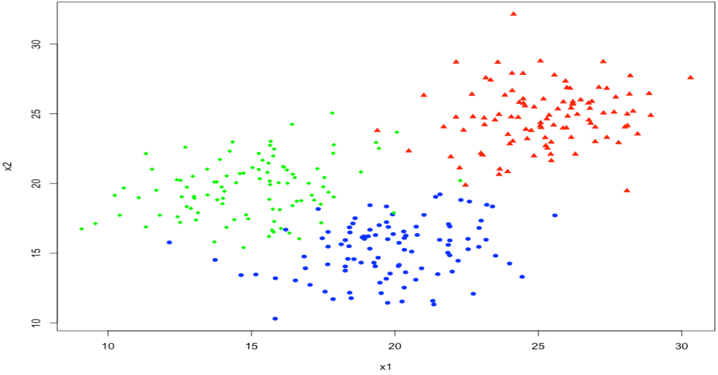
```
recall_score(y, y_pred)
```

```
0.92
```

Multiclass classification

- What if there are more than 2 classes?
- Google email classification: Primary, Social, Promotions, Forums
- Student grades classification: F, P, C, D, HD

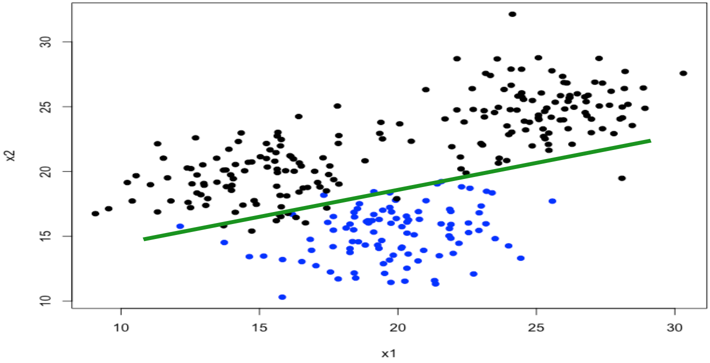
Multiclass classification



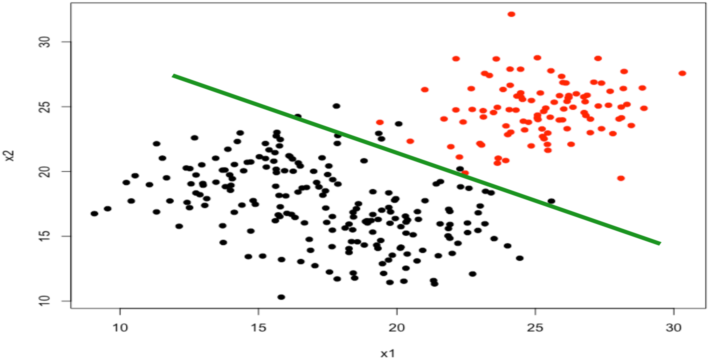
Multiclass classification

- Using binomial logistic regression
- We train a logistic regression classifier $f^{(i)}(x)$ for each class i to predict the probability that $y = i$.
- To use the classifier for prediction, for a new input x, select the class $max_i f^{(i)}(x)$
- Perform a prediction with each classifier

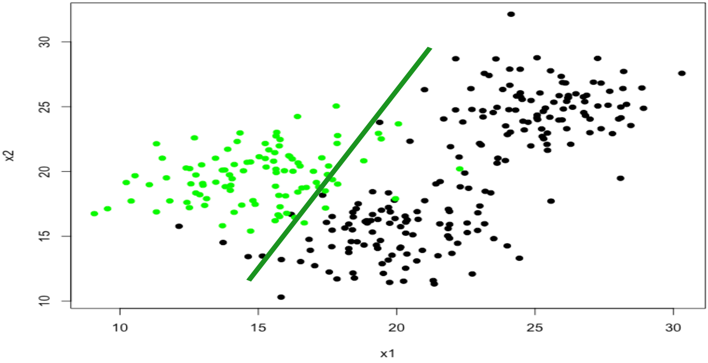
multiclass classification example



multiclass classification example



multiclass classification example



Softmax Regression

Softmax regression generalises logistic regression to multiclass without having to train and combine multiple logistic classifiers

- For a given number of classes k
- Given an instance $\mathbf{x}$, softmax regression first computes a score via some function $s_k(\mathbf{x})$
- $s_k(\mathbf{x}) = (\theta^{(k)})^T \cdot \mathbf{x}$
    - Softmax score for class k
    - each $\theta^{(k)}$ is unique parameter vector for each class k

Softmax function

- estimates the probability of belonging to the k^{th} class via the softmax function $\sigma(t)$
- $\hat{p}_k = \sigma(s(x))_k = \frac{e^{(s_k(x))}}{\sum_{j=1}^{k}(e^{(s_j(x))})}$
- The function computes the exponential of every score, then normalizes them (dividing by the sum of all the exponentials)
- k is the number of classes
- $s(\mathbf{x})$ is a vector containing the scores of each class for the instance $\mathbf{x}$
- $\sigma(s(\mathbf{x}))_\mathbf{k}$ is the estimated probability that the instance $\mathbf{x}$ belongs to class k, given the scores of each class for that instance

Softmax Regression classifier

- The Softmax Regression classifier predicts the class with the highest estimated probability (which is simply the class with the highest score)
- The prediction $\hat{y}$ is based on the maximum probability
- $\hat{y} = argmax_k \, \sigma(s(x))_k = argmax_k s_k(x) = argmax_k((\theta^{(k)})^T \cdot x)$
- The *argmax* operator returns the value of a variable that maximizes a function
- Here, it returns the value of k that maximizes the estimated probability $\sigma(s(x))_k$

The cross-entropy cost function

The objective is to have a model that estimates a high probability for the target class

- Minimize the cross-entropy cost function can achieve this objective
- The cross-entropy cost function
- $J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} log(\hat{p}_k^{(i)})$
- $y_k^{(i)}$ is the target probability that the ith instance belongs to class k
- $y_k^{(i)}$ is usually 1 or 0, depending on whether the instance belongs to the class
- if k = 2, this cost function is equivalent to the Logistic Regression

The cross-entropy gradient vector

- The cross entropy gradient vector for class k
- $\nabla_{\theta^{(k)}} J(\Theta) = -\frac{1}{m} \sum_{i=1}^{m} (\hat{p}_k^{(i)} - y_k^{(i)}) x^{(i)}$
- Can compute the gradient vector for every class
- Use Gradient Descent to find the parameter matrix Θ that minimizes the cost function

Softmax regression in scikit-learn

- Use Softmax Regression to classify the iris flowers into all three classes
- In scikit-learn, the class LogisticRegression uses the one-versus-all by default
- Can change to softmax by setting the *multi_class* hyperparameter to "multinomial"
- Specify a solver that supports softmax regression
- solver="lbfgs"

The code: an example

```
X = iris["data"][:, (2, 3)]
y = iris["target"] # petal length, petal width

softmax_reg =
LogisticRegression(multi_class="multinomial",
solver="lbfgs", C=10)

softmax_reg.fit(X, y)

softmax_reg.predict_proba([[5, 2]])

#array([[6.38014896e-07, 5.74929995e-02, 9.42506362e-01]])

softmax_reg.predict([[5, 2]])

#array([2])
```