DATASCI 207

Nedelina Teneva, PhD
nteneva@berkeley.edu

School of Information, UC Berkeley

Announcements

- **Finish group selection by next week!**
  - Fill in the Logistics Sheet
- **Homeworks:**
  - Check out the grading schemas on gradescope
  - Ensure that you've submitted the code you think you submitted
  - If I make a manual update to your grade, the score may flip back and forth (alas!)
- **Let me know if there is specific material you'd like to see (more of)!**

Hyper Parameter Optimization/Tuning

Hyper Parameters - Logistic Regression Example

Fig from https://sebastianraschka.com/pdf/lecture-notes/stat479fs18/10_eval-cv_notes.pdf

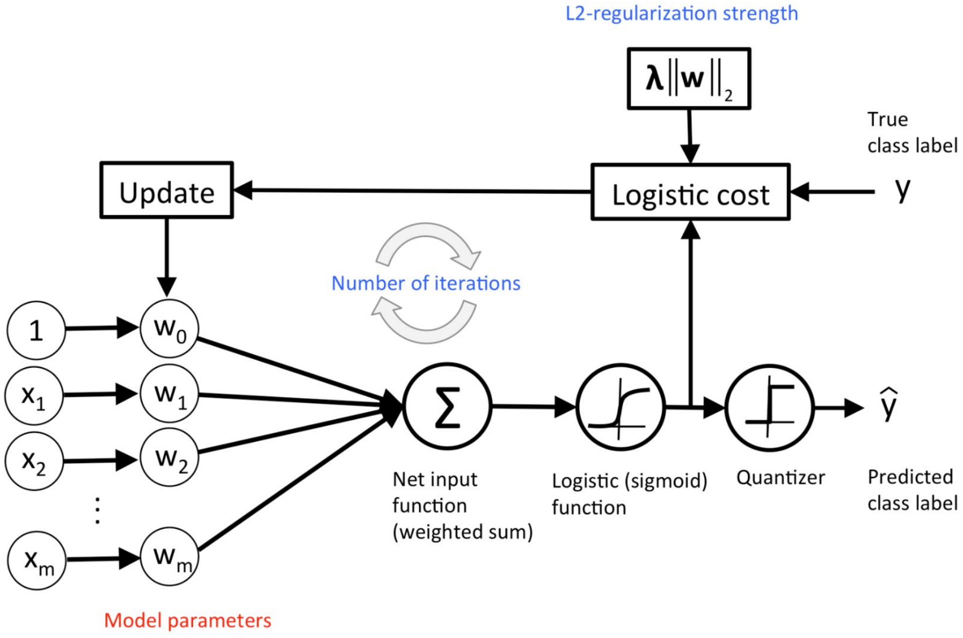


Figure 2: Conceptual overview of logistic regression.

Lambda is the C argument in `sklearn.linear_model.LogisticRegression` from Question 4 in LogisticRegresssionExercise.ipynb

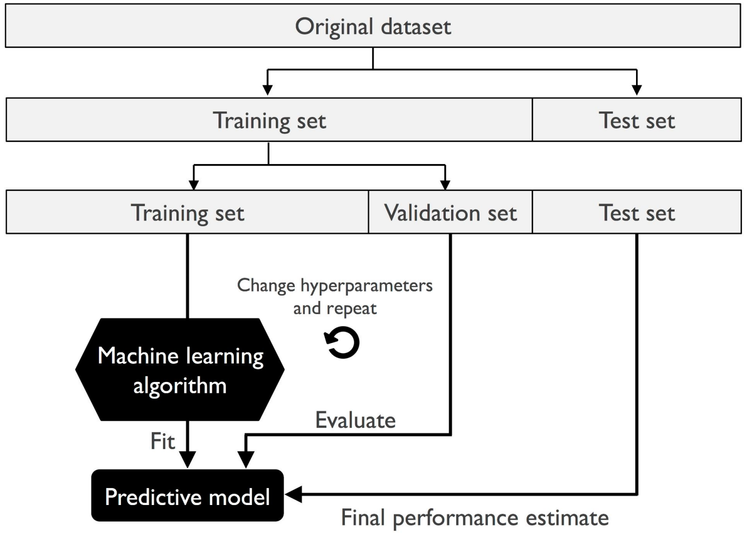
Hold out method

Hyper Parameter Optimization:

- **How do we select best hyper parameters for our algorithm?**
- **Idea**: investigate how the model predictions change as we change the value of hyperparameters (a.k.a., tuning parameters)
- Relies on 3 data groups: training set(s), validation set(s), test set

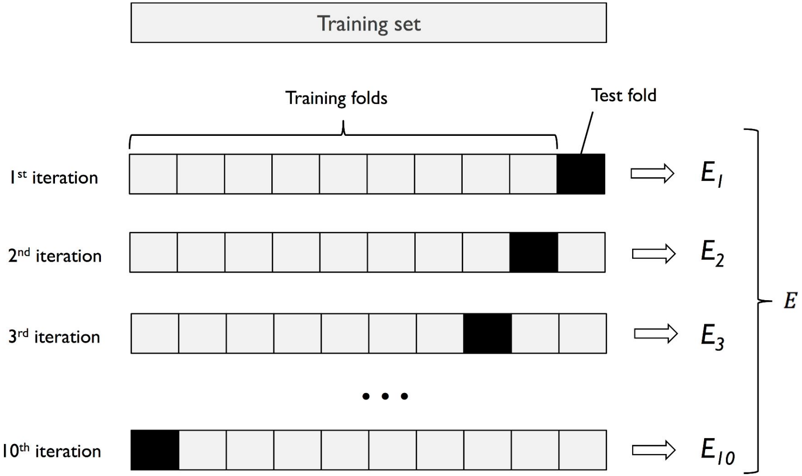
<u>Hold out Method (main steps)</u>

1. repeatedly train on the train dataset using different hyperparameter values
2. test the performance on the development set and choose the hyperparameter values that lead to best predictions
3. once you find the hyperparameter values that satisfy you, estimate the model's generalization performance on the test dataset.



K-fold Cross Validation

Figure from Raschka's book



1. randomly splits the training data into K folds w/o replacement.
2. use K-1 folds for training, and 1 fold for testing.
3. repeat 2. K times, to obtain K models and performance estimates.
4. Typically, K=10 or 5

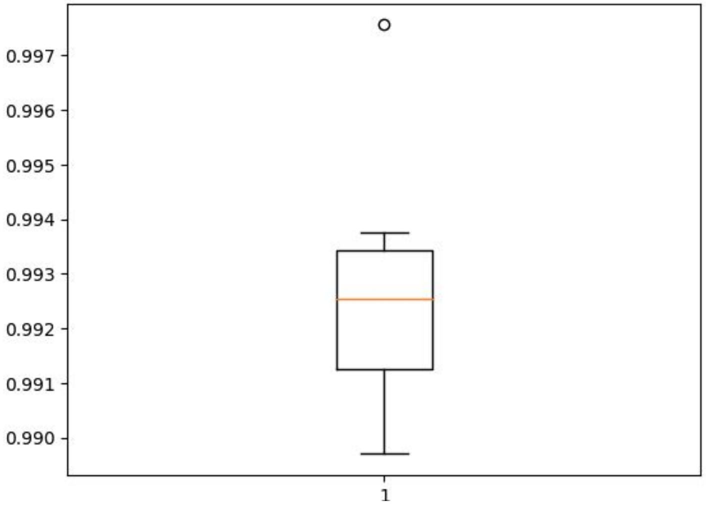
```
### quick test(simple error estimation)
### fit on the first 20K examples, evaluate on the remaining ones; report the accuracy through clf.score
clf = LogisticRegression()
clf.fit(M_scaled[:20000], l[:20000])
print (clf.score(M_scaled[20000:], l[20000:]))

### Real reporting (rigerous error estimation using cross validation -- more on this later!)

scores = cross_val_score(clf, M_scaled, l, cv=10)
print("Cross validation scores: ", scores)

#boxplot of the scores to visualize mean and std
plt.boxplot(scores)
plt.show()
```

```
0.9932756703993927
0.9893963627747171
Cross validation scores:  [0.98969911 0.99755965 0.99050976 0.99349241 0.99295011 0.99376356
 0.99213666 0.99186551 0.99105206 0.99322126]
```



Concept Recap

- prediction error
- accuracy
- confusion matrix
- grid search
- k-fold cross-validation
- holdout method

	Class 1 (+) Predicted	Class 2(-) Predicted
Class 1 (+) Actual	TP	FN
Class 2 (-) Actual	FP	TN

class 1 (+); class 2 (-)

Accuracy (Classification Rate):
(TP + TN) / (TP + TN + FP + FN)
(# correctly classified examples/ # examples)

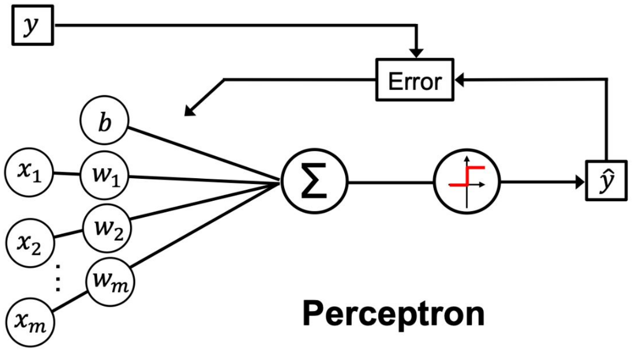
Recall: TP/ (TP + FN)
(# correctly classified pos. examples/# pos. examples)

Precision: TP/ (TP + FP)
correctly classified + examples/ # of predicted + examples

See: https://ibug.doc.ic.ac.uk/media/uploads/documents/ml-lecture3-2014.pdf

Neural Networks - Perceptron

Perceptron from Scratch - code



Let j represent the j-th features (m total). Let i represent the i-th observation (n total). η is the learning rate.

`Net input function`: $z = w_0 \cdot 1 + w_2 \cdot x_1 + w_2 \cdot x_2 + \ldots + w_m \cdot x_m$

`Threshold function`:

$$\tau(z) = \begin{cases} 1, \text{if } z \geq 0, \\ -1, \text{ otherwise} \end{cases}$$

`Compute error`: $(y\_actual_i - \tau(z_i))$, where $\tau(z_i)$ is y_pred_i

`Update weights`:

$$w_j := w_j + \Delta w_j, \forall j \in 1, \ldots, m$$

$$\Delta w_j = \eta(y\_actual_i - \tau(z_i)) \cdot x_{i,j}, \forall j \in 1, \ldots, m, \forall i \in 1, \ldots, n$$

The threshold function is used as an activation function. We'll see that for more complex neural models that's not always the case (they have both an activation and a threshold function)

Additional resources
https://nasirml.wordpress.com/2017/11/19/single-layer-perceptron-in-tensorflow/
https://towardsdatascience.com/the-magic-behind-the-perceptron-network-eaa461088367