In Section 3.2 of previous chapter we briefly discussed machine learning for complex models i.e. learning for structured instances and learning pipeline models. In this chapter, we discuss pipeline models in detail. As stated earlier, the main interest of this work is the use of machine learning techniques for natural language processing applications. Here we discuss the use of machine learning for an important natural language application i.e. information extraction. In Section 4.1 we provide an introduction about pipelining. In Section 4.2 we give a general overview of the information extraction process along with an example to show how the process will work. Section 4.3 discusses pipelining and machine learning and shows the steps of pipelining using active learning. In Section 4.4 we discuss stages of information extraction used in pipelining. In Section 4.5 we discuss various evaluation measures that are used to check the efficiency of machine learning models.

4.1. Introduction

Pipelining is a process in which a complex task is divided into many stages that are solved sequentially. A pipeline is composed of a number of elements (processes, threads, co routines, etc.), arranged in such a way so that the output of each element is fed as input to the next in the sequence. Many machine learning problems are also solved using a pipeline model. Pipelining plays a very important role in applying the machine learning solutions efficiently to various natural language processing problems. The use of pipelining results in better performance of these systems. However, these systems usually result in considerable computational complexity. A distinguishing feature of applications requiring pipeline models is that they often require significant quantities of labeled data to learn accurately, motivating the study of active learning in such scenarios. For this reason researchers were motivated for using active learning for these systems. Reason of using active learning is that these algorithms perform better than the traditional learning algorithms keeping the training data same. In this chapter we discuss an active learning strategy for pipelining of an important natural language processing task i.e. information extraction. The work described in this chapter has been previously published [Khan and Quadri, 2012a].

A number of natural language processing applications use machine learning algorithms. These applications include parsing, semantic role labeling, information extraction, etc. Using a machine learning algorithm for one natural language processing task often requires the output from
another task. Thus we can say these tasks are dependent on one another and therefore must be pipelined together. Therefore, a pipeline organization is used to model such situations. The benefit of using such an organization includes its ease of implementation and the main drawback is accumulation of errors between the stages of the pipeline that considerably affects the value of the results [Bunescu, 2008]. Pipelining has been used for a number of natural language applications e.g. bottom-up dependency parsing [Chang et al., 2006], semantic role labeling [Finkel et al., 2006]. A bidirectional integration of pipeline models has been developed as a solution to the problem of error accumulation in traditional pipelines [Yu and Lam, 2010]. In this chapter we show pipelining of information extraction. Although work has been done earlier in this regard which show pipelining of entity detection and relation extraction stages of information extraction. Here we theoretically discuss about including part-of-speech tagging stage of information extraction into the pipeline.

4.1.1. An Example of Pipelining

The primary motivation for modeling complex tasks as a pipelined process is the difficulty of solving such applications with a single classifier. For explaining the process of pipelining we will take an example of entity extraction as in Section 3.2. We will consider a sentence as shown in Figure 4.1. In this case, a pipeline model would first learn an entity identification (segmentation) classifier and use this as input into an entity labeling classifier, which is then assembled into a two stage pipeline system.

The primary requirement of a pipeline model is that the feature vector generating procedure for each stage is able to use the output from previous stages of the pipeline, \( \Phi^{(j)}(x, y^{(0)}, \ldots, y^{(j-1)}) \). To train a pipeline model, each stage of a pipelined learning process takes \( m \) training instances \( S^{(j)} = \{(x_1^{(j)}, y_1^{(j)}), \ldots, (x_m^{(j)}, y_m^{(j)})\} \) as input to a learning algorithm \( A^{(j)} \) and returns a classifier, \( h^{(j)} \), which minimizes the respective loss function of the \( j \)th stage. Once each stage of the pipeline model classifier is learned, global predictions are made sequentially with the expressed goal of maximizing performance on the overall task, resulting in the prediction vector \( \hat{y} = h(x) = [\text{argmax}_{y'} y^{(j)}(x^{(j)}) \] where \( j = 1 \) to \( J \) and \( y' \in Y^{(j)} \).
4 INFORMATION EXTRACTION AND MACHINE LEARNING-A PIPELINED APPROACH

Jake works in Calgary, Alberta

Segmentation

[Jake] works in [Calgary] [Alberta]

Named Entity Classification

[Jake]_{person} works in [Calgary]_{location} [Alberta]_{location}

Figure 4.1: Pipelined Segmentation and Entity Detection

4.1.2. Why Active Learning

An important aspect of pipelined approaches is the corresponding high cost associated with obtaining sufficient labeled data for good learning performance. The active learning protocol minimizes this problem by allowing the learning algorithm to incrementally select unlabeled examples for labeling by the domain expert with the goal of maximizing performance while minimizing the labeling effort [Cohn et al., 1996]. While receiving significant recent attention, most active learning research focuses on new algorithms as they relate to a single classification task. This work instead assumes that an active learning algorithm exists for each stage of a pipelined learning model and develops a strategy that jointly minimizes the annotation requirements for the pipelined process. In active learning the learning algorithm is capable of selecting additional instances to be labeled by maintaining access to the annotator. Thus active learning provides a way to reduce the labeling costs by labeling only the most useful instances for learning. Active learning reduces the amount of user effort required to learn a concept by reducing the number of labeled examples required [Arora and Agarwal, 2007]. In this learning technique, the learner is responsible for actively participating in the collection of the training examples i.e. obtaining the training set. The learner is capable of selecting a new input, observing the resulting output and including the new example based on the input and output into its training set. An important question that arises here is how to choose which input to try next
[Cohn et al., 1996]. The learner uses some strategies for choosing the examples. The examples are chosen by making queries to the expert. The query strategy frameworks that have been used are uncertainty sampling [Lewis and Gale, 1994] and query-by-committee [Seung et al., 1992].

4.2. Simple Architecture of Information Extraction

Information extraction (IE) can be defined as a process which involves automatic extraction of structured information such as entities, relationships between entities, and attributes describing entities from unstructured and/or semi-structured machine-readable documents [Sarawagi, 2008]. It can also be defined as a process of retrieving relevant information from documents. Applications of IE include news tracking [Turmo et al., 2006], customer care [Bhide et al., 2007], data cleaning [Sarawagi and Bhamidipaty, 2002], and classified ads [Michelson and Knoblock, 2005]. Figure 4.2 shows a simple architecture of information extraction system [Bird et al., 2006]. The overall process of information extraction is composed of a number of subtasks such as segmentation, tokenization, part of speech tagging, named entity recognition, relation extraction, terminology extraction, opinion extraction, etc.

![Figure 4.2: Simple Architecture of Information Extraction System](image)

These subtasks of information extraction can be implemented using a number of different algorithms e.g. list-based algorithms for extracting person names or locations [Watanabe et al.,]
2009], rule-based algorithms for extracting phone numbers or mail addresses, and advanced machine learning and statistical approaches for extracting more complex concepts. Sentence segmentation is the process of breaking the text into component sentences. It is the process of determining the longer processing units consisting of one or more words. It consists of recognizing sentence boundaries between words in different sentences. Since most written languages have punctuation marks which occur at sentence boundaries, sentence segmentation is frequently referred to as sentence boundary detection, sentence boundary disambiguation, or sentence boundary recognition. All these terms refer to the same task: determining how a text should be divided into sentences for further processing. Tokenization breaks the text into meaningful elements such as words, symbols. It is the process of breaking up the sequence of characters in a text by locating the word boundaries, the points where one word ends and another begins. For computational linguistics purposes, the words thus identified are frequently referred to as tokens. In written languages where no word boundaries are explicitly marked in the writing system, tokenization is also known as word segmentation, and this term is frequently used synonymously with tokenization. This is followed by part-of-speech tagging which labels these tokens with their POS categories. An example of applying these steps to a piece of text is shown below in Figure 4.3

Jake works in Calgary, Alberta with his brother Micheal.

<table>
<thead>
<tr>
<th>Jake</th>
<th>works</th>
<th>in</th>
<th>Calgary</th>
<th>Alberta</th>
<th>with</th>
<th>his</th>
<th>brother</th>
<th>Micheal</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>VB</td>
<td>P</td>
<td>NP</td>
<td>NP</td>
<td>P</td>
<td>DET</td>
<td>NP</td>
<td>NP</td>
</tr>
</tbody>
</table>

**Figure 4.3: Tokenization and Labeling**

This is followed by entity detection. It is the process of identifying the entities having relations between one another, e.g. considering the above sentence, entities are detected as follows:

Jake ➔ PERSON  Calgary ➔ LOCATION  Alberta ➔ LOCATION  Micheal ➔ PERSON

**Figure 4.4: Entity Detection**
Finally, after entities have been identified, the relations that exist between them are extracted in the relation detection step as follows:

\[
\{\text{Jake, Calgary}\} \rightarrow \text{works_in} \\
\{\text{Jake, Micheal}\} \rightarrow \text{brother_of} \\
\{\text{Calgary, Alberta}\} \rightarrow \text{located_in} \\
\{\text{Jake, Alberta}\} \rightarrow \text{works_in}
\]

Using pipelining in modeling the process of information extraction has resulted in an increase in efficiency. A lot of work has been done in this regard. Efficient information extraction pipelines have been developed that have resulted in the efficiency gains of up to one order of magnitude [Henning et al., 2011]. A pipeline-based system has been developed for automated annotation of Surgical Pathology Reports [Kevin et al., 2004]. There has been a lot of research in the field of information extraction using supervised machine learning. A number of supervised approaches have been proposed for the task of relation extraction which consists of some feature based methods [Kambhatla, 2004; Zhao and Grishman, 2005] and kernel methods [Lodhi et al., 2002; Bunescu and Mooney, 2005]. However, supervised methods have a number of disadvantages. First of all, we cannot extend these methods to define new relations between the entities due to lack of new labeled data as supervised methods have a predefined set of labeled data. Same problem occurs if we wish to extend the entity relations to higher order. Also for large input data these methods are computationally infeasible [Bach and Badaskar, 2007]. One of the main disadvantages of using supervised methods is the high cost associated with them as they require large amounts of annotated data. Active learning [Settles, 2010] provides a way to reduce these labeled data requirements. These algorithms are capable of collecting new labeled examples for annotation by making queries to the expert. The main advantage of using pipelining is that when the pipelining process starts the examples that are selected first are those that are needed at the beginning phases of pipeline followed by those that are needed later.
4.3. Pipelining and Machine Learning

In the supervised machine learning problem a function maps the inputs to the desired outputs by determining which of a set of classes a new input belongs to. This is determined on the basis of the training data which contains the instances whose class is known e.g. classification problem. The mapping function can be represented by $f$, $h$ denotes the hypothesis about the function to be learned. Inputs are represented as $X = (x_1, x_2, \ldots, x_n)$ and outputs as $Y = (y_1, y_2, \ldots, y_n)$ [Nilsson, 2005]. Therefore, hypothesis or the prediction function can be written as

$$h : X \rightarrow Y$$

$h$ is the function of vector-valued input and is selected on the basis of training set of $m$ input vector examples i.e.

$$X = (x_1, x_2, \ldots, x_n) \rightarrow h \rightarrow h(X)$$

Training set = \{ $X_1, X_2, \ldots, X_m$ \}

Therefore, the predicted value can be given as

$$y = h(x) = \arg\max_{y' \in Y} f(x, y')$$

In case of pipelining, we have different stages. Let there be $N$ stages. Therefore, each stage $n$ depends on the previous $(n-1)$ stages i.e.

$$x, y^{(0)}, \ldots, y^{(n-1)} \rightarrow x^{(n)}$$

Therefore, in case of pipelining the predicted value can be written as

$$y = h(x) = [\arg\max f^{(n)}(x^{(n)}, y')]$$

where $n = 1, \ldots, N$.

As discussed earlier in this chapter, active learning algorithms reduce the number of labeled examples needed to learn any concept by collecting new unlabelled examples for annotation [Thompson et al., 1999]. In active learning, the learner examines the unlabeled data and then queries only for the labels of instances which it considers to be informative. Therefore, an active learner learns only what it needs to in order to improve, thus reducing the overall cost of training.
an accurate system. In active learning the algorithm starts with a small number of labeled instances in the labeled training set \( L \). It then requests the labels for a few carefully selected instances from the unlabeled pool \( U \), learns from the query results, and then leverages its newly-found knowledge to choose which instances to query next. In this way, the active learner aims to achieve high accuracy using as few labeled instances as possible. The examples are selected from the unlabelled data source \( U \) and are then labeled and added to the set of labeled data \( L \) [Settles, 2010]. Figure 4.5 shows the process of active learning [Settles, 2009]. The examples are selected by making queries to the expert. There are many ways to select query instances, most of which stem from the uncertainty principle in experimental design and statistics [Federov, 1972]. One strategy for pool-based active learning is uncertainty sampling [Lewis and Gale, 1994]. It queries the instance that the model is least certain how to label. For probabilistic binary classifiers, this means querying the instance \( x \in U \) with the posterior probability \( P(y = 1 \mid x; \theta) \) that is closest to 0.5 (i.e., the most ambiguous instance). Query strategies that have been used earlier are uncertainty sampling and query by committee [Seung et al., 1992]. In both these strategies the point is to evaluate the informativeness of the unlabeled examples.

![Figure 4.5: Process of Active Learning](image)

The most informative instance or best query is represented as \( x^*_A \), where \( A \) represents the query selection method used [Settles, 2010]. In uncertainty sampling, the algorithm selects that example about which it is least confident. In that case,
This approach is often straightforward for probabilistic learning models. For example, when using a probabilistic model for binary classification, an uncertainty sampling strategy simply queries the instance whose posterior probability of being positive is nearest 0.5 [Lewis and Gale, 1994; Lewis and Catlett, 1994]. For many learning algorithms, a widely used method of uncertainty sampling is to select instances for which their predicted label is least confident, either from a probabilistic viewpoint or through a margin-based analogue [Lewis and Gale, 1994; Tong and Koller, 2000; Schohn and Cohn, 2000; Culotta and McCallum, 2005; Roth and Small, 2006b; Settles and Craven, 2008].

In case of margin sampling,

\[ x^*_{M} = \arg\min_{x} P_{\theta}(y_1 | x) - P_{\theta}(y_2 | x) \]  

where \( y_1 \) and \( y_2 \) are first and second most probable class labels [Scheffer et al., 2001].

Another uncertainty sampling strategy that uses entropy as uncertainty measure,

\[ x^*_{H} = \arg\max_{x} - \sum_{i} P_{\theta}(y_i | x) \log P_{\theta}(y_i | x) \]  

where \( y_i \) represents all the class labels [Settles, 2010]

The entropy-based approach can be generalized easily to probabilistic multi-label classifiers and probabilistic models for more complex structured instances, such as sequences [Settles and Craven, 2008] and trees [Hwa, 2004]. An alternative to entropy in these more complex settings involves querying the instance whose best labeling is the least confident:

\[ \Phi_{LC}(x) = 1 - P(y^*|x), \]

where \( y^* = \arg\max P(y|x) \) is the most likely class labeling. This sort of strategy has been shown to work well, for example, with conditional random fields or CRFs [Lafferty et al., 2001] for active learning in information extraction tasks [Culotta and McCallum, 2005; Settles and Craven, 2008].

Scoring functions are also used for selecting the examples to be labeled or annotated. Scoring functions are used for mapping an abstract concept to a numeric value. Here, the idea is to
calculate the score values for each instance to be labeled and the one with the minimum value is selected i.e.

\[ x^* = \text{argmin } q(x) \]

where \( x \) is selected from the unlabeled data \( U \). The key difference between active learning and standard supervised learning is a querying function, which when provided with the data \( U \) and the learned classifier \( h \) returns a set of unlabeled instance from \( U \). These selected instances are labeled and added to the supervised training set \( L \) used to update the learned hypothesis.

Therefore, for each stage \( n \) of the pipeline, there is a separate querying function i.e. \( q^{(n)} \), and after combining all these functions we get,

\[ x^* = \text{argmin} \sum q^{(n)}(x) \]

where \( n = 1, ..., N \) and \( x \) belongs to \( U \) and \( N \) is the total number of stages of a pipeline. The pipelining process using active learning consists of the following steps:

a. As discussed earlier, each stage \( n \) of the pipeline has its own querying function \( q^{(n)} \) and learner \( f^{(n)} \). First of all, for each stage \( n \), the hypothesis function as well as the querying function is estimated.

b. The unlabelled examples or instances are then selected by the learner from unlabeled data \( U \) and after labeling are added to labeled data \( L \) for each stage \( n \) of the pipeline.

c. As \( L \) changes after annotation of new instances, hypothesis is modified accordingly for each stage \( n \).

d. The process is repeated until the final hypothesis is obtained after all the \( N \) stages of pipeline have been completed.

### 4.4. Stages of Information Extraction used in Pipelining

Pipelining has been applied to information extraction earlier where the focus has been on entity detection and relation extraction. But as far as part-of-speech tagging is involved, not much has been done towards including it in the pipelining process of information extraction. Each stage of a pipeline is dependent on the earlier stages. In pipelining of information extraction, entity detection and relation detection highly depend on part-of-speech tagging. As discussed earlier,
part-of-speech tagging labels each word or phrase of a sentence with its POS category. It helps in recognizing different usages of the same word and assigns a proper tag e.g. in the sentences below the word “protest” has different usages:

The protest is going on. (Noun)

They protest against the innocent killings. (Verb)

Including part-of-speech tagging in the pipeline using active learning will result in the performance gain as the machine learning methods used for part-of-speech tagging have resulted in more than 95% accuracy. Moreover, in any natural language there are a number of words that are part-of-speech ambiguous (about more than 40%) and in such cases automatic POS tagging makes errors and hence require the use of machine learning techniques for tagging.

As discussed earlier, part-of-speech tagging labels each word or phrase of a sentence with its POS category, entity detection identifies the entities having relationships between one another in the sentence and relation detection extracts those relationships. Hence, in all these processes sentences are selected and annotated for all stages of the pipeline.

4.4.1. Including POS Tagging in Pipelining

Part-of-speech tagging (POS tagging), also called grammatical tagging or word-category disambiguation, is the process of marking up a word in a text (corpus) as corresponding to a particular part of speech, based on both its definition, as well as its context i.e. relationship with adjacent and related words in a phrase, sentence, or paragraph. Once performed by hand, POS tagging is now done in the context of computational linguistics, using algorithms which associate discrete terms, as well as hidden parts of speech, in accordance with a set of descriptive tags. POS-tagging algorithms fall into two distinctive groups: rule-based and stochastic. E. Brill's tagger, one of the first and widely used English POS-taggers, employs rule-based algorithms. Different methods of POS tagging are Rule-Based POS tagging e.g., ENGTWOL [Voutilainen, 1995], transformation-based tagging e.g. Brill’s tagger [Brill, 1995], and stochastic (probabilistic) tagging e.g. TNT [Brants, 2000]. POS tagging is used for a number of purposes e.g. it can help in determining authorship i.e. finding out are any two documents written by the same person (forensic linguistics) and it can help in speech synthesis and recognition. Labeling
natural language data with part-of-speech tags can be a complicated task, requiring much effort and expense, even for trained annotators. Several efforts, notably the Alembic workbench [Day et al., 1997] and similar tools, have provided interfaces to aid annotators in the process. Automatic POS tagging of text using probabilistic models is mostly a solved problem but requires supervised learning from substantial amounts of training data. Previous work demonstrates the suitability of Hidden Markov Models for POS tagging [Kupiec, 1992; Brants, 2000]. More recent work has achieved state-of-the-art results with Maximum entropy conditional Markov models (MaxEnt CMMs, or MEMMs for short) [Ratnaparkhi, 1996; Toutanova & Manning, 2000; Toutanova et al., 2003]. Part of the success of MEMMs can be attributed to the absence of independence assumptions among predictive features and the resulting ease of feature engineering.

In this section we theoretically show how active learning would be applied to POS tagging. As discussed earlier, first the informativeness of the unlabeled instances, sentences in our example, would be evaluated. Sentences would be selected from the unlabeled data and annotated/labeled by the annotator i.e. each word in the sentence would be tagged by its appropriate POS category. The annotated sentences will then be added to the labeled data. In Query By Uncertainty (QBU) approach, the informativeness of the unlabeled instances/examples is determined by evaluating the entropy- a measure of uncertainty associated with a random variable. In our example, these unlabeled instances are sentences. Therefore, we have to evaluate the entropy of sequence of words $w_i$ in a sentence of length $n$, i.e.

$$H(w_1,w_2,\ldots,w_n) = -\sum p(w_1,w_2,\ldots,w_n) \log p(w_1,w_2,\ldots,w_n)$$

From equation (2) we get,

$$x_H^* = -\sum p(y_i | x) \log p(y_i | x)$$

for each word $w_i$ of the sentence, $pos_i$ represents the part-of-speech tag for that word. Thus, the querying function for the part-of-speech tagging stage will be given as

$$q_{pos} = -\sum p(pos_i | w_i, y_i, pos_{i-1}, pos_{i-2}) \log p(pos_i | w_i, y_i, pos_{i-1}, pos_{i-2})$$

where $i = 1$ to $n$ and $pos_{i-1}$ and $pos_{i-2}$ represent the tags of previous two words.
4.4.2. Active learning for Entity and Relation Detection

For this stage too QBU approach will be used which selects those unlabeled examples/instances about which the learner is least confident. According to equation (1), the best query in case of multi class uncertainty sampling is given by

\[ x^{*\text{M}} = \arg\min \ P_0 (y_1 \mid x) - P_0 (y_2 \mid x) \]

where \( y_1 \) and \( y_2 \) are the first and second most probable class labels. Accordingly, the querying function for the entity and relation detection stage of information extraction can be given as

\[ q_{\text{ERD}} = \arg\min \ p(y \mid x_i) - p(y' \mid x_i) \]

or

\[ q_{\text{ERD}} = \arg\min \ [f(x_i, y) - f(x_i, y')] \]

\( i \) = 1 to \( n \) and \( y \) and \( y' \) are the first and second most probable class labels.

For all the stages, the performance would be calculated using three metrics i.e. precision, recall and F-measure. For POS tagging, precision would be calculated as number of correctly retrieved tags divided by the total number of retrieved tags. Recall would be calculated as number of correctly retrieved tags divided by the actual number of tags. For entity detection, precision would be calculated as the number of correctly extracted entities divided by the total number of extracted entities and recall would be calculated as number of correctly extracted entities divided by the actual number of entities. For relation extraction, precision would be calculated as the number of correctly extracted relations divided by the total number of extracted relations and recall would be calculated as the number of the correctly extracted relations divided by the actual number of relations. F- Measure for all these stages is equal to \( 2 \times \text{precision} \times \text{recall} / \text{precision} + \text{recall} \).

4.5. Evaluation Measures

This section outlines various evaluation measures that are used for checking how well a model performs. For a particular label of interest, we are provided with a set of actual positives (e.g., objects that belong to that label) contained within the data set. The model then makes a set of predicted positives (e.g., the objects it assigns to that label) for the same data set. The actual and predicted label groupings can be thought of as indicator variables, and their cross product results
in four important values: $tp$ (the number of true positives), $fp$ (false positives), $tn$ (true negatives), and $fn$ (false negatives). Figure 4.6 [Settles, 2008] illustrates the relationship between these numbers. A basic evaluation measure is accuracy $= \frac{tp+tn}{tp+fp+tn+fn}$. Basically, this measure represents the fraction of objects that the model labels correctly. In some problems, however, the data may be highly skewed, e.g., there might be nine times as many negative objects as positives. In a case like this, accuracy is a poor evaluation measure because a model that labels everything negative will still have accuracy $= 0.9$. In these situations, it is common instead to use precision, $P = \frac{tp}{tp+fp}$, the fraction of predictions that are correct, and recall, $R = \frac{tp}{tp+fn}$, the fraction of actual positives that are correctly predicted. Because of the inherent trade-off between precision $P$ and recall $R$, a summary statistic called the F-Measure $= \frac{2 \times P \times R}{P + R}$ is commonly used when both are considered equally important. A final evaluation measure is the area under the Receiver Operating Characteristic (ROC) curve. An ROC curve measures the rate of true positives vs. false positives as a threshold is varied across a measure of confidence in its predictions (e.g., the model’s posterior probability of the target label). It is regarded as a more appropriate measure than accuracy for some machine learning applications [Provost et al., 1998]. The area under the curve AUROC, also called the Wilcoxon signed-rank test, can be interpreted as the probability that the model will rank a randomly chosen positive object higher than a randomly chosen negative.

![Venn diagram](image)

**Figure 4.6:** A Venn diagram illustrating the relationship between actual and predicted positives.
The various overlaps define regions of tp (true positives), fp (false positives), tn (true negatives) and fn (false negatives).

Since it is trivial for a model to do well on the labeled data L that was used to train it, the practice of randomly partitioning data into a training set and an evaluation set is used, which do not overlap. In this way, the model is properly evaluated on new instances it has never seen before. To account for the effects of randomized partitioning, it is common to repeat an experiment for several runs and average the results. One particular way of doing this is cross-validation. In five-fold cross-validation, for example, the data is split into five partitions or folds. Then the five experiments are run for which each fold is held aside for evaluation, and the remaining four folds are used for training; then results are averaged across all folds.