ABSTRACT

This senior design project consists of examining a statistical model (Canonical Correlational Analysis), using it to predict different properties of the words in corpora (for example, Wikipedia and Project Gutenberg) such as, what entity type (E.g., person, place, organization ...) what the words are, what they link to, and what part of speech they comprise. We have built a system that, given a large sample text acquired from (mostly) Project Gutenberg, will remove unnecessary tags, tokenize the words (using the de-facto PennTree Tokenization), tag these words with part-of-speech (POS) tags, create state vectors for each word, run regressional analysis by plugging the POS and state vectors into a linear model, and finally predict and disambiguate properties of these words. We do this last part, using a method based on Canonical Correlation Analysis (CCA), which is a generalization of Principle Component Analysis (PCA) to pairs of matrices. CCA based methods are an alternative to the more commonly used method based on Hidden Markov Models (HMMs).

We present a Multi-View learning (MVL) method based on CCA - an alternative to the currently popular Hidden Markov Model (HMM). Running several trials under this method to predict properties of words, will provide insight (for certain linear models) into the advantage of employing CCA followed by linear regression.

The advantages of using MVL with CCA become apparent when it comes to the problem of scaling. Since CCA and linear regression is linear, it allows us to work with samples that are many times larger than those sizes that can be handled by non-linear methods. The fact that MVL with CCA scales well under large sets of data outweigh the disadvantage that certain non-linear methods may provide more accurate results for certain problems. Also, implementing CCA is rather straightforward compared to the complex coding required by some non-linear methods.

The fact that MVL can be used to address problems that are currently modeled by HMMs is yet another advantage of multi-view learning. MVL is optimal under the assumptions of linearity and Markov Property - that given the present, the future does not depend on the past. This is precisely the standard Markovian assumption used in HMMs. Hence MVL has the potential to be an efficient alternative method for problems currently modeled by HMMs.

To examine the MVL approach with CCA followed by linear regression, we have implemented a system, which can be trained (on a large input text), then subsequently be used to predict part-of-speech labels given a small sample text. In this paper we discuss the following:

1. Overview of the implemented MVL system.
2. Specifics of training the CCA model and calculations that yield a prediction.

1.1 Markov Property

A stochastic process \( \{X_n\} \) (sequence of random variables) on the state space \((S, \mathcal{S})\) - where \(S\) is the set of states and \(\mathcal{S}\) computes the directions of maximal correlation between a pair of matrices. These matrices, in our case, represent smoothed versions of the training text (a huge excerpt from corpora, such as Wikipedia or Project Gutenberg), where the training text is smoothed to capture structural patterns in the language influenced by both before (past) and after (future) a certain word in the whole text. Hence by employing CCA on these two "views" (past and future smooths), we obtain a latent structure that is common to both. Then we use linear regression to predict part-of-speech labels as a state-space vector.

The advantages of using MVL with CCA become apparent when it comes to the problem of scaling. Since CCA and linear regression is linear, it allows us to work with samples that are many times larger than those sizes that can be handled by non-linear methods. The fact that MVL with CCA scales well under large sets of data outweigh the disadvantage that certain non-linear methods may provide more accurate results for certain problems. Also, implementing CCA is rather straightforward compared to the complex coding required by some non-linear methods.

The fact that MVL can be used to address problems that are currently modeled by HMMs is yet another advantage of multi-view learning. MVL is optimal under the assumptions of linearity and Markov Property - that given the present, the future does not depend on the past. This is precisely the standard Markovian assumption used in HMMs. Hence MVL has the potential to be an efficient alternative method for problems currently modeled by HMMs.

To examine the MVL approach with CCA followed by linear regression, we have implemented a system, which can be trained (on a large input text), then subsequently be used to predict part-of-speech labels given a small sample text. In this paper we discuss the following:

1. Overview of the implemented MVL system.
2. Specifics of training the CCA model and calculations that yield a prediction.
is the σ-field generated by $S$ adapted to a filtration $\{F_n\}$, is said to have Markov Property if for every set $A \in S$:

$$\mathbb{P}(X_{n+1} \in A | F_n) = \mathbb{P}(X_{n+1} \in A | X_n) \quad (1)$$

Where $\mathbb{P}$ is a probability measure[5]. Loosely speaking, this is the property of having no memory, since the conditional probabilities of future states depend only on the present state and are independent of the past states. Such processes may be time discrete or continuous and state discrete or continuous. An example of a time and state continuous Markovian process (stochastic process with the Markov property) is Brownian Motion. Our interests are with Markov Chains, which are a time discrete Markovian processes on a discrete space.

### 1.1.1 Markov Chain

A Markov Chain, is defined on a finite or countable state space $S$, initial distribution $\{\nu_i\}_{i \in S}$ (where each $\nu_i \geq 0$ and $\sum_i \nu_i = 1$), and transition probabilities $\{p_{ij}\}_{i,j \in S}$ (where each $p_{ij} \geq 0$, and $\sum_j p_{ij} = 1$ for each $i \in S$). It can be thought of as the representation of a particle moving randomly in the space $S$. $\nu_i$ is the probability that the particle starts off at the point $i$ in space, and $p_{ij}$ is the probability that the particle will move from point $i$ to point $j$ [17]. Markov chains are stochastic processes, and can be defined as a sequence of random variables $\{X_t\}$ on $S$ such that for any natural number $n$ and any $i_0, i_1, ..., i_n \in S$ the following holds:

$$\mathbb{P}(X_0 = i_0, X_1 = i_1, ..., X_n = i_n) = \nu_{i_0} p_{i_0 i_1} p_{i_1 i_2} ... p_{i_{n-1} i_n}$$

We note that this is the discrete counterpart to equation (1), hence a Markov Chain obeys the Markov Property. A Markov Process is a continuous time Markov Chain, that is, it is a continuous time process on a discrete or countable state space.

### 1.1.2 Hidden Markov Model

The Hidden Markov Model is a stochastic model on a Markov Process. As opposed to the regular Markov Model, in which the states are observable and the transition probabilities are known, in HMM, however, the states are “hidden”, meaning they either cannot be observed or are unobserved. However, the outcomes and the transition probabilities from each state are observable. Hence each state is a probability distribution over the outputs. The Hidden Markov Model was introduced and studied in the late 1960s has a probability distribution over the outputs. The Hidden Markov Model is the statistical model of choice in applications requiring unsupervised learning.

### 1.2 Multi-view learning

Multi-view learning is the idea we claim will not only be faster than Markov models (as has been suggested), but will also produce better results. By incorporating for example real valued vectors to represent state, we could obtain a much more compact model than that offered by standard Markov Models. For example,

$$[0.0, \ldots, 1.0, \ldots, 0] \text{ (Single State)}$$

$$[0.482, -7.261, \ldots, 2.272] \text{ (Multi State)}$$

The idea of Multi-view learning is, that some commonality between two different views of a single object defines a representation which can be used to characterize the object. For example, in the two-view setting, there are two co-occurring views $X_1$ and $X_2$ of the data, and a target variable $Y$ of interest. Two natural underlying assumptions arise from this setting.

The first assumption is that either one of the views is sufficient to predict $Y$. Here, the complexity of the learning problem can be reduced by the elimination of subspaces of each view, which do not agree with each other. This can be done using unlabeled data, and a wide class of algorithms ([22] [2] [3] [14] [11] [9] [21] [20] [10] [19]).

The second assumption is that on some set of “hidden variables” the two views are independent. This is meant to capture the intuition that there is “hidden” state that generates the two views. It is this hidden structure that is relevant for predicting target variables. If we can capture that information about this “hidden state” using the correlation structure between both views, then this information could be used in subsequent supervised learning tasks [2] [1]. We focus on this second idea.

### 1.3 Canonical Correlation Analysis (CCA)

At the heart of Multi-View learning lies an important technique of applied linear algebra known as CCA [8]. It is the analog of a technique known as PCA for pairs of matrices.

PCA computes the directions of maximum covariance between elements in a single matrix. It does this by computing the eigenvectors of the correlation matrix of a single matrix. Thus it can be cast as an eigenvalue problem on a covariance matrix. However, compared to CCA, it wastes dimension (CCA reduces the dimension of the original data set), hence it is not as efficient as the CCA.

We discuss the mathematics behind the CCA using arrays (i.e. one dimensional vectors). Let $X = (X_1, X_2, ..., X_m)$ and $Y = (Y_1, Y_2, ..., Y_n)$ be random variables in $\mathbb{R}^m$ and $\mathbb{R}^n$ respectively with mean zero. That is they are measurable maps, $X : \mathbb{R}^m \to \mathbb{R}$ and $Y : \mathbb{R}^n \to \mathbb{R}$. Then CCA finds the directions $W_X$ and $W_Y$ that maximizes the correlation between the projections $x = W_X^T X$ and $y = W_Y^T Y$ (x and y are often referred to as canonical correlates)[6]. Hence, we must maximize the Pearson product-moment correlation...
Coefficient for $x$ and $y$ defined as follows,

$$\rho_{x,y} = \frac{\text{Cov}(x,y)}{\sigma(x)\sigma(y)} = \frac{E(xy) - E(x)E(y)}{\sqrt{E(x^2)E(y^2)}} \quad (2)$$

Where $E$ and $\sigma$ stands for the expected value and standard deviation respectively. Since $X$ and $Y$ were assumed to have mean zero, equation (2) reduces to

$$\rho_{x,y} = \frac{E(xy)}{\sqrt{E(x^2)E(y^2)}} \quad (3)$$

$$= \frac{E(Wx^t Y^t Wx)}{\sqrt{E(Wx^t Xx^t Wx)E(Wy^t Yy^t Wv)}} \quad (4)$$

$$= \frac{W^t x^t \Gamma_{x,y} Wx}{\sqrt{W^t x^t \Gamma_{x,x} Wx W^t y^t \Gamma_{y,y} Wv}} \quad (5)$$

Where the $\Gamma_{x,y}$ in equation (5) is the cross-covariance, which is defined as the matrix where the $(i,j)$th element is the covariance between the $i$th and $j$th element of $X$ and $Y$ (i.e. Cov($X_i, Y_j$)).

To maximize $\rho_{x,y}$, we set the derivative of $\rho_{x,y}$ (equation (4)) with respect to $Wx$ and $Wy$ equal zero [6], and we obtain:

$$\Gamma_{x,x}^{-1} \Gamma_{x,y} \Gamma_{y,x} Wx = \rho_{x,y} Wx \quad (6)$$

$$\Gamma_{y,y}^{-1} \Gamma_{y,x} \Gamma_{x,y} Wy = \rho_{x,y} Wy \quad (7)$$

This is the eigenvalue equation, where $\rho_{x,y}$ is the eigenvalue of the linear transformation $\Gamma_{x,x}^{-1} \Gamma_{x,y} \Gamma_{y,x}$ corresponding to the eigenvector $Wx$, and of the linear transformation $\Gamma_{y,y}^{-1} \Gamma_{y,x} \Gamma_{x,y}$ corresponding to the eigenvector $Wy$ [6].

CCA finds a set of basis vectors such that these basis vectors are orthonormal and such that the subspaces spanned by these vectors are maximally correlated [7]. If we let $\Pi X_i$ denote the projection of an observation $X_i$ onto the canonical correlates (referred as $x$ and $y$ above). The dimension of $\Pi X_i$ is determined by how many basis vectors are used in the projection. This projection can be viewed as the “state” characterizing the object since $\Pi X_i$ is a low dimensional representation of the observation. This state can then be used either to compute similarity to states for other observations, or as features in a regression to predict, for example, a label for the observation. CCA looks in many ways similar to the widely used PCA. CCA can be viewed as computing the principle components of the correlation between the views $X$ and $Y$. However, CCA offers many advantages over PCA. CCA is invariant to scale and affine transformations; thus, unlike PCA, when using CCA one does not need to worry about rescaling the data.

### 1.4 Exponential Smoothing

Smoothing, in statistics, is a way to eliminate noise and irregularities in a time series data (observations which have time or spatial ordering), while keeping important intrinsic properties of the data. Exponential smoothing is a specific type of smoothing where events further away from the “present” are given exponentially decreasing weights. In contrast, a Simple Moving Average Smoothing is a technique where events are given equal weights and simply averaged over $t+1$ observations. Hence if we let $\{x_n\}$ represent a time series data, where each $x_n$ is an observation at time $n$, then the smooths (which we call $\{s_n\}$) are found using the following simple formula:

$$s_n = \frac{1}{t+1} \sum_{k=0}^{t} x_{n-k}$$

The major drawback of this method being that we cannot capture the fact that often, events far away from the present have less impact on the current observation, hence must be given less weight. An improvement is the Weighted Moving Average Smoothing, in which one first chooses a vector of weighting factors $\{\alpha_0, \alpha_1, \ldots, \alpha_t\}$ such that $\sum_{k=0}^{t} \alpha_k = 1$. Then the average is computed by weighing the $t+1$ observations according to the corresponding $\alpha_k$. The following formula shows how:

$$s_n = \sum_{k=0}^{t} \alpha_k x_{n-k}$$

Hence by choosing a decreasing sequence of weighting factors, one can give more importance to the observations closer to the present and less to the observations further away from the present. However, these Moving Average Smoothing methods have a major disadvantage that since the first $t$ observations are required for computing the smooth $s_n$, it cannot be used on the terms $0, 1, \ldots, t$.

Exponential smoothing addresses these shortcomings by calculating the smooths recursively as follows:

$$s_n = \alpha x_{n-1} + (1 - \alpha)s_{n-1}$$

where $s_0 = x_0$ and $0 < \alpha < 1$

Here $\alpha$ is a smoothing factor (we often call this the “lag”).

If $\alpha$ is closer to 1 then Exponential Smoothing gives greater weight to the events that are closer to the “present” and if $\alpha$ is closer to 0 then this method gives less weight to the recent observations. From the formula given above, it is easy to observe that Exponential Smoothing only requires two observations to compute a smooth. The Kalman Filter, (see section 1.5) is a closely related method that attempts to calculate values close to the observations by using time series data.

### 1.5 The Kalman Filter

The Kalman filter is a recursive estimator. In order to compute an estimate for the current state with the Kalman Filter one only needs the estimated state from the previous iteration and the current measurement. This means that no history of observations and estimates is required. This filter has “Predict” and “Update” phases, where the “Predict” phase uses an estimate from previous iterations to provide a current estimate, and the “Update” phase uses the current prediction with the observed state at this stage to provide a refinement of the state estimates.

### 2. RELATED WORK

A variety of techniques have been proposed over the years to make use of unlabeled or partially labeled data to improve machine learning. Co-training [2] [18] uses two different views of the same data, thus resembles our proposed model closely. However, co-training makes stronger demands on...
the data that is used and the algorithms to handle these types of data are easily broken. Other multi-task methods take a complementary view, using the same features to predict multiple labels, divided into “main” and “auxiliary” tasks. Caruana [1997], Ando and Zhang [2005], Zhang et al. [2005], multi-task methods assume that the same latent structure that predicts the auxiliary task labels will also be predictive of the main task. Rather than assuming that the features are divided into complementary sets, they assume that the same features can be applied to different tasks.

The method, Ando and Zhang [2007] assume conditional independence where \( X_1 \) and \( X_2 \) are assumed to be independently conditioned on \( Y \). This is closer to the spirit of this proposal. Ando and Zhang [2007] do not explicitly consider CCA, but their results show that CCA can be used to project the views down to a lower dimensional space, such that this projection does not lose the predictive information about the target \( Y \).

2.1 Drawbacks

**Ando and Zhang [2007]**

The drawback to the assumption in this method, is that conditional independence based on the observed target (as opposed to based on the hidden state) is too stringent.

**Kakade and Foster [2007]**

The drawback to the redundancy assumption in this method is that often both predictors are not equally good (i.e. due to occlusion).

In theory we may expect a significant improvement when we have both views, however we hope that both views can still be used in an unsupervised manner. We will address both of the drawbacks mentioned above by relaxing our assumptions.

2.2 Advantages of CCA

CCA offers a variety of advantages. As mentioned above it is invariant to scale and affine transformations. Furthermore if the views of the data contain a large number of features that are irrelevant for the classification of the object while still being correlated with each other, CCA will capture the common features of the different views, being able to reject extraneous dimensions. On the other hand, PCA will waste many dimensions trying to describe irrelevant features. The next example shows the advantage of CCA over PCA on a large data set.

**Example 2.2.1**

Suppose both your views are composed of video and audio visual streams. One approach would be to use PCA on a very large single matrix using a huge number of dimensions to describe the scene. On the other hand, a much more efficient approach would be to use CCA which will realize that most of the visual scene is entirely uncorrelated with the audio and such projections would not be included in the canonical basis.

CCA is ideally suited to estimate state for linear dynamical systems such as Kalman filters, since the Markov assumption makes the past and future observations be independent conditional on the current state. CCA has been used in time series, such as in the systems identification literature [13] [15] [12]. More recently, work in Hsu et al.[4] shows how CCA can be used to learn HMMs. We will attempt to characterize how the multi-view assumption makes each problem easier in terms of better capturing states as well as computational complexity.

3. SYSTEM MODEL

At its core, multi-view learning is simply CCA on the two views followed by regression or clustering to predict the label. Different algorithms can be used for CCA and for the regression/clustering components. We are focusing on sparse CCA and sparse regression and testing a variety of such methods.

In this project we combine three closely coupled aspects:

1. Learning state (an object trained on a large data set using CCA) in language and how it can be efficiently estimated and what that state represents in terms of syntax (parts of speech) and semantics.

2. Implementation of a pipeline that will efficiently scale to very large data sets.

3. Demonstrations of the efficacy of multi-view learning on machine reading comprehension prediction and disambiguation.

A generic pipeline has been put together such that the exchange of different components of this system is made easy. By doing so, we may easily experiment with different algorithms and implementations to improve efficiency and performance. Moreover, by decomposing this pipeline into sensible stages, we have decoupled key aspects as to not waste time and resources in generating multiple sets of the same data and performing unnecessary calculations.

This is crucial for this project because we deal with extremely large quantities of raw data (several gigabytes or more of compressed excerpt from corpus), thus generating objects and models are very expensive operations that should be performed (in an ideal scenario) only once per data set. At this size, functions that are seemingly simple (such as the tokenization of text) can present themselves as a potential bottleneck and can give rise to memory issues if not designed properly.

The key to designing a seamless and efficient pipeline was to break the whole process into stages, and tweak its components with the objective of resulting in an accurate and efficient predictive model that will estimate state representations in terms of syntax and semantics. The pipeline that we built to this purpose is displayed below (in both training mode Figure 3.1a and Figure 3.1b):
4. SYSTEM IMPLEMENTATION

The MVL with CCA system that we have built consists of two major components, namely, training and predicting. In order to ensure accurate and efficient predictions, we must first train the CCA model as well as the labels (coefficients) and store the result in disk. Hence when one runs the prediction part of the pipeline, the trained CCA model

4.1 Detailed Explanation of Pipeline (Figure 3.1a and Figure 3.1b)

1. The input to run_mvl.py is a text file containing the file names separated by new lines and a boolean parameter that specifies whether the run is “training” or “prediction”. The file names point to the files from which we draw the text for the run (which may either be training or prediction). Obviously, for “training” the text files will be large in size, and for prediction they are smaller samples. The main function then creates a list structure that contains as the file names as entries, which we refer to as a “document list”, and calls the right type of MVL simulation we intend to run (whether it is missing words or POS).

2. Every word in every file in the document list provided above is scanned and added to a dictionary where a certain number of them (for example one hundred) is assigned a unique integer ID. The more words we choose the less sparse the matrices that represent them become, hence there exists a trade-off between how rich of a vocabulary the model captures versus computational complexity. Choosing the most common words is optimal from the point of view of capturing the linguistic properties. From this point on these (one hundred) words have numerical identifiers and the rest of the words are identified as zero. We refer to this as the X_dictionary. The X_dictionary is used to map each word into an integer, which produces a sequence of integers, which we refer to as int_doc. Depending on the dictionary used (X_dictionary or Y_dictionary) we name the int_doc as int_doc_x and int_doc_y respectively. For example if the text is

\[ It \text{ takes its name from the first six characters seen. } \]

and the dictionary maps each word to an integer as follows:

<table>
<thead>
<tr>
<th>It</th>
<th>takes</th>
<th>its</th>
<th>name</th>
<th>from</th>
<th>the</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
</tbody>
</table>

Then the int_doc produced is

\[ \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \]
\[ 5 \quad 2 \quad 1 \quad 0 \quad 0 \quad 6 \]
Now there are two possibilities, the run is intended for:
we proceed to do the following:
we read a pre-computed CCA model from disk and go
prediction of properties (POS or missing words) then
produced by this example would look as follows:

Figure 4.1 - Matrix for sample sentence

\[
\begin{array}{c|cccccccc}
\text{It} & \rightarrow & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
\text{takes} & \rightarrow & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\text{its} & \rightarrow & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
\text{name} & \rightarrow & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
\text{from} & \rightarrow & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{the} & \rightarrow & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{first} & \rightarrow & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{six} & \rightarrow & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{characters} & \rightarrow & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
\text{seen} & \rightarrow & 0 & 0 & 0 & 1 & 0 & 0 & 0
\end{array}
\]

3. Now there are two possibilities, the run is intended for prediction of properties (POS or missing words) then we read a pre-computed CCA model from disk and go to step 4. If training is the purpose, then it is assumed that the file list we were given is the training data and we proceed to do the following:

3a. In order to obtain a CCA we will require the eigenvectors to a matrix given by a product of square smoothing matrices. In order to do this the `compute_correlations` function uses the `X_dictionary` and the sample text to build a matrix $M$ as shown in Figure 4.1, then it computes a left smooth and a right smooth, which are referred to as the left view $L$ and right view $R$ respectively. Given $\alpha$, the smoothing rate, the exponential smoothing is done as follows:

$$L_{[t,]} = (1 - \alpha)L_{[t-1,]} + \alpha M_{[t,]}$$

$$R_{[t,]} = (1 - \alpha)R_{[t-1,]} + \alpha M_{[t,]}$$

Where the notation $L_{[t,]}$ represents the $[t+1]^{th}$ row of the matrix $L$, $R_{[t,]}$ represents the $[t-1]^{th}$ row of the matrix $R$, and finally $M_{[t,]}$ represents the $th$ row of the matrix $M$.

The dimensions of the left ($L$) and right ($R$) view matrices can become rather large, hence we multiply them by their transpose to obtain a square matrix of a smaller dimension and produce the matrices $LL, RR, LR$,

$$LL = L^t L$$

$$RR = R^t R$$

$$LR = L^t R$$

These matrices are, respectively, the square (transpose times original) of the left smooth, the right smooth, and finally the product of the transpose of the left smooth times the right smooth.

With the above in mind, the Canonical Components are the eigenvectors $\phi$ of the matrix $(L^tL)^{-1}L^tR(R^tR)^{-1}R^tL$, hence if we let $\Lambda$ be the eigenvalue,

$$(L^tL)^{-1}L^tR(R^tR)^{-1}R^tL\phi = \Lambda \phi$$

Where, as mentioned above, $L$ is the left view, $R$ is the right view.

3b. The CCA, which we call `leftCCs` is essentially a matrix composed of the eigenvectors $\phi$ with the $k$ largest $\phi$'s by magnitude.

3c. Now we calculate a state matrix (state), in which each row is the estimated state of each token influenced by the words before it. To obtain state, we multiply $L$ by the `leftCCs`.

$$\text{state} = L \cdot \text{leftCCs}$$

3d. Since we are carrying out training in this sequence of steps we are now interested in generating our regression coefficients.

The function `train_emissions()` takes the matrix `state` and a sample $Y$ matrix built from a dictionary which is based either on the same logic as the $X$ dictionary (in the case of missing words) or on the results of pretagged data by our POS tagger, TNT. Then the linear algebra package returns a coefficient matrix $\beta$ to be stored alongside our CCA for future use by doing a simple linear regression,

$$\beta = (\text{state}'\text{state})^{-1}\text{state}'Y$$

4. For predictions, the pre-computed `leftCCs` and the coefficients $\beta$ are read from disk, and a left smooth matrix $L$ is computed (as discussed in part [3a]) for the sample text input. Multiplying $L$ times the loaded `leftCCs` we obtain the state matrix for the sample input.

5. With $\beta$ and state in hand we calculate the prediction $\hat{Y}$ by essentially undoing the linear regression done during training, but with the new `state` of the sample data.

$$\hat{Y} = \text{state} \cdot \beta$$

This predicted matrix $\hat{Y}$ may then be translated (depending on whether we are doing POS predictions or missing words)

6. Lastly, an error check is run by using the Root Mean Square (RMS) on $Y - \hat{Y}$, where $Y$ is the original matrix and $\hat{Y}$ is the predicted matrix. If $Y$ and $\hat{Y}$ are an $[n \times m]$ matrix, then the RMS is calculated as follows,

$$\text{ERROR RMS} = \sqrt{\frac{\sum_{k=1}^{n} \sum_{j=1}^{m} (Y_{k,j} - \hat{Y}_{k,j})^2}{n \cdot m}}$$

RMS measures the magnitude of the variation hence will give us a measure of how close or how far our prediction $\hat{Y}$ is from the actual $Y$. The smaller the RMS error the more closer to the actual value we are, and
the bigger the RMS error, it means that our prediction is further from the actual value.

4.2 Tweaks

Figure 4.1 - Old Pipeline of Multi-view Learning System

In this section we will discuss some major performance tweaks which were added to the pipeline along the way in order to allow for faster, more efficient, and larger runs which were necessary to generate competitive models from vast amounts of data.

4.2.1 Faster Tagging

The first largest bottle neck was Stage 3b (see Figure 4.1) where tokens were being tagged with POS tags. Originally the Stanford POS Tagger was being used, however, this proved to be inefficient in large sample texts. As we can see in Figure 4.2 below, the Stanford tagger takes 163.38 seconds to tag 100 words, and the time grows exponentially as we take larger and larger sample texts. In fact, to tag a whole book (Pride and Prejudice) the Stanford Tagger took 2819.58 seconds (about 0.7 hours). This may seem like a plausible number, but considering the fact that the book was only 701kb in size, compared to the gigabytes of data that we would be working with, the efficiency of the Stanford Tagger would be impractical for our purposes.

Figure 4.2

Aside the slow speed of the Stanford Tagger, another issue that came up with using this tagger was the fact that the Stanford Tagger used an internal tokenizer to tokenize text. This would later cause problems when syncing vectors in the state labeled, and POS tagged files. Moreover, having to tokenize the same text twice, using two potentially different tokenizers would not scale well when dealing with large data sets. The solution to these problems was to use a different POS tagger. The TNT Tagger (as described in Section 4-System Implementation) was not only faster, but also decoupled the process of tokenization and tagging. Hence we had control over which tokenization standards to use, and would only have to tokenize the text once. Figure 4.3 shows the speed of the TNT Tagger, when it was run with the same set of texts as the Stanford Tagger. Notice how, while still being exponential in time complexity, the TNT tagger is running several orders of magnitude faster.

Figure 4.3

The main reason for such a dramatic difference in speed between the two taggers is that the stanford tagger builds a statistical model (maximum entropy or logistic regression). Whereas the tnt tagger does a table look up of the relevant n-grams. Table look-up is faster. In a logarithmic scale, we can readily observe (see Figure 4.4 below) that the TNT Tagger would be an optimal choice.

Figure 4.4

4.2.2 Faster Linear Algebra

A second bottleneck emerged to haunt us when generating a CCA (which requires some heavy linear algebra package matrix computations) running in the interpreted python was
just not cutting it. In order to generate these CCA models from a lot of data (the key to obtaining good state and properly capturing the most semantics in our corpora) the implementation of the CCA changed from python to C++. The C++ implementation was able to overcome this difficulty and generate much more solid CCA models for use in the system.

4.2.3 An Iterative Approach

One final change made to the computationally intense portion of the pipeline was making the MVL algorithm iterative in nature. In this approach every token of input is accompanied not only by a state vector but also by an attribute vector (estimated as an average of state vectors computed for the word in iterations and used as criteria for stopping the iterative algorithm thereby giving us a more reliable state). The essence of the iterative approach is as follows:

1. As usual we take the input text and transform it into an integer sequence representation (alongside a matrix representation) but we keep track of a separate dictionary referred to as the “attribute dictionary” which initially maps every token to a vector (of the same dimension as the state vector would be for this run) of random numbers taken uniformly from the interval (0, 1) (this is our initial guess, typical of iterative algorithms).

2. We define the state of each token to be the attribute vector stored in the “attribute dictionary” entry for that token.

3. We now carry out exponential smoothing as in the original algorithm but do so on the state estimate matrices instead of the matrix representation of the integer sequence. We carry out the smoothing, as usual, for both states before and after each token to get our usual left and right view pairs. The formula, with smoothing coefficient (or lag) α is as before:

\[
L_{[t, \cdot]} = (1 - \alpha)L_{[t-1, \cdot]} + \alpha M_{[t, \cdot]}
\]

\[
R_{[t, \cdot]} = (1 - \alpha)R_{[t-1, \cdot]} + \alpha M_{[t, \cdot]}
\]

4. Now using the CCA we estimate (or re-estimate, if not the first iteration) the state of each token using the familiar

\[
\text{state} = L \cdot \text{leftCCs}
\]

5. At this point we estimate the attributes for each token again by averaging the states estimated for that particular token.

6. We compute the change in attribute vectors (numerically by using the usual Euclidean norm) and halt the algorithm if and only if the change in norm from the previous iteration is bounded by some positive tolerance ϵ. If it is not, we go back to step 2.

5. RESULTS

When tested, the system has come to perform very well. Accuracy has been high in almost all instances and requiring very little training when it comes to the actual regression. The whole work really does go into building a CCA matrix - the larger (and hence more time consuming) this is, the better the results become. What is best is that this arduous process must only be carried out once - having it run in C++ rather than python was an excellent decision. To start off we set the system up to disambiguate between whether words are nouns or not. We ran the prediction engine with varied number of tokens used to train the regression coefficients and obtained the following accuracy results (with a lower than random or most-frequent guessing RMSE). A graph of the results follows:

Figure 5.1 Accuracy of System for Disambiguating Whether Noun or Not

<table>
<thead>
<tr>
<th>Number of training tokens</th>
<th>Accuracy (Noun Disambiguation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2011</td>
<td>83.7</td>
</tr>
<tr>
<td>10091</td>
<td>84.2</td>
</tr>
<tr>
<td>50591</td>
<td>85.1</td>
</tr>
<tr>
<td>151528</td>
<td>86.4</td>
</tr>
</tbody>
</table>

The zoom on the vertical axis serves to show that the performance does increase (albeit trivially) with a larger training set for coefficients (plus or minus random variation of course as we can see with the right-most bar). The fact that the changes in performance are minimal with an increase in training set size for the coefficients lets us cut down some time when training this part of the pipeline as merely over one thousand tokens suffice for a strong enough linear model most of the time.

For the same dictionary (disambiguating between nouns and not nouns) we tested the system on 200 different popular English works of literature. The following is a boxplot of the accuracy for the 200 books:
Even the worst performance was way above the accuracy of guessing one way or the other and the mean centered pretty well in the high 80s and low 90s of accuracy percentage.

While a lot of testing was conducted on this particular property set (noun or not noun), the generic nature of the system allowed us to substitute in some other dictionaries to attempt prediction over a larger set of possibilities. The following graph (discussed below) illustrates the performance over several different types of prediction.

**Figure 5.3 Accuracy Across Different Prediction Dictionaries**

For comparison the first bar is the noun or not a noun dictionary. The second bar attempts another interesting possibility. It attempts to distinguish between whether every token is a verb or not, and if it is, what type of verb it is (the acronyms VB, VBN, and VBD stand respectively for verb, past participle verb, and past verb). As one can see, the performance is still excellent, nearly 80%. In this example, however, we noticed that guessing for neither of the three performed at 92% accuracy - but this is because these are not all the types of verbs that the TNT tagger handles (there are other types like VBQ and VBO, which respectively stand for a verb ending in 'ing' and a verb ending in 's' and even then all of these do not constitute a majority of the words (sentences normally contain only one verb). Due to this, in this case such guessing outperforms the model. The final bar refers to the accuracy of disambiguating between all 42 tags provided by the TNT part of speech tagger. While the accuracy seems to be low at 29%, guessing for the most common of them would perform at about 24% accuracy - so beating this reveals that the system is working well. A much larger CCA trained over a cluster of computers over days would perform even better than this accuracy - the CCA we trained was only trained on one computer for several hours.

Overall the graphs reveal some fault, but overall, they demonstrate that the system is able to quickly and accurately disambiguate between linguistic properties of text as we expected it to.

6. FUTURE WORK

Seeing as the system as it stands produces reliable results, a first idea for future work which we would have carried out if we had more time would be to expedite the computation of the canonical correlation matrix. The algorithm (even the tweaked iterative version) form the hotspot of the system and future work would probably focus a lot of effort on them. Running this procedure in as distributed and parallel as possible would prove instrumental to make it possible to generate much larger and more statistically capable models to gain even better prediction capabilities.

Other work would involve making the system more interactive, smoother, and better overall structured. For example, one possibility would be to rewrite the system using C++ even though the computationally intense part has already been rewritten in C++. Experimenting with different CCAs is something which unfortunately we did not have much time to do as these are extremely time consuming to generate. Data from different CCAs would provide great insight into how much better the system can be and what the right trade-offs for CCA size and reliability versus computational effort involved really are.

7. CONCLUSION

Overall this year long project was very exciting. It was a demanding and difficult project, however, and it required an incredible amount of attention to detail and became acquainted with legacy code. The legacy code presented a relatively large challenge - as it is something we were not used to encountering in large quantities (the main repository out of which we operated had over ten thousand revisions).

While much work was required, the system performed with a high degree of accuracy and was fast considering it was written with a lot of disk IO and primarily in python.
8. REFERENCES