Homework 1

Pedro Matias
pmatias@uci.edu

1 Preliminaries

A basic data analysis, summarized in fig. 1, showed that the dataset is unbalanced when it comes to total number of documents/words/letters, where higher numbers of these correlate with democrat affiliation. The average number of words per document, letters per document and letters per word is pretty identical across across the candidates. Indeed, initial experiments showed that assigning weights inversely proportional to each label frequency results (more often than not) in better accuracies, so we adopted this strategy throughout our experiments. During these initial experiments, we gained further information that made us adopt the following strategies in later experiments:

	democrat	#words	#docs	#words/#docs	#chars	#chars/#words	#chars/#docs
CLINTON_PRIMARY2008	1	33716	1441	23.397641	153001	4.537935	106.176960
OBAMA_PRIMARY2008	1	19884	846	23.503546	89922	4.522330	106.290780
MCCAIN_PRIMARY2008	0	9207	408	22.566176	43583	4.733681	106.821078
EDWARDS_PRIMARY2008	1	7874	340	23.158824	36096	4.584201	106.164706
RICHARDSON_PRIMARY2008	1	7504	336	22.333333	36095	4.810101	107.425595
GIULIANI_PRIMARY2008	0	5728	247	23.190283	25798	4.503841	104.445344
GINGRICH_PRIMARY2012	0	4109	181	22.701657	19070	4.641032	105.359116
ROMNEY_PRIMARY2012	0	3902	173	22.554913	18358	4.704767	106.115607
SANTORUM_PRIMARY2012	0	3523	149	23.644295	15585	4.423787	104.597315
THOMPSON_PRIMARY2008	0	3337	145	23.013793	15223	4.561882	104.986207
HUCKABEE_PRIMARY2008	0	3093	130	23.792308	13547	4.379890	104.207692
ROMNEY_PRIMARY2008	0	1555	69	22.536232	7209	4.636013	104.478261
PERRY_PRIMARY2012	0	1523	68	22.397059	7265	4.770190	106.838235
PAUL_PRIMARY2012	0	1441	60	24.016667	6320	4.385843	105.333333
BIDEN_PRIMARY2008	0	1319	58	22.741379	6110	4.632297	105.344828
BACHMANN_PRIMARY2012	0	1068	48	22.250000	5176	4.846442	107.833333
PAWLENTY_PRIMARY2012	0	840	40	21.000000	4163	4.955952	104.075000
HUNTSMAN_PRIMARY2012	0	620	28	22.142857	3065	4.943548	109.464286
CAIN_PRIMARY2012	0	409	17	24.058824	1745	4.266504	102.647059

Figure 1: Dataset analysis.

- using 5-fold **cross-validation**, scored using the accuracy averaged across folds, since the small labeled-to-unlabeled ratio could promote overfitting. We used *stratified* folds to promote even label frequencies (since data is unbalanced). We also merged train and dev data for this purpose.
- using **TF-IDF**, as opposed to bag-of-words (counting or binary) model
- using both word-unigrams and **word-bigrams**, as opposed to just word-unigrams. In our case, the dependencies between consecutive words modeled by bigrams features helped the results slightly. We did not have time to try character n-grams
- using `nltk.word_tokenize` on **lower-case** text, as opposed to `Scikit-learn`'s default tokenizer
- not removing **stopwords**, since early experiments did not make worthy improvements, but also because we (i) used TD-IDF (which already discounts words with high document-frequencies), but also (ii) entirely removed words with high document-frequencies cutoffs (see section 2)
- using L2-regularization, as opposed to L1, given early experiment results
- using a maximum of 100 **iterations** for Logistic Regression convergence, as opposed to larger values (which did not increase the cross-validation accuracy and were slower) or smaller values (for which we underfit). When combining TF-IDF with other features (see section 3), we used larger values to avoid underfitting.

2 Supervised

We experimented with different configurations and noticed an increase in accuracies for higher values of C, the inverse of regularization strength, as depicted in fig. 2. It is interesting to note that changes in accuracy caused by varying document-frequencies cutoffs (i.e. the minimum document-

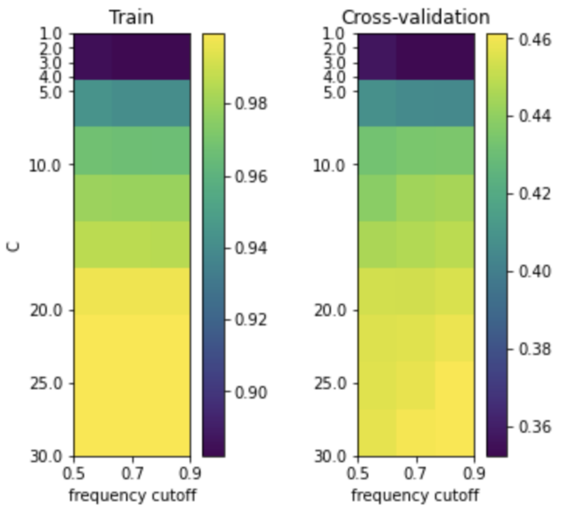

Figure 2: Varying document-frequency cutoff for removing words (x-axis) against regularization levels (y-axis) for Logistic Regression (higher values of C correspond to less regularization). See section 1 for remaining hyperparams configuration.

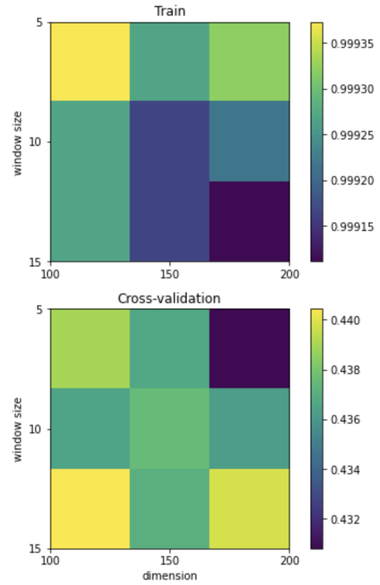

Figure 3: Accuracy varying context window sizes and output embedding dimensions.

frequency of a word considered for removal) was only noticeable for larger values of C, where accuracies were higher for higher cutoffs (i.e. less words were removed). A possible explanation is the fact that the stratregy of assigning higher weights (give less regularization) to words works best when the vocabulary size considered is indeed higher.

Indeed, increasing C even further resulted in even better results, until C assumes values bigger than 100, at which point the accuracy starts decreasing. Our best configuration, with respective accuracy scores was the following (see section 1 for the remaining hyperparams setting):

C	80
frequency cutoff	0.9
train accuracy	0.999
cross-validation accuracy	0.464
Kaggle accuracy	0.491

3 Semi-supervised

We tried incorporating word2vec embeddings, using Gensim's implementation. We generated these embeddings by training on both the labeled and unlabeled data provided. We also tried GloVe's pre-computed word2vec embeddings (Wikipedia 2014 + Gigaword 5, for 50-dimension), but results didn't differ significantly from results given by embeddings trained on the corpus itself. To combine the embeddings of each word in a single document, we tried the following approaches, while varying the number of dimensions (100, 150, 200) and the context window sizes (5, 10, 15, 20) (see fig. 3):

- Coordinate-wise **average** of all embeddings in the document
- Coordinate-wise **TF-IDF weighted average** of all embeddings in the document
- Coordinate-wise **sum** of all embeddings in the document
- Coordinate-wise **minimum** and/or **maximum** of all embeddings in the document

The best type of aggregation was TF-IDF weighted average. Close to this was regular average. The remaining types of aggregation had very small accuracies (less than half).

Unfortunately, we were not able to find a good configuration of hyperparams for TF-IDF weighted average, and our best model delivered the following accuracies (we increased the number of iterations from 100, since the increased number of features requires more time to avoid underfitting):

C	95
frequency cutoff	0.9
#iterations	200
w2v aggregation	TF-IDF weighted avg.
train accuracy	0.999
cross-validation accuracy	0.453
Kaggle accuracy	0.476

Indeed, the embeddings produced are not representative of each presidential candidate – see section 3.1 for a comparison between the average embedding for each presidential candidate.

3.1 Clustering

Next, we tried the following approach:

1. use a **clustering** technique to cluster the union of labeled (train+dev) and unlabeled data, into K clusters

2. determine a **mapping** between clusters and presidential candidates, using the labeled data. This mapping can be one-to-one, one-to-many or even many-to-one, depending on K

3. label the unlabeled data according to the previous mapping

4. re-train the previous best supervised classifier using both the initial labels and the ones generated in the previous step

To perform clustering, we resorted to lower-dimensional word2vec embeddings, using Gensim implementation. As in the previous section, we generated these by training on both the labeled and unlabeled data provided.

Before implementing the idea above mentioned, we examined the quality of embeddings produced by our supervised classifier, in the sense of how good they cluster the labeled data – see section 3.1 for details. As illustrated, the average embeddings for each candidate are very similar to each other, suggesting that labeled "clusters" are overlapping quite a bit, yielding bad representative embeddings. The similarity between candidates seems even more pronounced when comparing using cosine similarity and this may be due to the fact that it ignores vectors norms.

To evaluate the quality of our clusterings (using K-Means and Gaussian Mixture Model (GMM)), we estimated the likelihood that the average candidate belongs to each of the clusters generated – see fig. 5 for details. Particularly, we computed the probabilities $P(X_i \mid Z_j)$ that a document X_i is labeled with candidate k, given that we know it belongs to cluster Z_j and we summed these up, for each candidate k, to obtain $P(C_k \mid Z_j)$, which essentially models the distribution over candidates for each cluster Z_j. By Bayes' Rule and the assumption that X follows a uniform distribution

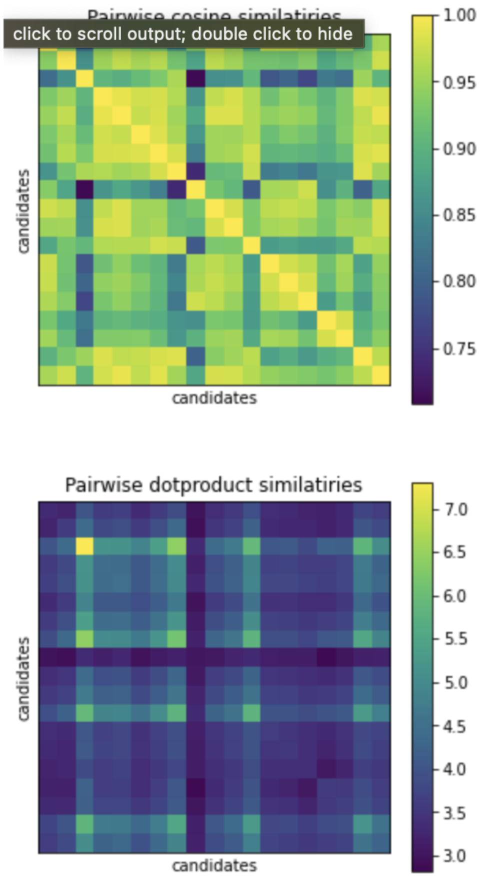

Figure 4: Similarities between the average (a.k.a. centroid) embedding for each presidential candidate, suggest that embeddings are not very representative of each candidate. The embeddings have dimension 200, were produced using a context window of size 20 and were combined using TF-IDF weighted average (other kinds of aggregations with lower dimensions/windows had similar or worse results).

$\mathcal{U}(1, 19)$, the quantities $P(X_i \mid Z_j)$ are proportional to the *membership probabilities* $P(Z_j \mid X_i)$, which can be obtained directly from the weights used for K-Means/GMM. The takeaway here is that the clusters generated were not good, since they contain (with some exceptions) the same proportion of candidates. Worth noting, however, is the fact that GMM gives more representative clusters when compared to K-Means, so we used this one to retrain our supervised classifier. Unfortunately, we weren't able to find a good configuration that allowed us to improve the Kaggle accuracy obtained using the supervised classifier. It is also possible that the clusterings obtained can be improved with further fine-tuning of K-Means/GMM hyperparams, which we did not conduct due to time restrictions. Our best model is given below (see section 1 for details on remaining hyperparams setting). The difference between cross-validation and Kaggle accuracies suggests overfitting occurrence.

C	80
frequency cutoff	0.9
#iterations	200
train accuracy	0.999
cross-validation accuracy	0.616
Kaggle accuracy	0.467

Implementation details.

For K-Means **weights initialization**, we tried k-means++ algorithm, random initialization, and using the labeled centroids (average embedding for each candidate – see section 3.1). All of these seemed to perform similarly. Regarding GMM, the parameters were initialized in a way that every mixture component has zero mean and identity covariance. The **mapping** from cluster centroids to candidates was obtained by sampling, for each cluster centroid Z_j, according to the probabilities $P(C_k \mid Z_j)$ computed. We avoided selecting the candidate C_k with maximum presence in Z_j to prevent having most clusters labeled with the most popular candidates (which tend to be democrats, see figs. 1 and 5). We set the number of clusters for both K-Means and GMM to $K = 19$ and, for the sake of analysis, $K = 2$. The latter case of $K = 2$ emphasized even more the difficulty of clustering, since we obtained membership probabilities close to 50% (i.e. $P(C_k \mid Z_1) \approx P(C_k \mid Z_2)$, for each k).

4 Statement of collaboration

Briefly discussed with Ramtin how to approach the semi-supervised model, namely which approach would be more interesting or more likely to work.

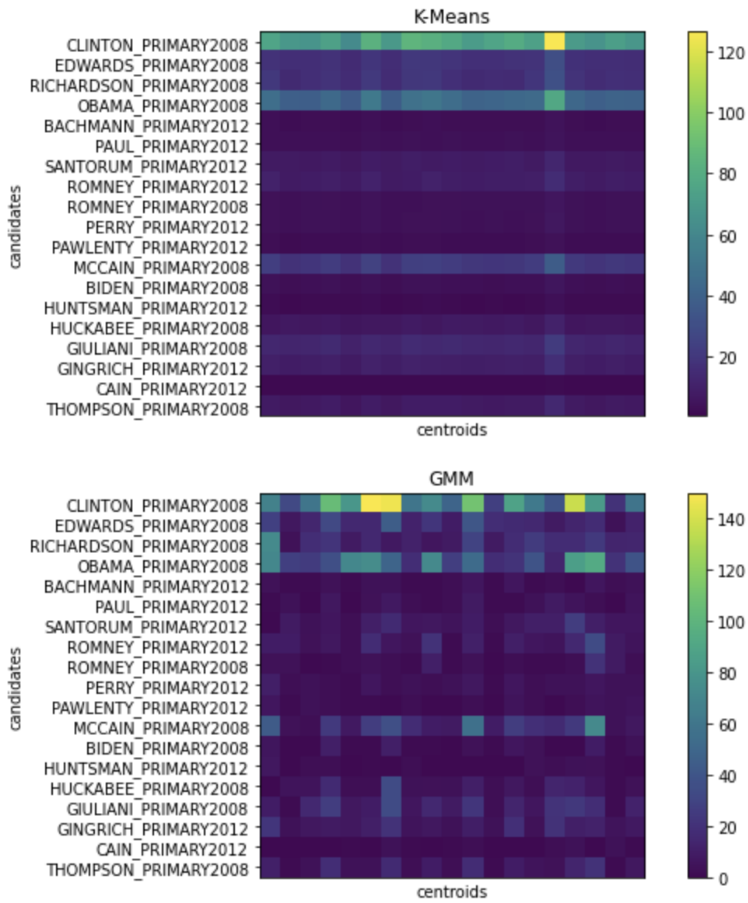

Figure 5: Each row i contains the membership probabilities $P(Z_j \mid C_k)$, representing the probability that the average candidate k belongs to the j^{th} cluster, out of 19 clusters generated, using K-Means (initialized with k-means++) and GMM (initialized for 0 means and identity covariance). We “joined” the democrats together in the top-4 rows, to assess any affiliation discrepancies – we obtained higher membership probabilities due to the unbalance in the data which favors democrats more over republicans.