1 NAIVE BAYES CLASSIFICATION DESIGN

1a) $P(X|Y) = \frac{P(X \cup Y)}{P(Y)} = \frac{P(X \cup Y)}{P(X)} = P(Y|X)$
$P(X|Y)P(Y) = P(Y|X)P(X)$
$P(Y|X) = \frac{P(X|Y)P(Y)}{P(Y)}$

1b) This equation is always true and makes no assumptions on Y, X only that P(X) and P(Y) > 0.

1c) $P(Y \mid X_1, X_2, ..., X_{|W|}) = \frac{P(X_1, X_2, ..., X_{|W|}|Y)P(Y)}{P(X_1, X_2, ..., X_{|W|})}$

1d) $P(Y)$ is the probability of decade Y_i ocurring amongst all the movies, the prior probability. $P(Y \mid X_i)$ is the probability the movie i is in decade Y given the attributes of movie i (feature vector X_i). This is the posterior probability and what we want to maximize, i.e if a movie plot contains "hello", "radio" etc what is the probability of classing that movie as decade Y?. Finally $P(X_i \mid Y)$ are the likelihoods of seeing a particular word X_i in decade Y.
$P(Y) = \frac{N_y}{N}$ where N_Y is the number of movies in decade Y and N is total number of movies.
$P(X_i \mid Y) = \frac{W_w y}{\sum_{w \in W} W_w y}$ where $W_y w$ is the number of times a word w appears in training reviews from decade y, including multiple counts.

1e) $P(X_1, X_2, ..., X_{|W|} \mid Y) = P(X_1 \mid Y)P(X_2, ..., X_n \mid Y, X_1)$
$= P(X_1 \mid Y)P(X_2 \mid Y, X_1)P(X_3, ..., X_n \mid Y, X_1, X_2)$
$= P(X_1 \mid Y)P(X_2 \mid Y, X_1)...P(X_{|W|} \mid Y, X_1, X_2, X_3, ..., X_{n-1})$

1f) This is equation is still always true, since we have only applied the chain rule expanding upon our previous equation above. This calculations holds true for the joint probability model and applying the definition of conditional probability over and over again. We have not assumed any independence yet between the random variables.

1g)

1h) The Naive Bayes assumptions means that the features are independent given the class, i.e assumes conditional independence between X_i and X_j for $i \neq j$ given Y. It is violated when we have duplicate copies of features across the data set. Here the model would assume that two feature vectors are identical and would consider them both as equal evidence.

1i) $P(Y \mid X_1, X_2, ..., X_{|W|}) = \frac{P(Y)P(X_1|Y)P(X_2|Y)P(X_3|Y)...P(X_n|Y)}{P(X_1, X_2, ..., X_{|W|})}$
$P(Y \mid X_1, X_2, ..., X_{|W|}) = \frac{P(Y)\prod_{i=1}^{n}(X_i|Y)}{P(X_1, X_2, ..., X_{|W|})}$

2 TIME FOR SOME DATA

3 FINDING ICONIC WORDS

4 THE MOMENT OF TRUTH

4a) Using sklearn we achieved an accuracy of nearly 68%. Our implemented version got an accuracy of 47.79%. This is more than the 3% range but we did not include stop words in our own implementation so there is a chance that we could get closer to sklearn.

4b) The underlying difference between them is in the assumptions they make in regards to the distribution of the likelihood probabilities $P(X_i \mid Y)$. Under Gaussian NB, the likleihoods are assumed to be a Gaussian distribution. Multinomial NB assumes a multinomial distribution amongst the conditional probabilities, each feature gets a count associated with it. And finally Bernoulli NB assumes that each feature has a Bernoulli distribution, or that multiple features only take values 0 or 1 (Boolean). The main difference between Bernoulli and Multinomial is that when calculating the probabilities Multinomial distributions ignore that a particular feature i had no count while Bernoulli would associate a penalty with that. The closest one for us would be Multinomial, because we actually keep track of counts of words that appear in each decade, as opposed to just being 0 or 1. Main difference from the our implementation and sklearn's of Multinomial is that they smooth out the occurrences and log probabilities.

5 MAXIMUM MARGIN CLASSIFICATION

5a) We chose to use hashing vectorizer for training our bag of words. We chose to use hashing vectorizer from sklearn, with 80,000 features. Our original vector had 120,000 unique features (after eliminating stop words) and we thought that k=80,000 is a decent enough number of features to reduce dimensionality and get good enough fit. For the second part of this question, we also tested out random projections. After defaulting to random projections it reduces our dimensionality from (54000, 118720) to (54000, 9340).

5b) We had to pass `with_ mean=False` in our standard scaler because it was throwing an error for sparse matrix data.

5c) None of the classifiers outperform sklearn's implementation of Naive Bayes, which had the highest accuracy at 68%. Our implemented version from problem 2 had an accuracy of 47%. It only outperforms PerceptronL2 with 25 iterations, Nearest Neighbors, We write the results to a file called results.txt of all the classifiers and their accuracy. Here are the results:
Ran classifier SGDClassifier: achieved accuracy 60.41%
Ran classifier Linear SVC: achieved accuracy 60.78%
Ran classifier SVC Kernel RBF: achieved accuracy 62.54%
Ran classifier PerceptronL1: achieved accuracy 61.16%
Ran classifier PerceptronL2: achieved accuracy 46.42%
Ran classifier Nearest Neighbors: achieved accuracy 17.16%
Ran classifier Ridge Classifier: achieved accuracy 41.33%

5d)

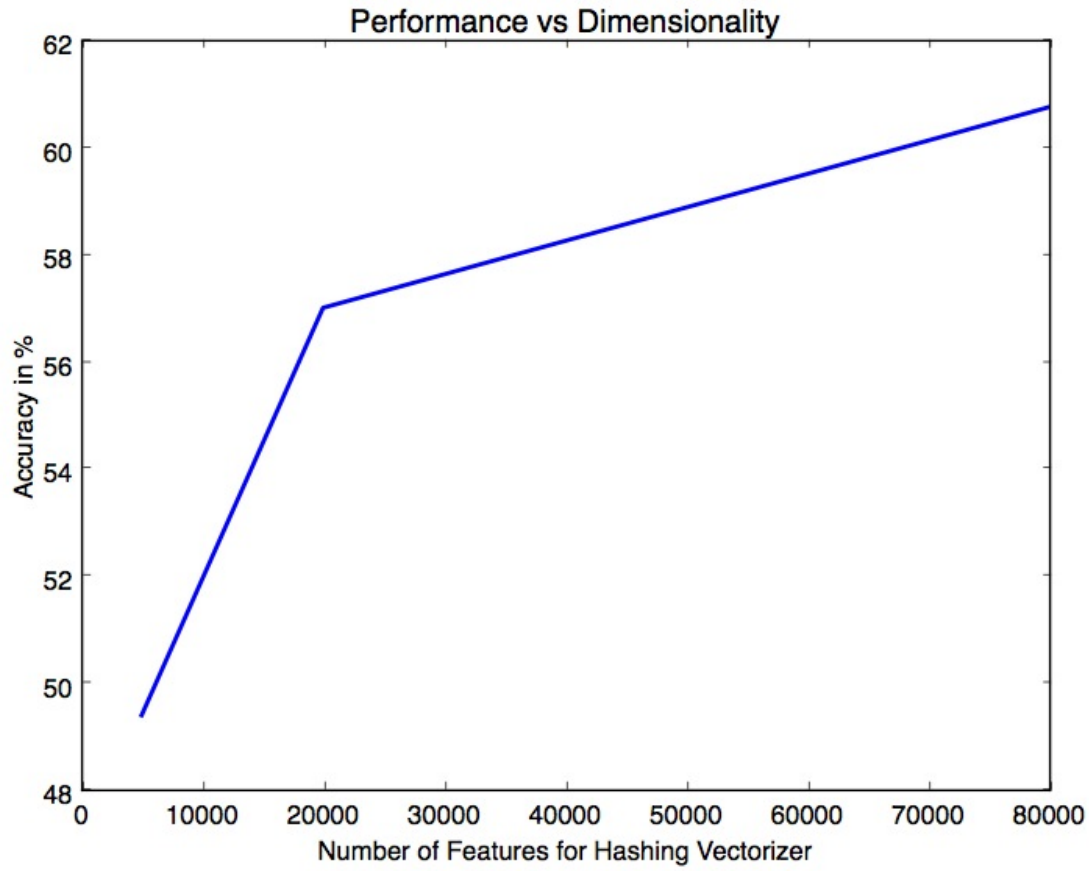
- SGDClassifier: it is highly efficient and very customizable, it had one of the lowest running times from our batch. In SGD we run through the training set in accordance to the gradient of the error of one particular training example. That is, we compute the direction that minimizes the objection function the most (min error). In SGD we approximate the amount of which to descend upon. It is stochastic because the estimated step computing in each training example is thought of to be a RV, this also helps to increase computational efficiency. The L2 is how we penalize the errors in the objective function.
- Linear SVC: runs a support vector machine classification with a linear kernel. The decision rule depends on the support vectors, where it tries to find the maximum distance separating hyperplane. The linear kernel works well when the number of features is larger than the number of samples, which is the case we have here. The method is

very efficient because it applies the "kernel trick" where it can compute efficiently dot products in higher dimensional spaces.

- RBF SVM: instead of a linear support vector this uses a radial basis function kernel. The RBF function acts like a low-pass signal filter. It selects samples that are smooth, typically used to classify images. It doesn't necessarily apply much to text classification but still can separate the data.
- PerceptronL1: as seen in class this classifier iteratavely runs through all the training samples and it learns by finding errors. Each time a training example has been misclassifies, it adds it to the weight vector to attempt to classify that example again. This one is also computationally efficient because we don't update samples that were correctly classified.
- PerceptronL2: Same as above however this one stops after only 25 iterations of the algorithm. That is way the accuracy is so much lower than the fully converged attempt from above. This one also uses the L2 norm or regularization term.
- Nearest Neighbors: This classifiers finds the k nearest neighbors for a new sample point according to shortest distance, and predicts the label of the majority of the adjoining neighbors. The input is k which finds the k nearest neighbors to selected sample, default is 5.
- Ridge Classifier: uses ridge regression during training and then classifies an instance according to some threshold. Like in most linear models, we are also training to find a hyperplane to linearly separate the data. Ridge regression/classification also is good at avoiding overfitting by regularizing the weights to keep them small (non-zero but small). As we also saw in lecture Ridge regression with a kernel is a special case of HMSVM with a hinge loss function.

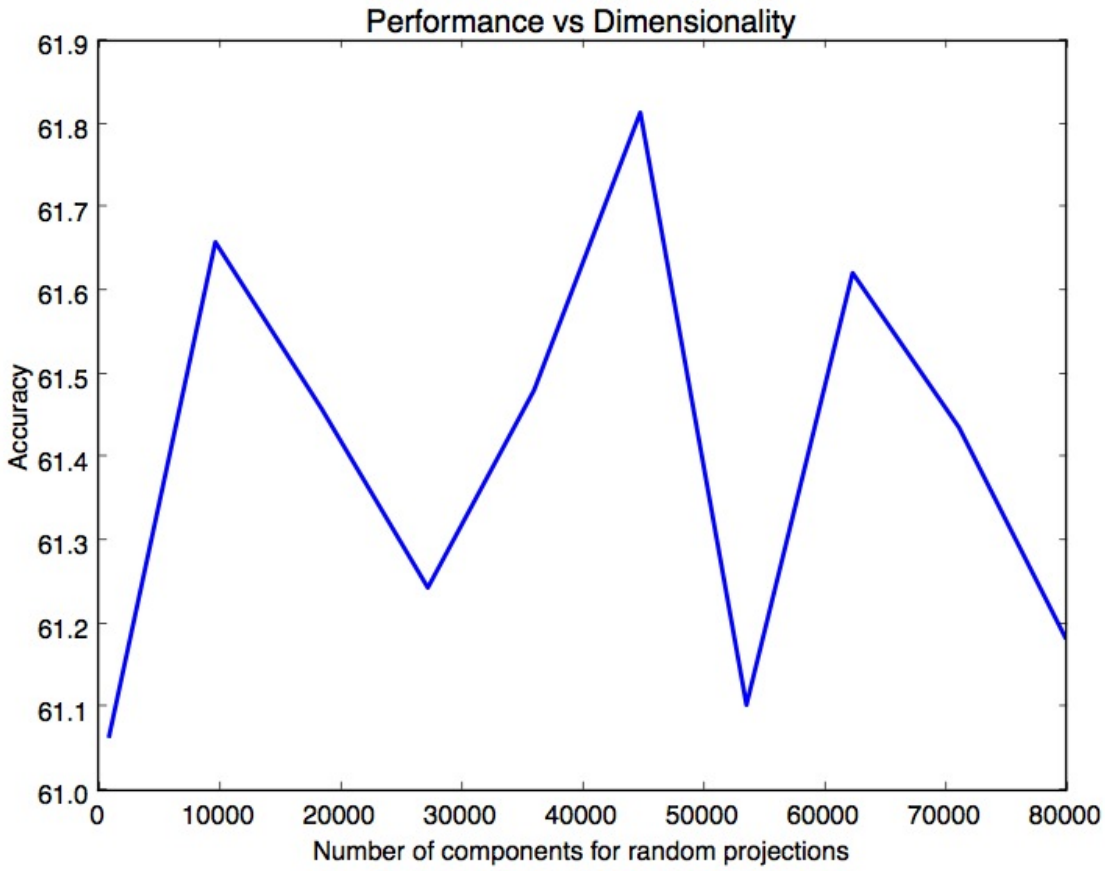
5e) We will compare our worst performing classifier, kNN to one of the best performing LinearSVC. The curse of dimensionality really hits hard on kNN as the number of necessary data points needed to calculate for good estimation grows exponentially and accuracy drops off as a result. Also, if the variables are discrete, then calculating Euclidean distance doesn't make most sense to predict a label of a movie. SVM is better suited to handle these because of the kernel trick and the regularization terms.

5f) We chose to very dimensionality in two ways for this question. The first way is with hashing vectorizer like we did previously, by varying the number of features below.



Ran Linear SVC with 5000 features Accuracy: 49.42%
Ran Linear SVC with 20000 features Accuracy: 57.03%
Ran Linear SVC with 80000 features 60.78%

As you can see accuracy definitely increases as we add more features (words) to our model. However, using random projections we got a slightly more different result when varying the number of components. We wanted to test using random projections after vectorizing our movie features and varying the degree of components for the sparse random projections module. When set to 'auto', the default dimensionality is going from (54000, 119030) to (54000, 9340) after the projections are applied to the vector. Using this as a gauge, we set our number of components to be spaced out from 1000 to 8000 equally spaced for to get 10 different component sizes. We ran it overnight and achieve some different results than hashing vectorizer. This time using this type of dimensionality reduction the Linear SVC model is somewhat robust. All of the accuracies stayed within a range of 60-61%.



5g) From the documentation, all sklearn classifiers are capable of multi class classification. One can individually modify the multi class behavior of a learner by changing the generalization error or handling computational resources. Naive Bayes and kNN are all inherently multi class, in the sense that when described in lecture the examples fit seemingly with non binary data (even though we saw spam and not spam in class). All of the other classifiers except for SVC use a one vs rest approach. This means that as we compare the N classes, we classify in the binary sense between N and the rest of N-1 classifiers. Then we chose the class with highest confidence after all combinations. From the documentation: "...SVC multi-class mode is implemented using one vs one scheme while LinearSVC uses one vs the rest".