THE JOURNEY FROM DATA TO MODEL

A Beginner's Guide to Applied Machine Learning

Justin Laughlin & Stuart Sonatina
University of California, San Diego

COGS 118A - Spring '17

Final Project

Contents

Abstract

Through the exploration of three different classification problems, the role of feature engineering was shown to be of greater importance than any other step in the process of training a classification model. First, the famous Titanic dataset was analyzed, manipulated, then used to train and test a random forest classifier with a success rate of 82%. Second, a Multinomial Naïve Bayes, along with a Stochastic Gradient Descent classifier were trained on a text dataset consisting of over 5000 text messages classified as "spam" or "ham". The F-scores of both classifiers were 0.94 and 0.97, respectively. Finally, the Facial Recognition Technology (FERET) database was skimmed for all forward facing subjects, then cropped down to the area of interest in order to train a Gaussian Naïve Bayes classifier that identified subjects wearing glasses. Principal Component Analysis (PCA) was utilized to reduce the dimensions of the problem from over 9600, to 120. The maximum F-score achieved was 0.91.

Introduction

Coming up with features is difficult, time-consuming, and requires expert knowledge. When we look at people doing "applied machine learning" most of that effort is designing features.

Andrew Ng, Machine Learning and AI via Brain simulations [1]

When first studying machine learning, many students mistakenly assume that the choice of algorithm is the most critical decision. While this is certainly an important factor, in practice the choice of classifier for a particular problem is typically clear; the most challenging yet arguably most important element in a successful statistical model is a comprehensive understanding of the data and feature choice/engineering which reflects this understanding. This paper shall explore several different datasets and classifiers but the primary emphasis will be on pragmatic techniques and methods involved in the process of solving real problems: data wrangling, data exploration, feature choice/engineering, dimensionality reduction, and appropriate classifier choice. Beyond the fact that it is a final project, the fundamental purpose of this paper is to educate the authors and hopefully assist the reader in their own personal journey for knowledge.

Methods

Software

All experiments were performed in Python using jupyter [2] along with the following packages: matplotlib [3], numpy [4], pandas [5], scikit-learn [6], scipy [8], and seaborn [9]. The jupyter notebooks used for each experiment can be found at github.com/justinlaughlin/cogs118a-final. Although this paper is intended to be comprehensive enough to be read as a standalone document - the authors highly recommend referencing the notebooks during reading to better understand the implementation.

Data Wrangling

The first step in any statistical modeling problem is to wrangle the data. This might at first glance seem a trivial task, but properly organizing the data into matrices for processing can make an enormous difference in the efficiency of training a classifier later on. Often datasets come with missing values, useless features, or even errors. These must all be corrected, eliminated, or dealt with in their own specific way. A popular tool for data wrangling is the Python package pandas. The process typically involves:

- Changing binary variables to numbers so that a statistical classifier can understand them, e.g.
$$\{male, female\} \rightarrow \{0, 1\}$$
- One-hot encoding categorical variables with more than two values where ordering has no importance, e.g. a feature such as $color = \{red, green, blue\}$ would be converted to the following:

Table 1: An example of one-hot encoding

Car	Red	Green	Blue
1	1	0	0
2	0	1	0
3	0	1	0
4	0	0	1

Note: In some categorical variables, ordering is an important feature and one-hot encoding is not recommended. For example in the titanic dataset, *Pclass* is a categorical variable representing socio-economic status which ranges from one to three. Unlike the example with colors above, ordering is relevant in this feature and one-hot encoding would remove valuable information.

- Manually removing obviously non-useful (or "noisy") features. This step is complemented by the use of automated dimensionality reduction algorithms such as Principle Component Analysis (PCA).
- Properly correcting errors in the dataset, such as missing values. e.g. simply replacing a "not a number" (NaN) with a zero might lead to erroneous results.

Data Exploration

Once the data has been sufficiently wrangled, one can generate a number of plots to explore the data and gain a greater understanding of the problem, and possibly an intuition for what features will be most important for that specific problem. This step in the process is crucial, as it is what separates a data scientist who merely sees machine learning as a "black box" from an excellent data scientist who can fine-tune a model based on strong intuitions of the data.

Exploring the data using creative charts and tables allow for different features to "jump out" depending on what is emphasized. This can lead to useful feature engineering that brings important aspects into the light. For example, figure 1 not only immediately highlights the enormous correlation between survival and being a woman, but also shows the high survival rate among male children. This might lead to a separation of adults and children since different features might play a different part in their survival. Having a parent/child might have a positive correlation with survival for children, but a negative one for adults.

Feature Selection/Engineering

Since the methods of machine learning can be applied to so many different types of problems, it's difficult to generalize the process of feature engineering. In an image dataset, feature engineering might entail looking for groups of pixels which create certain shapes, dense contours, or are similar to a certain color. This would require designing an algorithm that identifies these features once they're designed. In a text classification problem, a common technique is going beyond seeing each word as a feature and using word pairs as features as well. The word "dog" might imply the text is about animals, unless it's immediately following the word "hot". Using pairs, or "grams", would allow the classifier to see special meanings like that.

In an image classification problem, simple image manipulation such as cropping can be thought of as feature selection since thrown-out pixels are features that are being ignored. In the third experiment, the faces are not in the same position between each image, so the area of interest had to be chosen based on the nose, mouth, and eye coordinates. This led to a window that could be adjusted depending on if the feature being targeted was the glasses, mustache, or beard.

Classifier Choice

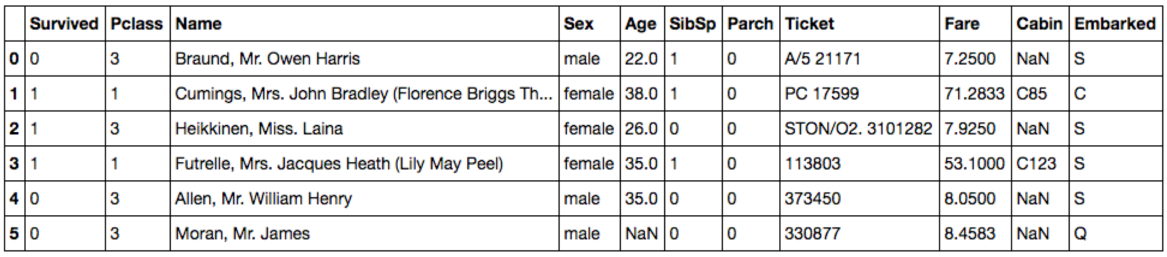
As noted earlier, the specific classifier chosen for each problem is rarely the main source of success. While it's possible to select a classifier that is completely wrong for the problem, once a classifier is declared "good enough", choosing a slightly better one might result in marginal gains, if any at all. The most dramatic improvements come from feature engineering with a good hypothesis as to why certain features are necessary, or what can be done to create better features.

Experiment 1: Titanic Dataset

Loading in Data

The first dataset chosen was the famous titanic dataset as it is quite straightforward and provides a simple framework to demonstrate fundamental data science techniques [10]. Illustrated below is a table of the first six passengers of the titanic dataset: whether or not a passenger survived is the class label, the other columns represent features [2].

Table 2: Features of first six passengers tabulated using a pandas dataframe [5]



	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
5	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q

Data Wrangling

Once the data has been loaded into Python, it must be cleaned (see Methods: Data Wrangling above). The following data wrangling decisions for this particular dataset and the thought processes behind them are presented below; they may be used as an example of how to approach similar problems but keep in mind that there are a multitude of correct methods when approaching a data science problem:

- *Survived* is the class label and therefore was separated into its own dataframe.
- *Pclass* represents socio-economic status and is a categorical variable with three options. Although it could be one-hot encoded it does have a natural ordering and therefore it was left as an integer.
- *Name* in its current state cannot be utilized and must be converted into a useful feature (or features) through feature engineering. There are a certain number of titles that appeared frequently which may provide insight, including "Dr", "Master", and "Rev". These were one-hot encoded into individual features and are discussed in greater detail in the Feature Engineering section.
- *Sex* was changed from $\{male, female\}$ to $\{0, 1\}$.

- *Age* contained many NaN values. Deleting these would drastically reduce the number of passengers, so they were converted to an intuitively reasonable value: the mean of all ages. Although this action does increase the bias of the estimator, in this scenario it was preferable to the alternative of deleting large quantities of samples.
- *Ticket* was removed as no obviously useful information could be extracted from it (although with some additional information on ticket number meaning, some creative feature engineering could perhaps result in some meaningful features).
- *Cabin* was also removed. In theory this could be a potentially valuable feature as certain cabins were much closer to the life rafts on the Titanic, however there are far too many missing data points (represented as NaNs) which significantly detracts from the utility of keeping this feature.
- *Embarked* represents the city each passenger departed from (S: Southampton, C: Cherbourg, Q: Queenstown), which has no natural ordering and therefore was was one-hot encoded.

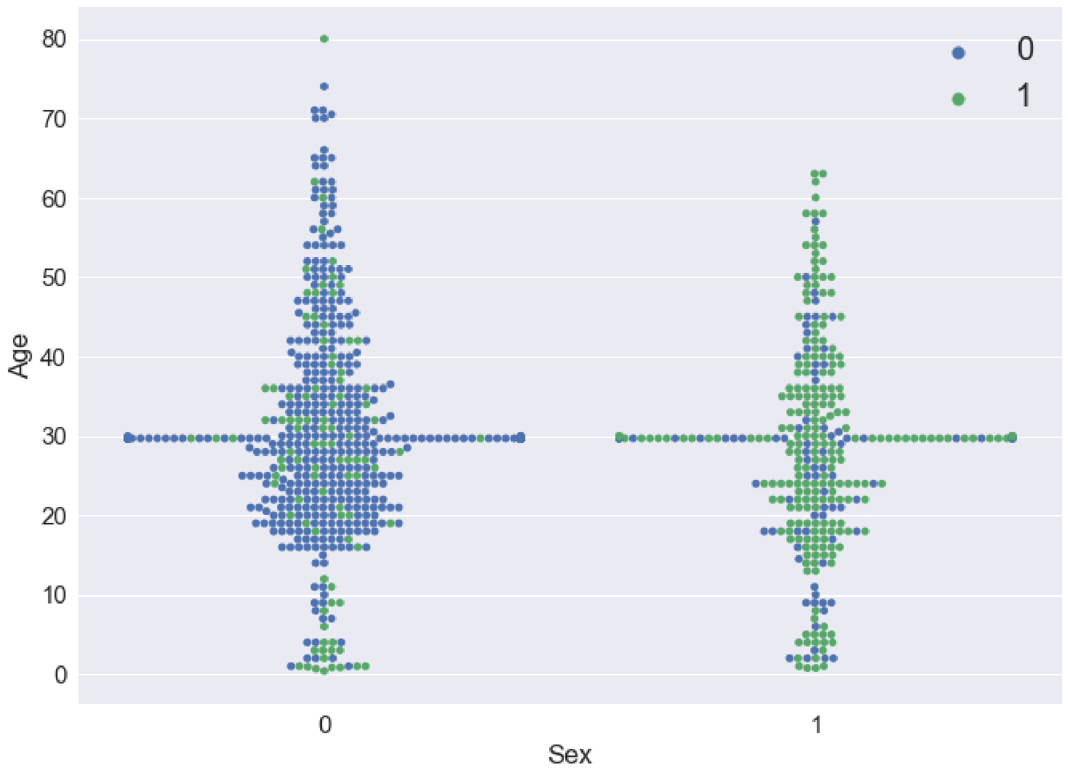


Figure 1: Dependence of age and sex on survival (0: male, 1: female)

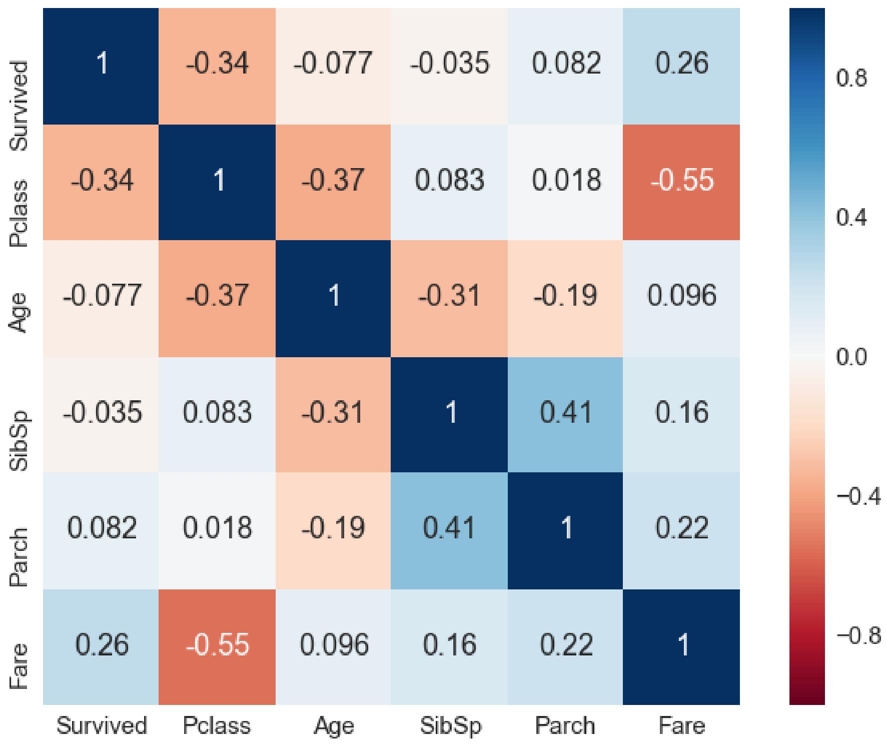
Exploring Features Using a Correlation Matrix

A simple plot which can aid in the exploration of the dataset is a correlation matrix. There are several different metrics used for correlation but typically Pearson's correlation coefficient, $\rho(X, Y)$, is used. The correlation between two variables, X and Y is defined as the covariance between them divided by the product of their standard deviations. Dividing by the product of standard deviations essentially non-dimensionalizes and normalizes the covariances so they may be compared. The mathematical expression is as follows:

$$\rho(X,Y) = \frac{cov(X,Y)}{\sigma_X \sigma_Y} = \frac{E\left[(X-\mu_X)(Y-\mu_Y)\right]}{\sigma_X \sigma_Y}$$

In Python the correlation matrix of a dataframe can easily be plotted using the following single line of code:

```
# df is the dataframe
sns.heatmap(df.corr(),square=True,annot=True,cmap="RdBu")
```



	Survived	Pclass	Age	SibSp	Parch	Fare
Survived	1	-0.34	-0.077	-0.035	0.082	0.26
Pclass	-0.34	1	-0.37	0.083	0.018	-0.55
Age	-0.077	-0.37	1	-0.31	-0.19	0.096
SibSp	-0.035	0.083	-0.31	1	0.41	0.16
Parch	0.082	0.018	-0.19	0.41	1	0.22
Fare	0.26	-0.55	0.096	0.16	0.22	1

Figure 2: Titanic dataset correlation matrix

A correlation matrix for the titanic dataset is shown in fig. 2. The diagonal should always consist of ones as each variable is always perfectly correlated with itself. The matrix is also symmetric as the same features are represented in both the rows and columns. What purpose does the correlation matrix serve for the data scientist? Intuitively, features which are highly correlated with "survival" are probably important features. Features which are highly correlated with each other may potentially be repetitive; the presence of several highly correlated features might indicate the need for a dimensionality reduction technique such as PCA.

The feature pairs with the highest correlation according to fig. 2 are $\{Pclass, Fare\}$, $\{SibSp, Parch\}$, and $\{Pclass, Age\}$. The high negative correlations of $\{Pclass, Fare\}$ and $\{Pclass, Age\}$ are reasonably intuitive: individuals with a higher socio-economic class ($Pclass = 1$) will likely pay higher fares and be older. However the relation between features $SibSp$ and $Parch$ are less intuitive. Both features group entities together that have drastically different interpretations. $SibSp$ is the total number of siblings *and* spouses and individual has on board, while $Parch$ is the total number of parents *and* children an individual has on board.

Clearly a young individual with $Parch = 2$ has two parents on board as they cannot be parents themselves. While an older individual with $Parch = 2$ could either have two children or two elderly parents on board. The same value for $Parch$ has taken on two distinct meanings which indicates it may be useful to engineer a new feature.

Feature Engineering

Parents vs. Children: Extracting More From *Parch*

It is hypothesized that many parents might sacrifice their own life for the sake of their progeny. Fig 1 has illustrated how the safety of women and children was given priority during the disaster - but what about children with parents versus those without? Perhaps a child traveling with a family friend is less likely to survive than a child traveling with both of their parents. Let us engineer two new features: $ChwPar$ and $ParwCh$ which indicate the number of parents a child has on board and the number of children a parent has on board, respectively (although ParwCh could also occasionally represent elderly parents of adults, for the sake of brevity we shall ignore this scenario).

```
# If individual is under 18 and Parch>=1 individual is a child with parent(s)
    on board
Xdf["ChwPar"] = ((Xdf["Age"]<=18) & (Xdf["Parch"]>=1)) * Xdf["Parch"]
# If individual is over 18 and Parch>=1 individual is a parent with children on
    board
```

```
Xdf["ParwCh"] = ((Xdf["Age"]>18) & (Xdf["Parch"]>=1)) * Xdf["Parch"]
# Drop the feature "Parch" as it has been split into two different features
Xdf = Xdf.drop(["Parch"], axis=1)
```

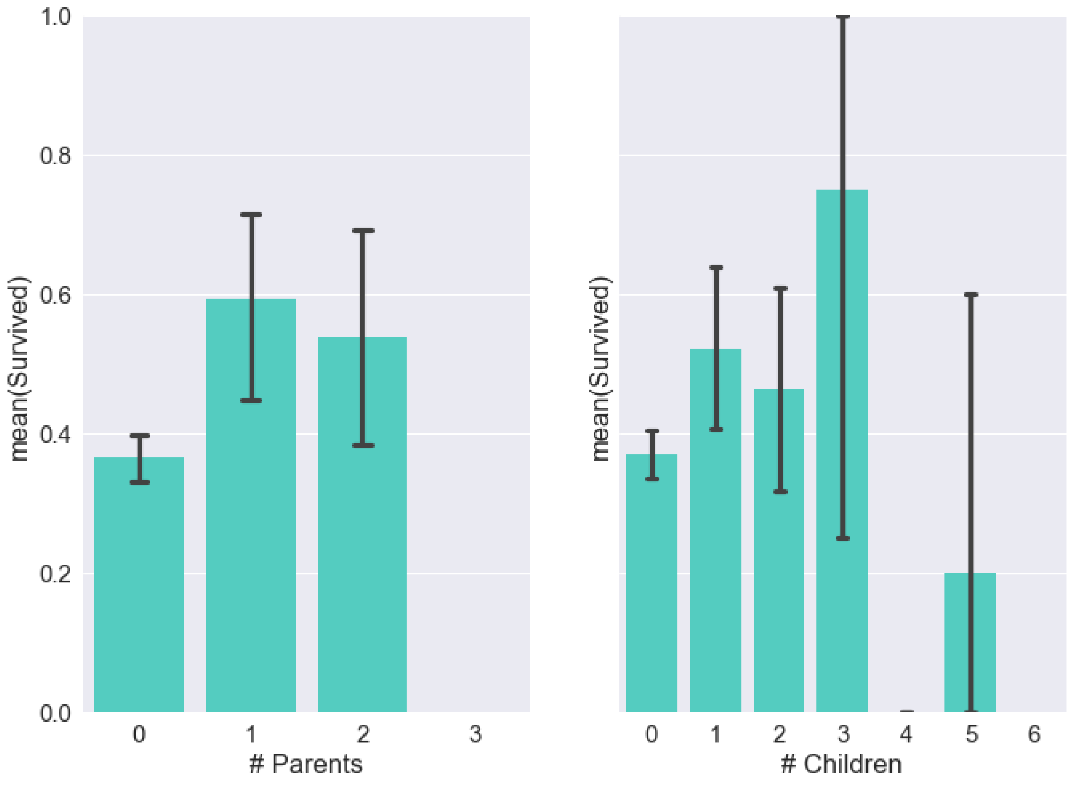


Figure 3: Survival rate of children with parents [left] vs. parents with children [right]

Extracting Titles From *Name*

As mentioned in the data wrangling section, titles were extracted from $Name$ as the title of an individual may be a good predictor of survival rate. Since each title in $Name$ begins with ", " and ends with a ". ", titles can easily be extracted using the following lines of code:

```
# Different titles with frequency of each
titles = df['Name'].str.split(", ", expand=True)[1].str.split(". ",
    expand=True)[0]
titles.groupby(titles).count()
```

The results are displayed in table 3.

Table 3: Titles of individuals and their respective number of occurrences

Capt	1
Col	2
Don	1
Dr	7
Jonkheer	1
Lady	1
Major	2
Master	40
Miss	182
Mlle	2
Mme	1
Mr	517
Mrs	125
Ms	1
Rev	6
Sir	1
the Countess	1

Mr, Mrs and Miss are the three most common titles, however this information is most likely already captured by *Sex* and *SibSp*. The next three titles with the greatest number of occurrences are Master, Dr, and Rev; these three titles were one-hot encoded as their own features. Table 4 shows the survival rates of these three titles compared to the mean of all other individuals. Some interesting observations can be made. Rev is short for "Reverend", which is typically a title given to leaders of some Christian churches; perhaps these Reverends were more selfless than the average individual as not a single one survived. Individuals with the title "Master" were far more likely to survive; "Master" was a term given to young boys who were also the eldest son [13].

Table 4: Survival rate based on title

	Number of Individuals	Mean Survival
Master	40	0.575
Dr	7	0.429
Rev	6	0.000
No Rare Title	838	0.384

After all the data wrangling and feature engineering has been completed, table 5 shows a subset of the resulting dataframe before training a classifier. These features are what were actually used to develop a model for predicting survival in the Titanic sinking.

Training Classifier

Due to their ease of use, immunity to overfitting, and simplicity of concept, Random Forest was chosen as a starter classifier for the Titanic dataset. The algorithm, a simple

Table 5: Cleaned dataframe with additional engineered features

	Pclass	Sex	Age	SibSp	Parch	Fare	EmbCherb	EmbQueen	EmbSouth	ChwPar	ParwCh	Master	Dr	Rev
0	3	0.0	22.0	1	0	7.2500	0	0	1	0	0	0	0	0
1	1	1.0	38.0	1	0	71.2833	1	0	0	0	0	0	0	0
2	3	1.0	26.0	0	0	7.9250	0	0	1	0	0	0	0	0
3	1	1.0	35.0	1	0	53.1000	0	0	1	0	0	0	0	0
4	3	0.0	35.0	0	0	8.0500	0	0	1	0	0	0	0	0

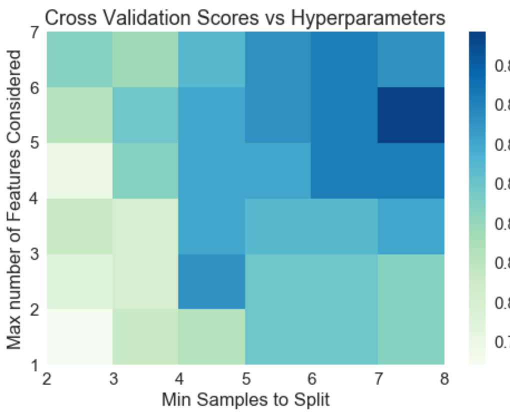
decision tree classifier used on subsets of the data, was implemented by utilizing a function from the scikit-learn package: RandomForestClassifier. [6]

The hyperparameters to optimize for were the max number of features considered for each split, and the minimum number of samples required to split again. The lower the number of features considered, the lower the bias for the entire classifier. The variance can then be lowered by utilizing a large number of these trees an averaging out their decision boundaries. The size of the nodes is determined by the minimum number of samples required to split again. This allows for more or less precise decision boundaries, which in turn affects bias and variance as well. The few the number of samples left in each node, the more over-tuned each classifier can be. This lowers bias for each weak classifier, but after averaging over many classifiers (in this case 1000), that bias can be eliminated, while still preserving the low variance from such over-tuned classifying.

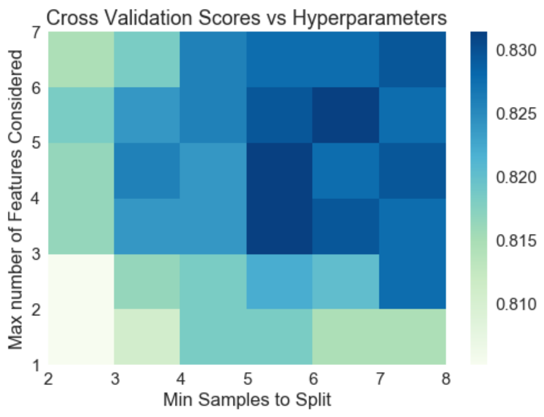
Results

As figure 4 shows, the benefits of tuning the hyperparameters to specific optimal values are not very large. There appears to be a gradient towards a less-tuned weak classifier (with a large number of features considered in the 80/20 split), but it's difficult to know how much of this variation is random noise with such small deviation. The difference between the optimal parameters and the worst ones is still only about 3-4%.

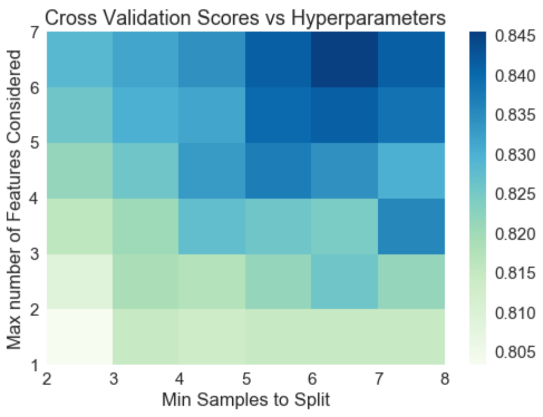
The 80/20 split appears to have the clearest gradient with a high score of around 84%. A score of 82% seems to be the maximum most models have achieved on this popular dataset, and the error is likely on the order of 2 percent, so these results are satisfactory. The next two datasets will illustrate the range of possible accuracies in different machine learning problems.



(a) Train/Test: 40/60



(b) Train/Test: 60/40



(c) Train/Test: 80/20

Figure 4: Results of Random Forest Classifier grid search on the Titanic dataset

Experiment 2: Spam vs Ham Dataset [Text Classification]

Introduction

As a short foray into another type of classification, a text dataset was selected in order to try out two classic text classifiers: Naïve Bayes, versus the versatile Support Vector Machine (SVM). The dataset is a collection of 5572 real text messages labeled as "spam" or "ham", indicating their purpose as soliciting or not, respectively. [11]

Before any classifier can be trained on a text dataset, the size of the feature set must be handled. Looking at a collection of documents as a "bag of words", the number of features is equal to the number of distinct words in the set. With a simple dataset of about 10,000 samples, 100,000 unique words would mean storing 10,000 x 100,000 float32s, which comes out to about 4 GB. This would need to be stored in RAM since the computer would be doing operations with this array. Obviously, this is beyond the capability of most computers to handle easily.

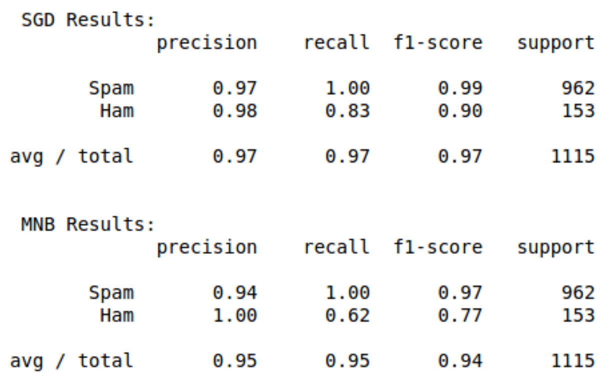
Since most of the documents would only contain a small fraction of those unique words, these matrices would be quite sparse. SciPy's "sparse" module takes advantage of this fact, and hence the actual size of the resulting feature vectors are quite manageable. The next step is to build a dictionary of features (tokenizing), normalize for frequency by dividing document frequency into the frequency of each sample's words, then training a classifier using these transformed matrices. This is all covered extensively in scikit-learn's user guide for working with text data, which was followed in order to train the two classifiers present in this experiment. [7]

Beginning with Multinomial Naïve Bayes (MNB) and immediately following with Stochastic Gradient Descent (SGD), both classifiers were trained on 80% of this transformed dataset and compared to each other using their F1-score on a 20% test set. MNB uses the Bayes theorem, which assumes the frequency of each feature (in this case, word) is independent from the existence of other features. SGD, on the other hand, is a linear classifier that estimates the gradient of the loss each sample at a time. The purpose of this experiment was merely to observe the power of classifiers to model text, so no further exploration was performed.

Results

Table 6 shows the impressive ability of both classifiers to identify a spam text message after being trained on only 4000 messages. Again the choice of the classifier, while non-negligible, has little effect on the success of the model. The SVM did score a few points higher than the MNB, but this small of a deviation is difficult to separate from random noise or chance. The receiver operating characteristic curve is another way to observe the true and false positive rates, and differentiate between the two classifiers. The SGD classifier clearly performed better on this dataset, but it's impossible to extrapolate this performance to another dataset with different features. It's important to try different classifiers on the way to training a model for each unique dataset, but the focus shouldn't be placed on classifier choice as much as feature engineering.

Table 6: Precision, Recall, and F-Scores for both classifiers



```
SGD Results:
             precision    recall  f1-score   support

       Spam       0.97      1.00      0.99       962
        Ham       0.98      0.83      0.90       153

avg / total       0.97      0.97      0.97      1115

MNB Results:
             precision    recall  f1-score   support

       Spam       0.94      1.00      0.97       962
        Ham       1.00      0.62      0.77       153

avg / total       0.95      0.95      0.94      1115
```

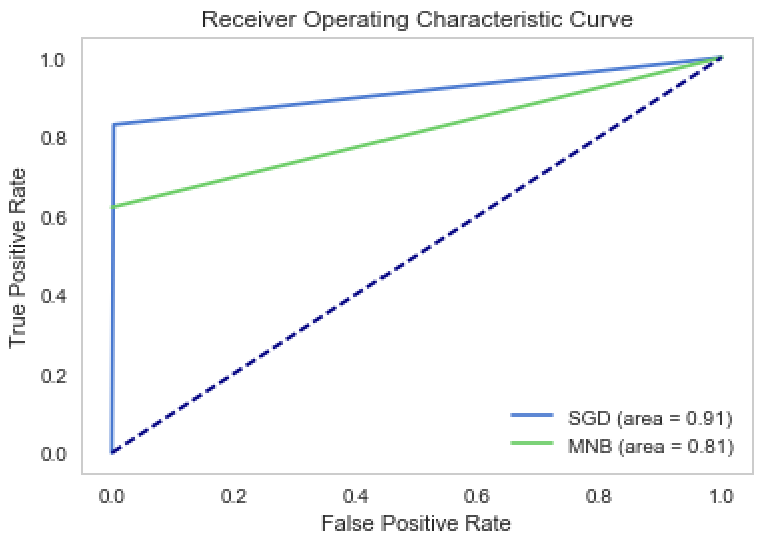


Figure 5: Receiver Operating Characteristics Curve of the Stochastic Gradient Descent and Multinomial Naïve Bayes Classifiers

Experiment 3: FERET Image Dataset

Introduction

A standard RGB image consists of three two-dimensional matrices of unsigned 8 bit integers (ranging from 0 to 255). Each of the three matrices represents the intensity of red, green, and blue, respectively. In order to transform each image into a "sample point" the matrices must be flattened so that each individual pixel is a feature. To simplify the data, the images were converted to grayscale in order to reduce the number of features to precisely the number of pixels - in this dataset n_pixel $= 256 \times 384 = 98,304$. Classifying an extremely high-dimensional dataset such as this directly without feature engineering or dimensionality reduction can lead to significant computational costs and low performance.

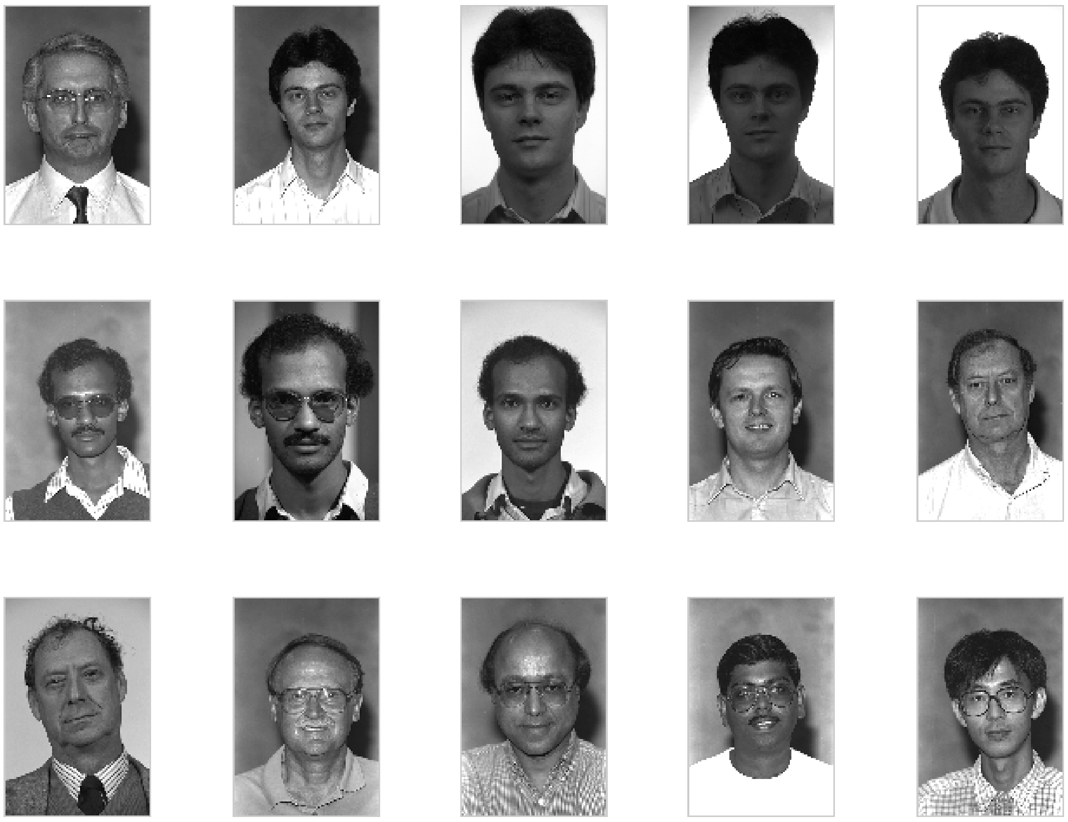


Figure 6: 15 faces from the FERET Image Dataset

PCA and Eigenfaces

In most image classification problems, the useful feature-space is far more constrained than what having tens or hundreds of thousands of features might suggest. For example, in this dataset every picture consists of a face looking forward; interesting features exist at a higher level than the individual pixel: features such as contours, clusters of similar colors, pixel intensity gradients, and much more. Principle Component Analysis (PCA) is a useful feature engineering technique which can reduce the dimensionality of the problem substantially. The primary idea behind PCA is to first find the direction with the highest variance; this becomes the first eigenvector or feature. All directions orthogonal to this first eigenvector are then searched to find the direction with the second highest variance. This process is continued until $nPCA$ eigenvectors are constructed, which are then used as features in the reduced dimensionality feature space. These eigenvectors represent basic ideas of faces from which actual faces can be constructed using a simple linear combination: as such they are called eigenfaces.

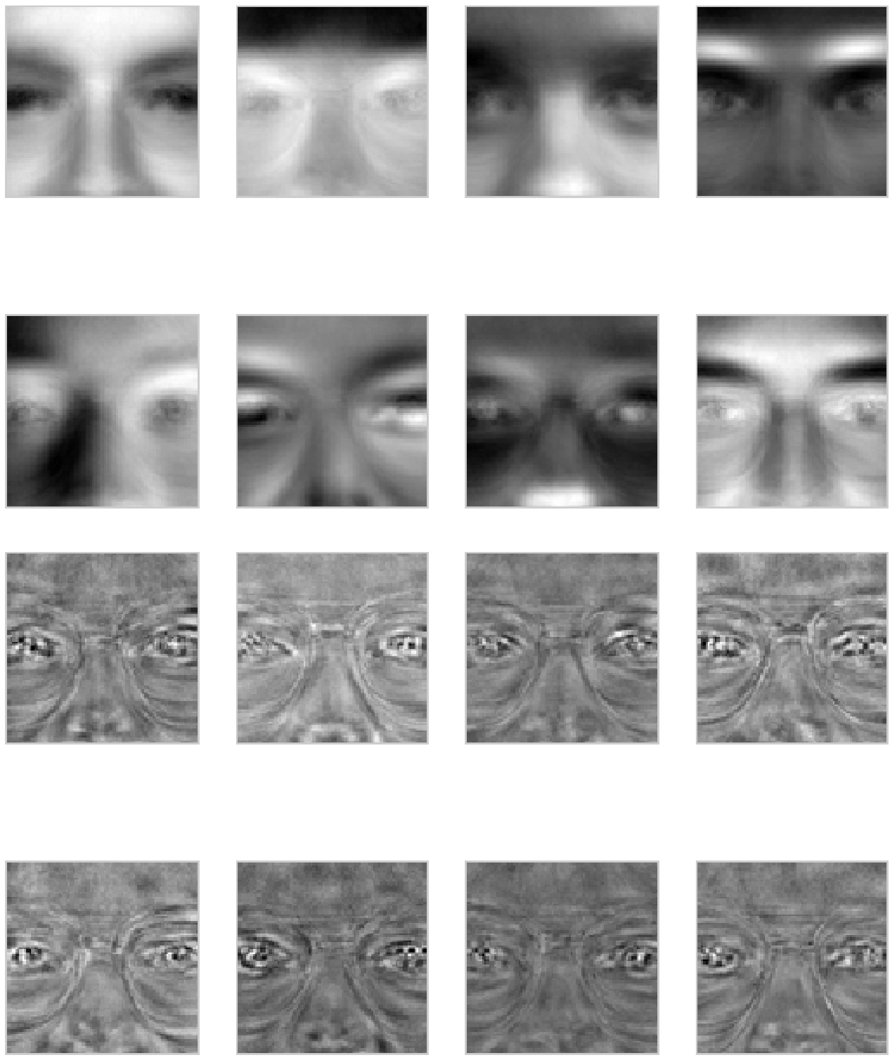


Figure 7: First eight eigenfaces vs last eight eigenfaces

Figure 7 shows the first and last eight eigenfaces when $nPCA$ was set to 120. Because PCA begins by finding the directions with the highest variance the first eigenfaces are quite distinct and various facial features can be recognized. However the later eigenfaces are bases in directions with much lower variance and therefore start to become more repetitive and less useful. To visualize the variance captured by each eigenface, it is useful to plot the eigenvalue for each eigenface (figure 8). One can observe that the usefulness of additional eigenfaces exponentially diminishes. This is excellent, as what was once a 98,304 dimension feature space has been reduced by almost three orders of magnitude!

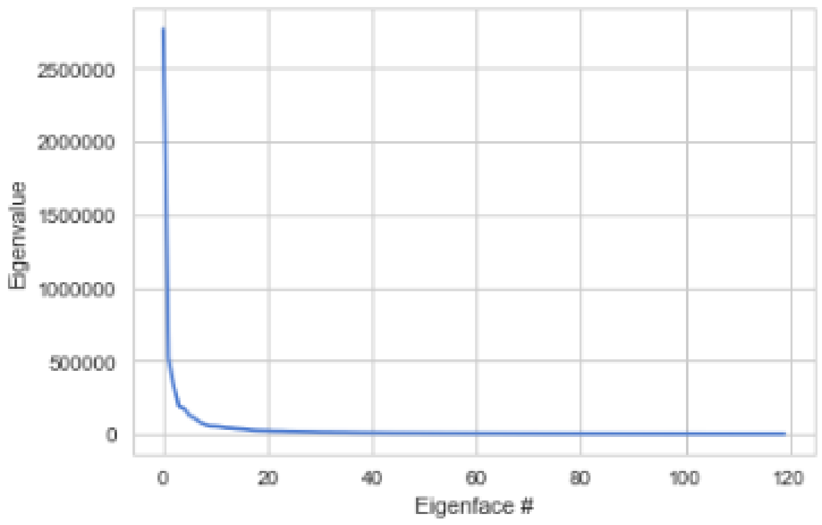


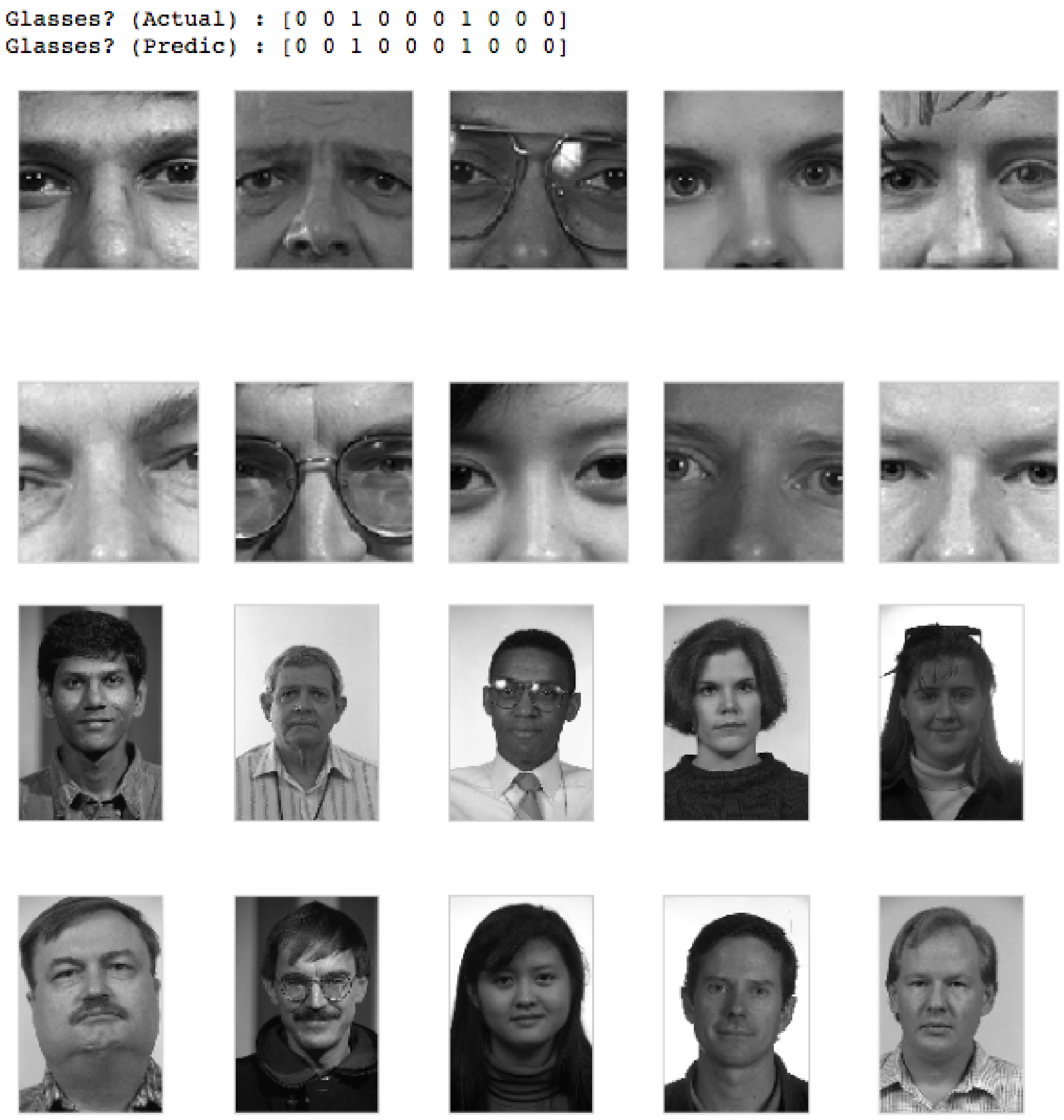
Figure 8: Eigenvalues vs. eigenfaces

Modifying Images

For the purposes of this experiment the authors decided to classify individuals wearing glasses vs. those not wearing glasses. The dataset contained 1207 correctly labeled images. Prior to PCA being performed, the images were modified so only the useful portions were included; i.e. identifying where the eyes were and reducing the image to a rectangular section surrounding them (figure 9). An immense benefit of the FERET dataset is that there is an abundance of information such as coordinates of facial features so no algorithms were needed to identify areas surrounding an individual's eyes. Any additional pixels outside of this region are most likely irrelevant to the problem and only add difficulties for the classifier. The number of pixels extracted for each picture was kept constant at 9600 so that their feature spaces pre-PCA were the same.

Results

As table 8 shows, the model performed exceptionally well. For a simple implementation of a Gaussian Naïve Bayes (GNB) on small, greyscale images to identify glasses, the performance was impressive. Initially, the different sizes, orientations, and colors of the faces seemed too varied to produce useful results, but after careful cropping and image selection, the classifier was able to pick up the glasses with much more precision. The similar shape of most of the glasses in this set (taken in the mid '90s) helps as well. The impression of the glasses on the eigenfaces in figure 7 shows how similar in shape and



```
Glasses? (Actual) : [0 0 1 0 0 0 1 0 0 0]
Glasses? (Predic) : [0 0 1 0 0 0 1 0 0 0]
```

Figure 9: Images automatically cropped to only include eyes [top] vs full images [bottom]

location most of the glasses were.

Due to the complexity of processing this dataset, only one classifier was trained on the resulting eigenvector array. This allowed for only a simple comparison between different training/testing splits. Table 8 highlights the similarities between different split ratios. Again, the hard (and salient) part – feature engineering – has been done already, so small changes in classifiers, test/train splits, or other parameters seem to have small effects. The problem is simply easier once the images had been oriented and cut in a useful way.

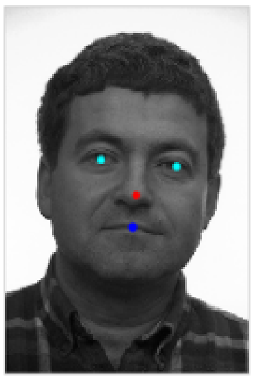
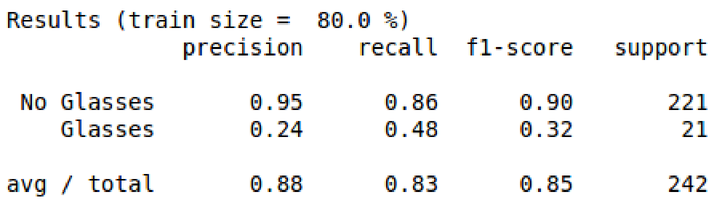


Figure 10: Coordinates of eyes, nose and mouth used to locate the range of interest

Table 7: Precision, Recall, and F-Score Gaussian Naïve Bayes on original, uncropped images

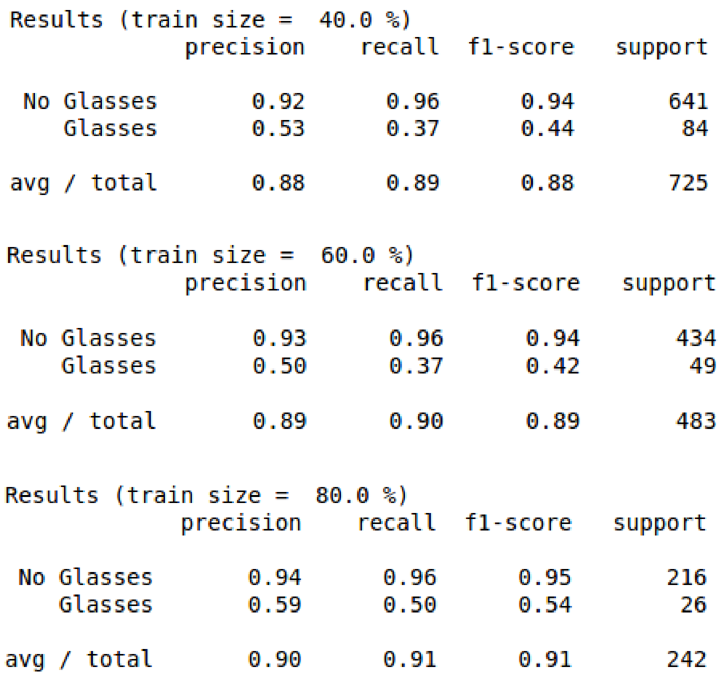


```
Results (train size =  80.0 %)
             precision    recall  f1-score   support

 No Glasses       0.95      0.86      0.90       221
    Glasses       0.24      0.48      0.32        21

avg / total       0.88      0.83      0.85       242
```

Table 8: Precision, Recall, and F-Score of Gaussian Naïve Bayes on the cropped FERET dataset



```
Results (train size =  40.0 %)
             precision    recall  f1-score   support

 No Glasses       0.92      0.96      0.94       641
    Glasses       0.53      0.37      0.44        84

avg / total       0.88      0.89      0.88       725

Results (train size =  60.0 %)
             precision    recall  f1-score   support

 No Glasses       0.93      0.96      0.94       434
    Glasses       0.50      0.37      0.42        49

avg / total       0.89      0.90      0.89       483

Results (train size =  80.0 %)
             precision    recall  f1-score   support

 No Glasses       0.94      0.96      0.95       216
    Glasses       0.59      0.50      0.54        26

avg / total       0.90      0.91      0.91       242
```

Acknowledgements

Portions of the research in this paper use the FERET database of facial images collected under the FERET program, sponsored by the DOD Counterdrug Technology Development

Program Office [15, 16].

Conclusion

After witnessing the performance of several types of classifiers on a few significantly different categories of classification problems, the rhetoric on the importance of feature engineering rings true. Most notably, utilizing exactly the same classifier, PCA dimension reduction, and hyperparameters, the ability of the GNB to correctly identify a subject wearing glasses more than doubled after feature engineering. This kind of improvement is not likely to result from a clever algorithm or finely tuned decision boundary.

Feature engineering is, in essence, a redefinition of the problem. Classifying a dataset using nothing but features in a matrix is a narrow, precisely-defined problem, ideal for solving with a computer. It is a mistake, however, to rely on the computer to see patterns or irregularities that aren't obvious or easily formulated. It takes a keen eye, and insight born of experience, to see where the hidden correlations are. To deploy novel methods of data visualization in order to intuit relationships and associations. To be able to gain a level of understanding of the given information through exploration of the data, then turn that given information into new sets of useful features that highly correlate with the class of the sample.

This project was a short cruise through the waters of applied ML. The famous Titanic dataset was used as an introduction to the entire procedure from data analysis and exploration, to feature engineering, to training a model and testing results. It was followed by a dip into the world of text classification, one of the largest applications of machine learning in the real world, as Google has made abundantly clear over the past two decades. Finally, and most extensively, over a thousand images were processed and analyzed to develop rudimentary computer vision for identifying subjects wearing glasses. These three experiments allowed the authors to see ML applied to unique problems, then analyze which parts of the process yielded the largest returns. With any luck, you, dear reader, will have gained a bit of insight as well.

Bibliography

[1] DB Tsai. "2013-08-01 Prof. Andrew Ng: "Deep Learning: Machine learning via Large-scale Brain Simulations"." *YouTube*. YouTube, [Online; accessed December 20, 2018].

[2] Thomas Kluyver, Benjamin Ragan-Kelley, et al. Jupyter Notebooks - a publishing format for reproducible computational workflows, Positioning and Power in Academic Publishing: Players, Agents and Agendas, 87-90. 10.3233/978-1-61499-649-1-87, http://ebooks.iospress.nl/publication/42900/ [Online; accessed December 20, 2018].

[3] John D. Hunter. Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering, 9, 90-95 (2007), DOI:10.1109/MCSE.2007.55, http://aip.scitation.org/doi/abs/10.1109/MCSE.2007.55/ [Online; accessed December 20, 2018].

[4] Stéfan van der Walt, S. Chris Colbert and Gaël Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation, Computing in Science & Engineering, 13, 22-30 (2011), DOI:10.1109/MCSE.2011.37, http://aip.scitation.org/doi/abs/10.1109/MCSE.2011.37 [Online; accessed December 20, 2018].

[5] Wes McKinney. Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference, 51-56 (2010), http://conference.scipy.org/proceedings/scipy2010/mckinney.html/ [Online; accessed December 20, 2018].

[6] Fabian Pedregosa, Gaël Varoquaux, et al. Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research, 12, 2825-2830 (2011), http://jmlr.org/papers/v12/pedregosa11a.html/ [Online; accessed December 20, 2018].

[7] http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html

[8] Jones E, Oliphant E, Peterson P, et al. SciPy: Open Source Scientific Tools for Python, 2001-, http://www.scipy.org/ [Online; accessed December 20, 2018].

[9] Michael Waskom, Olga Botvinnik, et al. seaborn: v0.7.1 (June 2016), https://zenodo.org/record/54844/ [Online; accessed December 20, 2018].

[10] https://www.kaggle.com/c/titanic/data

[11] https://www.kaggle.com/uciml/sms-spam-collection-dataset

[12] https://github.com/bytefish/facerec

[13] Leslie Dunkling, Dictionary of Epithets and Terms of Address (2012).

[14] Shakhnarovich Gregory, El-Yaniv Ran, Baram Yoram. Smoothed Bootstrap and Statistical Data Cloning for Classifier Evaluation. 2001, http://ttic.uchicago.edu/ gregory/papers/icml2001.pdf

[15] P.J. Phillips, H. Wechsler, J. Huang, P. Rauss. "The FERET database and evaluation procedure for face recognition algorithms", Image and Vision Computing J, Vol. 16, No. 5, pp. 295-306, 1998.

[16] P.J. Phillips, H. Moon, S.A. Rizvi, P.J. Rauss. "The FERET Evaluation Methodology for Face Recognition Algorithms", IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. 22, pp. 1090-1104, 2000.